KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Traffic Analysis Attacks and Defenses on Anonymizing Networks

Lennart Oldenburg

Dissertation presented in partial fulfillment
of the requirements for the degree of
Doctor of Engineering Science (PhD): Electrical
Engineering

Supervisor:
Prof. dr. Claudia Diaz

June 2025

TRAFFIC ANALYSIS ATTACKS AND DEFENSES ON ANONYMIZING NETWORKS

Lennart OLDENBURG

Supervisor:
Prof. dr. Claudia Diaz

Members of the
Examination Committee:
Prof. dr. ir. Jean Berlamont, chair
Prof. dr. ir. Bart Preneel
Prof. dr. Danny Hughes
Prof. dr. Güneş Acar
(Radboud University)
Prof. dr. Florentin Rochet
(Université de Namur)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

June 2025

© 2025 Lennart Oldenburg
Uitgegeven in eigen beheer, Lennart Oldenburg, Kasteelpark Arenberg 10, box 2452, Leuven (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission from the publisher.

Preface

I arrived at COSIC, ready to start my PhD, in the beginning of January 2020. I had imagined the approaching years to be full of stimulating and challenging discussions in front of whiteboards, at shared lunches, or over drinks at night. Academia! And indeed, things were looking to go this way—until, about ten weeks into my PhD, COVID forced everyone to isolate indoors. That the ensuing five and a half years nonetheless were full of joy and connection, meaningful work, and personal as well as professional growth is largely thanks to the amazing people around me, to whom I owe a great debt of gratitude.

First, I would like to thank my promoter, Claudia Diaz. Thank you, Claudia, for promptly inviting me for an interview-seminar presentation when I first contacted you. I was immediately impressed (still am) by the ease and speed with which you dissected my thoughts, your enthusiasm for the field of anonymous communication, and your interest in working with me. Thank you for teaching me the practice of research over these years and your trust and confidence in my developing into an independent researcher.

A PhD does not come into being without a jury. I am deeply thankful to Bart Preneel, Danny Hughes, Güneş Acar, and Florentin Rochet for the time and dedication that you all poured into this task, I have benefitted greatly from the various discussions over the years. Many thanks also to Jean Berlamont for guiding the jury as chair.

To my co-authors over the years, Florian Tschorsch, Güneş Acar, Marc Juarez, and Enrique Argones Rúa: None of the works in this thesis would have been possible without you and I am deeply grateful for everything you all have taught me. Florian, you opened up this academic path for me by supervising my master's thesis and writing this thesis' first paper with me. Güneş, thank you for treating me as a peer, literally three months after I had started at COSIC. Marc, you have been an amazing mentor over these past years, from whom I have learnt a lot. I consider myself fortunate to have worked with you. Enrique, thank you for always being available for in-depth explanations of intricate details of our works and for regularly checking in with me when you passed my office. Thank you to Claudia, Güneş, Marc, and Enrique for understanding my uneasiness with taking airplanes due to ecological reasons and for adjusting publication targets as a

result, I truly appreciate it. I am happy to report that for the purposes of conducting and disseminating the work that became this PhD, I did not take a single flight.

COSIC is a truly amazing place to work at, with colleagues who are simultaneously brilliant and kind. My deepest gratitude to everyone who I was fortunate to spend time with while at COSIC, many of you have become good friends. I want to particularly thank my fellow B01.27ers, Iness, Mahdi, and Tim—thank you for sharing an office with me over the years. Thanks also to the further Privacy Team members Rafa, Debo, Asuman, Piyush, Alireza, and Devashish for our lively research group and shared desire to bring privacy-enhancing technologies about.

My heartfelt thanks also to everyone from the COSIC support team: Péla, Wim, Dana, Anouk, Elsy, Vanessa, and Saartje. Without your kind help and extensive knowledge, I (and I assume many of my fellow PhD students) would be utterly lost in the thicket that is logistics, travel arrangements, financial matters, scheduling, research reporting and management, and funding obligations.

I want to thank the Research Foundation - Flanders (FWO) for their support in the form of a PhD Fellowship fundamental research. And I would like to express my deep gratitude to Bart for continuing to fund my PhD work (and life) once my FWO funding ended, without questions asked.

I was fortunate to participate in the doctoral summer school on sustainable ICT (SICT) 2021. This week of input and exchange on this critically important nexus was eye-opening and inspirational to me. So much so, that I became a core member of the team behind SICT and co-organized the editions for 2022 and 2023. I am very grateful for the many wonderful people I have met in the context of SICT: Thibault, Lou, Wiebe, JT, Fieke, Louis, Blandine, Maxime, Aina, Sara, Loïk, Noémie, and others.

PhD work can at times be lonely. I am lucky to have had amazing friends, close by and far away, with me on this journey, who swiftly put an end to any feelings of loneliness and made this whole thing worth pursuing. Henry, we have known each other now for close to eighty-five percent of our lives, which is mind-blowing, a little frightening (because of what that means in terms of age, not us…), but most definitely amazing. Thank you for always being there for me, whether there are only a few hundred kilometers between us or thousands. Matze, thank you for being such a great friend, with whom laughter comes naturally and deep talk is easy. Nadine, thank you for your friendship, the many great conversations over the years, and hosting us in Berlin (also thanks Matze).

Omar, we have started our PhDs almost at the same time, which meant that the ups and downs of our PhD trajectories often aligned. During the COVID lockdowns, we were part of each other's "corona bubble" and so we went through significant chunks of those strange times together. Thank you for your trust and friendship, for which I am deeply grateful. I am so happy that you and Sohaila decided to spend your lives together, because that meant that since then you, Sohaila, have become a good friend as well, for

which I consider myself lucky.

Thank you to my wonderful Berlin friends, who followed my journey from afar but continued to include me in their lives: Max and Katharina, Gabriel and Kim, Andy and Evi, and Max. I am lucky to count you as my friends and hope that I now have more time to visit again (knock on wood). And thank you to my Hamburg friends, Jonas and Tati (and Louie), with whom face-to-face time is rare but precious and enriching.

I am grateful for having such an amazing group of people around me here in Leuven. Thanks to Aster, for many great hanging-outs, laser tag sessions, and your friendship. Yeasle and Tim, you two are incredibly warm and dedicated people, thank you for all your time and support. Johan and Merle, thank you for the shared summer evenings, brunch gatherings, and cinema visits, with lots of laughter and good conversations (and thanks for nerding out with me, Johan). Thank you, Mahdi and Niloofar, for being the kind and funny people in my life that you are. Thank you, Dennis, for your support and friendship. Thank you, Erik, for pushing through the final stages of our PhDs together.

I would not be where or who I am today without the unwavering love and support of my family. Thank you to my mother, Doris, and Torsten, and to my father, Stefan, and Antje, for setting me on a path of curiosity and learning, for believing in me and encouraging me. Ich kann mich immer auf euch verlassen und ihr wisst gar nicht, wie sehr dieser Support mich trägt. Vielen Dank für alles, was ihr für mich getan habt und tut.

To my brother, Felix, and my sister, Aune, thank you for being the funniest, smartest, warmest, and most sincere siblings one could hope for. Hab euch lieb, ihr seid bärenstark. Danke, dass ihr immer für mich da seid. Thank you, Julius, for your warm support and presence in my life.

Shout-out to the two coolest and sassiest guinea pigs who have ever allowed humans to take care of them, Inge and Millie. Home office was never alone office, as you two would keep one good company, occasionally reminding one (gently) that breakfast or dinner really was quite late now.

The final words of gratitude are reserved for my partner, Alex. I cannot thank you enough for everything you have done for me. You have endured me in my various stressball states, reassured me when my confidence was wavering, shared in my struggles, and celebrated the wins with me. You make my life meaningful and rich, interesting and fun—and just the right amount chaotic. Thank you for being the intelligent, warm, curious, and witty person you are. I love you.

Lennart Oldenburg
Leuven, June 2025

Popularized Abstract

Any time we send and receive packets over a network such as the Internet (e.g., when we browse the web), we send network metadata together with the actual contents of our packets. Network metadata are data about a packet, e.g., in which direction it travels along a network link, at which time, and how large it is. Even if we take care to encrypt the contents of our packets so that intermediaries through which they travel cannot access them, we unfortunately almost never protect our packets' metadata.

This can make us vulnerable to traffic analysis attacks. A malicious intermediary can record the metadata of our packets and—given enough of them—is able to learn a lot about our online and offline lives. For example, which webpages we are accessing and in which order, who our close contacts are, and at which times we work, relax, and sleep. This may allow such malicious intermediary to learn sensitive details about our health status, sexual orientation, political views and affiliations, and religious beliefs.

If we want to protect our packets' network metadata, we can redirect them through anonymizing networks such as Tor (the most popular anonymizing network to date) or Nym (a modern anonymizing network with stronger metadata protections than Tor). Tor and Nym work by sending our packets through a chain of multiple servers, so that no single one of them knows both who sent a packet and who is going to receive it. Additionally, Nym also purposefully delays packets on their way through the servers, so that telling which packet entering a server is which packet exiting the server becomes difficult. However, in order for anonymizing networks to be effective tools for online anonymity, the largest number of people possible need to use them. Thus, they need to make trade-offs that accommodate as many potential use cases as possible. These trade-offs make them more vulnerable to traffic analysis again.

In this thesis, we evaluate a number of threats to anonymizing networks, among them, three traffic analysis attacks against Tor and Nym. We gain a data-grounded understanding of the real-world threats these attacks pose and which countermeasure that can be integrated into an anonymizing network best defends against these attacks. Our work has had practical impact. Tor now comes equipped with a defense limiting the

impact of our guard discovery attack. The Nym team is developing defenses which are inspired by the ones we evaluated and lower the impact of our flow correlation attack.

Gepopulariseerde Samenvatting

Telkens wanneer we pakketten verzenden en ontvangen via een netwerk zoals het internet (bijvoorbeeld wanneer we op het web surfen), verzenden we naast de daadwerkelijke inhoud van onze pakketten ook netwerkmetadata. Netwerkmetadata zijn gegevens over een pakket, bijvoorbeeld in welke richting het zich langs een netwerkverbinding verplaatst, op welk tijdstip en hoe groot het is. Zelfs als we ervoor zorgen dat de inhoud van onze pakketten versleuteld is, zodat tussenpersonen die ze doorgeven er geen toegang toe hebben, beschermen we helaas bijna nooit de metadata van onze pakketten.

Dit kan ons kwetsbaar maken voor aanvallen waarbij verkeersstromen worden geanalyseerd. Een kwaadwillende tussenpersoon kan de metadata van onze pakketten registreren en—als hij er genoeg heeft—veel te weten komen over ons online en offline leven. Bijvoorbeeld welke webpagina's we bezoeken en in welke volgorde, wie onze naaste contacten zijn en wanneer we werken, ontspannen en slapen. Hierdoor kan een dergelijke kwaadwillende tussenpersoon gevoelige details te weten komen over onze gezondheidstoestand, seksuele geaardheid, politieke opvattingen en overtuigingen, en religieuze overtuigingen.

Als we de netwerkmetadata van onze pakketten willen beschermen, kunnen we ze omleiden via anonimiserende netwerken zoals Tor (het populairste anonimiserende netwerk tot nu toe) of Nym (een modern anonimiserend netwerk met sterkere metadata-bescherming dan Tor). Tor en Nym werken door onze pakketten via een keten van meerdere servers te verzenden, zodat geen enkele server weet wie een pakket heeft verzonden en wie het gaat ontvangen. Bovendien vertraagt Nym ook opzettelijk pakketten op hun weg door de servers, zodat het moeilijk wordt om te bepalen welk pakket een server binnenkomt en welk pakket de server verlaat. Om anonimiserende netwerken echter effectieve hulpmiddelen voor online anonimiteit te maken, moeten ze door zoveel mogelijk mensen worden gebruikt. Daarom moeten ze compromissen sluiten om zoveel mogelijk potentiële gebruikssituaties mogelijk te maken. Deze compromissen maken ze weer kwetsbaarder voor verkeersanalyse.

In dit proefschrift evalueren we een aantal bedreigingen voor anonimiserende netwerken, waaronder drie verkeersanalyse-aanvallen op Tor en Nym. We verkrijgen een op gegevens gebaseerd inzicht in de reële bedreigingen die deze aanvallen vormen en welke tegenmaatregelen die in een anonimiserend netwerk kunnen worden geïntegreerd, het beste bescherming bieden tegen deze aanvallen. Ons werk heeft praktische gevolgen gehad. Tor is nu uitgerust met een verdediging die de impact van onze guard discovery-aanval beperkt. Het Nym-team ontwikkelt verdedigingen die zijn geïnspireerd op de verdedigingen die wij hebben geëvalueerd en die de impact van onze flow correlation-aanval verminderen.

Abstract

Anonymizing networks (ANs) prevent intermediaries and network-level observers from linking the senders of packets to their recipients, enabling users to communicate anonymously by proxying their packet flows through them. These systems thus aim to protect against adversaries collecting packet network metadata and conducting traffic analysis attacks on these metadata records. However, how well ANs are able to achieve this objective has been a topic of active research since the field was created in 1981.

This thesis investigates threats and remedies to the ability of ANs to provide their anonymous routing functionality to users, e.g., because adversaries aim to render them unusable or aim to recover the obfuscated sender-recipient relationships by deploying traffic analysis attacks. We start with an argument for how lack of fault tolerance may degrade the anonymity an AN is able to provide before turning to empirical attack analyses against two practical and deployed ANs: the frontier onion routing implementation and most popular anonymity network to date, Tor, as well as the state-of-the-art deployment of continuous-time mixnets, Nym.

In our first study, we conduct a planet-scale evaluation of how packet latencies diverge from expectation when network failures occur. Failures may be of benign origin or they may be induced purposefully and in a targeted way by adversaries aiming to disrupt the AN. We call on AN designers to include resilience measures from the start, which may also help with broadening their user bases and thus increasing their anonymity sets.

For our second line of research, we conduct attacks and evaluate defenses against the two mentioned prominent anonymizing networks: Tor and Nym. By assuming the role of a traffic analysis adversary, we empirically assess the threat that two different types of attack—guard discovery and passive flow correlation—pose to the anonymity of the respective AN's users and how to best mitigate them.

Guards form the first hop on any circuit a Tor user builds and remain fixed for long periods of time. Given that guards know the IP addresses of their users, Tor aims to prevent entities that should not know this from learning a user's guards. We present a fast, stealthy, and affordable guard discovery attack that enables certain adversaries to

learn the relays chosen by victim Tor Browser users into this sensitive circuit position. Tor now comes with Vanguards-Lite, a countermeasure that limits the impact our and similar guard discovery attacks have.

We conduct two studies evaluating passive flow correlation attacks and defenses against state-of-the-art mixnet Nym. This thesis thus contributes to the empirical understanding of this foundational attack, which aims to link a packet flow entering the AN to its version exiting the AN. We also contribute to the safety of Nym users, as Nym is planning to adopt variants of some of the defenses we considered. The two studies evaluate Nym (and more generally, continuous-time mixnets of similar design) in two different instantiations: first, when both directions of traffic are shaped to adhere to a constant bandwidth on average (bidirectionally regulated mixnet), and second, when packet emissions in the downstream direction of flows are not regulated (downstream-unregulated mixnet).

We find that despite the strong traffic shaping behavior present in Nym's bidirectionally regulated variant the drift between packets at the same index in the ingress and the egress flow gives away if the two flows are correlated or not. We show how statistical as well as machine learning classifiers can be constructed to effectively match—and thus deanonymize—the communicating endpoints on Nym. However, our attack is based on strict assumptions and we discuss how defenses that result in these assumptions being invalidated can diminish the effectiveness of our attack.

Finally, we assess passive flow correlation against Nym in its current downstream-unregulated configuration, where its public Internet proxies (network-requesters) send packets as quickly as possible to the requesting users. Here, the shape of the transferred flows becomes a signal that can be exploited for flow correlation. The adversary compares flows entering the mixnet to flows exiting the mixnet based on their shape, i.e., the distribution of their packets over time. We show that in the absence of additional defenses, the adversary is able to correlate flow pairs quickly and with high precision. However, our extensive countermeasure evaluation shows that the correct configuration of one or more defenses can be effective in mitigating this threat. We discussed the results of both studies with the Nym team and they are in the process of developing variants of some of the countermeasures we evaluated.

Beknopte samenvatting

Anonimiserende netwerken (AN's) voorkomen dat tussenpersonen en waarnemers op netwerkniveau de afzenders van pakketten aan hun ontvangers kunnen koppelen, waardoor gebruikers anoniem kunnen communiceren door hun pakketstromen via deze netwerken te routeren. Deze systemen zijn dus bedoeld om te beschermen tegen tegenstanders die metadata van pakketnetwerken verzamelen en op basis van deze metagegevens verkeersanalyses uitvoeren. Hoe goed AN's dit doel kunnen bereiken, is echter al sinds het ontstaan van dit vakgebied in 1981 een onderwerp van intensief onderzoek.

Dit proefschrift onderzoekt bedreigingen en remedies voor het vermogen van AN's om hun anonieme routeringsfunctionaliteit aan gebruikers te bieden, bijvoorbeeld omdat tegenstanders deze onbruikbaar willen maken of de versluierde relaties tussen afzender en ontvanger willen achterhalen door middel van verkeersanalyseaanvallen. We beginnen met een betoog over hoe een gebrek aan fouttolerantie de anonimiteit die een AN kan bieden, kan aantasten, waarna we ons richten op empirische aanvalsanalyses van twee praktische en geïmplementeerde AN's: de state-of-the-art onion routing-implementatie en het populairste anonimiteitsnetwerk tot nu toe, Tor, evenals de state-of-the-art implementatie van continuous-time mixnets, Nym.

In onze eerste studie voeren we een wereldwijde evaluatie uit van hoe pakketvertragingen afwijken van de verwachtingen wanneer er netwerkstoringen optreden. Storingen kunnen een onschuldige oorzaak hebben, maar ze kunnen ook opzettelijk en gericht worden veroorzaakt door tegenstanders die het AN willen verstoren. We roepen AN-ontwerpers op om bij het ontwerp de veerkracht te versterken, wat ook kan helpen om hun gebruikersbestand te verbreden en zo hun anonimiteitssets te vergroten.

Voor onze tweede onderzoekslijn voeren we aanvallen uit en evalueren we verdedigingsmaatregelen tegen de twee genoemde prominente anonimiserende netwerken: Tor en Nym. Door de rol van een verkeersanalyse-tegenstander aan te nemen, beoordelen we empirisch de bedreiging die twee verschillende soorten aanvallen—guard discovery en passieve flow correlation—vormen voor de anonimiteit van de gebruikers van de

respectieve AN's en hoe deze het best kunnen worden beperkt.

Guards vormen de eerste hop op elk circuit dat een Tor-gebruiker opbouwt en blijven gedurende lange tijd vast. Aangezien guards de IP-adressen van hun gebruikers kennen, wil Tor voorkomen dat entiteiten die dit niet mogen weten, de guards van een gebruiker te weten komen. We presenteren een snelle, heimelijke en betaalbare guard discovery-aanval waarmee bepaalde tegenstanders de relays kunnen achterhalen die door slachtoffers van Tor Browser zijn gekozen voor deze gevoelige circuitpositie. Tor wordt nu geleverd met Vanguards-Lite, een tegenmaatregel die de impact van onze en soortgelijke guard discovery-aanvallen beperkt.

We voeren twee studies uit waarin we passieve flow correlation-aanvallen en verdedigingen tegen het state-of-the-art mixnet Nym evalueren. Dit proefschrift draagt dus bij aan het empirische begrip van deze fundamentele aanval, die tot doel heeft een pakketstroom die het AN binnenkomt te koppelen aan de versie die het AN verlaat. We dragen ook bij aan de veiligheid van Nym-gebruikers, aangezien Nym van plan is varianten van enkele van de door ons overwogen verdedigingen in te voeren. De twee studies evalueren Nym (en meer in het algemeen continuous-time mixnets met een vergelijkbaar ontwerp) in twee verschillende instantiaties: ten eerste wanneer beide richtingen van het verkeer worden gevormd om gemiddeld een constante bandbreedte aan te houden (bidirectioneel gereguleerd mixnet), en ten tweede wanneer pakketemissies in de downstream-richting van flows niet worden gereguleerd (downstream-ongereguleerd mixnet).

We hebben vastgesteld dat ondanks het sterke traffic shaping-gedrag in de bidirectioneel gereguleerde variant van Nym, de drift tussen pakketten met dezelfde index in de ingangs- en uitgangsstroom wegvalt als de twee stromen al dan niet gecorreleerd zijn. We laten zien hoe statistische en machine learning-classificatoren kunnen worden geconstrueerd om de communicerende eindpunten op Nym effectief te matchen—en dus te deanonymiseren. Onze aanval is echter gebaseerd op strikte aannames en we bespreken hoe verdedigingen die deze aannames ongeldig maken, de effectiviteit van onze aanval kunnen verminderen.

Ten slotte beoordelen we passieve flow correlation tegen Nym in zijn huidige downstream-ongereguleerde configuratie, waarbij de openbare internetproxies (network-requesters) pakketten zo snel mogelijk naar de verzoekende gebruikers sturen. Hier wordt de vorm van de overgedragen flows een signaal dat kan worden gebruikt voor flow correlation. De tegenstander vergelijkt flows die het mixnet binnenkomen met flows die het mixnet verlaten op basis van hun vorm, d.w.z. de verdeling van hun pakketten in de tijd. We tonen aan dat de tegenstander zonder aanvullende verdedigingsmaatregelen in staat is om flow pairs snel en met hoge precisie te correleren. Uit onze uitgebreide evaluatie van tegenmaatregelen blijkt echter dat de juiste configuratie van een of meer verdedigingsmaatregelen deze dreiging effectief kan beperken. We hebben de resultaten van beide studies besproken met het Nym-team en zij zijn bezig met de ontwikkeling van varianten van enkele van de door ons geëvalueerde tegenmaatregelen.

Contents

Popularized Abstract **v**

Gepopulariseerde Samenvatting **vii**

Abstract **ix**

Beknopte samenvatting **xi**

Contents **xiii**

List of Abbreviations **xxi**

List of Figures **xxiii**

List of Tables **xxv**

I Traffic Analysis Attacks and Defenses on Anonymizing Networks **1**

1 Introduction **2**
1.1 The Problem with Metadata 4
1.2 The Threat of Traffic Analysis 5
1.3 Anonymous Communication 7
1.4 Research Questions 8
1.5 Structure of This Thesis 10
Bibliography 11

2 Anonymizing Networks **15**
2.1 System Model 15
2.2 Anonymity Properties 16
2.3 Adversaries 19
2.3.1 Network Visibility 19

2.3.2 Insider Visibility . . . 20
2.3.3 Mode of Operation . . . 21
2.3.4 Objectives . . . 22
2.3.5 Common Adversaries . . . 23
2.4 Anonymity Metrics . . . 24
2.4.1 Metrics for Binary Classification . . . 24
2.4.2 Metrics Involving Practical Constraints . . . 27
2.5 Connection Between Anonymity and Resilience . . . 28
Bibliography . . . 30

3 Onion Routing **33**
3.1 Low-Latency, Connection-Oriented: Onion Routing . . . 33
3.2 Tor . . . 35
3.2.1 Onion Services . . . 38
3.3 Attacks and Defenses on Tor . . . 39
3.3.1 Flow Correlation . . . 41
3.3.2 Website Fingerprinting . . . 42
3.3.3 Guard Discovery . . . 44
Bibliography . . . 46

4 Mixnets **55**
4.1 High-Latency, Packet-Oriented: Mixnets . . . 55
4.1.1 Mixnet Verifiability . . . 57
4.1.2 Mixnet Topologies . . . 58
4.1.3 Mix Flushing Strategies . . . 59
4.1.4 Traffic Shaping and Cover Packets . . . 60
4.2 Nym . . . 61
4.3 Attacks on Mixnets and Defenses . . . 66
4.3.1 Flow Correlation and Related Passive Attacks . . . 66
4.3.2 Flow Correlation Countermeasures . . . 69
4.3.3 Blending and Other Active Attacks . . . 71
Bibliography . . . 74

5 Contributions **79**
5.1 “Strong Anonymity is not Enough: Introducing Fault Tolerance to Planet-Scale Anonymous Communication Systems” . . . 79
5.2 “From ‘Onion Not Found’ to Guard Discovery” . . . 80
5.3 “MixMatch: Flow Matching for Mixnet Traffic” . . . 81
5.4 “Shift Your Shape: Correlating and Defending Mixnet Flows Based on Their Shapes” . . . 83
Bibliography . . . 85

6 Conclusions and Directions for Future Research **87**

6.1 Conclusions . . . 87
6.2 Future Research Directions . . . 91
Bibliography . . . 95

II Publications **97**

7 Strong Anonymity is not Enough: Introducing Fault Tolerance to Planet-Scale Anonymous Communication Systems **98**
7.1 Introduction . . . 99
7.2 Anonymity Foundations . . . 100
7.3 ACS Selection and Configuration . . . 101
7.4 Evaluation . . . 103
7.4.1 Failure Scenarios . . . 103
7.4.2 Metrics . . . 103
7.4.3 Experimental Setup . . . 104
7.4.4 Results . . . 104
7.5 Related Work . . . 109
7.6 Conclusions . . . 110
Bibliography . . . 111

8 From "Onion Not Found" to Guard Discovery **115**
8.1 Introduction . . . 116
8.2 Background and Related Work . . . 118
8.3 Guard Discovery Attack . . . 121
8.3.1 Adversarial Objectives and Capabilities . . . 121
8.3.2 Attack Overview . . . 122
8.3.3 Characteristics of Target Cell Pattern . . . 124
8.4 Tuning the Attack . . . 125
8.4.1 Attack Parameters . . . 125
8.4.2 Maximizing the Victim Lookup Rate . . . 128
8.4.3 Scriptless Attack . . . 129
8.5 Attack Evaluation . . . 129
8.5.1 Estimating Tor's Noise Lookup Rate . . . 130
8.5.2 Experimental Setup . . . 132
8.5.3 Attack Success . . . 133
8.5.4 Time to Generate Attack Keys . . . 136
8.6 Countermeasures . . . 137
8.7 Discussion . . . 142
8.8 Conclusions . . . 144
8.9 Acknowledgements . . . 144
Bibliography . . . 146
8.A Adversarial Cell Pattern Characteristics with Shadow . . . 151

8.A.1 Is the Target Cell Pattern Deterministic? . . . 151
8.A.2 Is the Target Cell Pattern Unique? . . . 152
8.B Establishing the Most Effective Set of Attack Parameters . . . 153
8.C Monetary Cost of the Attack . . . 154
8.D Attack Key Precomputation . . . 155
8.E Onion Security Icons . . . 155

9 MixMatch: Flow Matching for Mixnet Traffic 157
9.1 Introduction . . . 160
9.2 Background and Related Work . . . 162
9.2.1 Flow Correlation Against Tor . . . 162
9.2.2 Mixnets and Nym . . . 163
9.2.3 Flow Correlation Against Mixnets . . . 164
9.3 Flow Matching on Mixnets . . . 164
9.3.1 System Model . . . 164
9.3.2 Threat Model . . . 165
9.3.3 Endpoints Closed Sets and Flow Merging . . . 167
9.3.4 Packet Alignment . . . 169
9.4 Statistical Classifier . . . 169
9.4.1 Statistical Detector . . . 169
9.4.2 Combination of Payload and Acknowledgment Flow Scores . . . 171
9.5 Deep Learning Classifiers . . . 172
9.5.1 Deep Learning . . . 172
9.5.2 Input Representation . . . 172
9.5.3 Window Partitioning . . . 173
9.5.4 Drift Network . . . 173
9.5.5 Shape Network . . . 174
9.6 Data Collection . . . 177
9.7 Experimental Results . . . 180
9.8 Discussion . . . 185
9.8.1 Flow Matching Effectiveness Against Nym . . . 185
9.8.2 Limitations and Assumptions of our Attack . . . 186
9.8.3 Countermeasures . . . 187
9.9 Conclusions . . . 188
Bibliography . . . 190
9.A Brief Introduction to Tor . . . 194
9.B Flow Correlation Attacks Against Connection-Based Anonymous Communication Systems . . . 194
9.C Data Collection Setups . . . 195
9.D Additional Experimental Results . . . 195
9.E Details on the DL Classifiers . . . 199
9.E.1 Network Architecture . . . 200
9.E.2 Hyperparameter Tuning . . . 200

9.F Impact of Base Rate (Simultaneous Communications) 202
9.G Hardware Specifications 202

10 Shift Your Shape: Correlating and Defending Mixnet Flows Based on Their Shapes 205
10.1 Introduction 206
10.2 Problem Statement 209
10.2.1 System Model and the Nym Network 209
10.2.2 Adversarial Model 213
10.3 Shape-Based Correlation Attack and Defenses 215
10.3.1 Correlating Mixnet Flows Based on Their Shapes 215
10.3.2 Countermeasures Against Flow Correlation 219
10.4 Data Collection Infrastructure 220
10.5 Undefended and Defended Datasets 222
10.5.1 Undefended Dataset: *default* 222
10.5.2 Defended Datasets 223
10.6 Empirical Evaluation 227
10.6.1 Experimental Setup 227
10.6.2 Results 232
10.7 Discussion 239
10.7.1 Likelihood of the Attack 239
10.7.2 Mitigation Strategies 240
10.7.3 Threat Model 241
10.7.4 Limitations 242
10.7.5 Responsible Disclosure 243
10.8 Related Work 243
10.8.1 Comparison With MixMatch 243
10.8.2 Traffic Analysis Attacks on Anonymity Systems 244
10.9 Conclusions 246
Bibliography 248

List of Abbreviations

ACK Acknowledgment Packet. 212, 214, 215, 220, 223, 226, 230, 237, 240

ACS Anonymous Communication System. 99–106, 109, 110

AN Anonymizing Network. ix–xii, 7–10, 15–29, 33–35, 39, 58–60, 62–64, 68, 79, 80, 88, 89, 91–93

ANN Artificial Neural Network. 215, 224, 245

AS Autonomous System. 20, 109

AUC Area Under the Curve. 26, 27

CDF Cumulative Distribution Function. 106, 108, 135, 136, 138

CNN Convolutional Neural Network. 162, 208, 218, 226, 230

CPIR Computational Private Information Retrieval. 88, 99–102, 109, 110

CPU Central Processing Unit. 103–106, 128, 202

DC-net Dining Cryptographer Network. 7, 8, 100, 101

DL Deep Learning. 41, 43, 44, 159, 172–176, 180, 186, 202, 204, 244–246

DNS Domain Name System. 3, 38, 211

DoH DNS over HTTPS. 3

DoS Denial of Service. 38, 40, 58, 88, 92, 109

ELU Exponential Linear Unit. 200

FN False Negative. 25, 26

FP False Positive. 25, 26, 135

FPR False Positive Rate. 26, 159, 161, 162, 180–183, 185

GCP Google Cloud Platform. 101, 103, 104, 110

GPU Graphics Processing Unit. 199, 230, 231

HSDir Hidden Service Directory. 36–39, 81, 89, 116–125, 130–137, 142–144, 152, 154, 155

HTTP Hypertext Transfer Protocol. 82, 130, 131, 143, 144, 177–179, 195

HTTPS Hypertext Transfer Protocol Secure. 3, 62, 65, 211

IP Internet Protocol. ix, xii, 4, 8, 36, 38, 44, 89, 117, 118, 120

IPD Inter-Packet Delay. 41

ISP Internet Service Provider. 20, 22, 24

ITPIR Information-Theoretic Private Information Retrieval. 101, 109

IXP Internet Exchange Point. 20

KDE Kernel Density Estimation. 216, 217, 223, 245

NTP Network Time Protocol. 104

PIR Private Information Retrieval. 7, 8, 101, 102

PN Predicted Negative. 25

PP Predicted Positive. 25

PR Precision-Recall. 26, 27

PR-AUC Area Under the Precision-Recall Curve. 206, 208, 227, 228, 232–239

PRNG Pseudorandom Number Generator. 199

RAM Random-Access Memory. 102–106, 128, 155, 202, 203

ReLU Rectified Linear Unit. 200

ROC Receiver Operating Characteristic. 26, 27, 180–184, 197–199

RTT Round-Trip Time. 185

SGD Stochastic Gradient Descent. 201

SMPC Secure Multi-Party Computation. 7, 8

SMTP Simple Mail Transfer Protocol. 5

SSD Solid-State Drive. 222

SSH Secure Shell Protocol. 6

SURB Single-Use Reply Block. 64, 65, 70, 212, 213, 219–221, 241

TCP Transmission Control Protocol. 4, 194

TLS Transport Layer Security. 3, 4, 242

TN True Negative. 25–27

TP True Positive. 25, 26, 135

TPR True Positive Rate. 26, 27, 159, 161, 162, 180, 181, 183, 185, 202

UDP User Datagram Protocol. 5

URL Uniform Resource Locator. 119, 141, 221, 222, 226, 230, 231

vCPU Virtual Central Processing Unit. 128, 155, 222

VoIP Voice over Internet Protocol. 6

VPN Virtual Private Network. 62, 207

List of Figures

2.1 Anonymizing network model for packet flows 16
2.2 Adversarial network visibility 19
2.3 Adversarial insider visibility 20
2.4 Adversarial mode of operation 21
2.5 Abstract binary classifier for traffic analysis 25
2.6 ROC and PR curve examples 26

3.1 Three-hop regular Tor circuit 37
3.2 Six-hop rendezvous Tor circuit 39
3.3 Website fingerprinting attack scenario 43

4.1 Worst-case adversary against Chaumian mix cascade 56
4.2 Mixnet topologies 58
4.3 Nym mixnet architecture with two types of network-requester 65
4.4 Flow correlation attack 67

7.1 Resource demands when sending and receiving messages 105
7.2 End-to-end latencies when sending and receiving messages 107

8.1 Tor network model 121
8.2 Step-wise guard discovery attack overview 123
8.3 Potential false positive lookups 130
8.4 Attack success depending on attack time 134
8.5 Precomputation time for attack onion service public keys 137
8.6 Token bucket defense effectiveness 138
8.7 Vanguards-lite defense effectiveness 139
8.8 Effectiveness of token bucket combined with Vanguards-lite 141
8.9 Target cell pattern of victim lookup 151
8.10 Victim lookup rate from different attack parameters 154
8.11 Onion icons in use by Tor Browser 156

9.1 System model . . . 165
9.2 Flow merging operation . . . 168
9.3 Adapted triplet network for the shape classifier . . . 175
9.4 Attack performance depending on number of observations . . . 181
9.5 Attack performance depending on use of cover traffic . . . 182
9.6 Attack performance depending on per-mix packet delay . . . 183
9.7 Attack performance depending on size of endpoints set . . . 184
9.8 Attack performance depending on real-world network effects . . . 184
9.9 Isolated setup . . . 195
9.10 Live-network setup . . . 196
9.11 Download duration per dataset . . . 196
9.12 Attack performance depending on packet type . . . 197
9.13 Attack performance depending on threshold locality . . . 198
9.14 Attack performance depending on aggregation rules . . . 198
9.15 Attack performance depending on input representation . . . 199
9.16 Observations needed for high recall and precision at different base rates 202
9.17 Shape and drift neural network architectures . . . 204

10.1 System model . . . 210
10.2 Shape of flow for one particular webpage visit . . . 216
10.3 Flow processing procedure . . . 217
10.4 Shape-based window-wise comparison of flows . . . 218
10.5 Shape of split flow versus split flow plus injected packets . . . 225
10.6 Final flow correlator model (discriminator) . . . 229
10.7 Final packet injector model (generator) . . . 230
10.8 Adversarial flow correlation performance over datasets *splitS_injI* . . . 233
10.9 Adversarial flow correlation performance on datasets *splitS_injI* . . . 238

List of Tables

2.1 Adversarial objectives 22

7.1 First Wasserstein distances between latency distributions 108

8.1 Lookup rates depending on injection rate and onion service version . . 126
8.2 Lookup rates depending on injection rate and victim setting 127
8.3 Lookup statistics at hidden service directory 131
8.4 Time to successful attack 135

9.1 Hyperparameter search spaces and chosen values 201

10.1 Hyperparameter search spaces and chosen values 228
10.2 Adding unrelated traffic versus *default* 235
10.3 Higher delay and larger packet size versus *default* 236
10.4 Usage of acknowledgment packets versus *default* 237
10.5 Attack performance of countermeasure-unaware adversary 239

Part I

Traffic Analysis Attacks and Defenses on Anonymizing Networks

Chapter 1

Introduction

For many of us today, our lives take place to a significant extent online. We have come to rely on the Internet for even the most privacy-sensitive activities—perhaps without worrying much about the implications. The average adult now spends more than one quarter of their day online (6:38 hours, Q3/2024) [29]. Whether we are mapping routes between the stops of our days' agendas, looking for romantic partners, communicating with loved ones or complete strangers via messenger services and social networks, researching answers to professional and very private questions, spending money, or staying informed—sending and receiving packets is now essential to everyday life.

While being able to use the Internet undoubtedly carries great benefits, the sum of our online activities (Matz et al. call this our "digital footprint" [16]) says a lot about who we are, who we associate with, what makes up our daily lives, and what we will be doing next. So much so that if enough of these data are known to other entities, they may be able to nudge us on political decisions,[1] inundate us with hyper-targeted advertisements to extract money from us [5, 14, 31], precisely track our movements (e.g., back and forth between home and an abortion clinic),[2] algorithmically reject prospective tenants due to bad credit scores [25], leverage background information to identify us in large datasets [18], or hike insurance premiums based on behavioral data.[3] Beginning in 2013, the documents leaked by Edward Snowden demonstrated how rich a target the digital data of a country's or bloc's citizens is for state-sponsored (mass) surveillance and digital

[1] https://web.archive.org/web/20250329183511/https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election

[2] https://web.archive.org/web/20250328090435/https://krebsonsecurity.com/2024/10/the-global-surveillance-free-for-all-in-mobile-ad-data/

[3] https://web.archive.org/web/20250320191820/https://www.nytimes.com/2024/03/14/technology/gm-lexis-nexis-driving-data.html

intelligence gathering (so called *signals intelligence* or *SIGINT*).[4,5]

Since then, a large-scale effort across the technical community has resulted in impressive gains towards a more secure and private Internet, focusing on making the *contents* of what we send over the Internet inaccessible to anyone but us and the intended recipients (*content confidentiality*). As reported by Let's Encrypt (an organization providing TLS certificates free of charge), the period between October 2013 and October 2020 saw an almost $55\,\%$ rise in pageloads over HTTPS in Firefox.[6] By increasing the adoption of DNS over HTTPS (DoH) [12], we can ensure that no intermediary on the path between us and the chosen DoH resolver has plaintext access to our DNS queries. Messaging services that have adopted the Signal Protocol[7] or other *end-to-end encryption* schemes (e.g., Messaging Layer Security [2]) keep our digital conversations private between only the respective participants.

While some of the above listed advances require user action to materialize (e.g., installing an end-to-end encrypted messaging application instead of an insecure one), many do not, as they become available as privacy-preserving alternatives in the protocols and lower layers end-user-facing applications build upon. US-based Firefox users, for example, use DoH by default for their DNS queries as of early 2020.[8] It is a great achievement that—with some effort and willingness to resist the allures of the worst-offending digital products—users today can drastically lower the exposure of their communications' contents to intermediaries or unintended parties.

However, achieving widespread confidentiality of data transmission content is only one part of the challenge we face to meaningfully improve online privacy. Another, often neglected, part concerns the threat a different type of data—so called *metadata*—pose when they are left unprotected. Metadata are *data about the data* we are exchanging and thus naturally arise along the paths of data transmissions. This thesis investigates aspects of the threat that metadata present to online privacy and offers insights into select avenues for remedy.

[4] https://web.archive.org/web/20250210033909/https://www.aclu.org/nsa-documents-released-to-the-public-since-june-2013

[5] https://web.archive.org/web/20250330091155/https://www.theguardian.com/world/2013/jun/06/nsa-phone-records-verizon-court-order

[6] https://web.archive.org/web/20250328094048/https://letsencrypt.org/stats/

[7] https://web.archive.org/web/20250330133850/https://signal.org/docs/

[8] https://web.archive.org/web/20250321103434/https://blog.mozilla.org/en/products/firefox/firefox-continues-push-to-bring-dns-over-https-by-default-for-us-users/

1.1 The Problem with Metadata

Metadata refer to any auxiliary data associated with some *primary data*. If you think of a book in a library, its contents could be considered the primary data while publisher, date of publication, and loan status would be examples of metadata associated with the book. In the context of network transmissions and therefore this thesis, we are specifically concerned with the risks arising from *network metadata*. The term network metadata denotes any kind of data that an entity involved in the transmission of a network packet (e.g., a TCP or IP packet) is able to observe *at the network transport level*. Crucially in the context of risks arising from the analysis of network metadata, an entity involved in a packet's transmission does not need access to the packet's *payload* (i.e., the part of the packet carrying its primary data) in order to collect the packet's network metadata.

Consider the example of an email submitted from one Internet user (the sender) to their email server, for it to ultimately be displayed to another Internet user (the recipient). At the application level, the email consists of the message the sender wants to communicate to the recipient and any number of headers such as sender email address, recipient email address, subject, and timestamp (many more exist). Thinking about primary data versus metadata from the perspective of email, it is clear that the message forms the primary data while the headers represent metadata associated with it. However, care must be taken when analyzing this example from a different layer, e.g., the network transport level. At this layer, the entire email (its message and any associated header) is encapsulated as the payloads of the transferred packets. This means that the whole email becomes payload for the network transport level and new metadata relevant to this layer are generated (explained below) and possibly attached to the packets.

Assuming the sender uses an email protocol that properly implements transport encryption (e.g., via TLS), the payload will not be visible to any network-level entity located on the path between sender and email server (e.g., an access point, router, or middlebox). Assuming that the hardness assumptions hold on which the employed cryptographic schemes are based, trying to break the payloads' encryption is a fruitless endeavor for such intermediary. Hereinafter, we thus assume that packet payloads are *inaccessible* to any network intermediary.

What are visible to intermediaries, however, are network metadata that are required to facilitate the packets' transmissions or that can be observed from the packets directly. Similar to what we write on the envelope of a letter to enable its correct delivery by postal services, network packets need routing information in order to arrive at their destination (the email server in our example, as the first hop on the email's path to the recipient). If the network intermediary has access to the TCP level of packet transmissions, this amounts to the so called *five-tuple* [15]: source IP address (e.g., `1.2.3.4`), source port number (e.g., `37200`), destination IP address (e.g., `9.8.7.6`), destination port number (e.g., `465`), and protocol identifier (e.g., TCP). Note how, if Internet best practices are observed,

such five-tuple allows an educated guess which application layer protocol caused the associated packet's transmission (in this case, SMTP), due to the port numbers used. Apart from these necessarily accessible routing metadata, an intermediary observing a network link learns (at least) two additional network metadata about any passing packet: the packet's size and the precise timestamp at which the packet arrives (or leaves). This is similar to how a postal service may record the timestamp of when a letter first enters their system (e.g., upon submission in a postal office) and how they can weigh a letter (e.g., to determine the service charge or plan delivery logistics).

Where we start to strain the analogy to letters, however, is when we factor in that most network packet transmissions are not isolated events (like most letters or even broadcasting a single UDP packet are). For most modern end-user-facing Internet-based applications (e.g., email, web browsing, messaging, media streaming), we need to exchange *packet flows*, i.e., few to very many packets in sequence. This is because network packets are of a (relatively small) fixed size and any payload larger than one packet can carry (e.g., a song, a webpage) needs to be split across multiple packets. Additionally, we may create dependencies between packets at the application layer, e.g., due to implementing *protocols* between two or more participants, resulting in one participant's next transmission depending on the input of another participant. At this point, we realize that these dependencies between packets result in metadata *patterns* at the network level that—given appropriate analysis—have the potential to allow far-reaching inferences about the participants involved, protocols and applications used, and actions performed in the packet flows that cause them.

1.2 The Threat of Traffic Analysis

A *traffic analysis* attack constitutes the process of inferring higher-level properties, actions, or involved participants of one or more packet flows purely from their associated network metadata. By conducting traffic analysis, an entity with access to network metadata leverages that data and becomes an *adversary* or *attacker*. While it is possible for entities external to (i.e., uninvolved in) packet flows to attack them using traffic analysis (e.g., by purchasing network metadata from other sources),[9,10,11] for this thesis, we assume that the entity conducting traffic analysis is in fact an intermediary at the network transport level of the packet flows under attack.

[9] https://web.archive.org/web/20250328085952/https://www.vice.com/en/article/data-brokers-netflow-data-team-cymru/

[10] https://web.archive.org/web/20250430095022/https://www.vice.com/en/article/us-military-bought-mass-monitoring-augury-team-cymru-browsing-email-data/

[11] https://web.archive.org/web/20250414131956/https://www.vice.com/en/article/fbi-bought-netflow-data-team-cymru-contract/

The attack can be conducted as close as possible to the time the network metadata was collected ("online", "live", or "real-time") or arbitrarily long after the time of collection ("offline"). An adversary conducting traffic analysis may be collecting network metadata only at a single network location or aggregate them across multiple network vantage points. While traffic analysis is based only on network metadata, some adversarial vantage points may in fact be located at the level of participants at the application level, depending on the concrete attack (e.g., a malicious web server as an aide in conducting a traffic analysis attack against web-browsing users). The more metadata an entity has available to conduct traffic analysis on, the higher in quality (precision, reach, level of detail) are the inferences. At the extreme, the indiscriminate recording and analyzing of the metadata of a large number of parties of interest (e.g., an entire population) is part of an effort commonly referred to as *mass*, *bulk*, or *dragnet surveillance*, or *pervasive monitoring* [10].

Traffic analysis was initially developed as a tool for military purposes [6]. However, with the advent of networked communication, the technique quickly found application also in civilian contexts. Early examples of traffic analysis over packet flows that are characteristic of modern Internet uses include the attack by Song et al. [28] enabling inferences about passwords transmitted over SSH, the works of Bernaille et al. [3, 4] on classifying the carried application based on initial packets, and identifying spoken phrases in a VoIP application based on sequences of packet lengths by Wright et al. [30]. Nowadays, traffic analysis attacks have a broad set of statistical and machine learning methods at their disposal [23] in the perpetual cat-and-mouse game between improved traffic analysis attacks and subsequent defenses to them.

It may be tempting to dismiss the threat to privacy that traffic analysis poses due to these attacks "only" drawing from network metadata, which can seem of lower quality than the contents of communications. However, this does not bear out in reality, as for example Prof. Edward W. Felten explained in 2013: "Metadata can now yield startling insights about individuals and groups, particularly when collected in large quantities across the population. It is no longer safe to assume that this 'summary' or 'non-content' information is less revealing or less sensitive than the content it describes" [11]. And he continued: "Given limited analytical resources, analyzing metadata is often a far more powerful analytical strategy than investigating content: It can yield far more insight with the same amount of effort" [11]. The 2013 MetaPhone study by Mayer et al. [17] strikingly confirmed these statements empirically using a crowdsourced telephone metadata collection. As did General Michael Hayden, a former director of both the NSA and the CIA (two intelligence organizations that have taken metadata collection and analysis to population scale), stating in no uncertain terms: "We kill people based on metadata" [13].

1.3 Anonymous Communication

We say that users communicate anonymously if they cannot be linked to their communication partners based on the packets they send or receive. Clearly, this depends on the entity attempting the linking, the adversary. Thus, a refined definition of anonymous communication might be: users communicate anonymously with respect to a concrete adversary if this adversary, after having deployed traffic analysis, is unable to link users to their communication partners better than using *a priori knowledge* only, i.e., what the adversary knew before applying traffic analysis. If the adversary has no a priori knowledge, users communicate anonymously if, after traffic analysis, the adversary is not able to link users to packets with probability better than random guessing.

No one can be anonymous on their own (thus, "anonymity loves company" [8]), so users have to organize into *anonymizing networks (ANs)*—the more users, the better. ANs are the primary technical response and countermeasure to the risk of traffic analysis based on network metadata [27]. At an abstract level, ANs act as *anonymous routing layers*: they are responsible for transporting application-level data that participants in the network want to communicate to other participants while hindering intermediaries from inferring information about properties, behaviors, and relationships at that application layer. ANs thus offer *proactive* protection against incursions into Internet users' privacy, which is one of their most salient differentiators from the privacy protections that legislation and regulation can afford, which can only threaten retribution after a violation occurred.

While only the composition of many different building blocks results in a functional AN, the anonymity properties an AN can provide are determined by the specific *anonymity mechanism* it relies on at its core. Anonymity mechanisms consume one or more resources—packet end-to-end latency through the network, bandwidth, computational effort—in order to achieve their goal of providing the users of the ANs with the desired anonymity properties. With now more than four decades of research in the field of anonymous communication, a number of anonymity mechanisms are available to AN designers: *onion routing*, *mixnets*, *dining cryptographer networks (DC-nets)*, *private information retrieval (PIR)*, and *secure multi-party computation (SMPC)*.

The body of literature on proposed AN designs is therefore huge. One useful way to distinguish them is on their ability to adapt well to changes in size and composition of their user bases. This extends all the way to the size and churn of "Internet-scale" applications such as messaging services. An AN based on an anonymity mechanism that scales well is able to add or remove infrastructure (e.g., service nodes) to handle the change in load gracefully, while the impact on user experience (e.g., end-to-end latency of packets) that these changes entail is relatively modest. We care about this quality in an AN as it indicates real-world applicability of the design and thus its potential to meaningfully improve online privacy. Into this category fall onion routing networks (*connection-oriented*) and mixnets (*packet-oriented*), and in fact we see readily available,

deployed systems for both in the form of Tor [9][12] and Nym [7][13].

Onto the other side of this distinction fall AN designs based on DC-nets, PIR, and SMPC, which face challenges of scale and dynamicity of the contexts that large real-world applications work in. We refer to [1, 24, 26] for explanations of these concepts and discussions of their properties, and focus on onion routing networks and mixnets in this thesis, which we discuss in more detail in Chapter 3 and Chapter 4, respectively.

1.4 Research Questions

This thesis contributes to the understanding of the following research questions (RQs).

RQ1: How Do Network Failures Degrade The Usability Of Anonymizing Networks?

Onion routing networks and mixnets are large distributed systems, in which failures of some kind (e.g., bugs in source code, faulty network links, failing network cards in intermediate routers, emergency maintenance downtimes) will occur. Depending on the specific failure, its impact on the objective of the AN can be devastating, if the AN is not prepared. In the worst case, the AN ceases to operate, i.e., does not route packets anonymously for its users anymore. This can have serious negative consequences for the privacy of the AN's users: they may switch to less private means of communication (thus making themselves more vulnerable to traffic analysis) or stop communicating entirely. We want to understand how and in which quality failures on the network level (packet losses, high amounts of delay) impact the ability of ANs to achieve their service objective and thus their usability.

RQ2: How Vulnerable Are Tor Users To Guard Discovery And How Do We Protect Them?

To send packets anonymously, Tor users build virtual connections (circuits) through the AN that (typically) consist of three relays. Each relay on a circuit only knows its predecessor and successor, but never both the user and their destination (e.g., a web server on the public Internet). We call the relay succeeding the user on a circuit their guard. Guards play an important role in protecting their users' anonymity. They are the only Tor relays (except for bridges, which we ignore as they are only used when Tor is blocked) that know the actual network locations (IP addresses) of Tor users. Discovering someone's guards, i.e., learning their guard even though one should not know this information, is thus a serious threat to Tor users. An adversary successfully conducting a guard discovery attack can now concentrate their efforts on a single target relay on their way to ultimately deanonymize the guard's user(s). We ask how much of a

[12] `https://web.archive.org/web/20250401185036/https://www.torproject.org/`
[13] `https://web.archive.org/web/20250406004702/https://nym.com/`

threat this type of attack is, in particular, against Tor Browser users visiting a webpage modified by the adversary. In order to disable or lower this threat, we also aim to understand which countermeasures are available and how effective they are.

RQ3: How Resistant Is State-Of-The-Art Mixnet Nym Against Passive Flow Correlation Attacks And How Can We Increase Its Resistance Further?

Mixnets are AN designs intended to withstand very strong adversaries trying to deanonymize the mixnets' users. Among the many possible mixnet designs, continuous-time mixnets are promising as they hold the potential for strong network-level privacy while being compatible with the requirements of more latency-sensitive applications (for which packet delivery times beyond a few seconds are a non-starter). Nym is a state-of-the-art implementation of this line of mixnet research and is deployed as a usable service. As a system with real users, Nym has to evolve its components if it becomes necessary. One of Nym's core components, the network-requester, was recently changed not to shape the traffic it sends through the mixnet back to Nym users anymore. This change allows us to evaluate the threat of passive flow correlation against two variants for how the current frontier in continuous-time mixnets (exemplified by Nym) may be implemented. Each of the following two questions addresses one of these two variants.

RQ3(a): What If Both Mixnet Traffic Directions Are Shaped To Hide Patterns? In the first continuous-time mixnet variant, both directions of traffic on packet flows proxied to public Internet services are shaped according to a Poisson process. This corresponds to Nym in its version before the above mentioned change to its network-requesters. Here, passive flow correlation is severely complicated for an adversary controlling some of Nym's gateways. We ask if passive flow correlation in the form of flow matching (where we assume that endpoints communicate exclusively in groups and no packets are lost) is nonetheless still possible and if so, to what extent. Additionally, we aim to identify countermeasures to passive flow matching against Nym with traffic regulation in both directions.

RQ3(b): What If One Direction Is Not Shaped At All? In the second continuous-time mixnet variant, the component of the mixnet that proxies traffic to the public Internet does not regulate packet emissions anymore and thus sends packets from the public Internet as fast as possible through the mixnet to the requesting users. This corresponds to Nym after the above mentioned network-requesters change was implemented. The threat of passive flow correlation against the mixnet's users may be increased significantly here and we aim to understand if that is the case for Nym and how much so. We also identify a set of defenses Nym can implement and deploy together and thus aim to understand how well they work and in which configuration.

1.5 Structure of This Thesis

This thesis is organized into two parts. Part I contains Chapters 1 to 6 and motivates and contextualizes the research this thesis produced. Chapter 1 introduces the risks to online privacy when intermediaries are able to leverage collected network metadata in traffic analysis, explains how anonymizing networks (AN) can help in reducing this threat, and states the research questions this thesis answers. Chapter 2 provides an overview over what makes up ANs conceptually, which properties we aim for with these systems, adversarial capabilities, and how binary classification is used to assess adversarial performance. Deepening our understanding of one branch of ANs, we discuss onion routing and its main representative, Tor, in Chapter 3. In Chapter 4, we take a look at an AN type with the potential for stronger anonymity compared to onion-routing-only ANs: mixnets and state-of-the-art representative Nym. Chapter 5 overviews the contributions this thesis makes to the literature. Finally, Chapter 6 presents conclusions and insights from this thesis to the stated research questions and outlines directions for future research.

Part II contains the four publications this thesis contributes to the literature:

1. "Strong Anonymity is not Enough: Introducing Fault Tolerance to Planet-Scale Anonymous Communication Systems" [22] in Chapter 7,
2. "From 'Onion Not Found' to Guard Discovery" [19] in Chapter 8,
3. "MixMatch: Flow Matching for Mixnet Traffic" [20] in Chapter 9,
4. "Shift Your Shape: Correlating and Defending Mixnet Flows Based on Their Shapes" [21] in Chapter 10.

Bibliography

[1] A. Back, U. Möller, and A. Stiglic. "Traffic Analysis Attacks and Trade-Offs in Anonymity Providing Systems". In: *Information Hiding*. Ed. by I. S. Moskowitz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 245–257. ISBN: 978-3-540-45496-0 (p. 8).

[2] R. Barnes, B. Beurdouche, R. Robert, J. Millican, E. Omara, and K. Cohn-Gordon. *The Messaging Layer Security (MLS) Protocol*. RFC 9420. July 2023. DOI: 10.17487/RFC9420. URL: https://www.rfc-editor.org/info/rfc9420 (p. 3).

[3] L. Bernaille and R. Teixeira. "Early Recognition of Encrypted Applications". In: *Passive and Active Network Measurement*. Ed. by S. Uhlig, K. Papagiannaki, and O. Bonaventure. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 165–175. ISBN: 978-3-540-71617-4 (p. 6).

[4] L. Bernaille, R. Teixeira, and K. Salamatian. "Early Application Identification". In: *Proceedings of the 2006 ACM CoNEXT Conference*. CoNEXT '06. Lisboa, Portugal: Association for Computing Machinery, 2006. ISBN: 1595934561. DOI: 10.1145/1368436.1368445. URL: https://doi.org/10.1145/1368436.1368445 (p. 6).

[5] B. Cyphers and G. Gebhart. *Behind the One-Way Mirror: A Deep Dive Into the Technology of Corporate Surveillance*. Tech. rep. Electronic Frontier Foundation, 2019 (p. 2).

[6] G. Danezis and R. Clayton. "Introducing Traffic Analysis". In: *Digital Privacy*. Ed. by A. Acquisti, S. Gritzalis, C. Lambrinoudakis, and S. di Vimercati. 1st. Auerbach Publications, 2007. DOI: 10.1201/9781420052183. URL: https://doi.org/10.1201/9781420052183 (p. 6).

[7] C. Diaz, H. Halpin, and A. Kiayias. *The Nym Network*. Tech. rep. Version 1.0 as of February 26, 2021. Nym Technologies SA, Feb. 2021. URL: https://web.archive.org/web/20250403044143/https://nym.com/nym-whitepaper.pdf (p. 8).

[8] R. Dingledine and N. Mathewson. "Anonymity Loves Company: Usability and the Network Effect". In: *5th Annual Workshop on the Economics of Information Security, WEIS 2006, Robinson College, University of Cambridge, England, UK, June 26-28, 2006*. 2006. URL: https://econinfosec.org/archive/weis2006/docs/41.pdf (p. 7).

[9] R. Dingledine, N. Mathewson, and P. Syverson. "Tor: The Second-Generation Onion Router". In: *13th USENIX Security Symposium (USENIX Security '04)*. USENIX Association, 2004 (p. 8).

[10] S. Farrell and H. Tschofenig. *Pervasive Monitoring Is an Attack*. RFC 7258. May 2014. DOI: 10.17487/RFC7258. URL: https://www.rfc-editor.org/info/rfc7258 (p. 6).

[11] E. W. Felten. *Written Testimony to United States Senate, Committee on the Judiciary, Hearing on Continued Oversight of the Foreign Intelligence Surveillance Act*. Oct. 2013. URL: https://web.archive.org/web/20250306210459/https://www.cs.princeton.edu/~felten/testimony-2013-10-02.pdf (visited on 03/14/2023) (p. 6).

[12] P. E. Hoffman and P. McManus. *DNS Queries over HTTPS (DoH)*. RFC 8484. Oct. 2018. DOI: 10.17487/RFC8484. URL: https://www.rfc-editor.org/info/rfc8484 (p. 3).

[13] Johns Hopkins University on YouTube. *The Johns Hopkins Foreign Affairs Symposium Presents: The Price of Privacy: Re-Evaluating the NSA*. 2014. URL: https://www.youtube.com/watch?v=kV2HDM86XgI&t=1072 (visited on 03/14/2023) (p. 6).

[14] D. Marti, F. Lin, M. Schwartz, and G. Fahs. *Who Shares Your Information With Facebook?* Tech. rep. Consumer Reports, Jan. 2024 (p. 2).

[15] P. Matthews, I. van Beijnum, and M. Bagnulo. *Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers*. RFC 6146. Apr. 2011. DOI: 10.17487/RFC6146. URL: https://www.rfc-editor.org/info/rfc6146 (p. 4).

[16] S. C. Matz, R. E. Appel, and M. Kosinski. "Privacy in the age of psychological targeting". In: *Current Opinion in Psychology* 31 (2020). Privacy and Disclosure, Online and in Social Interactions, pp. 116–121. ISSN: 2352-250X. DOI: https://doi.org/10.1016/j.copsyc.2019.08.010. URL: https://www.sciencedirect.com/science/article/pii/S2352250X19301332 (p. 2).

[17] J. Mayer, P. Mutchler, and J. C. Mitchell. "Evaluating the privacy properties of telephone metadata". In: *Proceedings of the National Academy of Sciences* 113.20 (2016), pp. 5536–5541. DOI: 10.1073/pnas.1508081113. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1508081113. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1508081113 (p. 6).

[18] A. Narayanan and V. Shmatikov. "Robust De-anonymization of Large Sparse Datasets". In: *2008 IEEE Symposium on Security and Privacy (SP)*. 2008, pp. 111–125. DOI: 10.1109/SP.2008.33 (p. 2).

[19] L. Oldenburg, G. Acar, and C. Diaz. "From "Onion Not Found" to Guard Discovery". In: *Proceedings of the 22nd Privacy Enhancing Technologies Symposium (PETS 2022)*. Vol. 2022. 2022, pp. 522–543. DOI: 10.2478/popets-2022-0026 (p. 10).

[20] L. Oldenburg, M. Juarez, E. Argones Rúa, and C. Diaz. "MixMatch: Flow Matching for Mixnet Traffic". In: *Proceedings of the 24th Privacy Enhancing Technologies Symposium (PETS 2024)*. Vol. 2024. 2024, pp. 276–294. DOI: 10.56553/popets-2024-0050 (p. 10).

[21] L. Oldenburg, M. Juarez, E. Argones Rúa, and C. Diaz. "Shift Your Shape: Correlating and Defending Mixnet Flows Based on Their Shapes". Under submission to *IEEE Transactions on Dependable and Secure Computing* since March 31, 2025. 2025 (p. 10).

[22] L. Oldenburg and F. Tschorsch. "Strong Anonymity is not Enough: Introducing Fault Tolerance to Planet-Scale Anonymous Communication Systems". In: *Proceedings of the 16th International Conference on Availability, Reliability and Security*. ARES '21. Association for Computing Machinery, 2021. DOI: 10.1145/3465481.3469189 (p. 10).

[23] E. Papadogiannaki and S. Ioannidis. "A Survey on Encrypted Network Traffic Analysis Applications, Techniques, and Countermeasures". In: *ACM Comput. Surv.* 54.6 (July 2021). ISSN: 0360-0300. DOI: 10.1145/3457904. URL: https://doi.org/10.1145/3457904 (p. 6).

[24] J.-F. Raymond. "Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems". In: *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability Berkeley, CA, USA, July 25–26, 2000 Proceedings*. Ed. by H. Federrath. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 10–29. ISBN: 978-3-540-44702-3. DOI: 10.1007/3-540-44702-4_2. URL: https://doi.org/10.1007/3-540-44702-4_2 (p. 8).

[25] E. Rosen, P. M. E. Garboden, and J. E. Cossyleon. "Racial Discrimination in Housing: How Landlords Use Algorithms and Home Visits to Screen Tenants". In: *American Sociological Review* 86.5 (2021), pp. 787–822. DOI: 10.1177/00031224211029618. URL: https://doi.org/10.1177/00031224211029618 (p. 2).

[26] S. Sasy and I. Goldberg. "SoK: Metadata-Protecting Communication Systems". In: *Proceedings of the 24th Privacy Enhancing Technologies Symposium (PETS 2024)*. Vol. 2024. 2024, pp. 509–524. DOI: 10.56553/popets-2024-0030 (p. 8).

[27] F. Shirazi, M. Simeonovski, M. R. Asghar, M. Backes, and C. Diaz. "A Survey on Routing in Anonymous Communication Protocols". In: *ACM Comput. Surv.* 51.3 (June 2018). ISSN: 0360-0300. DOI: 10.1145/3182658. URL: https://doi.org/10.1145/3182658 (p. 7).

[28] D. X. Song, D. Wagner, and X. Tian. "Timing Analysis of Keystrokes and Timing Attacks on SSH". In: *10th USENIX Security Symposium (USENIX Security '01)*. USENIX Association, 2001 (p. 6).

[29] We Are Social and Meltwater. *Digital 2025: Global Overview Report*. Feb. 2025. URL: https://web.archive.org/web/20250316083744/https://datareportal.com/reports/digital-2025-global-overview-report (visited on 04/01/2025) (p. 2).

[30] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson. "Spot Me if You Can: Uncovering Spoken Phrases in Encrypted VoIP Conversations". In: *2008 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2008, pp. 35–49. DOI: 10.1109/SP.2008.21 (p. 6).

[31] S. Zuboff. *The Age of Surveillance Capitalism: The Fight for a Human Future at the New Frontier of Power*. New York: PublicAffairs, 2019. ISBN: 9781610395694 (p. 2).

Chapter 2

Anonymizing Networks

In order to be able to evaluate traffic analysis attacks and defenses on anonymizing networks, we need to define (a) the system model in which the AN works at a useful level of abstraction, (b) the ideal anonymity properties an AN may aim to provide to its users, (c) the capabilities of adversaries to collect network metadata for analysis, and (d) the metrics by which we judge the success of the adversaries and AN in achieving their objectives. This chapter introduces each of the four components in turn.

2.1 System Model

In this chapter, we treat ANs as "black box" network services that provide the aforementioned anonymous routing layer functionality (see Section 1.3) for transporting packet flows (in contrast to single packets). Later, in Chapter 3 and Chapter 4, we detail the inner workings of the two AN approaches this thesis focuses on: onion routing networks and mixnets, respectively.

We visualize the AN system model in Figure 2.1, in which the *users* U_i, with $i \in \mathbb{N}^+$, are connected via an *anonymizing network*. Users are represented as rectangles with dotted borders. Internally, each U_i consists of an *application level* (represented as encircled "AL") and the AN's *network endpoint* (represented as the ends of the pipes opening out into the AN). A user's application layer maintains a bidirectional packet flow (represented as the double arrow) through the AN. The colored bars on top of a double arrow are packets sent by the user's application layer to the anonymizing network, while the gray bars underneath a double arrow represent packets arriving from the anonymizing network at the user's application layer. The height of each bar indicates the size of the packet. We

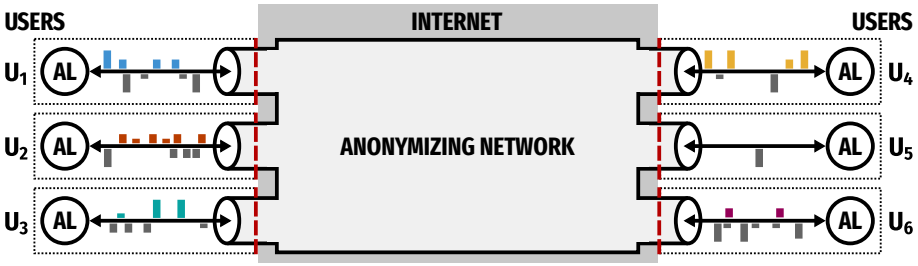


Figure 2.1: Model of an anonymizing network providing anonymous routing layer functionality to its users who are assumed to send and receive flows of packets. The boundary between a user's (trusted) device and any other network entity (including network intermediaries) is marked as a dashed line in red.

make no assumptions as to which kind or how many applications cause the displayed bidirectional packet flow at each user. Note that by user we mean an entity using the AN. We thus do not distinguish between the human and the human's computing device used to connect to the AN, as this distinction is not relevant from the network perspective. And an AN-enabled program running on a server is also considered a user.

Most ANs are implemented as *overlay networks*. An overlay network is a set of nodes that forms a virtual network topology at the application layer, in which the underlying network distances between nodes on the overlay network are hidden from the overlay network's members. Figure 2.1 shows an AN implementing the *client-server* model, in which participants are distinguished into *clients* (U_i in the figure) and *servers* (we show them as the opaque anonymizing network connecting the users). Clients consume and infrastructure nodes provide the anonymous routing layer functionality.

Alternatively, ANs also can be and historically often have been implemented as *peer-to-peer* networks. Here, the AN stipulates that each user who sends or receives packets through it (consumer) also becomes a routing member (facilitator) of the overlay network, i.e., peers both send and receive packets for their own application layer as well as route packets for other peers' application layers. The users shown in Figure 2.1 are either clients (if the AN distinguishes roles) or peers (if the AN is a peer-to-peer network), but not infrastructure nodes.

2.2 Anonymity Properties

Assume for a moment that users in Figure 2.1 communicate in pairs, with the traffic flowing as is depicted, and that the AN forwards packets instantaneously and without

loss. We can see how a network-only adversary recording timestamp and size for packets right after they have left the users' devices and right before they arrive at the users' devices would have an easy time identifying which two users form a pair: at each recipient, simply check which of the recorded metadata sequences for packets going into the AN is an exact match and thus identifies the (only) sender. The purpose of the AN is to make it so that users can send and receive packets for their application layer without adversaries conducting traffic analysis attacks learning which users are communicating with each other. The ideal anonymous routing layer, transporting packets from a set of senders to a set of recipients as in Figure 2.1, would be able to hide the following properties (inspired by [10]) from even the strongest adversaries (see Section 2.3):

1. who the non-adversarial sender of any set of packets received by a user was,
2. who the non-adversarial recipient of any set of packets sent by a user will be,
3. who communicates with whom when the adversary is neither sender nor recipient,
4. and whether any application-layer communication occurred at all.

Note how the mentions of "set" in the ideal anonymity properties (1) and (2) above encompass both single-packet observations as well as multi-packet observations (which may form flows at the application layer) collected over potentially long periods of time.

Note that in most cases it is not a design goal to hide the fact that a user is connected to an AN. Most ANs publicly announce how to connect to them, which allows capable adversaries to monitor (at least) their interface nodes (i.e., the nodes "at the edge" of the AN). A notable exception (which is out of scope for this thesis) constitutes the case when an AN also strives to provide *censorship circumvention* or *censorship resistance* (see for context [7, 19, 20]),[1] in which case additional protections may be needed.

Pfitzmann and Hansen [10] provide more precise terminology for the above ideal anonymity properties. Their definitions of *unlinkability*, its negation *linkability*, and *unobservability* have widely influenced the academic field and we restate them below and apply them to the concrete properties we expect of our ideal functionality above.

They say: "*Unlinkability* of two or more items of interest (IOIs, e.g., subjects, messages, actions, …) from an attacker's perspective means that within the system (comprising these and possibly other items), the attacker cannot sufficiently distinguish whether these IOIs are related or not" [10]. Linkability follows immediately as the negation of unlinkability: "*Linkability* of two or more items of interest […] from an attacker's perspective means that within the system […], the attacker can sufficiently distinguish whether these IOIs are related or not" [10].

[1] `https://web.archive.org/web/20250328104940/https://refraction.network/`

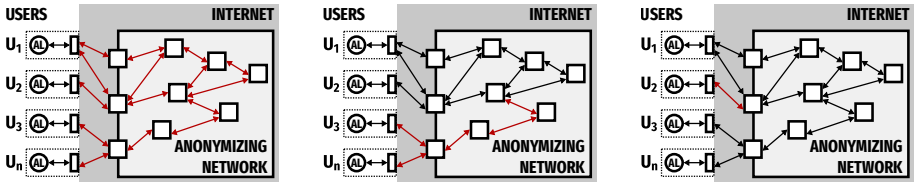
By casting our ideal anonymity properties above as "items of interest" (e.g., for property (1): "who sent this set of packets received by this user"), we can apply the concept of unlinkability to express ideal anonymity property (1): *sender anonymity* [10] against a particular adversary means that this adversary is unable to sufficiently distinguish whether a set of packets received by a user is related to one particular sender, i.e., the adversary is unable to link these items of interest and they are thus unlinkable. The definition of *recipient anonymity* [10] as an expression of ideal anonymity property (2) follows accordingly (a set of packets sent by a user being related to a particular recipient).

Clearly, ideal anonymity properties (1) and (2) cannot hold for packets that the adversary sends and receives between their own users, which is why we restrict these properties to non-adversarial senders and recipients, respectively. For ideal anonymity property (3), we assume the adversary not to be involved in the packet exchanges and can use this property to reason about the anonymity of the respective senders and recipients. Expressing this anonymity property in terms of unlinkability, we say that *relationship anonymity* [10] against a concrete adversary who is neither the sender nor the recipient for the items of interest means that this adversary is unable to sufficiently distinguish whether one or more participants are related to one other participant, i.e., these items of interest are unlinkable for the adversary.

Ideal anonymity property (3) thus hides the *link* between the observations of who will receive a set of packets arriving at the AN and who sent a set of packets departing from the AN. In this property lies the potential for an appropriately capable AN to provide *sender-recipient relationship anonymity* towards strong adversaries that are not involved in the packet exchanges. This notion is thus often called *third-party anonymity* [11] and one of the most important anonymity targets that ANs can strive for.

All the above properties are relative to *anonymity sets* of the respective entity or entities of interest. For example, the true sender of a set of packets exiting an AN can only hope to be anonymous with respect to having sent this set of packets among other participants that the adversary in question considers to be likely alternatives. This set of likely alternatives is called the *sender anonymity set* [10], i.e., the set of AN participants that the adversary deems to be potential senders of the set of packets in question. The *recipient anonymity set* and *relationship anonymity set* are defined analogously [10].

Finally, in order to realize ideal anonymity property (4), we turn to *unobservability*. In the context of a packet exchange between a sender and a recipient, unobservability distinguishes between the participants involved in the exchange and any entity outside of the exchange (e.g., network-only observers, infrastructure nodes, other users). Towards uninvolved parties (third parties), unobservability stipulates that they must be unable to detect whether any genuine application-level packet is sent at all [10]. Given that clearly some packets were observed, this property in practice means that third parties must not be able to tell genuine application-level packets from "noise" packets, i.e., packets containing random or garbage data. This first property is thus called *undetectability* [10].



(a) Global, $\mathcal{N} = g$ (b) Partial, $\mathcal{N} = p$ (c) Local, $\mathcal{N} = l$

Figure 2.2: Network visibility $\mathcal{N} \in \{ g, p, l \}$ of the adversary (in red) into an AN.

However, sender and recipient in a packet exchange can obviously tell genuine packets from noise. Towards them, unobservability provides the respective notion of anonymity. For example, it is undetectable if senders send (*sender unobservability* [10]) and recipients receive (*recipient unobservability* [10]). Regarding the property most relevant with respect to third party adversaries, *relationship unobservability*, Pfitzmann and Hansen state: "*Relationship unobservability* then means that it is sufficiently undetectable whether anything is sent out of a set of could-be senders to a set of could-be recipients" [10]. And with that, unobservability perfectly captures our ideal anonymity property (4).

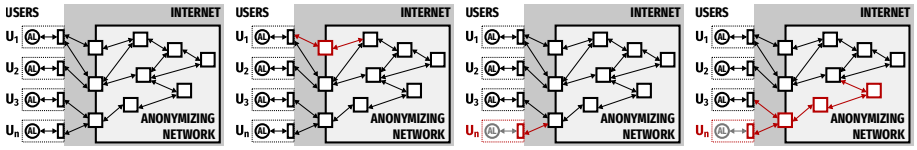
2.3 Adversaries

A concrete AN adversary $\mathbb{A} = (\mathcal{N}, \mathcal{F}, \mathcal{M}, \mathcal{O})$ is defined by their capabilities $\mathcal{N}$, $\mathcal{F}$, and $\mathcal{M}$, and their attack objectives $\mathcal{O}$, which aim at subverting the AN's anonymity properties as just defined in Section 2.2. $\mathcal{N}$ refers to the adversary's **visibility at the network level**, $\mathcal{F}$ to their **insider visibility**, and $\mathcal{M}$ to the **mode** in which the adversary operates. After in turn introducing each element making up this adversarial four-tuple, we showcase commonly used adversaries with different assignments for each of them.

2.3.1 Network Visibility

Adversarial network visibility $\mathcal{N}$ describes the fraction of network links underlying an AN that the adversary can observe, i.e., record packet metadata (timestamp, direction, size) for. These network links run between the AN's infrastructure nodes and between the AN and all its users. We visualize the three discussed levels of network visibility for characterizing AN adversaries in Figure 2.2.

First, if the fraction amounts to all links, we speak of a *global* network adversary (represented as $\mathcal{N} = g$, in Figure 2.2a). A global network adversary is able to observe all network links and thus any packet flowing through the anonymizing network on



(a) External, $\mathcal{I} = 0$ (b) Infrastructure, $\mathcal{I}{=}i$ (c) User, $\mathcal{I} = u$ (d) Both, $\mathcal{I} = i{+}u$

Figure 2.3: Insider visibility $\mathcal{I} \in \{ 0, i, u, i{+}u \}$ of the adversary (in red) into an AN.

any hop. While it is not certain that any single real-world attacker ever is truly able to monitor all network links of an AN, many large entities may come dangerously close to this concept. For example, operators of large *autonomous systems (AS)*, of important *internet exchange points (IXP)*, or of the Internet backbone, or nation-state intelligence agencies (particularly if they cooperate with other countries' intelligence agencies) may be capable of monitoring enough of the network links underlying an AN that it makes sense to approximate their network visibility as global.

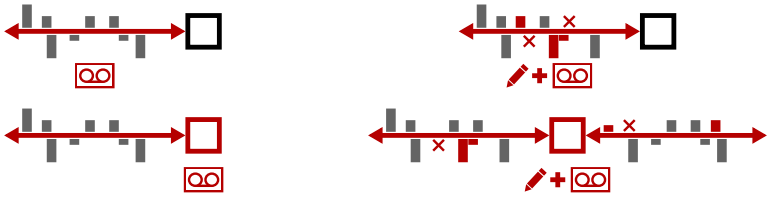
Second, a *partial* network adversary ($\mathcal{N} = p$, in Figure 2.2b) has network link visibility into a significant part of the AN, which means they may observe multi-hop packet routes through the AN but possess nowhere near global network visibility. Any regular AS or large *internet service provider (ISP)* falls into this category.

Third, a *local* network adversary ($\mathcal{N} = l$, in Figure 2.2c) has "spotlight-only" visibility into the AN, i.e., local to hyper-local network monitoring abilities. For example, the ISP of U_2 in Figure 2.2c, operating the link between the user and the AN's interface node, might be considered a local network adversary if they turned malicious. Operators of network access points (e.g., coffee shops, airports) are other examples for this category.

2.3.2 Insider Visibility

An adversary may also control (operate or compromise) one or more participants of the AN under consideration. In peer-to-peer ANs, the adversary immediately becomes part of the network simply by joining the AN with their own nodes. However, becoming part of the infrastructure of an AN that does distinguish users from service nodes is oftentimes also not difficult for the adversary. This is due to client-server ANs commonly relying on their community to contribute infrastructure nodes, which the adversary may thus choose to do. Additionally, if the adversary has the means to launch legal, social, or technical attacks against AN participants of interest, the adversary may be able to obtain the desired insider visibility by compromising existing (e.g., high-bandwidth) nodes.

If the adversary is not an insider of the AN under attack, we refer to them as being *external*



(a) Passive, $\mathcal{M} = p$ (b) Active, $\mathcal{M} = a$

Figure 2.4: Mode of operation $\mathcal{M} \in \{ p, a \}$ of adversary (in red) in an AN. The scenario at the top of each mode of operation shows the case for the external network adversary, while the scenario at the bottom of each mode visualizes the case for the insider adversary.

($\mathcal{I} = 0$, in Figure 2.3a). An example of this is a network-only adversary, observing network links but without control of any members of the AN's overlay network. Once the adversary does control at least one AN participant, we say the adversary is an *insider*, as they become privy to the overlay network's application layer at these AN members. If the AN operates in the client-server model, the adversary may operate or compromise either only *at least one* infrastructure node ($\mathcal{I} = i$, in Figure 2.3b), only *at least one* user ($\mathcal{I} = u$, in Figure 2.3c), or *at least one* user and *at least one* infrastructure node ($\mathcal{I} = i{+}u$, in Figure 2.3d).

We note three important details. First, by controlling at least one AN participant, the adversary is also able to monitor all network links (all incoming and outgoing packets) of this node, i.e.: $\mathcal{I} \in \{ i, u, i{+}u \} \implies \mathcal{N} \in \{ l, p, g \}$. Second, while the access of an insider adversary on any packet exchange where they are not one of the involved users is still restricted to packet metadata, we are now considering packets at the overlay network level and not (only) the physical network level anymore. As a result, such insider adversary may additionally collect (and use) whichever metadata are generated when sending packets between hops on the overlay network, which may result in richer and cleaner metadata sequences collected for the observable packet flows. Third, we exclude the case where all but one of the AN users are under adversarial control, as in this case, anonymity for the single remaining (honest) user is impossible.

2.3.3 Mode of Operation

The adversary can either be *passive* ($\mathcal{M} = p$, in Figure 2.4a) or *active* ($\mathcal{M} = a$, in Figure 2.4b) at the respective components under their control. A passive adversary restricts themself to *recording* (tape symbols in Figure 2.4) the packets' metadata that pass through the resources they observe or control (network links or AN participants or both), while otherwise behaving as their role prescribes. In contrast, an active adversary

Table 2.1: Attack objectives $\mathcal{O}$ that a traffic analysis AN adversary may have.

Symbol	Description
c	Detect if a particular user is *connected* to AN
up	Learn network *usage patterns* of user (intensity, duration, frequency)
sn	Identify *sender node* of particular packet(s)
rn	Identify *recipient node(s)* of particular packet(s)
t	Determine if user contacts set of *targets*
srr	Determine if two users have a *sender-recipient relationship*
all	Recover *all* (observed) sender-recipient relationships

may make use of their ability to *manipulate* (pencil symbols in Figure 2.4) the packet flows passing through them in order to reach their objectives, e.g., to cause disruptions, induce *watermarks* (identifiable patterns in packet flows), or otherwise manipulate packet flows favorably for their purposes of attacking. Depending on the extent of their ability, such adversary may delay, drop, insert (e.g., duplicate), or modify the packets. Making use of just one of these capabilities suffices for the adversary to be called active.

While it may be nearly impossible to detect the presence of a passive adversary, an active adversary risks detection by other participants of the AN, if appropriate cryptographic precautions are in place and participants have an idea of the behavior they expect from other AN participants. The distinction between a passive and an active adversary is thus oftentimes more a matter of whether the adversary is willing to risk detection than one of aptitude. Even an external adversary controlling only a network link (e.g., an ISP) has the ability to tamper with the carried packet flows (e.g., delay or drop packets), however, may choose to operate in passive mode to avoid getting caught.

2.3.4 Objectives

In Table 2.1, we present a list of objectives $\mathcal{O}$ that an adversary conducting a traffic analysis attack against an AN may pursue. The objectives are not mutually exclusive and many adversaries will pursue multiple objectives at once. Any AN adversary relevant for this thesis uses their network visibility $\mathcal{N}$, their insider visibility $\mathcal{I}$, and their mode $\mathcal{M}$ to try and subvert the anonymity properties from Section 2.2 that the AN under attack aims to achieve and that stand in the way of the adversary realizing their objectives. For example, an adversary with $\mathcal{O} = sn$ launching a traffic analysis attack against an AN which provides its users sender unobservability deploys their capabilities to negate the AN's anonymity properties (1) and (4), i.e., the AN's ability to hide who the sender of

one or more packets was and whether packets are genuine application-level packets or just noise. We discuss more complete adversary examples in Section 2.3.5 below.

Adversaries aim to recover the ability to observe, detect, or link the sender(s), recipient(s), or relationship(s) of interest that the AN under attack has worked to hide by means of its anonymity mechanism (see Section 2.2). Towards this, adversaries may exploit deficiencies (which are often explicitly made trade-offs) in the AN's design that lead to small amounts of information leaking each time an event occurs in the AN (e.g., a packet traversing one hop). Given enough time for their attack, sufficiently much information may have leaked and been accumulated by the adversary to realize their objective(s). We then say that the adversary *deanonymized* the users targeted by the attack. Note that we only list the objectives of importance to this thesis in Table 2.1. While other motivations for conducting traffic analysis attacks may exist, they are not relevant for us here.

2.3.5 Common Adversaries

One commonly assumed adversary is the **global passive adversary** $\mathbb{A}_{\mathrm{GPA}} = (\, g, \mathcal{I} \in \{\, 0, i, u, i{+}u \,\}, p, \{\, c, \mathit{up}, \mathit{sn}, \mathit{rn}, t, \mathit{srr}, \mathit{all} \,\}\,)$. While insider visibility may vary widely between concrete instances of the $\mathbb{A}_{\mathrm{GPA}}$, emphasis with this adversary lies on their *omnipresence*, allowing them to observe every packet sent in the AN while not giving away their presence by becoming active. Given this vast reach and the fact that they are typically not assumed to be a single person, the objectives of the $\mathbb{A}_{\mathrm{GPA}}$ may be many and diverse, as represented in their large set for $\mathcal{O}$. The cross-jurisdictional intelligence alliance *Five Eyes*[2] is often assumed to possess $\mathbb{A}_{\mathrm{GPA}}$ capabilities.

Two **flow correlation adversaries** are relevant to this thesis: the insider variant $\mathbb{A}_{\mathrm{IFC}} = (\, p, \mathcal{I} \in \{\, i, u, i{+}u \,\}, p, \{\, \mathit{srr}, \mathit{all} \,\}\,)$ and the network-only variant $\mathbb{A}_{\mathrm{EFC}} = (\, p, 0, p, \{\, \mathit{srr}, \mathit{all} \,\}\,)$. These adversaries observe packet flows *at the edge* of the AN under attack, either by controlling nodes interfacing between users and the AN or by observing network links between users and the AN's interface nodes. Their goal is to link (as many) packet flows going into the AN to their counterparts departing from the AN, thus recovering sender-recipient relationships. These adversaries remain passive, to stay undetected for as long as possible in order to maximize the time they are able to collect packet metadata. While they can typically start applying their attack even with limited network visibility (e.g., by controlling just two interface nodes), the usefulness of their attack (particularly towards objective *all*) increases dramatically the more of the AN's edge nodes these adversaries cover. The $\mathbb{A}_{\mathrm{IFC}} = (\, p, i, p, \{\, \mathit{srr}, \mathit{all} \,\}\,)$ is the adversary we evaluate in Chapter 9, while we assume $\mathbb{A}_{\mathrm{IFC}} = (\, p, i{+}u, p, \{\, \mathit{srr}, \mathit{all} \,\}\,)$ to be the adversary for Chapter 10.

[2] `https://web.archive.org/web/20250321040930/https://privacyinternational.org/learn/five-eyes`

The important quality of **watermarking adversaries** is their ability to induce patterns into packet flows that they can monitor for at other eavesdropping locations, i.e., they are active and observe more than one of the AN's network links. Concrete watermarking adversaries, however, vary widely in their level of network or insider visibility. As one example, we deploy watermarking adversary $\mathbb{A}_{\mathrm{W}} = (\, p, i{+}u, a, \{\, sn, rn \,\}\,)$ in our study described in Chapter 8. This adversary is able to trigger a particular packet exchange to occur with their user-level access to overlay network flows. Then, the adversary looks for the pattern of this packet exchange with as many infrastructure nodes of a certain type as they are able to deploy in order to identify a specific node on the packets' route. However, watermarking can also work well exclusively at the network level (i.e., $\mathcal{I} = 0$).

Finally, **website fingerprinting adversaries** $\mathbb{A}_{\mathrm{WF}} = (\, l, 0, p, t \,)$ compare a user's bidirectional packet flow with the AN to network profiles from a database containing such fingerprints for a number of target webpages. The adversary has constructed this database ahead of the attack and, with it, aims to identify which of the target webpages the user is visiting, if any. As an example, the $\mathbb{A}_{\mathrm{WF}}$ can manifest in the form of a malicious ISP, who is targeting one or more of their users connected to the AN.

2.4 Anonymity Metrics

Over the course of research into the field of anonymous communication, many metrics have emerged to measure the anonymity provided by an AN [3, 8, 9]. These can be used to assess the strength of an AN in the face of a concrete adversary and compare different AN designs regarding their strength. Metrics based on the information-theoretic concept of *entropy* [4, 14] are oftentimes used to quantify the uncertainty of an adversary at singling out the sender or the recipient of a packet from among all candidates. They are thus useful when reasoning about attacks where the adversary decides which participant is the likeliest according to a probability distribution. Here, it is implicitly assumed that the adversary has a global view on participants and always chooses an assumed sender or recipient for a packet. If these assumptions do not apply, we have to use different metrics. One common way to model a traffic analysis attack is in the form of a *binary classifier*, deciding whether a candidate pair of flows is indeed related, and we discuss the metrics used to evaluate these classifiers in the following.

2.4.1 Metrics for Binary Classification

A binary classifier computes a score in $[\,0.0, 1.0\,]$ that expresses how close the classifier considers the submitted sample to be to either of its two classes, 0.0 or 1.0. In the context of traffic analysis attacks, the class represented by 0.0 may mean that the two flows under consideration are deemed definitely not to be related by the binary classifier, while 1.0

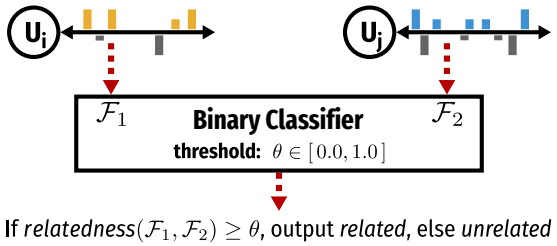


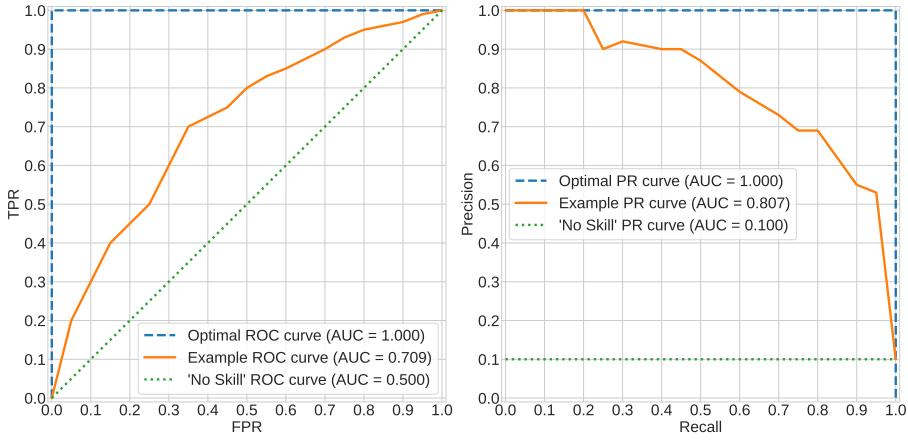
Figure 2.5: Model of how a binary classifier is used to decide whether two packet flows are related in some form or not.

then means that the classifier is absolutely certain that the two flows are related. Any score between these two extremes enables the classifier to express how confident it is in the class membership prediction (which, depending on how the binary classifier is realized, may or may not represent a probability of class membership).

Classifiers are assessed *empirically* and we compute metrics for binary classification [17] to judge how well (or not) the AN provides anonymous communication to its users. We do so by collecting a dataset of samples (i.e., pairs of packet flows) on the AN for which we know each sample's *ground truth*, i.e., whether it is a *positive* (with ground truth label P) or a *negative* (ground truth label N). Then, the binary classifier is trained to its best ability on one part of the dataset and tested for how well it performs on another part of the dataset, such that we judge the classifier's performance on data it has not yet seen.

During testing, the classifier assigns a numeric score to each sample submitted to it. By adopting a threshold $\theta \in [0.0, 1.0]$, we turn the score into a binary prediction. For example, any sample with a score equal to or above θ is predicted to be a positive (called *predicted positive*, with label PP), while any sample with a score below θ is predicted to be a negative (*predicted negative*, label PN). The sample is a *true positive* (TP) if PP $=$ P and a *true negative* (TN) if PN $=$ N. In these cases, the classifier performed well as it predicted in line with ground truth. However, the classifier may also make mistakes and mislabel samples. A misclassified negative (PP $=$ N) is called a *false positive* (FP), while a misclassified positive (PN $=$ P) is a *false negative* (FN). Choosing θ in $[0.0, 1.0]$ thus allows the operator of the classifier to decide whether to prioritize accepting false positives or accepting false negatives.

The canonical use of binary classifiers for traffic analysis (e.g., any flow correlation adversary from Section 2.3.5) is to have them decide whether two packet flows are related by learning how packet flows are transformed in their observable network metadata when they traverse an AN. Such a classifier thus expresses how confident it is that one specific packet flow at one end of the AN is some other specific packet flow at the other end of the AN. We visualize this binary classifier application for determining



(a) ROC curve examples.

(b) PR curve examples.

Figure 2.6: Example ROC and PR curves, including their respective AUC values.

the relatedness of two packet flows at an abstract level (including θ) in Figure 2.5.

With all TP, TN, FP, and FN obtained on the testing dataset and at specific threshold θ, we can compute different aggregate metrics that express how well the binary classifier performs overall. The *true positive rate* (TPR) [13, 17], also called *recall* or *sensitivity*, describes how many of the positive samples the classifier identified as positives: $\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP+FN}}$. Analogously, the *false positive rate* (FPR) [17] captures how many of the negatives were falsely labelled as positives: $\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{TN+FP}}$. *Precision* [17, 18] captures how many of the predicted positives were truly positives: $\mathrm{Precision} = \frac{\mathrm{TP}}{\mathrm{TP+FP}}$. And *accuracy* [17] additionally considers negatives, expressing the ratio of correctly labelled samples out of all: $\mathrm{Acc} = \frac{\mathrm{TP+TN}}{\mathrm{TP+TN+FP+FN}}$.

While useful, the above metrics all depend on threshold θ that determined how many samples were classified correctly and incorrectly. We consider these metrics in the context of attacks on ANs and the prediction threshold is entirely up to the adversary, who may wish to select the threshold according to their preferences. For example, the adversary may prioritize detecting more of all positives while accepting also more negatives predicted as positives or to have high confidence in the predictions being correct. Two graphical measures are commonly used to facilitate the adversary's decision on making this trade-off, i.e., choosing the value for θ: the *receiver operating characteristic (ROC)* curve [6, 12, 17] and the *precision-recall (PR)* curve [17]. We visualize optimal, example, and baseline curves for both measures in Figure 2.6.

The ROC curve plots a classifier's TPR against its FPR (see Figure 2.6a), allowing the

adversary to decide how to trade off producing more true positives in exchange for also increasing the number of false positives. An optimal classifier—making no mistakes—obtains an ROC curve maximally to the top-left corner of the plot, while a classifier without any skill achieves exactly the bottom-left to top-right diagonal through the ROC plot. If a classifier shows skill, its ROC curve will thus lie between the "no skill" classifier's ROC curve and the optimal classifier's ROC curve. All three example ROC curves are shown in Figure 2.6a. Any ROC curve further to the top-left of the ROC plot is better than another ROC curve closer to the diagonal, allowing to compare the performance of different classifiers on the same setting. However, doing so graphically may not always be conclusive. Here, the *area under the ROC curve (ROC-AUC)* score can help by providing a single value that can be compared to other ROC-AUCs.

ROC curves and ROC-AUCs are useful when working with *balanced* datasets, in which the number of positives and negatives are equal. However, the ROC curve is invariable to changes in the ratio of positives to negatives [17]. This makes the ROC curve and ROC-AUC not the ideal tool to decide on θ when the positive class is a rare event that the adversary cares tremendously about. On such *imbalanced* datasets, the adversary can misjudge classifier performance when only relying on ROC curves and ROC-AUC.

On imbalanced datasets, the PR curve and its *area under the PR curve (PR-AUC)* value (shown in Figure 2.6b) are preferable. The PR curve plots precision versus recall (TPR) and therefore does not involve the TN performance of a classifier, i.e., how well it predicts negative samples. Changing the ratio of positives to negatives leads a classifier's PR curve and thus PR-AUC value to change as well. Any skillful classifier achieves a PR curve above the horizontal line for the baseline classifier predicting with base rate precision of the positive class. An optimal classifier achieves a PR curve pegged to the top and right borders of the plot, as it always achieves perfect precision and recall. See Figure 2.6b for exemplary visualizations of these PR curves and their PR-AUC values.

Binary classification metrics—PR-AUC in particular—are a good fit for judging the performance of an AN attack and potential defenses to it. Traffic analysis attacks can be modelled well as binary classifiers, datasets are often imbalanced, and the tension between attack and defense is immediately visible: improving the attack increases the adversarial classifier's PR-AUC, while deploying an effective defense reduces it.

2.4.2 Metrics Involving Practical Constraints

While it is useful to reason about the level of anonymity an AN is able to provide or the success of an attack at compromising this anonymity by way of the above described binary classification metrics, we often want to go beyond these "clean room" metrics and involve practical constraints such as *duration of metadata collection or attack* or *probability of compromise* in our assessments. In particular, when we need to prioritize between multiple angles of attack based on their impact or relevance to a concrete deployed AN.

The first constraint we consider is the *time the adversary is able to collect or attack* the packet metadata for. Given that, in practice, adversarial resources are also often limited and one can assume that the adversary would want to optimize the resources they are willing to spend on any one packet flow or user, this metric allows to factor in this adversarial budgeting logic. We may say that "the adversary achieves this level of attack performance after collecting or attacking packet flows for that amount of time" to connect binary classification performance with the time over which they were established. If it is shown that even short periods of metadata collection or attack lead to high attack performance, this can underscore the risk of a particular attack and help raise awareness of the issue with the AN's designers.

Secondly, traffic analysis adversaries are parameterized, among other things, by their visibility into the AN's network or participants. This means that the adversary needs to acquire a particular vantage position in the network or at the AN level in order to be able to launch their attack. Once acquired, two attacks conducted over vastly different vantage positions may produce relatively similar binary classification results (e.g., on equal-sized testing datasets). However, factoring in the *difficulty or probability of the adversary to acquire the respective vantage position* may be important additional detail when judging the probability of compromise and thus real-world impact of the attack.

2.5 Connection Between Anonymity and Resilience

The final aspect we consider for anonymizing networks generally is the connection between the overall anonymity an AN can provide to its users and its *resilience* [2]. Resilience (or *fault tolerance*) refers to the AN's ability to cope with failures, i.e., its ability to continue achieving its purpose (routing packets anonymously) when things go wrong. By their nature, ANs are networked services between multiple network participants. Oftentimes, their anonymity mechanism is in fact based on their being distributed systems, in which a large to a very large number of network nodes must cooperate in order to facilitate the overall system objective [15]. Given that failures are always a possibility (even in a single computing node), if we both increase the number of computing nodes to that of a very large system and add on top the complexities of a planet-spanning network (the Internet), the occurrence of failures is not a question of if but when and how severe.

Broadly, failures can arise from two sources: from *computing node errors* (e.g., hardware defects, programming errors) and from *network errors* (e.g., lost packets, broken network links). It is possible to preempt both types of failure to such a degree that their effect on the overall system is limited. But both also have the potential to "bubble up" from their low level to the system level and cause an outage, if the distributed system was not prepared to handle them. Additionally, while they can occur due to *benign*, random

reasons (e.g., hard disks failing due to age), failures can just as well be induced *intentionally* by some actor inside or outside the system with the aim to disrupt it. In the context of these distributed systems being ANs, benign and intentionally caused failures can have a devastating impact on the overall goal of the service. For the purposes of this discussion, it is irrelevant if a particular failure is in fact due to a benign or an intentional cause. The fact that the failure occurred and needs to be reacted to, is the important part here.

Consider for a moment the following to be true: users in a large and distributed AN are either *high-risk* users or *low-risk* users. High-risk users, e.g., human rights activists under repressive regimes or investigate journalists in corrupt countries, depend (not exclusively) on anonymous communication tools to safeguard their and their sources' safety. Low-risk users want to benefit from and strengthen the AN's anonymity by contributing to its anonymity sets but they do not fear repercussions if they instead sent their packets through less anonymous or even unprotected networks. We would like the AN to service as many users as possible to increase the potential for maximal anonymity sets and for it to remain *available* for operation, i.e., not to cease working correctly.

Now consider an AN with low resilience. Once failures manifest (which, as we stated above, is a certainty) and the AN is not able to react to them in a way ensuring its continued availability, it becomes unavailable to some or even all users. Low-risk users may feel annoyed, retry a few times, but if the outage persists may switch away to less anonymous or unprotected networks to continue their communications. High-risk users cannot do the same without potential harm to themselves or their connections, which means they cannot communicate at all. Thus, benign or intentional failures have caused high-risk users to cease communicating online and exposed low-risk users to higher risks of traffic analysis because they switched to less anonymous or even unprotected networks, severely reducing both groups' privacy. In contrast, not only is a resilient AN able to continue routing packets anonymously even in the face of failures, it may also attract a larger user base than the fragile AN due to its higher reliability and dependable performance level—which in turn may lead to larger anonymity sets again [1, 5, 16].

Bibliography

[1] A. Back, U. Möller, and A. Stiglic. "Traffic Analysis Attacks and Trade-Offs in Anonymity Providing Systems". In: *Information Hiding*. Ed. by I. S. Moskowitz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 245–257. ISBN: 978-3-540-45496-0 (p. 29).

[2] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz. "Denial of Service or Denial of Security?" In: *Proceedings of the 14th ACM Conference on Computer and Communications Security*. CCS '07. Alexandria, Virginia, USA: Association for Computing Machinery, 2007, pp. 92–102. ISBN: 9781595937032. DOI: 10.1145/1315245.1315258. URL: https://doi.org/10.1145/1315245.1315258 (p. 28).

[3] C. Diaz. "Anonymity Metrics Revisited". In: *Anonymous Communication and its Applications*. Ed. by S. Dolev, R. Ostrovsky, and A. Pfitzmann. Vol. 5411. Dagstuhl Seminar Proceedings (DagSemProc). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2006, pp. 1–6. DOI: 10.4230/DagSemProc.05411.2. URL: https://drops.dagstuhl.de/entities/document/10.4230/DagSemProc.05411.2 (p. 24).

[4] C. Díaz, S. Seys, J. Claessens, and B. Preneel. "Towards Measuring Anonymity". In: *Privacy Enhancing Technologies*. Ed. by R. Dingledine and P. Syverson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 54–68. ISBN: 978-3-540-36467-2 (p. 24).

[5] R. Dingledine and N. Mathewson. "Anonymity Loves Company: Usability and the Network Effect". In: *5th Annual Workshop on the Economics of Information Security, WEIS 2006, Robinson College, University of Cambridge, England, UK, June 26-28, 2006*. 2006. URL: https://econinfosec.org/archive/weis2006/docs/41.pdf (p. 29).

[6] P. A. Flach. "ROC Analysis". In: *Encyclopedia of Machine Learning and Data Mining*. Ed. by C. Sammut and G. I. Webb. Boston, MA: Springer US, 2017, pp. 1109–1116. ISBN: 978-1-4899-7687-1. DOI: 10.1007/978-1-4899-7687-1_739. URL: https://doi.org/10.1007/978-1-4899-7687-1_739 (p. 26).

[7] S. Khattak, T. Elahi, L. Simon, C. M. Swanson, S. J. Murdoch, and I. Goldberg. "SoK: Making Sense of Censorship Resistance Systems". In: *Proceedings of the 16th Privacy Enhancing Technologies Symposium (PETS 2016)*. Vol. 2016. 2016, pp. 37–61. DOI: 10.1515/popets-2016-0028 (p. 17).

[8] T. Lu, Z. Du, and Z. Jane Wang. "A Survey on Measuring Anonymity in Anonymous Communication Systems". In: *IEEE Access* 7 (2019), pp. 70584–70609. DOI: 10.1109/ACCESS.2019.2919322 (p. 24).

[9] S. J. Murdoch. "Quantifying and Measuring Anonymity". In: *Data Privacy Management and Autonomous Spontaneous Security*. Ed. by J. Garcia-Alfaro, G. Lioudakis, N. Cuppens-Boulahia, S. Foley, and W. M. Fitzgerald. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 3–13. ISBN: 978-3-642-54568-9 (p. 24).

[10] A. Pfitzmann and M. Hansen. *A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management*. v0.34. Aug. 2010. URL: http://dud.inf.tu-dresden.de/literatur/Anon%5C_Terminology%5C_v0.34.pdf (pp. 17, 18, 19).

[11] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis. "The Loopix Anonymity System". In: *26th USENIX Security Symposium (USENIX Security '17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1199–1216. ISBN: 978-1-931971-40-9. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/piotrowska (p. 18).

[12] "ROC Curve". In: *Encyclopedia of Machine Learning and Data Mining*. Ed. by C. Sammut and G. I. Webb. Boston, MA: Springer US, 2017, pp. 1116–1116. ISBN: 978-1-4899-7687-1. DOI: 10.1007/978-1-4899-7687-1_735. URL: https://doi.org/10.1007/978-1-4899-7687-1_735 (p. 26).

[13] "Sensitivity". In: *Encyclopedia of Machine Learning and Data Mining*. Ed. by C. Sammut and G. I. Webb. Boston, MA: Springer US, 2017, pp. 1152–1152. ISBN: 978-1-4899-7687-1. DOI: 10.1007/978-1-4899-7687-1_751. URL: https://doi.org/10.1007/978-1-4899-7687-1_751 (p. 26).

[14] A. Serjantov and G. Danezis. "Towards an Information Theoretic Metric for Anonymity". In: *Privacy Enhancing Technologies*. Ed. by R. Dingledine and P. Syverson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 41–53. ISBN: 978-3-540-36467-2 (p. 24).

[15] M. van Steen and A. S. Tanenbaum. *Distributed Systems*. 4th ed. 2023. ISBN: 978-90-815406-4-3. URL: https://distributed-systems.net (p. 28).

[16] P. Syverson. "A Peel of Onion". In: *Proceedings of the 27th Annual Computer Security Applications Conference*. ACSAC '11. Orlando, Florida, USA: Association for Computing Machinery, 2011, pp. 123–137. ISBN: 9781450306720. DOI: 10.1145/2076732.2076750. URL: https://doi.org/10.1145/2076732.2076750 (p. 29).

[17] A. Tharwat. "Classification assessment methods". In: *Applied Computing and Informatics*. Vol. 17. 1. Emerald Publishing Limited, 2021, pp. 168–192. DOI: 10.1016/j.aci.2018.08.003. URL: https://doi.org/10.1016/j.aci.2018.08.003 (pp. 25, 26, 27).

[18] K. M. Ting. "Precision". In: *Encyclopedia of Machine Learning and Data Mining*. Ed. by C. Sammut and G. I. Webb. Boston, MA: Springer US, 2017, pp. 990–990. ISBN: 978-1-4899-7687-1. DOI: 10.1007/978-1-4899-7687-1_658. URL: https://doi.org/10.1007/978-1-4899-7687-1_658 (p. 26).

[19] M. C. Tschantz, S. Afroz, N. withheld on request, and V. Paxson. "SoK: Towards Grounding Censorship Circumvention in Empiricism". In: *2016 IEEE Symposium on Security and Privacy (SP)*. 2016, pp. 914–933. DOI: 10.1109/SP.2016.59 (p. 17).

[20] M. Wu, J. Sippe, D. Sivakumar, J. Burg, P. Anderson, X. Wang, K. Bock, A. Houmansadr, D. Levin, and E. Wustrow. "How the Great Firewall of China Detects and Blocks Fully Encrypted Traffic". In: *32nd USENIX Security Symposium (USENIX Security '23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 2653–2670. ISBN: 978-1-939133-37-3. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/wu-mingshi (p. 17).

Chapter 3

Onion Routing

Onion routing [18, 47, 59] relies on the primitive of *onion encryption*, i.e., the multiple encryption of data with different cryptographic key material for each encryption layer, in order to hide routing information along a packet's path. Onion routing then has each intermediary along the packet's path "peel off" (thus the name) their respective encryption layer, which reveals only the subsequent intermediary to pass the packet on to. Onion routing is an essential building block for many AN designs. We introduce the concept below before discussing the dominant AN implementation of onion routing: Tor [14]. Due to its large user base, attacks and defenses on Tor have received a lot of academic attention, and we overview both aspects below.

3.1 Low-Latency, Connection-Oriented: Onion Routing

The origins of onion routing and ANs built primarily around this primitive (the authors tracked the evolution of the designs in *generations*) trace back to the mid- to late-1990s [8, 15, 18, 47, 59, 60]. At that time, several AN designs and prototypes for *high-latency* communications already existed, e.g., designs based on the packet-oriented mixnet concept (discussed in Chapter 4). However, AN designs for the packet flows of networked applications that require *low packet latency* (e.g., web browsing, interactive remote shell) and reduced computational load were missing.

This gave rise to the development of the concept of onion routing, in the beginning at the Naval Research Laboratory (NRL) in Washington, D.C., in late-1995 [8, 60]. The concept works as follows: an *initiator* (e.g., a user) selects a path of multiple overlay network hops (called *onion routers*) to a *responder* (e.g., a web server) in order to establish

a low-latency virtual connection (called *circuit*) through the selected onion routers. For as long as the circuit exists, this path from initiator to responder stays the same, hence the connection-oriented nature of onion routing networks. Onion routers are assumed to already have created long-lived encrypted connections with their neighboring onion routers, over which the circuits that onion routing users build are multiplexed. Each onion router in the network possesses a cryptographic key pair of which the public key is (assumed to be) known to anyone wanting to build a circuit through the node.

Users first go through connection setup for a new circuit before they can send data over the circuit. Connection setup initializes the circuit along the path of chosen onion routers and establishes a symmetric key for a stream cipher between the user and each hop. Compared to ANs relying heavily or exclusively on public-key cryptography for each packet, this gives onion routing networks a significant performance advantage from faster cryptographic operations, particularly so over two decades ago. To obfuscate the location of packets on their circuit to network observers, data sent along the circuit (for setup, data transmission, padding, or teardown) is cut into *fixed-size cells*: an application-level data stream is split into multiple cell payloads and the final cell payload is padded to full length, if necessary. Each cell's payload and possibly associated length header (if the cell is a data cell) are wrapped in one layer of encryption for each onion router on the circuit in reverse order, i.e., starting from the onion router closest to the recipient, then the previous one to that, and continuing until the cell has been prepared for the onion router closest to the sender.

Upon receipt of a cell that a user created, an onion router "peels off" its layer from the cell's payload (and possibly the length header) by decrypting it. The remaining per-link headers not covered by onion encryption (i.e., circuit identifier, command, and possibly length) are decrypted following the link encryption scheme between each two neighboring onion routers. This reveals to each onion router only the necessary information—the identifier of the circuit on which the cell is travelling—for it to pass on the transformed cell to its next hop. And so, this cryptographic transformation of the part of each cell that travels end-to-end on the circuit prevents an observer at the link level from tracing packets based on their bit string representation on the network link. Onion routers also pad all outgoing cells to be of fixed cell size again.

Onion routing derives its anonymity properties from the facts that initiators choose the paths of their circuits through the overlay topology of onion routers and that packets cannot be traced between hops purely based on their size and link-level bit string representation. Onion routers only know the preceding hop and the succeeding hop on any circuit they are part of, but no participants beyond these limits. Thus, the first onion router on a circuit knows who the initiator is and which relay takes the second-hop position on the circuit, but no circuit participant beyond that (including the responder). The circuit's last onion router knows to which responder the packets will go and from which relay it received them, but not the initiator who created the packets in the first place. Some of the early onion routing papers [47, 59] even included minimal amounts

of *packet mixing* at onion routers for increased resistance to timing analysis. All packets received within a specified amount of time would be reordered subject to the constraint that packet order on each circuit was preserved.

Different AN implementations at various stages of deployment existed of these early onion routing generations, chiefly among them the NRL-developed prototypes. Other implementations of variations of the same concept include the Freedom Network [17], Tarzan [16], and MorphMix [48]. However, to date, the by far most relevant and most popular implementation of onion routing is Tor [14].

3.2 Tor

Tor [14] has become the version of the onion routing anonymity concept that turned theory and prototype implementations into the most widely deployed and used AN thus far. It made good on design omissions (e.g., network state maintenance and distribution) and shortcomings (e.g., lack of forward secrecy in shared key establishment) of earlier onion routing systems and thus proposed and continues to advance a comprehensive real-world AN. At the time of writing, according to its own metrics, Tor has more than two million directly connecting users,[1] consuming bandwidth of more than 300 Gbit/s [2] provided by over 8,000 volunteer-operated relays.[3] While work on what would become Tor started in the early 2000s and first source code and network versions became available in fall of 2002,[4,5] the canonical publication detailing Tor (including its differences to earlier onion routing networks) was published in 2004 [14].

The onion routers (called *relays*) on the overlay network that Tor forms forward the users' application packet flows along circuits. Tor follows the client-server paradigm and clearly distinguishes between users (clients) and relays (servers), so Tor users are not required to also provide relay functionality when they access network resources through Tor. Each relay can theoretically connect to any other relay in a single overlay network hop, enabling a huge number of potential routes through the network. However, in practice, Tor places restrictions on which relay can occur in which position on a circuit and thus does not implement such *free-route* topology (see Section 4.1.2).

[1] `https://web.archive.org/web/20250417080854/https://metrics.torproject.org/userstats-relay-country.html`

[2] `https://web.archive.org/web/20250416133629/https://metrics.torproject.org/bandwidth-flags.html`

[3] `https://web.archive.org/web/20250416133847/https://metrics.torproject.org/networksize.html`

[4] `https://web.archive.org/web/20250402173003/https://www.torproject.org/about/history/`

[5] `https://web.archive.org/web/20250416144025/https://archive.torproject.org/websites/lists.torproject.org/pipermail/tor-dev/2002-September/002374.html`

One marked difference [14, 15, 60] to earlier onion routing designs is that circuit setup in Tor realizes session key establishment using Diffie-Hellman [13]. This makes Tor forward-secret, as it prevents network observers from storing entire circuit sessions and breaking and decrypting them later. The switch to Diffie-Hellman-based key establishment also means that circuit setup has to proceed in a *telescoping* fashion (originally published by Brown [5]), where circuits are built by successively reaching out to the next relay through the part of the circuit already created. Tor introduced *directory servers*, which collectively maintain and distribute the network's state to all participants at the trade-off that participants need to trust their joint operation to some degree. Users always download the entire network state (the *consensus* document) for any epoch they are part of, to obviate epistemic and other route selection attacks [9, 10, 62]. Komlo et al. [30] propose a mechanism that overcomes the scalability limitations inherent to centralized directory servers. Previous onion routing generations' ideas of packet mixing at relays and cover traffic between relays did not make it into Tor's generation of onion routing.

Tor distinguishes relays into different roles by assigning them *flags* in the consensus[6] and we introduce select ones relevant for this thesis below:

- **Guard:** Each Tor client chooses a small set of Tor relays that carry the "Guard" flag to act as their guards set for some weeks, during which time the first hop of any circuit the client builds is always a relay from this set. Authorities assign relays the "Guard" flag automatically when relays prove over some duration of time that they are reliable and fast (and were not configured not to act as guards).
- **Exit:** Tor relays with the "Exit" flag have routing policies configured that allow Tor clients to send traffic from them towards the public Internet. Relay operators thus decide whether or not their relay should be eligible for the "Exit" flag.
- **Middle:** Any Tor relay initially and always participates as a middle relay, i.e., a relay not in the edge positions of any circuits. Middle relays thus neither see the Tor user's nor the destination's IP address on any of the circuits they are part of.
- **Authority:** As of the time of writing, nine Tor relays are collectively responsible for publishing the consensus document, in which all available relays, important configuration flags, and auxiliary values are distributed.
- **HSDir:** All Tor relays carrying the "HSDir" flag together maintain the distributed *hidden service directory (HSDir)* for onion service descriptors (see Section 3.2.1 below for details on onion services). Relays are automatically assigned the "HSDir" flag by the authorities after proving their reliability and performance. Tor clients intending to connect to an onion service first need to obtain connection details

[6] `https://web.archive.org/web/20250407211551/https://consensus-health.torproject.org/`

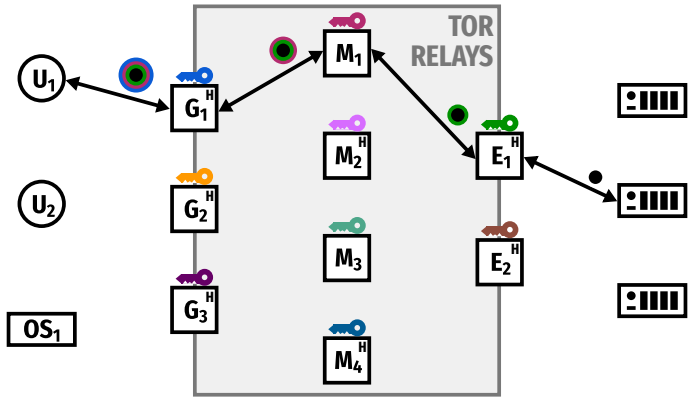


Figure 3.1: Tor user U_1 accessing a server on the public Internet through a three-hop Tor circuit: G_1 (acting as guard), M_1 (middle), E_1 (exit). Cells between the user and E_1 are layer-encrypted using symmetric keys established between the user and each of the circuit's relays. Cell size is used to indicate onion encryption along the circuit, not byte size at the network level. In reality, (most) cells are of fixed size on all Tor links.

about the onion service from some of these HSDirs. Onion services record their connection details in onion service descriptors and publish them to multiple distinct HSDirs throughout the network.

One of the primary use cases of Tor is browsing the web privately. Tor running on a user's device builds circuits of (by default) three hops to relay the communication between the user and, for example, web servers on the public Internet. We show a visualization of such three-hop circuit in Figure 3.1.

When Tor users decide which three relays from the current consensus will carry their next circuit, they take various considerations into account. The first hop on any of their circuits is always one of the relays from their small set of guards. Guards were introduced as a countermeasure to an adversary causing Tor clients to repeatedly build new circuits until the adversary occupied the first hop [42, 67, 68, 69]. The second hop is chosen from the list of all relays (all relays are middles), however, among other things, weighted by the middle's bandwidth. The final hop for circuits meant to carry traffic between a Tor client and a server on the public Internet has to be chosen from the set of exits in the consensus. For this position on a particular circuit, only exits are eligible that are willing to handle traffic towards the intended destination (through exit policies). This description can only serve as a high-level overview of the steps involved in connecting through Tor, as more than two decades of research and engineering work have gone

into improving the network. The official Tor specifications provide more detail.[7]

The other aspect of private web browsing concerns application-level protections against privacy-invasive techniques such as tracking and device fingerprinting. Since 2008, the Tor Project (the non-profit organization behind Tor) publishes *Tor Browser* as a defense against these threats. Tor Browser is a Firefox browser bundled with the Tor software and hardened against web tracking, fingerprinting, and other privacy threats.

3.2.1 Onion Services

A network service that wants to be reachable by Tor users can publicly announce its IP address (also called *network location*) via DNS like any other regular network service. This will make the service's IP address available to any party interested in initiating a connection to them. In this case, however, the network service does not benefit from the same protections (specifically, the ability to hide one's network location) that Tor affords its users. In order for network services to also enjoy location-hiding protections, Tor offers *onion services* (historically called *location-hidden* or just *hidden services* [14]).

Onion services rely on a set of protocols[8] that enable a network service to be reachable by Tor users from within the Tor network (Figure 3.2 shows such an ultimately established circuit). The network service builds three-hop circuits to selected Tor relays and announces to Tor users that they can be contacted anonymously at these relays. *Single-hop* onion services exist (which do not build three-hop circuits from their side), but we ignore them here due to server anonymity not being among their design goals. Once established, a Tor connection between a user and an onion service provides location hiding protections to both sides, which may be of great benefit to services otherwise faced with overpowering *denial of service (DoS)* attacks.

A network service intending to become an onion service first selects a number of Tor relays to act as its *introduction points* and builds three-hop circuits to each of them. It then assembles all information that clients need to connect to it (including the above mentioned introduction points) in its *onion service descriptor* and publishes this descriptor at a deterministic set of HSDirs within a particular time period. Tor users connect to an onion service by means of the service's *onion address* (e.g., `2gzyxa5ihm7nsggfxnu52rck2vv4rvmdlkiu3zzui5du4xyclen53wid.onion` for the Tor Project's web presence). An onion address contains the onion service's public key, allowing the Tor client to determine from which HSDirs to fetch the descriptor. Once the descriptor has been downloaded, the user creates a three-hop circuit to a relay acting as their *rendezvous point*. Then, the user reaches out to the onion service by building a circuit to one of the onion service's introduction points and through it

[7] `https://web.archive.org/web/20250406090952/https://spec.torproject.org/`

[8] `https://web.archive.org/web/20250327184149/https://spec.torproject.org/rend-spec/protocol-overview.html`

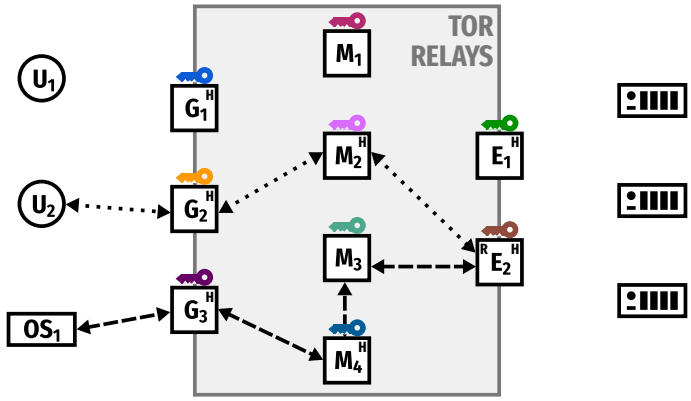


Figure 3.2: Six-hop rendezvous circuit established between Tor user U_2 and onion service OS_1 through Tor. We show the rendezvous circuit that carries the actual application-level data, after the descriptor download and introduction steps were completed (using separate circuits). The rendezvous relay (E_2, indicated by "R") was chosen by U_2 and thus forms the third hop of the user's circuit (marked as the dotted arrows). However, E_2 was not chosen by OS_1, which is why the onion service builds a four-hop circuit (marked as the dashed arrows) to the rendezvous relay. An "H" in a relay's top right corner indicates that this relay carries the "HSDir" flag.

extends an invitation to meet at the user-chosen rendezvous point. If the onion service accepts, it builds a circuit to the rendezvous point. The rendezvous relay fuses the two circuits and the user is now able to access the onion service through the resulting six-hop circuit. Importantly, the user is assured that they are indeed connected to the intended onion service. We visualize an exemplary established rendezvous circuit in Figure 3.2.

3.3 Attacks and Defenses on Tor

Tor being an onion routing network means that many of the limitations of the onion routing concept apply to Tor as well. In particular, being a low-latency, connection-oriented AN limits the adversaries that Tor can protect its users' anonymity from. Since early on [14, 47, 61], it was known that onion routing and other low-latency AN designs could not protect against adversaries who controlled both ends of a connection. Simple packet counting [52] or timing attacks [32, 37, 42, 53] reveal whether an observed link on one side is some other observed link on the other side of the network. Bauer et al. [2] showed that it used to not even be necessary to send any application-level data to conduct this attack, as only circuit setup provided enough information for the attack

to be successful. And active attackers can induce timing patterns into flows to then recognize these perturbations at another observation point in the network [6, 53].

Tor's threat model thus assumes that at most some but not all relays in any circuit may be adversarial or observed at the network level and Tor cannot defend against an adversary having compromised both the first and last relay [14, 27]. Tor thus does not aim to protect against the widely cited global passive adversary ($\mathbb{A}_{\mathrm{GPA}}$), but does allow for traffic tampering by the adversary. These choices regarding adversary strength were explicitly made with Tor's design goals of ease of use, light resource consumption, and performance as well as the state of knowledge at the time of inception in mind.

Tor evolved to defend against attacks that used to be threats but are less severe or disabled now.[9] Guards were introduced to preempt predecessor attacks [42, 67, 68, 69] and their number per user was adjusted in light of new insights.[10] Resource consumption verification was integrated as one defense to a grave DoS attack by Jansen et al. [26].[11] Certain cell types were disabled after a flow correlation attack was successful on the live network.[12] In response to many website fingerprinting attacks, Tor implemented initial defenses based on the WTF-PAD defense proposed by Juarez et al. [29].[13] And the set of onion service protocols was upgraded after studies such as the one by Biryukov et al. [4] demonstrated serious flaws in the previous version.[14]

However, precisely due to Tor's anonymity foundation (onion routing) and aversion to overheads from traffic analysis defenses (e.g., increased latency, bandwidth "wasted" for cover traffic), attacks remain that are realistic threats to Tor users. Das et al. [11] have proved that Tor cannot provide (indistinguishability-based) strong anonymity given its choice of latency and bandwidth overheads. Below, we introduce three real-world threats to Tor users (flow correlation, website fingerprinting, and guard discovery) and some countermeasures.

[9] https://web.archive.org/web/20250422155041/https://spec.torproject.org/proposals/344-protocol-info-leaks.html

[10] https://web.archive.org/web/20241227154748/https://spec.torproject.org/proposals/291-two-guard-nodes.txt

[11] https://web.archive.org/web/20250114015136/https://spec.torproject.org/proposals/289-authenticated-sendmes.txt

[12] https://web.archive.org/web/20250312025133/https://blog.torproject.org/tor-security-advisory-relay-early-traffic-confirmation-attack/

[13] https://web.archive.org/web/20250210230733/https://spec.torproject.org/padding-spec/circuit-level-padding.html

[14] https://web.archive.org/web/20250310182259/https://spec.torproject.org/rend-spec/index.html

3.3.1 Flow Correlation

Tor explicitly excludes scenarios in which an adversary has visibility over both ends of a circuit and conducts an attack to identify which packet flow going into Tor is which flow exiting Tor (known as flow correlation attacks) from its threat model. However, while protection against flow correlation is not a design goal of Tor, the threat these attacks pose to Tor's users may nonetheless be significant. Thus, much research has investigated flow correlation on Tor [2, 6, 7, 25, 27, 37, 39, 40, 41, 42, 50, 58] in order to understand the risks of this attack to users and explore potential countermeasures. While it is typically assumed that the adversary either controls interface nodes (guards, exits) or network links towards non-relay Tor participants to collect the metadata needed to correlate flows, Murdoch and Zieliński have shown [38] even centrally collected and much coarser-grained metadata to be sufficient for successful flow correlation attacks.

The specific mechanism by which the adversary arrives at a prediction for a packet flow pair (representing the two ends of a Tor circuit) is dependent on the adversary's abilities and objectives (see Section 2.3). Techniques [45] to determine the score for a candidate flow pair range from *packet counting* [52], to computing the *Spearman's rank correlation coefficient* [58] and *Pearson's correlation coefficient* [53] over samples of the two flows' time series or packet counts, to nowadays *deep learning (DL)* mechanisms such as *convolutional neural networks* [39, 41] and even *transformer models* [7]. Their historical grounding in measures of correlation (e.g., [6, 32, 53, 66]) explains why this score is often called the *correlation score* for a flow pair despite not necessarily computing statistical correlation anymore, as is the case with modern deep-learning-based mechanisms.

Given protection against flow correlation is a stated non-goal of Tor, we delay further discussion of this concept until Section 4.3.1. Instead, we briefly introduce two recent and state-of-the-art flow correlation attacks based on DL methods: DeepCorr [39] and DeepCoFFEA [41].

DeepCorr [39], published in 2018, was first to take advantage of the rapidly advancing capabilities of DL methods for flow correlation on Tor. Specifically, DeepCorr employs a convolutional neural network that takes *inter-packet delay (IPD)* vectors and *packet size* vectors of two candidate Tor flows (one on the guard side, one on the exit side) in both directions and computes their correlation score. In so doing, DeepCorr follows the traditional process for creating and assessing a capable machine learning model: (1) *tune* its *hyperparameters* (a set of variables defining essential aspects of the model such as the shape of filters, batch size, learning rate), (2) *train* the model on one split of the dataset, and (3) *test* the model's performance on a separate dataset split that it has not seen before. DeepCorr massively outperforms previous state-of-the-art flow correlation attacks, reinforcing the flow correlation threat to Tor users.

Improving upon DeepCorr, DeepCoFFEA [41] (published in 2022) also utilizes DL methods to first embed the two flows making up a candidate pair into a lower-

dimensional space, in which comparing embeddings can be done much more efficiently using distance metrics. Additionally, by splitting each flow into many windows, computing window-by-window correlation scores in the lower-dimensional space, and voting across all window-by-window scores for the final verdict on a pair of flows, DeepCoFFEA is able to heavily drive down the false positive rate. DeepCoFFEA handily outperforms the state-of-the-art (DeepCorr), while also demonstrating its effectiveness at defeating many countermeasures proposed in the literature (however, these were mostly designed against website fingerprinting, which we discuss in Section 3.3.2).

Tor has two options to counter flow correlation attacks such as DeepCorr and DeepCoFFEA: protect Tor flows by obfuscating more of their identifying characteristics (at the cost of higher latency or more wasted bandwidth or both) or make it less likely for any entity to obtain the network vantage position that allows them to conduct a flow correlation attack in the first place. While both DeepCorr and DeepCoFFEA showed that existing website fingerprinting countermeasures only mildly reduce the threat they pose to the privacy of Tor users, newer defenses such as DeTorrent [23] (through training a cover packet injector using *long short-term memory (LSTM)* in competition with such flow correlation attacks) are more successful and can offer meaningful protection. In the other direction, numerous proposals [1, 51, 57] have been published to alter Tor's path selection algorithm and reduce the likelihood that any entity gains the network visibility required to conduct flow correlation attacks on Tor users.

3.3.2 Website Fingerprinting

An attack very much within Tor's threat model is website fingerprinting. Here, the adversary runs at least one guard relay or observes at least one network link between a Tor user and their guard. A visualization of this setup is shown in Figure 3.3. Before launching the attack, the adversary collects a database worth of network metadata sequences, each characteristic for one in a set of monitored webpages. For this, the adversary visits each webpage multiple times through Tor using their own clients and learns the traffic fingerprint of each webpage. With such list of webpage fingerprints at hand, the adversary can check if any observed packet flow's network metadata matches one of the stored fingerprints. If it does, the adversary has (likely) identified the webpage the user is accessing through Tor. Naturally, website fingerprinting adversaries remain passive, as perturbing the signal would interfere with their ability to classify a user's visit correctly ($\mathbb{A}_{\mathrm{WF}}$ from Section 2.3.5).

One of the most salient differences between the two traffic analysis attacks (flow correlation and website fingerprinting) is the formulation of their classification problem: while flow correlation typically aims to answer the question whether two observed flows form the ingress and egress halves of a matched pair (a binary classification question), website fingerprinting aims to additionally assign the correct webpage label (a multi-class

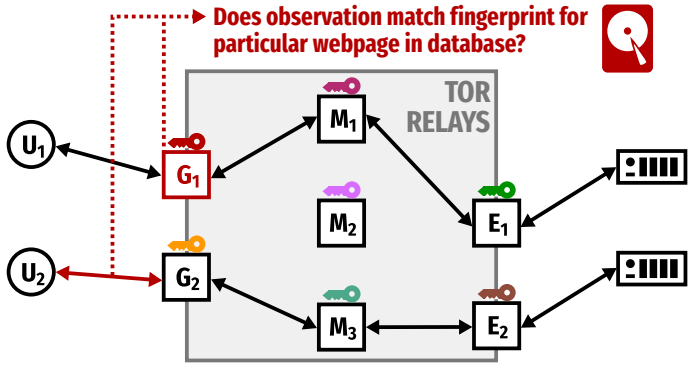


Figure 3.3: In a website fingerprinting attack, the adversary (marked in red) controls at least one guard (G_1) or observes at least one network link between a Tor user and their guard ($U_2 \Leftrightarrow G_2$). The adversary compares network metadata sequences for observed packet flows to network metadata sequences the adversary collected for a particular set of webpages in a database ahead of the attack. The insider adversary (guard) has the advantage of being able to distinguish circuits on the same network link, while the network-level adversary's view may be noisy due to circuit multiplexing.

classification problem). Investigations of this type of attack [36] predate even Tor [22, 56], however, research has ramped up [20, 21, 31, 34, 43, 44, 49, 54, 55, 63, 64] as Tor continued to rise in scale and popularity, not least due to web browsing being a core user activity that Tor aims to offer protection for.

While traditional machine learning mechanisms such as *support vector machines* [43, 44, 64], *k-nearest-neighbors* [63], and *random forests* [20] were successful in deanonymizing Tor users' webpage visits, recent studies [34, 49, 54, 55] have—like flow correlation attacks—adopted DL techniques for their superior performance. However, making inferences for real-world Tor usage from academic website fingerprinting attacks remains non-trivial. Questions regarding the impacts of most attacks' "lab setting" remain [28], e.g., *closed* versus *open world* evaluations (and their scale), the level and detectability of noise, and user behavior such as multi-tab browsing.

Website fingerprinting defenses have evolved—with varying degrees of success—together with the attacks they aim to protect against (see [33, Table 1] for a countermeasures comparison as of 2023). We highlight a few countermeasures below. WTF-PAD [29], based on adaptive padding by Shmatikov and Wang [53], was successful in crafting a low-impact (only a modest bandwidth increase, no delay) defense. This fact contributed to the Tor Project eventually basing their circuit padding scheme on WTF-

PAD.[15] The Walkie-Talkie defense by Wang and Goldberg [65] patches Tor Browser to communicate in *half-duplex* mode with web servers and shapes the resulting packet bursts to target shapes. FRONT by Gong and Wang [19] focuses cover packet injection on the beginnings of packet flows and is thus a defense with low bandwidth and no delay overhead, similar to WTF-PAD. Accepting low additional delay and moderate bandwidth overhead, RegulaTor by Holland and Hopper [24] represents a very capable defense which shapes the two directions of a flow into quickly decaying surges.

Taking the completely different approach of splitting connections across multiple circuits (i.e., across multiple guards) and thus possibly reducing the likelihood of adversarial observation (if the adversary is assumed to exclusively be an insider, at the guard position), De la Cadena et al. [12] report good website fingerprinting protection of their defense called TrafficSliver, at no delay or bandwidth overhead. However, Beckerle et al. [3] demonstrate that TrafficSliver (as well as other defenses that split traffic) performs much worse as a defense against their state-of-the-art DL-based attack, even on the dataset of the original publication. Finally, Pulls and Witwer [46] offer with Maybenot a general framework to evaluate website fingerprinting defenses.

3.3.3 Guard Discovery

To protect against attacks in which the adversary occupies the first and last hop in the users' circuits, Tor introduced guards. Users choose a small set of relays carrying the "Guard" flag and reuse these relays for the first hop on any circuit they build over an extended period of time (weeks). Except for bridges (which we ignore here for the sake of simplicity), guards are the only infrastructure nodes in Tor that know the true network location, i.e., IP address, of Tor users. As long as the guards a user selected are honest, the user is safe from many attacks that require the adversary to control the first hop [42, 67, 68, 69]. However, if the user samples adversarial relays to be part of their guards set or the adversary is able to compromise or coerce at least one of the user's guards, these attacks become possible and quickly lead to the deanonymization of the user.

Thus, guards clearly play an important role in safeguarding the anonymity of Tor users. And therefore, attacks successful at determining a Tor user's guard—*guard discovery attacks*—pose a serious threat to their privacy. Adversaries can launch guard discovery attacks to learn the relay in the first-hop position of a target circuit and thus uncover the guard of the Tor user who initiated this circuit. Subsequently, the adversary can focus their energy on compromising or coercing the identified guard in order to eventually fully deanonymize the Tor user.

[15] `https://web.archive.org/web/20250210230733/https://spec.torproject.org/padding-spec/circuit-level-padding.html`

Over the years, many vectors for discovering guards have been identified by researchers. Mittal et al. [35] measure the throughput of circuits to identify which relays are the circuits' bottlenecks. The most frequently observed bottleneck relay oftentimes is a user's guard. Biryukov et al. [4] monitor for a cell pattern on circuits between the target onion service and adversary-controlled relays to identify the onion service's guards.

Rochet and Pereira [50] exploit the flexibility of the Tor protocol (to remain forward compatible, Tor drops cells it does not understand) and public availability of relay bandwidth statistics to discover the guards of onion services. Through many separate circuits to the target onion service, the authors send a large number of cells that will get dropped by the onion service. If the onion service's bandwidth is lower than the bandwidth of its guard, a bug in the way Tor handles cell counters when recovering from an *"out of memory"* situation leads to a disparity in the reported incoming versus outgoing bandwidth of the onion service's guard in Tor's official bandwidth statistics. If the onion service's bandwidth is at least as large as the bandwidth of its guard, the above bug does not get triggered. However, sending cells that will silently get dropped to the onion service at peak rate of its guard for at least the duration of two bandwidth reporting windows leads the onion service's guard to report maximum bandwidth utilization in the official Tor bandwidth statistics. Either way, the adversary can use these abnormal bandwidth events as indicators that a relay acts as the targeted onion service's guard.

If a particular guard discovery attack relies on the adversary being next to the guard on a circuit in order to recognize a pattern transported through the circuit, the risk of this attack to Tor users can be lowered by extending the idea of guards also to later circuit positions. In the setting of protecting onion services and their users from guard discovery, the Tor Project has developed Vanguards[16] and Vanguards-Lite[17], which restrict the relay choices available to onion services and their users for the middle positions of the circuits they build. Vanguards extends the length of rendezvous circuits between Tor users and onion services from six to eight hops, by introducing an additional middle on either circuit on both sides of the rendezvous point. Vanguards restricts the relays eligible for the second-hop and (new) third-hop position on these circuits for different extended periods of time. In contrast, Vanguards-Lite does not introduce an additional hop into onion circuits but only restricts which relays can be chosen for the second-hop circuit position, similar to how guards restrict the relay choice for the first-hop circuit position. Vanguards is intended for long-lived onion services, while Vanguards-Lite targets onion service use cases with short lifetimes.

[16] `https://web.archive.org/web/20250304083918/https://spec.torproject.org/vanguards-spec/full-vanguards.html`

[17] `https://web.archive.org/web/20250319234942/https://spec.torproject.org/vanguards-spec/vanguards-lite.html`

Bibliography

[1] A. Barton and M. Wright. "DeNASA: Destination-Naive AS-Awareness in Anonymous Communications". In: *Proceedings of the 16th Privacy Enhancing Technologies Symposium (PETS 2016)*. Vol. 2016. 2016, pp. 356–372. DOI: 10.1515/popets-2016-0044 (p. 42).

[2] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker. "Low-Resource Routing Attacks Against Tor". In: *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society*. WPES '07. Alexandria, Virginia, USA: Association for Computing Machinery, 2007, pp. 11–20. ISBN: 9781595938831. DOI: 10.1145/1314333.1314336. URL: https://doi.org/10.1145/1314333.1314336 (pp. 39, 41).

[3] M. Beckerle, J. Magnusson, and T. Pulls. "Splitting Hairs and Network Traces: Improved Attacks Against Traffic Splitting as a Website Fingerprinting Defense". In: *Proceedings of the 21st Workshop on Privacy in the Electronic Society*. WPES '22. Los Angeles, CA, USA: Association for Computing Machinery, 2022, pp. 15–27. ISBN: 9781450398732. DOI: 10.1145/3559613.3563199. URL: https://doi.org/10.1145/3559613.3563199 (p. 44).

[4] A. Biryukov, I. Pustogarov, and R.-P. Weinmann. "Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization". In: *2013 IEEE Symposium on Security and Privacy (SP)*. 2013, pp. 80–94. DOI: 10.1109/SP.2013.15 (pp. 40, 45).

[5] Z. Brown. *Cebolla: Pragmatic IP Anonymity*. Tech. rep. 2002. 10 pp. (p. 36).

[6] S. Chakravarty, M. V. Barbera, G. Portokalidis, M. Polychronakis, and A. D. Keromytis. "On the Effectiveness of Traffic Analysis against Anonymity Networks Using Flow Records". In: *Passive and Active Measurement*. Ed. by M. Faloutsos and A. Kuzmanovic. Cham: Springer International Publishing, 2014, pp. 247–257. ISBN: 978-3-319-04918-2 (pp. 40, 41).

[7] T. Chawla, S. Mittal, N. Mathews, and M. Wright. "ESPRESSO: Advanced End-to-End Flow Correlation Attacks on Tor". In: *Proceedings of the 8th Asia-Pacific Workshop on Networking*. APNet '24. Sydney, Australia: Association for Computing Machinery, 2024, pp. 219–220. ISBN: 9798400717581. DOI: 10.1145/3663408.3665824. URL: https://doi.org/10.1145/3663408.3665824 (p. 41).

[8] B. Collier. *Tor: From the Dark Web to the Future of Privacy*. The MIT Press, Apr. 2024. ISBN: 9780262378925. DOI: 10.7551/mitpress/14907.001.0001. eprint: https://direct.mit.edu/book-pdf/2362316/book_9780262378925.pdf. URL: https://doi.org/10.7551/mitpress/14907.001.0001 (p. 33).

[9] G. Danezis and R. Clayton. "Route Fingerprinting in Anonymous Communications". In: *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06)*. 2006, pp. 69–72. DOI: 10.1109/P2P.2006.33 (p. 36).

[10] G. Danezis and P. Syverson. "Bridging and Fingerprinting: Epistemic Attacks on Route Selection". In: *Privacy Enhancing Technologies*. Ed. by N. Borisov and I. Goldberg. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 151–166. ISBN: 978-3-540-70630-4 (p. 36).

[11] D. Das, S. Meiser, E. Mohammadi, and A. Kate. "Anonymity Trilemma: Strong Anonymity, Low Bandwidth Overhead, Low Latency—Choose Two". In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 108–126. DOI: 10.1109/SP.2018.00011 (p. 40).

[12] W. De la Cadena, A. Mitseva, J. Hiller, J. Pennekamp, S. Reuter, J. Filter, T. Engel, K. Wehrle, and A. Panchenko. "TrafficSliver: Fighting Website Fingerprinting Attacks with Traffic Splitting". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS '20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1971–1985. ISBN: 9781450370899. DOI: 10.1145/3372297.3423351. URL: https://doi.org/10.1145/3372297.3423351 (p. 44).

[13] W. Diffie and M. Hellman. "New Directions in Cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638 (p. 36).

[14] R. Dingledine, N. Mathewson, and P. Syverson. "Tor: The Second-Generation Onion Router". In: *13th USENIX Security Symposium (USENIX Security '04)*. USENIX Association, 2004 (pp. 33, 35, 36, 38, 39, 40).

[15] M. Edman and B. Yener. "On Anonymity in an Electronic Society: A Survey of Anonymous Communication Systems". In: *ACM Comput. Surv.* 42.1 (Dec. 2009). ISSN: 0360-0300. DOI: 10.1145/1592451.1592456. URL: https://doi.org/10.1145/1592451.1592456 (pp. 33, 36).

[16] M. J. Freedman and R. Morris. "Tarzan: A Peer-to-Peer Anonymizing Network Layer". In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. CCS '02. Washington, DC, USA: Association for Computing Machinery, 2002, pp. 193–206. ISBN: 1581136129. DOI: 10.1145/586110.586137. URL: https://doi.org/10.1145/586110.586137 (p. 35).

[17] I. Goldberg and A. Shostack. *Freedom Network 1.0 Architecture and Protocols*. White Paper. Timestamp on report is 2001-10-10, however, the report's change history indicates changes were last made in 1999-11. Zero-Knowledge Systems, Nov. 1999. 23 pp. URL: https://web.archive.org/web/20211015202339/https://shostack.org/files/zks/zeroknowledgewhitepapers/arch-tech.pdf (p. 35).

[18] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. "Hiding Routing Information". In: *Information Hiding*. Ed. by R. Anderson. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 137–150. ISBN: 978-3-540-49589-5 (p. 33).

[19] J. Gong and T. Wang. "Zero-delay Lightweight Defenses against Website Fingerprinting". In: *29th USENIX Security Symposium (USENIX Security '20)*. USENIX Association, Aug. 2020, pp. 717–734. ISBN: 978-1-939133-17-5. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/gong (p. 44).

[20] J. Hayes and G. Danezis. "k-fingerprinting: A Robust Scalable Website Fingerprinting Technique". In: *25th USENIX Security Symposium (USENIX Security '16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 1187–1203. ISBN: 978-1-931971-32-4. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/hayes (p. 43).

[21] D. Herrmann, R. Wendolsky, and H. Federrath. "Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier". In: *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*. CCSW '09. Chicago, Illinois, USA: Association for Computing Machinery, 2009, pp. 31–42. ISBN: 9781605587844. DOI: 10.1145/1655008.1655013. URL: https://doi.org/10.1145/1655008.1655013 (p. 43).

[22] A. Hintz. "Fingerprinting Websites Using Traffic Analysis". In: *Privacy Enhancing Technologies*. Ed. by R. Dingledine and P. Syverson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 171–178. ISBN: 978-3-540-36467-2 (p. 43).

[23] J. K. Holland, J. Carpenter, S. E. Oh, and N. Hopper. "DeTorrent: An Adversarial Padding-only Traffic Analysis Defense". In: *Proceedings of the 24th Privacy Enhancing Technologies Symposium (PETS 2024)*. Vol. 2024. 2024, pp. 98–115. DOI: 10.56553/popets-2024-0007 (p. 42).

[24] J. K. Holland and N. Hopper. "RegulaTor: A Straightforward Website Fingerprinting Defense". In: *Proceedings of the 22nd Privacy Enhancing Technologies Symposium (PETS 2022)*. Vol. 2022. 2022, pp. 344–362. DOI: 10.2478/popets-2022-0049 (p. 44).

[25] A. Houmansadr and N. Borisov. "SWIRL: A Scalable Watermark to Detect Correlated Network Flows". In: *18th Annual Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, February 6–9, 2011*. The Internet Society, 2011. URL: https://www.ndss-symposium.org/wp-content/uploads/2017/09/houm.pdf (p. 41).

[26] R. Jansen, F. Tschorsch, A. Johnson, and B. Scheuermann. "The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network". In: *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23–26, 2014*. The Internet Society, 2014. URL: https://www.ndss-symposium.org/wp-content/uploads/2017/09/05_4_0.pdf (p. 40).

[27] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson. "Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries". In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. CCS '13. Berlin, Germany: Association for Computing Machinery, 2013, pp. 337–348. ISBN: 9781450324779. DOI: 10.1145/2508859.2516651. URL: https://doi.org/10.1145/2508859.2516651 (pp. 40, 41).

[28] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt. "A Critical Evaluation of Website Fingerprinting Attacks". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS '14. Scottsdale, Arizona, USA: Association for Computing Machinery, 2014, pp. 263–274. ISBN: 9781450329576. DOI: 10.1145/2660267.2660368. URL: https://doi.org/10.1145/2660267.2660368 (p. 43).

[29] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright. "Toward an Efficient Website Fingerprinting Defense". In: *Computer Security – ESORICS 2016*. Ed. by I. Askoxylakis, S. Ioannidis, S. Katsikas, and C. Meadows. Cham: Springer International Publishing, 2016, pp. 27–46. ISBN: 978-3-319-45744-4 (pp. 40, 43).

[30] C. H. Komlo, N. Mathewson, and I. Goldberg. "Walking Onions: Scaling Anonymity Networks while Protecting Users". In: *29th USENIX Security Symposium (USENIX Security '20)*. USENIX Association, Aug. 2020, pp. 1003–1020. ISBN: 978-1-939133-17-5. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/komlo (p. 36).

[31] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas. "Circuit Fingerprinting Attacks: Passive Deanonymization of Tor Hidden Services". In: *24th USENIX Security Symposium (USENIX Security '15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 287–302. ISBN: 978-1-939133-11-3. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/kwon (p. 43).

[32] B. N. Levine, M. K. Reiter, C. Wang, and M. Wright. "Timing Attacks in Low-Latency Mix Systems". In: *Financial Cryptography*. Ed. by A. Juels. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 251–265. ISBN: 978-3-540-27809-2 (pp. 39, 41).

[33] N. Mathews, J. K. Holland, S. E. Oh, M. S. Rahman, N. Hopper, and M. Wright. "SoK: A Critical Evaluation of Efficient Website Fingerprinting Defenses". In: *2023 IEEE Symposium on Security and Privacy (SP)*. 2023, pp. 969–986. DOI: 10.1109/SP46215.2023.10179289 (p. 43).

[34] N. Mathews, J. K. Holland, N. Hopper, and M. Wright. "LASERBEAK: Evolving Website Fingerprinting Attacks With Attention and Multi-Channel Feature Representation". In: *IEEE Transactions on Information Forensics and Security* 19 (2024), pp. 9285–9300. DOI: 10.1109/TIFS.2024.3468171 (p. 43).

[35] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov. "Stealthy Traffic Analysis of Low-Latency Anonymous Communication Using Throughput Fingerprinting". In: *Proceedings of the 18th ACM Conference on Computer and Communications Security*. CCS '11. Chicago, Illinois, USA: Association for Computing Machinery, 2011, pp. 215–226. ISBN: 9781450309486. DOI: 10.1145/2046707.2046732. URL: https://doi.org/10.1145/2046707.2046732 (p. 45).

[36] M. A. I. Mohd Aminuddin, Z. F. Zaaba, A. Samsudin, F. Zaki, and N. B. Anuar. "The rise of website fingerprinting on Tor: Analysis on techniques and assumptions". In: *Journal of Network and Computer Applications* 212 (2023), p. 103582. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2023.103582. URL: https://www.sciencedirect.com/science/article/pii/S1084804523000012 (p. 43).

[37] S. Murdoch and G. Danezis. "Low-Cost Traffic Analysis of Tor". In: *2005 IEEE Symposium on Security and Privacy (SP)*. 2005, pp. 183–195. DOI: 10.1109/SP.2005.12 (pp. 39, 41).

[38] S. J. Murdoch and P. Zieliński. "Sampled Traffic Analysis by Internet-Exchange-Level Adversaries". In: *Privacy Enhancing Technologies*. Ed. by N. Borisov and P. Golle. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 167–183. ISBN: 978-3-540-75551-7 (p. 41).

[39] M. Nasr, A. Bahramali, and A. Houmansadr. "DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. Toronto, Canada: Association for Computing Machinery, 2018, pp. 1962–1976. ISBN: 9781450356930. DOI: 10.1145/3243734.3243824. URL: https://doi.org/10.1145/3243734.3243824 (p. 41).

[40] M. Nasr, A. Houmansadr, and A. Mazumdar. "Compressive Traffic Analysis: A New Paradigm for Scalable Traffic Analysis". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 2053–2069. ISBN: 9781450349468. DOI: 10.1145/3133956.3134074. URL: https://doi.org/10.1145/3133956.3134074 (p. 41).

[41] S. E. Oh, T. Yang, N. Mathews, J. K. Holland, M. S. Rahman, N. Hopper, and M. Wright. "DeepCoFFEA: Improved Flow Correlation Attacks on Tor via Metric Learning and Amplification". In: *2022 IEEE Symposium on Security and Privacy (SP)*. 2022, pp. 1915–1932. DOI: 10.1109/SP46214.2022.9833801 (p. 41).

[42] L. Overlier and P. Syverson. "Locating Hidden Servers". In: *2006 IEEE Symposium on Security and Privacy (SP)*. 2006. DOI: 10.1109/SP.2006.24 (pp. 37, 39, 40, 41, 44).

[43] A. Panchenko, F. Lanze, A. Zinnen, M. Henze, J. Pennekamp, K. Wehrle, and T. Engel. "Website Fingerprinting at Internet Scale". In: *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21–24, 2016*. The Internet Society, 2016. URL: https://www.ndss-symposium.org/wp-content/uploads/2017/09/website-fingerprinting-internet-scale.pdf (p. 43).

[44] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. "Website Fingerprinting in Onion Routing Based Anonymization Networks". In: *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*. WPES '11. Chicago, Illinois, USA: Association for Computing Machinery, 2011, pp. 103–114. ISBN: 9781450310024. DOI: 10.1145/2046556.2046570. URL: https://doi.org/10.1145/2046556.2046570 (p. 43).

[45] E. Papadogiannaki and S. Ioannidis. "A Survey on Encrypted Network Traffic Analysis Applications, Techniques, and Countermeasures". In: *ACM Comput. Surv.* 54.6 (July 2021). ISSN: 0360-0300. DOI: 10.1145/3457904. URL: https://doi.org/10.1145/3457904 (p. 41).

[46] T. Pulls and E. Witwer. "Maybenot: A Framework for Traffic Analysis Defenses". In: *Proceedings of the 22nd Workshop on Privacy in the Electronic Society*. WPES '23. Copenhagen, Denmark: Association for Computing Machinery, 2023, pp. 75–89. ISBN: 9798400702358. DOI: 10.1145/3603216.3624953. URL: https://doi.org/10.1145/3603216.3624953 (p. 44).

[47] M. Reed, P. Syverson, and D. Goldschlag. "Anonymous Connections and Onion Routing". In: *IEEE Journal on Selected Areas in Communications* 16.4 (1998), pp. 482–494. DOI: 10.1109/49.668972 (pp. 33, 34, 39).

[48] M. Rennhard and B. Plattner. "Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection". In: *Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society*. WPES '02. Washington, DC: Association for Computing Machinery, 2002, pp. 91–102. ISBN: 1581136331. DOI: 10.1145/644527.644537. URL: https://doi.org/10.1145/644527.644537 (p. 35).

[49] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen. "Automated Website Fingerprinting through Deep Learning". In: *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18–21, 2018*. The Internet Society, 2018. URL: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_03A-1_Rimmer_paper.pdf (p. 43).

[50] F. Rochet and O. Pereira. "Dropping on the Edge: Flexibility and Traffic Confirmation in Onion Routing Protocols". In: *Proceedings of the 18th Privacy Enhancing Technologies Symposium (PETS 2018)*. Vol. 2018. 2018, pp. 27–46. DOI: 10.1515/popets-2018-0011 (pp. 41, 45).

[51] F. Rochet and O. Pereira. "Waterfilling: Balancing the Tor network with maximum diversity". In: *Proceedings of the 17th Privacy Enhancing Technologies Symposium (PETS 2017)*. Vol. 2017. 2017, pp. 4–22. DOI: 10.1515/popets-2017-0013 (p. 42).

[52] A. Serjantov and P. Sewell. "Passive Attack Analysis for Connection-Based Anonymity Systems". In: *Computer Security – ESORICS 2003*. Ed. by E. Snekkenes and D. Gollmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 116–131. ISBN: 978-3-540-39650-5 (pp. 39, 41).

[53] V. Shmatikov and M.-H. Wang. "Timing Analysis in Low-Latency Mix Networks: Attacks and Defenses". In: *Computer Security – ESORICS 2006*. Ed. by D. Gollmann, J. Meier, and A. Sabelfeld. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 18–33. ISBN: 978-3-540-44605-7 (pp. 39, 40, 41, 43).

[54] P. Sirinam, M. Imani, M. Juarez, and M. Wright. "Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. Toronto, Canada: Association for Computing Machinery, 2018, pp. 1928–1943. ISBN: 9781450356930. DOI: 10.1145/3243734.3243768. URL: https://doi.org/10.1145/3243734.3243768 (p. 43).

[55] P. Sirinam, N. Mathews, M. S. Rahman, and M. Wright. "Triplet Fingerprinting: More Practical and Portable Website Fingerprinting with N-shot Learning". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom: Association for Computing Machinery, 2019, pp. 1131–1148. ISBN: 9781450367479. DOI: 10.1145/3319535.3354217. URL: https://doi.org/10.1145/3319535.3354217 (p. 43).

[56] Q. Sun, D. Simon, Y.-M. Wang, W. Russell, V. Padmanabhan, and L. Qiu. "Statistical Identification of Encrypted Web Browsing Traffic". In: *Proceedings 2002 IEEE Symposium on Security and Privacy (SP)*. 2002, pp. 19–30. DOI: 10.1109/SECPRI.2002.1004359 (p. 43).

[57] Y. Sun, A. Edmundson, N. Feamster, M. Chiang, and P. Mittal. "Counter-RAPTOR: Safeguarding Tor Against Active Routing Attacks". In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, pp. 977–992. DOI: 10.1109/SP.2017.34 (p. 42).

[58] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal. "RAPTOR: Routing Attacks on Privacy in Tor". In: *24th USENIX Security Symposium (USENIX Security '15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 271–286. ISBN: 978-1-939133-11-3. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/sun (p. 41).

[59] P. Syverson, D. Goldschlag, and M. Reed. "Anonymous Connections and Onion Routing". In: *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No.97CB36097)*. 1997, pp. 44–54. DOI: 10.1109/SECPRI.1997.601314 (pp. 33, 34).

[60] P. Syverson. "A Peel of Onion". In: *Proceedings of the 27th Annual Computer Security Applications Conference*. ACSAC '11. Orlando, Florida, USA: Association for Computing Machinery, 2011, pp. 123–137. ISBN: 9781450306720. DOI: 10.1145/2076732.2076750. URL: https://doi.org/10.1145/2076732.2076750 (pp. 33, 36).

[61] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr. "Towards an Analysis of Onion Routing Security". In: *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability Berkeley, CA, USA, July 25–26, 2000 Proceedings*. Ed. by H. Federrath. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 96–114. ISBN: 978-3-540-44702-3. DOI: 10.1007/3-540-44702-4_6. URL: https://doi.org/10.1007/3-540-44702-4_6 (p. 39).

[62] Q. Wang, P. Mittal, and N. Borisov. "In Search of an Anonymous and Secure Lookup: Attacks on Structured Peer-to-Peer Anonymous Communication Systems". In: *Proceedings of the 17th ACM Conference on Computer and Communications Security*. CCS '10. Chicago, Illinois, USA: Association for Computing Machinery, 2010, pp. 308–318. ISBN: 9781450302456. DOI: 10.1145/1866307.1866343. URL: https://doi.org/10.1145/1866307.1866343 (p. 36).

[63] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg. "Effective Attacks and Provable Defenses for Website Fingerprinting". In: *23rd USENIX Security Symposium (USENIX Security '14)*. San Diego, CA: USENIX Association, Aug. 2014, pp. 143–157. ISBN: 978-1-931971-15-7. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/wang_tao (p. 43).

[64] T. Wang and I. Goldberg. "Improved Website Fingerprinting on Tor". In: *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*. WPES '13. Berlin, Germany: Association for Computing Machinery, 2013, pp. 201–212. ISBN: 9781450324854. DOI: 10.1145/2517840.2517851. URL: https://doi.org/10.1145/2517840.2517851 (p. 43).

[65] T. Wang and I. Goldberg. "Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks". In: *26th USENIX Security Symposium (USENIX Security '17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1375–1390. ISBN: 978-1-931971-40-9. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/wang-tao (p. 44).

[66] X. Wang, D. S. Reeves, and S. F. Wu. "Inter-Packet Delay Based Correlation for Tracing Encrypted Connections through Stepping Stones". In: *Computer Security — ESORICS 2002*. Ed. by D. Gollmann, G. Karjoth, and M. Waidner. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 244–263. ISBN: 978-3-540-45853-1 (p. 41).

[67] M. Wright, M. Adler, B. Levine, and C. Shields. "Defending Anonymous Communications Against Passive Logging Attacks". In: *2003 IEEE Symposium on Security and Privacy (SP)*. 2003, pp. 28–41. DOI: 10.1109/SECPRI.2003.1199325 (pp. 37, 40, 44).

[68] M. K. Wright, M. Adler, B. N. Levine, and C. Shields. "An Analysis of the Degradation of Anonymous Protocols". In: *9th Annual Network and Distributed System Security Symposium, NDSS 2002, San Diego, California, USA, February 6–8, 2002*. The Internet Society, 2002. URL: https://www.ndss-symposium.org/wp-content/uploads/2017/09/An-Analysis-of-the-degradation-of-anonymous-protocols.pdf (pp. 37, 40, 44).

[69] M. K. Wright, M. Adler, B. N. Levine, and C. Shields. "The Predecessor Attack: An Analysis of a Threat to Anonymous Communications Systems". In: *ACM Trans. Inf. Syst. Secur.* 7.4 (Nov. 2004), pp. 489–522. ISSN: 1094-9224. DOI: 10.1145/1042031.1042032. URL: https://doi.org/10.1145/1042031.1042032 (pp. 37, 40, 44).

Chapter 4

Mixnets

Mixnets (mix networks) use public-key cryptography for every packet, which—in contrast to connection-oriented onion routing networks—prevents intermediaries from knowing which packets belong to the same application-level flow. Mixnets are thus *packet-oriented*. Additionally, mixnets alter the packets' *timing patterns* by delaying them at intermediaries. While mixnets may rely on onion routing as a building block, they thus introduce (configurable but potentially considerable amounts of) *artificial delay* into data transmissions. The mixnet design space is vast and allows for very different assemblings of components into final systems. In this thesis, we focus on *continuous-time decryption mixnets* and their current state-of-the-art implementation Nym [14][1].

4.1 High-Latency, Packet-Oriented: Mixnets

In 1981, Chaum introduced mixnets [3] as a way to send anonymous email and thus launched the academic field of anonymous communication. In a mixnet, packets pass from senders to recipients through a series of intermediaries called *mixes*. An adversary observing a set of packets arriving at a mix and leaving the mix is unable to tell which packet arriving is which packet leaving based on the packets' bit string representations, timestamps, and sizes. Mixnets achieve this high level of resistance to attacks due to three characteristics. First, similar to onion routing networks, packets are *transformed cryptographically* at each mix. In decryption mixnets, each mix decrypts its layer of each packet like the forward direction in onion routing networks. Second, each mix *reorders* the packets within a set of concurrent packets. This steps adds *artificial delays* to packet transmissions that prevent linking packets based on their timestamps. Packets can either

[1] `https://web.archive.org/web/20250406004702/https://nym.com/`

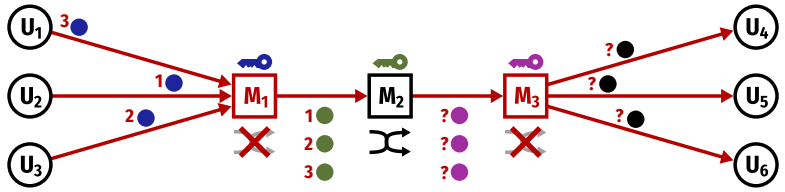


Figure 4.1: Worst-case scenario in terms of adversarial visibility (marked in red) into a Chaumian decryption mixnet with three mixes (which, for the purposes of this simple example, flush once three packets were received). Despite all but one mix being adversarial, the single honest mix (M_2) is enough to prevent the adversary from linking arriving to leaving packets at M_2 based on link-level representation or time.

be delayed as part of a *batch* or independently. Third, distinguishability of packets based on their size is prevented by ensuring that all packets are of *known equal size*. Through these three measures, mixnets have the potential to resist even very capable traffic analysis adversaries such as the global passive adversary ($\mathbb{A}_{\mathrm{GPA}}$ from Section 2.3.5).

Critically, mixnets are *packet-oriented*, in contrast to the connection-oriented nature of onion routing networks. The notion of a connection (circuit) does not exist at the mix level. Each packet traversing a mixnet thus requires the use of fresh cryptographic key material, dedicated to this particular packet only and never to be used again. Given that each processing of a packet by a mix involves at minimum one public-key operation, the average computational load that mixnets require of their mixes is typically higher than that of onion routing networks towards their routers.

Clearly, if the single mix of a proxy-like, single-mix mixnet is trustworthy (i.e., not controlled by and not colluding with an adversary), such single-mix mixnet is sufficient to achieve the mixnet concept's anonymity guarantees. As long as the single mix executes the above steps truthfully and keeps the packet batch permutations secret, packets flowing through this mix cannot be linked based on time or bit string representation.

However, the original mixnet paper [3] allows for insider adversaries, i.e., adversaries controlling participants of the overlay network, and it is indeed a common assumption that mixnet adversaries are participants themselves. If the adversary controlled the single mix mentioned above, they would know which packet arriving is which packet leaving and users would thus have no anonymity. To counteract this, Chaum proposes to lower the risk that this single point of trust poses by chaining multiple mixes in sequence, called a *cascade*. Packets flow from senders to the cascade's first mix, followed by the cascade's second mix, and so on, until they reach the last cascade mix and finally their intended recipients. Now, even if all but one mix in the cascade are adversarial and collude, the mixnet's unlinking properties are realized by virtue of this single honest

mix. This illustrates the trust model underlying the mixnet's anonymity guarantees: mixnet users need to trust that at least one mix on the paths chosen for their packets is honest—the so called *anytrust* property. We visualize this scenario in Figure 4.1.

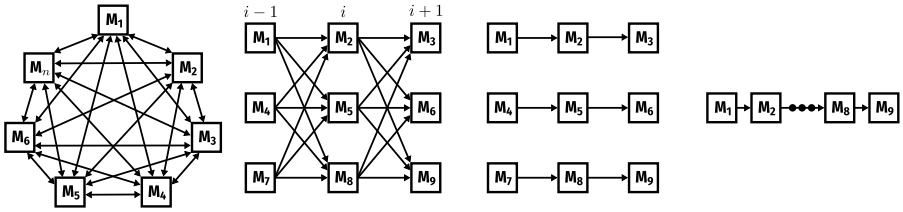
Many different options for each dimension of the mixnet design space (verifiability, network topology, flushing strategy, traffic shaping behavior, cover traffic, cryptographic packet format, active attack resistance) have been proposed over the decades since Chaum's initial paper. Concrete choices for each dimension of a realized mixnet have far-reaching implications in terms of the mixnet's usability for target network applications, level of anonymity provided, capability to scale, and more. We first cover the most important design decisions before introducing and situating state-of-the-art mixnet Nym in this mixnet design space.

4.1.1 Mixnet Verifiability

Verifiable mixnets [19, 20] require—in contrast to the mixnets used for online communication purposes considered thus far—that each mix can be held accountable for its processing of packets. Verifiable mixnets can be realized either as decryption mixnets (described above) or as re-encryption mixnets, in which senders submit packets encrypted to a key shared by a number of trustees to the mixes which re-encrypt the packets as they process them. Either variant for realizing a verifiable mixnet requires senders and mixes to publish their output packets to a public *bulletin board*. An authenticated channel is assumed to exist between each participant and the bulletin board. Each step of the mixnet protocol can then be checked to be correct with the help of the data stored on the bulletin board. Depending on the concrete approach for how verifiability is achieved, this checking process may involve computationally expensive proof techniques (e.g., zero-knowledge proofs). Packet delivery is significantly delayed so that each step of the mix protocol can be verified and misbehavers can be held accountable.

It is clear from the above that verifiable mixnets come with strong assumptions about the type of application that can make use of them sensibly. When packet frequency is low, delivery time of packets to their final purpose is irrelevant compared to the application objective, and correctness of the mix operation is required, verifiable mixnets offer a way for senders to anonymously submit packets to the application's purpose. For example, how long it takes for an electronic voting system to tally the voters' ballots and announce the final result is much less of a concern (e.g., fine if it takes hours) compared to the integrity needs of such system. Electronic voting is thus the primary application of verifiable mixnets, e.g., the Open Verificatum project[2] or the Helios Voting system.[3] The networked services we consider in this thesis—everyday online activities such as web

[2] `https://web.archive.org/web/20250118003712/https://www.verificatum.org/`
[3] `https://web.archive.org/web/20250410100959/https://vote.heliosvoting.org/`



(a) Free-route (b) Stratified (c) Multi-cascade (d) Cascade

Figure 4.2: Mixnet topologies from least restrictive (left) to most restrictive (right).

browsing, email, and financial transactions—are, however, not a good fit for verifiable mixnets and we thus do not consider verifiable mixnets further.

4.1.2 Mixnet Topologies

The choice for how to arrange the servers of an AN into the AN's overlay network topology has been a topic of academic debate [2, 37]. Any topology exists somewhere on the spectrum between "fewest restrictions on packet routes" and "most restrictions on packet routes". We illustrate key topologies on this spectrum in Figure 4.2.

Free-route ANs impose the fewest restrictions on network paths available for users to choose from. Every overlay network node is reachable by every other node in a single logical network hop, which is why Figure 4.2a is the only one with double arrows connecting network members. Free-route ANs may dictate the path length (i.e., number of hops on the overlay network) but typically do not. Instead, they either specify a default or a maximum path length, or both. Note that a free-route topology in a peer-to-peer AN means that every AN participant can reach every other participant in a single hop, which may present the system with serious network state distribution and discoverability problems. However, in an AN that does distinguish roles (e.g., users and servers), the free-route topology only exists between the servers. Up to the point of exhausting a node's resources (e.g., unused ports, memory, bandwidth) from open connections, free-route topologies can scale well with increasing demand on bandwidth, simply by adding more topology members. They offer good resistance to link failures (and thus, DoS attacks [37]), as ample alternatives exist, contributing to the AN's availability (see Section 2.5). However, the potentially very low amount of traffic per link in free-route networks (worst in peer-to-peer) may aid an adversary in deanonymizing users [36].

Packets in a *stratified* topology [15, 17] flow from left to right. Each mix in layer i can receive packets from any mix in layer $i - 1$ and can forward packets to any mix in layer $i + 1$. By having mixes verify that they receive packets only from the

correct previous nodes, the length of paths is thus fixed to the number of layers in the stratified topology. Topologies are public knowledge and it is therefore not considered an information leak that stratified topology members know which hop along packet paths they service. Compared to free-route topologies, stratified topologies impose significant route restrictions: much fewer routes are available for the same number of topology members. While this reduces the AN's tolerance to link failures, no single point of failure exists that would block the sending of packets through the AN and the path restrictions lead to traffic concentrating on many fewer links than in free-route networks. Additionally, scaling the network with user demand is straightforward: if demand goes up, nodes can be added to each layer to absorb the additional bandwidth usage and can be removed again once demand wanes.

In a mixnet with mixes arranged into *multiple cascades*, users select one of them to follow for each packet they send. Typically, mixes do not participate in more than one cascade at a time. Once a packet has entered one of the cascades its path is set and will not cross with packets sent to other cascades, leading anonymity sets to partition according to the selected cascade. While more cascades can be added to serve increased demand, anonymity sets will not grow with them and configuration needs to account for the slowest mix per cascade, as all packets going through the cascade have to pass through that node [6]. Finally, all mixnet packets going through a *single cascade* (one of the proposals in Chaum's initial work [3]) leads to them sharing the same anonymity set. This set can potentially be maximal and also very large in absolute terms—if the cascade is composed of very capable (i.e., powerful) mixes. However, this topology does not scale well with growing demand and needs to be prepared for the situation in which a single failing link or failing or misbehaving mix halts operation of the entire system.

4.1.3 Mix Flushing Strategies

The heart of a mixnet is the *flushing strategy* according to which its mixes alter the flow of packets. The choice of this strategy has significant impact on a mixnet's latency (and thus, user-perceived performance), resistance to attacks, and number and kind of applications that realistically can make use of the mixnet. However, before choosing the flushing strategy, mixnet designers need to decide whether the mixnet is supposed to operate in a *synchronous* or *asynchronous* fashion.

Synchronous mixnets assume a global clock dividing time into discrete steps (can be large or small) that all participants are aware of. Oftentimes, a single time step is called a *round*, so it is common to reason about, for example, the actions of a particular group of participants in round i. Asynchronous mixnets, on the other hand, make no assumptions about the existence of a global clock synchronizing all mixnet participants. Time is treated as continuous and mixnet participants can take actions at any time.

Next, the mixes' flushing strategy can either work over *batches* of packets or each packet independently, in *continuous time*. The three primary flushing strategies for *batching* mixnets are *threshold*, *timed*, and *pool* mixes [16, 35]. Any of these strategies can be deployed exclusively or as part of a hybrid strategy combining two or more of them. A threshold mix always waits for a target number of packets to arrive before assembling them into a batch and forwarding them. Thus, anonymity sets can be of constant size (up to the value of the threshold, if each packet comes from a different sender), but latency is not bounded (as batches can take vastly different amounts of time to fill up).

Timed mixes instead flush all currently assembled packets after a target duration has elapsed, e.g., every second. Anonymity sets can be minimal (size one) but mixing latency is predictable (if we ignore failures). Pool mixes add a packet buffer to either threshold or timed mixes, which retains a configurable number of packets across flushings. This introduces uncertainty for outside observers, as not all packets that went into a threshold pool mix or timed pool mix since the last flushing are going to exit as part of the next. Packets to retain in the pool are chosen uniformly at random from all packets currently in the mix (pooled packets and arrivals since the previous flushing). Chaum's original mixnet proposal [3] defined threshold mixes, while the Mixminion mixnet by Danezis et al. [5] implemented a hybrid of threshold pool and timed pool mixes.

In contrast to batching mixnets, continuous-time mixnets make the flushing decision on individual packets themselves. That means that each packet flowing through the mixnet is processed by each mix in isolation of every other (potentially concurrent) packet. Mixing occurs due to each (honest) mix holding back a packet for an amount of time determined by the packet's sender and encapsulated in the packet's appropriate encrypted layer. Continuous-time mixes were proposed as "Stop-and-Go" mixes by Kesdogan et al. [22], in which packets are delayed independently with delay drawn from an exponential distribution. Due to the *memorylessness property* of the exponential distribution [13], two packets in the buffer of an exponential mix are equally likely to be emitted next—independent of their arrival time [22]. If the arrival times of packets at an exponential mix follow a Poisson process, the exponential mix was shown to be the optimal mixing strategy (in the sense of maximizing entropy) for a given latency [8]. Loopix [33] is a continuous-time mixnet built using exponential mixes and so is Nym [14], as it in turn is inspired by Loopix. Das et al. [12] prove bounds for some anonymity properties afforded by continuous-time mixnets based on exponential mixes.

4.1.4 Traffic Shaping and Cover Packets

In the context of ANs, traffic shaping concerns the adherence to a target packet sending schedule of some or all of the network's participants. For the largest possible sender anonymity set, all AN users would need to send the same number of packets at the exact same times. As this is impossible in any real-world networked system with actual

users (e.g., because users choose when to participate and when not to), one way to approximate this ideal is to instruct users to shape their outgoing traffic according to some common schedule. For example, users might be compelled to send one packet per round in synchronous mixnets. Asynchronous mixnets may prescribe packet emission schedules that work well with their respective flushing strategy. For example, users of a continuous-time mixnet may be instructed to shape their packet emissions following a Poisson process. This way, the network activity of mixnet users can be made to look more like a target activity profile and they become less distinguishable.

One part of traffic shaping is thus the delaying of packets at users, when the input rate from the application level exceeds the stipulated mixnet-level packet rate. If more than one application-level packet is available for sending at the sending time determined by a user's egress scheduler, the excess packets will need to be held in a local queue until one send slot was available for each of them (which will happen with time). The other half of traffic shaping is the injection of *cover packets* whenever a send slot arrives and no application-level packet is ready to be sent. Cover packets are packets which do not carry application-level data as payload but random data to be discarded upon receipt. Cover packets can thus be used to fill empty slots in sending schedules, as, for example, in Chaum's initial mixnet design [3], where users were expected to send with the constant bandwidth of one packet per round. Using cover packets to fill gaps and thus send with (average) constant bandwidth, is also the only way for mixnets to achieve unobservability [32], i.e., the inability of an entity that is neither sender nor recipient to determine whether any application-level data was communicated at all (see Section 2.2).

In addition to their role in traffic shaping, cover packets can also be utilized towards thwarting active attacks. By sending looping *heartbeat* packets (going from themselves through the mixnet back to themselves), users and mixes have a way to determine whether they are under active attack (in which packets are dropped) and, if that is indeed the case, can take counteraction by ramping up the rate of injected cover packets [10]. The mixnet Loopix [33] derives its name from employing such looping cover packets from users and mixes to monitor for active attacks and contribute to anonymity sets.

4.2 Nym

By drawing inspiration from Loopix [33] and combining a number of best-in-class mixnet building blocks with a cryptocurrency-based incentivization scheme to form its general-purpose network privacy layer, Nym [14][4] represents the current state of the art for continuous-time mixnets. Nym's source code is public[5] and its deployed mainnet is available for packet mixing since 2022. Note that Nym is a system under

[4] `https://web.archive.org/web/20250406004702/https://nym.com/`
[5] `https://web.archive.org/web/20250425231337/https://github.com/nymtech/nym`

active development and is thus changing over time. The state of Nym discussed in the following reflects the system at the time when we conducted the two studies this thesis contributes to the literature (Chapter 9 and Chapter 10, respectively).

Like Loopix, Nym is a continuous-time mixnet built on exponential mixes arranged in a stratified topology for its attractive trade-off between number of paths and concentration of traffic on each link (see Section 4.1.2). In order to participate in an infrastructure role in Nym, operators need to possess some amount of Nym's cryptocurrency in order to make it expensive for any single entity to control a large share of the entire network. By algorithmically rewarding infrastructure operators for their (high-quality) service to the network, Nym aims to enable sustainable operation, in which infrastructure contributions do not need to be volunteered but pay off financially and operators are thus incentivized to contribute. Nym recently launched also a *virtual private network (VPN)* variant of its network, in which packets complete only two-hop paths without being delayed artificially. For the purposes of discussing and analyzing Nym in this thesis, we focus on Nym's mixnet variant and ignore Nym's incentivization efforts. Thus, hereafter, Nym refers to the system's continuous-time mixnet component.

The following Nym entities are relevant for the research presented in this thesis (we visualize these entities and parts of their interplay later in Figure 4.3):

- **Endpoints:** Endpoints are the entities that convert packets between the application level and the mixnet level (e.g., the ends of the AN pipes at the users in Figure 2.1), such that they can be routed through Nym's mixnet (if sent) or be relayed to a user-facing application on a Nym endpoint (if received). Thus, endpoints are the entities in Nym that can form mixnet-routable packets and be addressed by other endpoints as the mixnet-level recipients of their packets. This is in distinction to the notion of endpoints at other layers, e.g., web servers being endpoints for the end-to-end-encrypted application-layer protocol HTTPS. We distinguish user-operated from machine-operated endpoints. User-operated endpoints are the ones enabling a user's device to communicate via the Nym mixnet. Machine-operated endpoints enable networked services to communicate via Nym, either by integrating with Nym directly ("Nym-native" applications) or by proxying an application-layer protocol such as HTTPS through a *network-requester*. The network-requester endpoint acts as an automated conversion layer for packet flows between their Nym mixnet form and their public Internet form.
- **Mixes:** Nym's mixes are exponential mixes (as described before) and arranged in k layers (at the time of writing, $k = 3$). Operators who put sufficient cryptocurrency behind their nodes (a process called *bonding*) can put them up to be elected into the role of mixes for an upcoming epoch. The network state is maintained in a blockchain. At the time of writing, more than 300 mixes spread across multiple continents are part of the Nym network.

- **Gateways:** Endpoints in Nym do not communicate with mixes directly. Instead, gateways act as interface nodes between these two groups. Endpoints associate with gateways for packet buffering and bandwidth billing purposes. Gateways forward packets from the endpoints to the mixes in the first layer of the mixnet and relay packets from the last layer's mixes to the recipient endpoints (or hold on to them until an endpoint reconnects). While initially logically distinct, Nym by now allows for a unified mode in which both a gateway and a network-requester run on the same instance, thus adding *exit* functionality to gateways. We call a gateway that additionally offers network-requester functionality to Nym endpoints a *gateway-requester* or *gwreq*. At the time of writing, more than 200 gateways or gateway-requesters are available for endpoints to register with.

Nym strives to offer an anonymous routing layer to "a broad range of message-based applications and services" [14] and to offer unobservability to its endpoints. The first aspiration implies that Nym must choose a comparatively low average delay for which packets will get delayed at each mix, in order to satisfy the latency demands of the largest number of networked services possible. While Nym is not "low latency" in comparison to ANs such as Tor (which do not artificially delay packets at all), Nym does offer endpoint-to-endpoint packet transfers that are low in latency compared to other (historical) mixnet designs such as Mixminion [5][6]. While additional delay on the order of minutes may not matter when relaying emails via some mixnet, it certainly matters for latency-sensitive applications such as instant messaging and web browsing. Thus, endpoints in Nym sample the average delay μ with which packets will get delayed at each mix along their paths independently from an exponential distribution with mean $\mu = 50$ ms. With Nym's stratified topology currently consisting of $k = 3$ layers, this adds artificial mixing delay (in milliseconds) to packet transmissions following the Erlang distribution $E(3, \frac{1}{50})$, i.e., 150 ms of additional latency on average, with circa 86.6 ms standard deviation.

Nym's second aspiration—unobservability of endpoints—requires endpoints to shape their packet emission behavior and to fill gaps in the sending schedule (when no application-layer packet is available) with cover packets. Towards this, packet emissions of Nym endpoints are governed by Poisson process $Pois(\frac{1}{\lambda_{\mathbb{F}}} + \frac{1}{\lambda_{\mathbb{B}}})$, i.e., the superposition of $Pois(\frac{1}{\lambda_{\mathbb{F}}})$, with $\lambda_{\mathbb{F}} = 20$ ms, governing emissions from the endpoint's *foreground* queue and $Pois(\frac{1}{\lambda_{\mathbb{B}}})$, with $\lambda_{\mathbb{B}} = 200$ ms, for emissions from the endpoint's *background* queue. Application-level packets are always added to the foreground queue. If the foreground queue is empty at the next sending time, a loop cover packet is generated ad-hoc and sent in place of an application-level packet. Loop cover packets traverse Nym's mixnet indistinguishably from application-level packets and loop back to the sender. The endpoint's background queue exclusively emits loop cover packets.

[6] `https://web.archive.org/web/20250429122548/https://www.mixminion.net/manpages/mixminiond.conf.5.txt`

The procedure for a user to send the packet flow of an application on their device (e.g., a web browser) through Nym works as follows. The user's Nym endpoint splits the flow of application packets into *chunks* of a fixed size. Chunks that are too short are padded to this target size. Nym uses Danezis and Goldberg's *Sphinx* [9] as its cryptographic packet format and each chunk becomes the *payload* of a routable Sphinx packet. As a *source-routed* network, endpoints choose the Nym network paths for the Sphinx packets they intend to send. A path consists of five hops: the sender's gateway, the first-layer, second-layer, and third-layer randomly selected mix, and the recipient's gateway. Similar to onion routing, the endpoint layer-encrypts each chunk and associated routing header in reverse routing order according to the Sphinx packet creation algorithms. This step involves agreeing on a shared key with each mix along the path using public-key cryptography (Diffie-Hellman) and encoding the randomly sampled delay value into the header of the respective hop. The prepared Sphinx packet is appended to the endpoint's foreground queue. Loop cover packets are created in the same way with garbage data as payload. Upon receipt, loop cover packets are discarded.

Once the send time for a Sphinx packet arrives, the endpoint sends the packet to its first hop, the endpoint's gateway. After verifying that the endpoint has sufficient bandwidth available, the first-hop gateway injects the packet into Nym's mixnet by forwarding it to the user-selected first-layer mix. The first-layer mix processes the packet according to the Sphinx algorithms: it decrypts its layer, verifies integrity, and transforms header and payload into the next-hop packet. Importantly, the mix delays the packet for the specified amount of time before passing it on. Mixes at path positions three and four behave the same. The gateway of the packet's recipient processes the packet in the same way as the mixes, but does not delay it. Instead, the recipient's gateway recognizes that this is a packet destined for an endpoint (instead of a mix) and splits the packet's payload into an *acknowledgment* packet and the final payload data to be forwarded to the recipient.

Acknowledgment packets in Nym are realized using *single-use reply blocks (SURBs)* [5, 9]. SURBs are a mechanism in ANs by which a recipient can reply anonymously to a sender. In Nym, a SURB is a Sphinx header prepared by a sender and included in a payload packet. The recipient of the payload packet can use the enclosed SURB to attach a payload to it and form a valid packet that can be routed through the mixnet back to the sender. The recipient preparing the SURB-based payload packet neither knows the sender's gateway nor the path through the mixnet that the packet will take (except for the first-layer mix). Nym endpoints can thus be replied to anonymously by including SURBs in their payloads. One application of SURBs in Nym are acknowledgment packets. These are regular Sphinx packets (created from SURBs) but smaller than payload packets and carry only the information needed to acknowledge the receipt of a payload packet to the payload packet's sender. Senders include SURB-based acknowledgment packets in their payload packets. The gateways of the recipients split off the acknowledgments from the payload packets they receive and confirm delivery of these payload packets by sending the acknowledgments through the mixnet back to the sender.

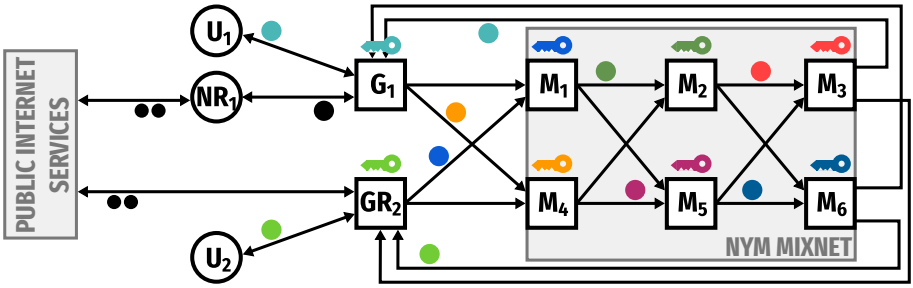


Figure 4.3: Overview of Nym's mixnet architecture, involving users U_u, gateways G_g, gateway-requesters GR_r, mixes M_m, network-requesters NR_n, and networked services on the public Internet. Users, network-requesters, and the network-requester components of gateway-requesters are endpoints from the perspective of the Nym mixnet. If U_1 and U_2 ran Nym-native applications, they could exchange packets directly. Shown, however, are the two ways for how users can access networked services on the public Internet: the legacy way where U_2 proxies through standalone network-requester NR_1 and the modern way where U_1 proxies through the network-requester embedded in gateway-requester GR_2. Packets on top of double arrows follow the arrow in the right direction, packets below them follow the arrow in the left direction. For illustration, the two packet paths purposefully do not mix while traversing the mixnet.

When a user wants to access a networked service unaware of Nym (e.g., a web server on the public Internet) anonymously through Nym, they can proxy their packet flow through a network-requester. As described before, the network-requester acts as a proxy translating between Nym's Sphinx packets and their application-layer form (e.g., HTTPS). Network-requesters are regular Nym endpoints that can be addressed in the same way as other user endpoints. Thus, the process for sending packets to network-requesters is the same as described above. If users do not trust the network-requesters they rely on for proxying, they can opt to always include enough SURBs as part of their requests so that the network-requesters can respond to them without the users having to disclose their Nym addresses (which include identifying information). While network-requester and gateway used to be distinct logical entities, between the time of our study presented in Chapter 9 and our study in Chapter 10, Nym enabled a unified operation, in the form of gateway-requesters, as described earlier. We depict a high-level overview of Nym's mixnet and the two ways for how packets can be proxied to the public Internet via network-requesters in Figure 4.3.

Finally, machine-operated endpoints in principle behave the same as user endpoints, i.e., they shape their egress traffic and prepare their packets in the same way. However, one crucial exception are network-requesters, which changed in a significant way between the time of our study presented in Chapter 9 and our study in Chapter 10. At the time of

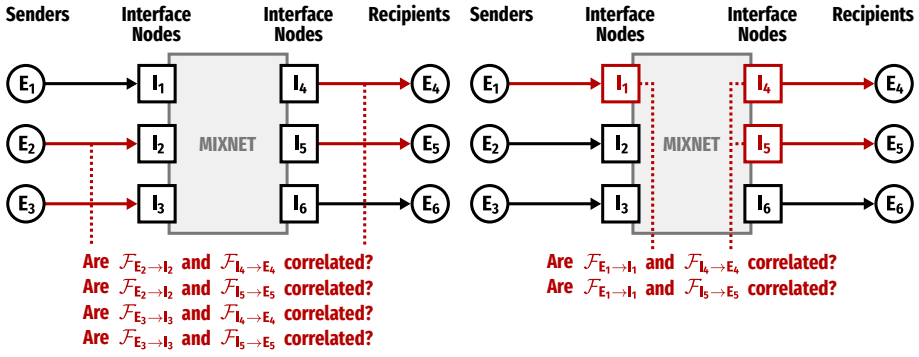
our study in Chapter 9, they indeed behaved the same as user endpoints. However, before we started our study presented in Chapter 10, network-requesters were changed to not shape their emissions from the foreground queue anymore, i.e., to abolish the Poisson-governed sending schedule and to instead send as fast as possible. This change was made in order to speed up data transmissions, as now bursty or large data transmissions from network-requesters to the requesting user endpoints complete much faster. However, this change has side effects and enabled the attack we present in Chapter 10.

4.3 Attacks on Mixnets and Defenses

In contrast to onion routing networks, mixnets are intended to resist traffic analysis attacks where the adversary observes both endpoints involved in a packet exchange. Even more so, Chaum lays out a very strong adversary against whom mixnets shall provide metadata protection in his original mixnet paper [3]: the adversary is able to observe all packets in the entire network (this is the $\mathbb{A}_{\mathrm{GPA}}$ from Section 2.3.5) and to corrupt or coerce all but one mix on the packets' paths. Recall from Section 2.3 that the adversary aims to recover who sends packets to whom and thus negate the mixnet's anonymity properties. It is to the adversary's advantage that most communications between endpoints happen in flows nowadays: flows consist of tens, hundreds, or even thousands of exchanged packets, each potentially leaking some amount of information about the associated sender-recipient pair. By analyzing observations for flows instead of single packets, the adversary is thus able to continually improve their inferences regarding the sender-recipient relationships. We introduce next the most relevant passive and active deanonymization attacks and discuss avenues for countermeasures.

4.3.1 Flow Correlation and Related Passive Attacks

In passive flow correlation attacks, we assume adversaries such as $\mathbb{A}_{\mathrm{IFC}}$ and $\mathbb{A}_{\mathrm{EFC}}$ from Section 2.3.5: adversaries with visibility over the mixnet's "edge", i.e., either controlling some or all of the mixnet's interface nodes (in Nym, these are the gateways and the gateway components of gateway-requesters) or observing some or all of the network links between the mixnet's interface nodes and the endpoints. Note that the attack can also be conducted using adversarial service endpoints to which other endpoints initiate connections (e.g., publicly announced network-requesters in Nym). While having such insider visibility reduces the noise in the adversary's observations significantly, the attack procedure is mostly the same. Flow correlation adversaries are clearly within the class of adversaries that mixnets are supposed to be resilient against. The attack focuses exclusively on what can be learned at the edge of a mixnet—i.e., treating the mixnet as a "black box" into which the adversary has no visibility—and proceeds in two phases.



(a) $\mathbb{A}_{\mathrm{EFC}}$ (b) $\mathbb{A}_{\mathrm{IFC}}$

Figure 4.4: Flow correlation attack settings against mixnets. The adversary (marked in red) correlates packet flows arriving at the mixnet with packet flows leaving the mixnet. Figure 4.4a shows the external (network-level) adversary $\mathbb{A}_{\mathrm{EFC}}$ (see Section 2.3.5) computing similarity scores between packet metadata flows $\mathcal{F}$ observed at the network level. Figure 4.4b depicts the insider adversary $\mathbb{A}_{\mathrm{IFC}}$ (see Section 2.3.5) controlling some of the mixnet's interface nodes (either mixes or other infrastructure nodes such as gateways in Nym or providers in Loopix), who computes similarity scores between packet metadata flows $\mathcal{F}$ observed at the mixnet level (thus containing much less noise).

First, a flow-correlating adversary collects the packet metadata (timestamp, direction, size) of all observable flows between endpoints and the mixnet's edge, potentially over a very long time. If the adversary is external to the mixnet under attack ($\mathbb{A}_{\mathrm{EFC}}$ in Figure 4.4a), they may at this point opt to preprocess the collected packet metadata in order to reduce the amount of noise that is caused by the translation of packets from the mixnet to the network level. An insider adversary ($\mathbb{A}_{\mathrm{IFC}}$ in Figure 4.4b) does not need to "clean up" packet metadata sequences in a similar way, as they already have a noise-free view on mixnet-level flows. In either case, the adversary may decide to store fewer than all available packet metadata to leverage for their attack.

Then, in the second phase, the adversary computes a similarity measure (a correlation score) for every sender-recipient pair of metadata sequences and decides whether the implicated endpoints indeed exchange packets, e.g., using a binary classifier with appropriate threshold (see Section 2.4.1). This similarity score is calculated independently for each sender-recipient candidate pair. The adversary can launch this attack as soon as they control at least one mixnet interface node used for sending packets to the mixnet and one interface node used for sending packets to endpoints. It often makes sense to test both directions of a candidate endpoint pair (endpoint E_1 sending to endpoint E_2 and E_2 sending to E_1) together, if communication is bidirectional. The

second phase of the attack can take place at any time after the first phase.

Typically, the outputs of phase one (i.e., the raw packet metadata sequences) are not used directly to conduct the attack. Instead, the adversary splits the sequences based on direction and size, thus obtaining timestamp-only packet metadata sequences for a particular direction and packet size. From these packet timestamp sequences, the adversary can prepare the inputs to the second stage. Because packets to the same recipient that have taken very different paths through the mixnet consolidate again on the final link from mixnet to recipient, many of the flow correlation methods effective against connection-oriented ANs such as Tor can be tried against mixnets as well. The adversary may compute similarity measures using traditional statistical or modern deep learning methods from: packet counts in specific time windows [34, 36, 38], inter-arrival times of packets [4, 26, 27, 28], or traffic density estimations of the flows [29].

Mixnets that do not perfectly hide statistical properties of the application-level flows transported through them (e.g., Loopix and Nym) will leak this kind of packet timing information about sender-recipient relationships to observers. This thesis contributes two flow correlation attacks exploiting precisely this timing information (see Chapter 9 and Chapter 10, respectively). However, even mixnets in which all endpoints send with the same rate may be susceptible to flow correlation of a different kind, called *intersection* or *disclosure attacks* [1, 7, 11, 17, 21, 25, 31]. The adversary exploits the fact that sender and recipient sets change over time as users join and leave the mixnet or switch conversation partners. By observing flows at the mixnet edge long enough, the adversary can progressively rule out potential communication partners for endpoints and thus become increasingly confident in the actual sender-recipient relationships.

Note that an adversary with visibility over the *entire* edge of the mixnet under attack has a *complete* view of all possible pairs. Thus far, we have assumed the flow correlation attack to be applied to each candidate flow pair independently, i.e., in isolation of other pairs and thus without the aim to arrive at a bipartite graph for senders and recipients. If, however, the adversary observes the entire mixnet edge (of which they can be sure of in mixnets that publicly announce their interface nodes, which are most), they can conduct a regular flow correlation attack against all possible sender-recipient pairs and declare the pairing with the highest similarity score to be correlated—without relying on an absolute threshold to determine similarity and decide if a flow pair is correlated.

Flow Matching as a Special Case of Flow Correlation

Flow matching refers to the special case of flow correlation that is *complete*. Here, we assume that groups of endpoints communicate exclusively amongst themselves and that all ingress packets eventually become egress packets (i.e., they do not get lost or leave the considered endpoints set). We introduce flow matching as part of Chapter 9, where we evaluate the threat it poses against bidirectionally regulated Nym. Flow matching

asks whether the merged egress flow (consisting of the egress flows of all considered endpoints) is a complete representation of the merged ingress flow (consisting of the ingress flows of all considered endpoints) after having traversed the mixnet.

A candidate ingress-egress flow pair, in which half of the packets indeed travel between the considered endpoints, is considered *not matched* under flow matching (as packets are unaccounted for and so the egress flow is not a complete representation of the ingress flow) while it is considered *correlated* under flow correlation (as packets are exchanged between the considered endpoints, even if not all of them). Clearly, packet loss presents a problem, as dropped packets violate the assumption that all ingress packets become egress packets while we would still consider the complete set of endpoints involved in an exchange to be communicating exclusively. As we demonstrate in Chapter 9, the conditions of real-world networks such as the Internet (which include packet losses) indeed degrade the ability of flow matching classifiers to correctly match flows.

4.3.2 Flow Correlation Countermeasures

Countermeasures to the threat of flow correlation fall into three different categories: (a) reduce the number of observations that an adversary can collect, (b) increase the amount of noise and distortion in the observations, or (c) reduce the likelihood of the conditions for the attack to occur. We overview each category in turn.

Reduce Number of Observations. Given that with each observation (i.e., observing the transfer of one packet over one hop) the adversary learns a little bit more about the sender-recipient relationships relayed through a mixnet, it is a worthwhile defense to reduce the number of observations that an adversary can collect over time. For the same amount of payload data transferred at the application level, increasing the size of the mixnet-level packets results in fewer packets exchanged between mixnet endpoints. Clearly, this increase in protection from flow correlation attacks comes at the expense of higher "wasted" bandwidth, as more bandwidth is spent on padding packets that are too short. Similarly, if previously distinguishable packet types (e.g., special-purpose packets such as acknowledgments or management packets of a smaller size than payload packets) can be removed from the mixnet level, this takes away observations from the adversary and thus prolongs the time the adversary needs to observe senders and recipients for.

Additionally, if the adversary observes only some part of the mixnet's edge, endpoints can split the packets of their flows between multiple interface nodes that inject them into the mixnet and between multiple interface nodes that forward the packets to the receiving endpoints. Assuming that not all interface nodes selected for this flow splitting procedure are adversarial (or at least, not all colluding), this technique exposes only a fraction of the observations about the original flow to the adversary. If the recipient's interface node and the receiving endpoint are the same entity (as is the case with Nym's gateway-requesters for requests proxied to the public Internet), whether flows can be

split between multiple interface nodes also when leaving the mixnet may depend on whether the application-level protocol allows for it.

Increase Amount of Noise and Distortion in Observations. This defense category concerns cover traffic, traffic shaping, and increasing the variance of the time it takes packets to traverse the mixnet (their mixing delay variance). By employing these defense techniques, packet flows look more uniform to the adversary when they enter the mixnet (due to traffic shaping and filling gaps with cover packets) and it is less predictable for the adversary when a packet entering the mixnet will exit it again (due to increased mixing delay variance). We thus increase the noise in adversarial observations that better hide the genuine communication contained in them and also distort the *shape* of flows (their distribution of packets over time) on their way through the mixnet. As a result, it is harder for the flow correlation adversary to identify which packet flow entering the mixnet corresponds to which packet flow exiting it and attack performance for the same number of observations drops compared to a mixnet without these defenses. Increasing the mixing delay variance can, for example, be achieved by choosing a pool mix instead of a threshold or timed mix in batching mixnets or by increasing the average per-mix packet delay μ in continuous-time mixnets such as Loopix or Nym.

Depending on whether the mixnet enforces limits on the number of packets each endpoint can send per time interval, different schemes for sending cover traffic exist, and cover packets can originate and terminate at various locations in the mixnet [30]. Cover packets may be sent by endpoints to terminate at endpoints, in which case they may loop back to the sender [14, 33] or go to an endpoint who is not the sender, or to be dropped by mixes along their paths [24]. Similarly, infrastructure nodes such as mixes may also send cover packets through the mixnet, again, either back to themselves [33] or to be dropped by some other participant [38]. Crucially, all types of cover packets need to be indistinguishable from genuine packets transporting application-layer data.

The schedule on which endpoints may send cover packets is closely related to whether they shape their packet emissions to follow some target profile. If the mixnet stipulates sending with constant bandwidth (or constant bandwidth on average in the case of Poisson-governed packet emissions), cover packets need to be employed to fill gaps, i.e., a cover packet needs to be generated and sent each time a send slot would otherwise go unused because no application-level packet was available. If endpoints can send at will (i.e., packet emissions do not follow a target shape), cover packets can also be sent following a variable-rate schedule that may depend on environmental factors and could even be optimized by training a neural network for it. Note that if SURBs are available in the mixnet, they allow for an amplification strategy, where endpoints can deposit a large number of SURBs at some auxiliary participant in the transmission of a single packet. They also enclose expected times for when the auxiliary participant (if honest) will send one cover packet per SURB through the mixnet back to the sender. This way, one packet sent in one direction turns into many packets sent in the reverse.

Finally, increasing the mixing delay variance helps confuse a flow correlation adversary by distorting the shape of each flow as it transits the mixnet and thus their similarity is decreased, which helps protect against this attack. However, we note two practical difficulties with this approach to countering flow correlation. First, the intended application use case of a mixnet may simply not allow for arbitrarily prolonged mixing delays, as timeouts may occur (too frequently). Second, users may perceive the performance of a mixnet with less predictable and higher mixing delays as intolerable, switch to alternative means for transporting their packets, and thus degrade overall anonymity of the mixnet. While mixnets should aim for as high a mixing delay variance as is practical for their use case, it may well be more advantageous to instead focus on attracting new and retaining existing users to grow the anonymity sets this way.

Reduce Likelihood of Attack Setting. In mixnets where users choose through which interface node to send or receive packets (e.g., gateways in Nym and providers in Loopix), trust-based selection of these nodes can be employed to reduce the likelihood of the preconditions for flow correlation to manifest. Through other channels, operators of interface nodes can build up a reputation of being trustworthy, e.g., by engaging in the mixnet's community, making their operation as transparent as possible, and publishing their high performance on measures of correct operation that the mixnet may gather. The easier it is for users to correctly decide for themselves through these side channels who among the interface node operators is trustworthy, the less likely an adversary posing as many independent operators or with an unconvincing public persona is going to be chosen for the sensitive interface node position required to control for the attack.

In mixnets where users do not choose the interface nodes themselves, trust-based election of nodes into these positions could still be leveraged, however, encoded in some technical way into the algorithm that decides who is going to operate the edge nodes. For example, a synchronous cascade mixnet may benefit from regularly rotating its network topology in a way that involves randomness and whichever technical proxy measure is available to assess how correctly infrastructure nodes have executed their roles [23]. Assuming some mechanism is implemented that makes it difficult for a single entity to spoof many distinct identities (e.g., because it is exceedingly expensive in some scarce or regulated resource), enforcing that first and last hop of the cascade be operated by different entities increases the work an adversary has to do in order to control both ends of the cascade.

4.3.3 Blending and Other Active Attacks

Passive attacks allow the adversary to remain undetected for a potentially long time, recording and analyzing packet metadata while otherwise following protocol. However, adversaries may also choose to become active and tamper with flows. For example, a flow correlation adversary choosing to become active can use their capabilities to delay,

drop, insert, or modify packets and induce identifiable patterns (watermarks) into flows observable at the edges. Watermarks can be particularly effective against continuous-time mixnets where endpoints shape their packet emissions, as they noticeably alter target flows and thus make them more distinguishable from unrelated flows.

A different kind of adversary with network visibility over the links between the mixes can mount a class of packet tracing attacks called *blending attacks* [1, 18, 34, 35]. A blending attack enables an adversary with this kind of network visibility into the mixnet to trace a packet from its sender through a synchronous mixnet to its recipient, thus defeating the anonymity objective of the mixnet. The adversary manipulates a batch of packets entering a mix so that the only unknown packet is the target one. For this, the adversary either delays all packets that are not the target packet so that the target packet is the only packet in the mix or floods the mix with so many packets of their own that the only other packet is the target packet. As the adversary now knows which packet leaving a mix is the target packet, they can follow the packet through the mixnet to its recipient by iteratively applying the same procedure to each mix along the packet's path. Depending on how they are realized, cover packets generated and injected by the mixes can complicate this attack for the adversary.

While active attacks can indeed be very effective and are expensive to protect against ahead of time (see verifiable mixnets in Section 4.1.1), they may be detectable and thus enable mixnet designers to specify protocols for participants to take when an active attack is detected. Looping cover packets can be used by either endpoints or mixes to ascertain whether they are under active attack, e.g., because the number of returning loops or their transmission times through the mixnet deviate (enough) from statistical expectation. Danezis and Sassaman's red-green-black mixes [10] implement this strategy. Red packets are looping cover packets sent by mixes together with the genuine black packets. If too few red packets return to a mix, the mix injects (more) green packets, which are cover packets addressed to some later stage in the mixnet topology and thus confuse a blending attack adversary. Loopix [33] also foresees mixes increasing their cover packet rate if they receive fewer than an expected number of looping cover packets back. Leibowitz et al. [23] combine loop cover packets with a reputation system for active attack resistance in their system Miranda.

Other active attacks are *packet replay* [9] and *packet tagging attacks* [34]. A mixnet not keeping track of already processed packets is vulnerable to packet replay attacks, in which an active adversary duplicates and resends a packet in order to learn where it is going. If the mixnet does not provide packet integrity protection between a sender and each hop along a sender-chosen path, it is vulnerable to packet tagging attacks. Here, the adversary modifies a packet lightly to use this modification to identify this packet at later hops. The adversary needs to have the appropriate visibility into the mixnet that enables them to observe the modifications caused by replaying or tagging packets at the targeted location along the packets' paths. Choosing a well-designed cryptographic packet format such as Sphinx [9] goes a long way in proofing a mixnet

against these active attacks, as a rigorously constructed packet format disables attacks such as replaying or tagging packets cryptographically.

Bibliography

[1] O. Berthold, A. Pfitzmann, and R. Standtke. "The Disadvantages of Free MIX Routes and How to Overcome Them". In: *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability Berkeley, CA, USA, July 25–26, 2000 Proceedings*. Ed. by H. Federrath. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 30–45. ISBN: 978-3-540-44702-3. DOI: 10.1007/3-540-44702-4_3. URL: https://doi.org/10.1007/3-540-44702-4_3 (pp. 68, 72).

[2] R. Böhme, G. Danezis, C. Díaz, S. Köpsell, and A. Pfitzmann. "On the PET Workshop Panel "Mix Cascades Versus Peer-to-Peer: Is One Concept Superior?"" In: *Privacy Enhancing Technologies*. Ed. by D. Martin and A. Serjantov. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 243–255. ISBN: 978-3-540-31960-3 (p. 58).

[3] D. L. Chaum. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms". In: *Commun. ACM* 24.2 (Feb. 1981), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/358549.358563. URL: https://doi.org/10.1145/358549.358563 (pp. 55, 56, 59, 60, 61, 66).

[4] T. Chawla, S. Mittal, N. Mathews, and M. Wright. "ESPRESSO: Advanced End-to-End Flow Correlation Attacks on Tor". In: *Proceedings of the 8th Asia-Pacific Workshop on Networking*. APNet '24. Sydney, Australia: Association for Computing Machinery, 2024, pp. 219–220. ISBN: 9798400717581. DOI: 10.1145/3663408.3665824. URL: https://doi.org/10.1145/3663408.3665824 (p. 68).

[5] G. Danezis, R. Dingledine, and N. Mathewson. "Mixminion: Design of a Type III Anonymous Remailer Protocol". In: *2003 Symposium on Security and Privacy, 2003.* 2003, pp. 2–15. DOI: 10.1109/SECPRI.2003.1199323 (pp. 60, 63, 64).

[6] G. Danezis. "Mix-Networks with Restricted Routes". In: *Privacy Enhancing Technologies*. Ed. by R. Dingledine. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 1–17. ISBN: 978-3-540-40956-4 (p. 59).

[7] G. Danezis. "Statistical Disclosure Attacks". In: *Security and Privacy in the Age of Uncertainty*. Ed. by D. Gritzalis, S. De Capitani di Vimercati, P. Samarati, and S. Katsikas. Boston, MA: Springer US, 2003, pp. 421–426. ISBN: 978-0-387-35691-4 (p. 68).

[8] G. Danezis. "The Traffic Analysis of Continuous-Time Mixes". In: *Privacy Enhancing Technologies*. Ed. by D. Martin and A. Serjantov. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 35–50. ISBN: 978-3-540-31960-3 (p. 60).

[9] G. Danezis and I. Goldberg. "Sphinx: A Compact and Provably Secure Mix Format". In: *2009 30th IEEE Symposium on Security and Privacy (SP)*. 2009, pp. 269–282. DOI: 10.1109/SP.2009.15 (pp. 64, 72).

[10] G. Danezis and L. Sassaman. "Heartbeat Traffic to Counter (n-1) Attacks: Red-Green-Black Mixes". In: *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society*. WPES '03. Washington, DC: Association for Computing Machinery, 2003, pp. 89–93. ISBN: 1581137761. DOI: 10.1145/1005140.1005154. URL: https://doi.org/10.1145/1005140.1005154 (pp. 61, 72).

[11] G. Danezis and A. Serjantov. "Statistical Disclosure or Intersection Attacks on Anonymity Systems". In: *Information Hiding*. Ed. by J. Fridrich. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 293–308. ISBN: 978-3-540-30114-1 (p. 68).

[12] D. Das, C. Diaz, A. Kiayias, and T. Zacharias. "Are continuous stop-and-go mixnets provably secure?" In: *Proceedings of the 24th Privacy Enhancing Technologies Symposium (PETS 2024)*. Vol. 2024. 2024, pp. 665–683. DOI: 10.56553/popets-2024-0136 (p. 60).

[13] F. M. Dekking, C. Kraaikamp, H. P. Lopuhaä, and L. E. Meester. "Continuous random variables". In: *A Modern Introduction to Probability and Statistics: Understanding Why and How*. London: Springer London, 2005, pp. 57–70. ISBN: 978-1-84628-168-6. DOI: 10.1007/1-84628-168-7_5. URL: https://doi.org/10.1007/1-84628-168-7_5 (p. 60).

[14] C. Diaz, H. Halpin, and A. Kiayias. *The Nym Network*. Tech. rep. Version 1.0 as of February 26, 2021. Nym Technologies SA, Feb. 2021. URL: https://web.archive.org/web/20250403044143/https://nym.com/nym-whitepaper.pdf (pp. 55, 60, 61, 63, 70).

[15] C. Diaz, S. J. Murdoch, and C. Troncoso. "Impact of Network Topology on Anonymity and Overhead in Low-Latency Anonymity Networks". In: *Privacy Enhancing Technologies*. Ed. by M. J. Atallah and N. J. Hopper. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 184–201. ISBN: 978-3-642-14527-8 (p. 58).

[16] C. Díaz and A. Serjantov. "Generalising Mixes". In: *Privacy Enhancing Technologies*. Ed. by R. Dingledine. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 18–31. ISBN: 978-3-540-40956-4 (p. 60).

[17] R. Dingledine, V. Shmatikov, and P. Syverson. "Synchronous Batching: From Cascades to Free Routes". In: *Privacy Enhancing Technologies*. Ed. by D. Martin and A. Serjantov. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 186–206. ISBN: 978-3-540-31960-3 (pp. 58, 68).

[18] C. Gülcü and G. Tsudik. "Mixing E-mail with BABEL". In: *Proceedings of Internet Society Symposium on Network and Distributed Systems Security*. 1996, pp. 2–16. DOI: 10.1109/NDSS.1996.492350 (p. 72).

[19] T. Haines, R. Goré, and B. Sharma. "Did you mix me? Formally Verifying Verifiable Mix Nets in Electronic Voting". In: *2021 IEEE Symposium on Security and Privacy (SP)*. 2021, pp. 1748–1765. DOI: 10.1109/SP40001.2021.00033 (p. 57).

[20] T. Haines and J. Müller. "SoK: Techniques for Verifiable Mix Nets". In: *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*. 2020, pp. 49–64. DOI: 10.1109/CSF49147.2020.00012 (p. 57).

[21] D. Kesdogan, D. Agrawal, and S. Penz. "Limits of Anonymity in Open Environments". In: *Information Hiding*. Ed. by F. A. P. Petitcolas. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 53–69. ISBN: 978-3-540-36415-3 (p. 68).

[22] D. Kesdogan, J. Egner, and R. Büschkes. "Stop-and-Go-MIXes Providing Probabilistic Anonymity in an Open System". In: *Information Hiding*. Ed. by D. Aucsmith. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 83–98. ISBN: 978-3-540-49380-8 (p. 60).

[23] H. Leibowitz, A. M. Piotrowska, G. Danezis, and A. Herzberg. "No Right to Remain Silent: Isolating Malicious Mixes". In: *28th USENIX Security Symposium (USENIX Security '19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1841–1858. ISBN: 978-1-939133-06-9. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/leibowitz (pp. 71, 72).

[24] B. N. Levine, M. K. Reiter, C. Wang, and M. Wright. "Timing Attacks in Low-Latency Mix Systems". In: *Financial Cryptography*. Ed. by A. Juels. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 251–265. ISBN: 978-3-540-27809-2 (p. 70).

[25] N. Mathewson and R. Dingledine. "Practical Traffic Analysis: Extending and Resisting Statistical Disclosure". In: *Privacy Enhancing Technologies*. Ed. by D. Martin and A. Serjantov. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 17–34. ISBN: 978-3-540-31960-3 (p. 68).

[26] M. Nasr, A. Bahramali, and A. Houmansadr. "DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. Toronto, Canada: Association for Computing Machinery, 2018, pp. 1962–1976. ISBN: 9781450356930. DOI: 10.1145/3243734.3243824. URL: https://doi.org/10.1145/3243734.3243824 (p. 68).

[27] S. E. Oh, T. Yang, N. Mathews, J. K. Holland, M. S. Rahman, N. Hopper, and M. Wright. "DeepCoFFEA: Improved Flow Correlation Attacks on Tor via Metric Learning and Amplification". In: *2022 IEEE Symposium on Security and Privacy (SP)*. 2022, pp. 1915–1932. DOI: 10.1109/SP46214.2022.9833801 (p. 68).

[28] L. Oldenburg, M. Juarez, E. Argones Rúa, and C. Diaz. "MixMatch: Flow Matching for Mixnet Traffic". In: *Proceedings of the 24th Privacy Enhancing Technologies Symposium (PETS 2024)*. Vol. 2024. 2024, pp. 276–294. DOI: 10.56553/popets-2024-0050 (p. 68).

[29] L. Oldenburg, M. Juarez, E. Argones Rúa, and C. Diaz. "Shift Your Shape: Correlating and Defending Mixnet Flows Based on Their Shapes". Under submission to *IEEE Transactions on Dependable and Secure Computing* since March 31, 2025. 2025 (p. 68).

[30] S. Oya, C. Troncoso, and F. Pérez-González. "Do Dummies Pay Off? Limits of Dummy Traffic Protection in Anonymous Communications". In: *Privacy Enhancing Technologies*. Ed. by E. De Cristofaro and S. J. Murdoch. Cham: Springer International Publishing, 2014, pp. 204–223. ISBN: 978-3-319-08506-7 (p. 70).

[31] S. Oya, C. Troncoso, and F. Pérez-González. "Meet the Family of Statistical Disclosure Attacks". In: *2013 IEEE Global Conference on Signal and Information Processing*. 2013, pp. 233–236. DOI: 10.1109/GlobalSIP.2013.6736858 (p. 68).

[32] A. Pfitzmann and M. Hansen. *A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management*. v0.34. Aug. 2010. URL: http://dud.inf.tu-dresden.de/literatur/Anon%5C_Terminology%5C_v0.34.pdf (p. 61).

[33] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis. "The Loopix Anonymity System". In: *26th USENIX Security Symposium (USENIX Security '17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1199–1216. ISBN: 978-1-931971-40-9. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/piotrowska (pp. 60, 61, 70, 72).

[34] J.-F. Raymond. "Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems". In: *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability Berkeley, CA, USA, July 25–26, 2000 Proceedings*. Ed. by H. Federrath. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 10–29. ISBN: 978-3-540-44702-3. DOI: 10.1007/3-540-44702-4_2. URL: https://doi.org/10.1007/3-540-44702-4_2 (pp. 68, 72).

[35] A. Serjantov, R. Dingledine, and P. Syverson. "From a Trickle to a Flood: Active Attacks on Several Mix Types". In: *Information Hiding*. Ed. by F. A. P. Petitcolas. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 36–52. ISBN: 978-3-540-36415-3 (pp. 60, 72).

[36] A. Serjantov and P. Sewell. "Passive Attack Analysis for Connection-Based Anonymity Systems". In: *Computer Security – ESORICS 2003*. Ed. by E. Snekkenes and D. Gollmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 116–131. ISBN: 978-3-540-39650-5 (pp. 58, 68).

[37] F. Shirazi, M. Simeonovski, M. R. Asghar, M. Backes, and C. Diaz. "A Survey on Routing in Anonymous Communication Protocols". In: *ACM Comput. Surv.* 51.3 (June 2018). ISSN: 0360-0300. DOI: 10.1145/3182658. URL: https://doi.org/10.1145/3182658 (p. 58).

[38] V. Shmatikov and M.-H. Wang. "Timing Analysis in Low-Latency Mix Networks: Attacks and Defenses". In: *Computer Security – ESORICS 2006*. Ed. by D. Gollmann, J. Meier, and A. Sabelfeld. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 18–33. ISBN: 978-3-540-44605-7 (pp. 68, 70).

Chapter 5

Contributions

In this chapter, we succinctly state the contributions to the literature included in this thesis and place them in the context of the preceding chapters.

5.1 "Strong Anonymity is not Enough: Introducing Fault Tolerance to Planet-Scale Anonymous Communication Systems"

In our first research question (Section 1.4) and in Section 2.5, we alluded to the relationship between the level of fault tolerance an AN possesses and the level of anonymity it is able to provide its users. Specifically, the negative connection: we posit that a lack of fault tolerance in systems where failures are inevitable (which most ANs are) will degrade the anonymity they can offer, due to unavailability or intolerable performance when failures occur. On this connection between anonymity and resilience in ANs, this thesis contributes the study [5] in Chapter 7, published in the *16th International Workshop on Frontiers in Availability, Reliability and Security (FARES 2021)*.

We subject three ANs to three different network failure settings in a planet-spanning evaluation testbed, in which we measure resource usage and *packet end-to-end latencies* (i.e., how long packets take to travel from sender through the AN to recipient) for three different user base sizes. The first failure setting sees, in fact, no (artificial) failures and thus serves as a baseline. The other two failure settings introduce various amounts of packet loss and high delay. Two of the evaluated ANs are state-of-the-art representatives of their respective approach, while the final one is a proof-of-concept

AN we implemented using a mixnet basis and simple fault tolerance measures.

The goal of this study is to evaluate the impact that network failures have on different approaches to and different levels of resilience in AN designs. By measuring each AN's deviation in packet end-to-end latencies under failures from its baseline performance, we obtain a method to reason about the AN's resilience in the form of user-perceived routing performance. We can visualize these differences graphically in cumulative distribution plots or compress them into numeric values by computing the *First Wasserstein distance* (or *earth mover's distance* [6]) between each failure and the baseline scenario. We find that the simple fault tolerance measures integrated into our prototype mixnet achieve their purpose and lead to lower distances of the failure scenario distributions from the baseline scenario than for the other two ANs. To designers of ANs, we recommend making resilience a first-class design principle by building in fault tolerance measures (e.g., forward error correction, redundant paths and packets) from the beginning.

This study is published as:

> L. Oldenburg and F. Tschorsch. "Strong Anonymity is not Enough: Introducing Fault Tolerance to Planet-Scale Anonymous Communication Systems". In: *Proceedings of the 16th International Conference on Availability, Reliability and Security*. ARES '21. Association for Computing Machinery, 2021. DOI: 10.1145/3465481.3469189

I am the first author of the publication and have undertaken the majority of the conceptual, implementation, evaluation, and writing-up work that went into it.

5.2 "From 'Onion Not Found' to Guard Discovery"

In the category of guard discovery attacks on Tor (see Section 3.3.3), this thesis contributes the study [2] in Chapter 8, published in the *Proceedings of the Privacy Enhancing Technologies Symposium (PETS 2022)*.

In this work, we present a guard discovery attack that is fast and stealthy for an adversary that operates a non-negligible but limited number of Tor relays and controls parts of a target webpage in some way. The attack exploits the fact that the cell pattern of lookups for non-existing onion service descriptors is distinguishable from other traffic at the second-hop position of these circuits. Our attack targets Tor Browser users, i.e., Tor users accessing webpages using Tor Browser. We assume that the adversary has some capacity to modify a target webpage and aims to discover the guards of the users visiting this webpage with Tor Browser. The adversary may obtain this webpage modification capability by either directly operating the webpage or by being an involved third party,

e.g., an advertiser. Additionally, the adversary operates a number of Tor relays that carry the "HSDir" flag and offer bandwidth as middle relays to the network.

The adversary launches the attack by embedding resources on non-existing onion services into the target webpage requested by a Tor Browser user. The user's Tor Browser creates multiple four-hop circuits to HSDirs per onion service address in order to download the associated onion service descriptor. Crucially, the referenced onion services do not exist. Their addresses were crafted by the adversary ahead of the attack in such a way that the adversary's HSDirs would be contacted by the targeted Tor Browser users when rendering the attack webpage. By recording characteristics about these lookup circuits both at their HSDirs and their middle relays, the adversary may determine that they occupy the second-hop circuit position on one of these lookups and take note of the relay in the first-hop position. By waiting for a relay to occur twice as the logged first-hop relay on these circuits, we show that the adversary is able to discover the guards of $50\,\%$ of the targeted Tor Browser users in 12.06 s with a true positive rate of 0.9978. We evaluate a token bucket that limits the number of lookup circuits a Tor Browser tab can create, the effect of Vanguards-Lite[1] in restricting second-hop relay choices, and a combination of the two as countermeasures to our attack. All defenses offer Tor Browser users some kind of protection, however, none is a perfect solution. Tor deploys Vanguards-Lite by default as of version `0.4.7`.

This study is published as:

> L. Oldenburg, G. Acar, and C. Diaz. "From "Onion Not Found" to Guard Discovery". In: *Proceedings of the 22nd Privacy Enhancing Technologies Symposium (PETS 2022)*. Vol. 2022. 2022, pp. 522–543. DOI: 10.2478/popets-2022-0026

I am the first author on the publication and have undertaken major parts of the conceptual, implementation, evaluation, and writing-up work that went into it.

5.3 "MixMatch: Flow Matching for Mixnet Traffic"

This thesis contributes two studies evaluating the threat of and defenses against passive flow correlation on state-of-the-art mixnet Nym. The first study [3] (Chapter 9) evaluates flow correlation (in its special case of flow matching) in the setting of bidirectionally regulated Nym and is published in the *Proceedings of the Privacy Enhancing Technologies Symposium (PETS 2024)*. The second one is summarized in the following section.

[1] `https://web.archive.org/web/20250319234942/https://spec.torproject.org/vanguards-spec/vanguards-lite.html`

As described in Section 4.3.1, a passive flow correlation adversary analyzes only the packet flow metadata they observe at the edge of the mixnet under attack. By either controlling some of the mixnet's edge nodes interfacing with actual users or monitoring the network links on this hop between users and the mixnet, the adversary aims to correlate a flow entering the mixnet with its transformed version exiting the mixnet. We present an evaluation of the flow matching threat using novel methods against the version of Nym in which packet emissions are shaped according to Poisson processes in both directions of bidirectional flows and a discussion of mitigations of the threat.

We let Nym endpoints under our control download a file via HTTP from our web server by proxying their requests through network-requesters that we set up, both in a lab setup without real-world network effects and on the live Nym network with these effects. Both directions of this exchange (upstream from endpoint to network-requester and downstream from network-requester to endpoint) are Poisson-governed and use cover traffic, and thus make correlation based on traffic shape features ineffective. We show this by demonstrating that DeepCoFFEA [1], the state-of-the-art flow correlation attack against Tor, is not well suited to this classification task on Nym. Despite Nym's rigorous traffic shaping, however, the adversary can still check if the difference between egress and ingress packet timestamps follows the expected distribution for matched flows or *drifts* away to decide whether or not a set of users is communicating.

Our attack relies on the assumption that endpoints communicate in closed sets, i.e., they exchange packets within their arbitrarily sized group, however, never with endpoints outside this set. We also assume that the adversary observes the entire set. Our *flow merging* technique yields unified ingress and egress flows for any endpoints set and thus leverages Nym's loop cover traffic as additional observations the adversary can utilize towards deanonymizing endpoints. We implement our attack as three classifiers: one using traditional statistical methods and two based on deep learning methods. Through extensive evaluations (including varying many of Nym's core parameters), we show that two of our classifiers perform very well at correctly matching flows—as long as no realistic network effects are present. On our final dataset, collected on the live Nym network and thus capturing realistic network effects such as packet losses, our classifiers perform much worse, indicating that invalidating our flow matching's assumptions may be a viable approach for a defense against the attack. We contribute an extensive discussion on the effectiveness, implications, and countermeasure avenues of our attack.

This study is published as:

> L. Oldenburg, M. Juarez, E. Argones Rúa, and C. Diaz. "MixMatch: Flow Matching for Mixnet Traffic". In: *Proceedings of the 24th Privacy Enhancing Technologies Symposium (PETS 2024)*. Vol. 2024. 2024, pp. 276–294. DOI: 10.56553/popets-2024-0050

This article won one of two *"Andreas Pfitzmann Best Student Paper Awards"* at PETS 2024.

I am the first author on the publication and have undertaken major parts of the conceptual, implementation, evaluation, and writing-up work that went into it.

5.4 "Shift Your Shape: Correlating and Defending Mixnet Flows Based on Their Shapes"

The second study [4] (Chapter 10) evaluates the threat of passive flow correlation (see Section 4.3.1) on downstream-unregulated Nym and is currently under submission to *IEEE Transactions on Dependable and Secure Computing (IEEE TDSC)*.

In this study, we evaluate passive flow correlation against Nym after one of Nym's core components, network-requesters, were changed in the following significant way: instead of shaping their packet emissions according to a Poisson process as any other Nym endpoint does, they now send packets in the direction of Nym endpoints (downstream) as quickly as possible. Recall that Nym endpoints may contact network-requesters in order to proxy requests destined for servers on the public Internet through Nym's mixnet and thus access these servers anonymously. The mentioned change turns Nym from a bidirectionally regulated into a downstream-unregulated mixnet, in which the variable rate of downstream packets (e.g., bursty web traffic) is not obfuscated anymore by traffic shaping from a Poisson process. Thus, correlating downstream flows observed at the edge of Nym's mixnet based on their *shape* (distribution of packets over time) becomes a threat and we systematically evaluate this threat and possible countermeasures.

In order for our attack to work, the adversary needs to control the first gateway-requester (called entry) and the last gateway-requester (called exit) on the user-chosen packet paths through the mixnet to servers on the public Internet. The adversary can acquire this vantage position by contributing at least two gateway-requesters to the Nym network. With these gateway-requesters present, users choosing one of the adversary's gateway-requesters as their entry and the network-requester component of one of the adversary's gateway-requesters as their exit are susceptible to our attack. The adversary records the Sphinx-level packet metadata for flows observed at their gateway-requesters and transforms them into their shape representations using a process we propose based on *kernel density estimation*. By passing them through a simple convolutional neural network, the adversary can compute a similarity metric for any candidate pair of entry-side and exit-side flow. We conduct extensive hyperparameter tuning in order to arrive at attack parameters that are very effective against these packet flows on Nym.

We evaluate our shape-based flow correlation attack empirically on a large number of datasets from data collected on the deployed Nym network. In contrast to MixMatch's [3] contrived application use case, we collect flows from endpoints we set up that visit popular webpages using an actual web browser (Firefox). In addition to a baseline dataset

(default Nym, i.e., its current configuration without additional defenses), we generate datasets that capture the effects of (combinations of) five different countermeasure approaches. We thus offer an extensive evaluation of this attack across 85 testing scenarios, one of which represents default Nym and the rest capturing the settings of other attack-defense configurations. We find that shape-based flow correlation is fast and effective against downstream-unregulated Nym unless defenses are deployed with high overhead or in conjunction with other defenses. An essential component of our contribution is an extensive discussion offering recommendations to the Nym team for how to make the vantage position our attack requires even less likely to occur and which defenses to deploy in which configuration in order to effectively lower the attack's success. Our study has had real-world impact already, as after discussions with us the Nym team is planning to deploy (variations of) some of the countermeasures we evaluated.

This study is currently under submission as:

> L. Oldenburg, M. Juarez, E. Argones Rúa, and C. Diaz. "Shift Your Shape: Correlating and Defending Mixnet Flows Based on Their Shapes". Under submission to *IEEE Transactions on Dependable and Secure Computing* since March 31, 2025. 2025

I am the first author on the manuscript and have undertaken major parts of the conceptual, implementation, evaluation, and writing-up work that went into it.

Bibliography

[1] S. E. Oh, T. Yang, N. Mathews, J. K. Holland, M. S. Rahman, N. Hopper, and M. Wright. "DeepCoFFEA: Improved Flow Correlation Attacks on Tor via Metric Learning and Amplification". In: *2022 IEEE Symposium on Security and Privacy (SP)*. 2022, pp. 1915–1932. DOI: 10.1109/SP46214.2022.9833801 (p. 82).

[2] L. Oldenburg, G. Acar, and C. Diaz. "From "Onion Not Found" to Guard Discovery". In: *Proceedings of the 22nd Privacy Enhancing Technologies Symposium (PETS 2022)*. Vol. 2022. 2022, pp. 522–543. DOI: 10.2478/popets-2022-0026 (pp. 80, 81).

[3] L. Oldenburg, M. Juarez, E. Argones Rúa, and C. Diaz. "MixMatch: Flow Matching for Mixnet Traffic". In: *Proceedings of the 24th Privacy Enhancing Technologies Symposium (PETS 2024)*. Vol. 2024. 2024, pp. 276–294. DOI: 10.56553/popets-2024-0050 (pp. 81, 82, 83).

[4] L. Oldenburg, M. Juarez, E. Argones Rúa, and C. Diaz. "Shift Your Shape: Correlating and Defending Mixnet Flows Based on Their Shapes". Under submission to *IEEE Transactions on Dependable and Secure Computing* since March 31, 2025. 2025 (pp. 83, 84).

[5] L. Oldenburg and F. Tschorsch. "Strong Anonymity is not Enough: Introducing Fault Tolerance to Planet-Scale Anonymous Communication Systems". In: *Proceedings of the 16th International Conference on Availability, Reliability and Security*. ARES '21. Association for Computing Machinery, 2021. DOI: 10.1145/3465481.3469189 (pp. 79, 80).

[6] Y. Rubner, C. Tomasi, and L. Guibas. "A Metric for Distributions with Applications to Image Databases". In: *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. 1998, pp. 59–66. DOI: 10.1109/ICCV.1998.710701 (p. 80).

Chapter 6

Conclusions and Directions for Future Research

We return to the research questions stated at the beginning of this thesis to draw conclusions on the extent to which this thesis has contributed to advancing our understanding and end on ideas for future studies in anonymous communications involving both attacks and countermeasures.

6.1 Conclusions

For a large share of today's global population, there is no escaping using the Internet (not that we may want to). Accessing the web, messaging friends and coworkers, purchasing goods and services, and many other online activities have rapidly become essential features of everyday life. And while by now we, in many cases, have powerful tools at our disposal to ensure that the *contents* of our online activities are protected against access or tampering by unintended parties, the same can unfortunately not be said about their *network metadata*. Malicious actors with access to network metadata can easily acquire far-reaching insights about our online (and thus, offline) lives through the threat of *traffic analysis*. Access to content is not needed or even wanted, as exploiting network metadata using traffic analysis can often be the more effective strategy for the amount of resources an adversary is willing or able to expend (see quote by Prof. Edward W. Felten in Section 1.2). Expanding and strengthening our ability to *send and receive packets anonymously* is thus central to advancing our privacy online.

The work of this thesis was conducted against this backdrop. Specifically, the threat

of and defending against traffic analysis when we use *anonymizing networks* that unlink packet senders from packet recipients. The aim of the research that became this thesis was twofold: first, to understand better the limits of ANs when they are being attacked or experience failures (which can be caused intentionally by an adversary) and second, to explore the remedies available to AN designers to alleviate the impact these attacks and failures have on the users of their systems. We approached these objectives by taking on the role of such adversary or fault injector across different AN implementations and empirically assessing how well the system under inspection coped, either in terms of resilience to failures or in ability to protect user anonymity under attack.

Concretely, we started out with an analysis of the tolerance of three different AN implementations to network failures. This thesis' first research question asked *how network failures degrade the usability of anonymizing networks* and our study [8] in Chapter 7 contributes to the answer by subjecting three AN implementations to various amounts of packet loss and delay while measuring their ability to route packets according to expectation. By contrasting a frontier mixnet implementation and a frontier *computational private information retrieval (CPIR)* implementation with a prototype mixnet we developed with straightforward fault tolerance measures built in, we show the benefits that increased resilience to failures brings for relaying packets with end-to-end latencies that are similar to the ones when no failures occur. Our argument is that a lack of fault tolerance and thus unpreparedness for when problems inevitably arise may degrade an AN's anonymity because some users may leave for less private means of communication, leaving behind users that cannot risk using less private networks and are thus forced to cease communicating entirely.

While our assumed scenario admittedly is bleak and thus may appear unreasonable, we note two observations. First, while failures can occur for benign, random reasons, they may just as well be caused intentionally by a (strategic) active adversary. The adversary can either use failures (e.g., dropping or delaying packets or crashing infrastructure nodes on purpose) in a targeted way to steer users to certain paths or nodes or frustrate them to the point of resignation, or they may also force the entire AN offline (akin to a DoS attack) if they occupy or can target bottleneck positions. Second, fault tolerance should not be considered only a virtue to have in crises, as the same principles enabling a system to absorb the loss of entire network links or topology members equally contribute to hiding small problems such as temporary congestion or node restarts to users. From the users' perspective, a resilient AN offers performance they can predict and thus rely on, which may help retain them and even attract new users, thus increasing the AN's anonymity by growing anonymity sets. We advise to build in fault tolerance mechanisms early on in AN design and recommend comparing deviation from baseline performance in terms of packet end-to-end latencies as a useful metric for assessing user-perceived performance of an AN as well as its ability to cope with failures.

In order to meaningfully contribute to raising online privacy, an AN offering anonymous packet routing to Internet users must be able to scale well and accommodate many

different application use cases so that anonymity sets can be large. Two strains for how to design ANs have thus far shown promise or even already proved their capability for practical large-scale deployment: connection-oriented and low-latency onion routing networks (with prominent deployed implementation Tor [3]) and packet-oriented and medium- to high-latency mixnets (with prominent deployed implementation Nym [2]). The remainder of this thesis has focussed on assessing threats and defenses on these two AN approaches, with the goal of improving the anonymity they can offer based on recommendations grounded in experiments.

Guards play a vital role in protecting Tor users from attacks known to be effective against connection-oriented ANs that are vulnerable to timing analysis (such as Tor). If the adversary was able to occupy the first and last hop on connections users build through these ANs, they would easily be able to link the two ends and thus deanonymize users. To protect against this threat, Tor users rely on guards, which are relays users select and fix for many weeks and use as their first hop on any connection they build. If their guards are honest, users are thus immune to these attacks as well as path rebuildings that an adversary may induce in order to steer users to their malicious guards. However, guard discovery now becomes a risk, i.e., attacks that allow an adversary to identify a target user's guard. Once identified, the adversary can focus their effort on compromising or coercing just this single Tor relay, after which they will be able to obtain the Tor user's IP address. Consequently, our second research question asks *how vulnerable Tor users are to guard discovery and how they can be protected*. Our study in Chapter 8 presents a novel guard discovery attack on Tor users that access webpages with Tor Browser and evaluates defenses to make it less likely and less effective.

We find that the guard discovery attack we describe is indeed fast, stealthy, and affordable, such that it presented a real risk to Tor users and could have been deployed by practical adversaries. Our attack assumes that the adversary somehow controls parts of a target webpage that Tor Browser users visit. Additionally, the adversary needs to operate a number of Tor relays such that they control a small part of Tor's distributed HSDir database and a target fraction of Tor's relay bandwidth. At the time of writing the study [5], we estimated that 257 relays would be required to approximately contribute five percent of Tor's relay bandwidth. The adversary uses these relays to be selected as the second hop on the circuits that our attack targets, i.e., next to the first hop, which may be the target user's guard.

Crucially, our attack does not require complex or expensive classifiers to detect the signal revealing a user's guard. Instead, it exploits tracking the cell pattern on targeted circuits and thus neither requires a training phase nor much computational power to run. We have disclosed our findings to the Tor developers responsibly. As of Tor in version `0.4.7`, Vanguards-Lite[1] is enabled by default, which limits the impact of

[1] `https://web.archive.org/web/20250319234942/https://spec.torproject.org/vanguards-spec/vanguards-lite.html`

our attack. Taking a step back, it is clear that our guard discovery attack represented but one vector for how to conduct it. It is likely that many more exist or may get introduced inadvertently as Tor is complex and constantly evolving. A systematic approach to identifying, classifying, and disabling potential vectors for attacks such as guard discoveries is indispensable and Tor developers have already laid the foundation for this effort.[2]

As this thesis' final component, we have assessed the threat of passive flow correlation attacks against mixnets. Flow correlation is a mainstay in the field of anonymity attacks on mixnets. It concerns the passive (or silent) linking of arriving to leaving packet flows at a mixnet's edge—the boundary between the mixnet's users and the mixnet's infrastructure nodes (e.g., mixes)—based on packet direction, timestamp, and size. While synchronous batching mixnets prescribing all endpoints to send with the same rate may offer resistance to brief flow correlation due batches containing the same number of packets per endpoint, they are also limited in applicability due to their typically low throughput and high delays. In contrast, asynchronous mixnets (e.g., continuous-time mixnets) may offer less resistance to flow correlation attacks but can be configured to accommodate more application requirements and thus become actually practicable tools. Nym [2] is a state-of-the-art and deployed implementation of a continuous-time mixnet, readily available to end users. This prompts us to ask *how resistant Nym is against passive flow correlation attacks and how its resistance can be increased further*, differentiating on which traffic directions are protected using Nym's Poisson-governed traffic shaping.

In the case that packets in *both directions* (from endpoints to Nym's network-requester services and back) are shaped according to Poisson processes, characteristic features of packet flows (e.g., bursts, silences) are substantially obfuscated and thus of diminished use for flow correlation. Nym once implemented this approach to traffic shaping and we evaluate this version of Nym in our study [6] in Chapter 9. We find that despite the absence of traffic features such as bursts or silences, the adversary is able to leverage the difference between the timestamps when packets leave the mixnet and the timestamps when they enter it to match communicating endpoints. This is because this difference follows a stationary distribution only for flows that are indeed matched and drifts away from this distribution when they are unmatched. We build effective adversarial classifiers that demonstrate the viability of our attack, assuming our strict assumptions on adversarial visibility and network conditions hold. When these assumptions are violated, our attack classifiers degrade in performance, and we offer countermeasure proposals based on this fact.

Finally, we also obtain empirical results for the case that *one direction of traffic is not shaped at all*. Nym recently moved to this model for their network-requesters, which now send packets in the direction of users as quickly as possible, i.e., not shaped according to

[2] `https://web.archive.org/web/20250422155041/https://spec.torproject.org/proposals/344-protocol-info-leaks.html`

a Poisson process at all anymore. This has repercussions for Nym's susceptibility to passive flow correlation, as now above mentioned traffic features such as bursts and silences filter through the mixnet and can be exploited by the adversary to link senders to recipients. We comprehensively evaluate this research question in our study [7] in Chapter 10, both in terms of optimizing the adversarial strategy as well as optimally defending against this attack. Our results demonstrate clearly how effective a tool for linking flows their shape is, when no additional protections against flow correlation are deployed on Nym. However, as our extensive evaluation of potential defenses shows, the adversary's success at correlating flows can be lowered, even substantially if Nym were to deploy multiple defenses at the same time and at considerable overhead. As of the time of writing, the Nym team is in the planning stages to deploy some (variations of) the defenses we discussed with them based on our study.

As an overall conclusion, repeated communication between endpoints is incredibly hard to hide (harder still if the adversary controls one of the involved endpoints), at least at levels of overhead compatible with sustainable operation at Internet scale and for many different applications. This is one general takeaway from the studies presented in this thesis. It is not a new insight—the attacks we investigate have been conceived of many years ago and have been studied by many researchers—but our empirical results offer timely and forceful reminders how difficult a task it is to hide a long signal in background noise.

We find that our empirical results complement and underscore theoretical results about these issues well. The former through implementing and evaluating actual attacks that could be deployed by realistic adversaries and thus have the potential to serve as impetus for AN designers to enhance their systems. And the latter by putting numbers to risks formulated in the abstract that turn factors into "this many seconds until fifty percent of the targeted users are compromised" or "deploying this defense lowers the system's vulnerability by that many percentage points". In the process, we have placed emphasis on making source code and datasets used to arrive at our results publicly available in the hopes that others may find them useful, critically explore them, and possibly even take them further. We are delighted to see the impact our work has had already in that it has spurred discussions with teams from prominent AN implementations and in some cases already contributed to changes being deployed.

6.2 Future Research Directions

We conclude Part I by pointing to future research directions we deem worth exploring.

Network-Level Passive Flow Correlation on Nym. Both of our works assessing passive flow correlation against Nym have been conducted from an insider position of the adversary: we either assumed them to operate some gateways (Chapter 9) or

some gateway-requesters (Chapter 10). This is advantageous for the adversary as it leads them to observe packet metadata at the Sphinx packet level of the mixnet, which is free of the artifacts resulting from converting Sphinx packets to network-link-level packets. If the adversary additionally controls one of the involved endpoints (e.g., the gateway-requester adversary we assumed in Chapter 10), they are able to tell which of the ongoing flows a received packet belongs to. Thus, it remains an open question which level of passive flow correlation performance a network-only (external) adversary would be able to achieve against Nym. It is important that future research investigates this question, as it is crucial to our understanding of the threat these attacks pose.

Flow Correlation on Nym by Active Adversaries. Equally important is the analysis of active adversaries conducting flow correlation against Nym users. Active adversaries use their modification capabilities to induce patterns on flows entering the mixnet that make it even easier for them to recognize the packet flows when they exit the mixnet again. Targets of active attacks range from individual flows to the entire network (e.g., DoS). The adversary risks getting detected if they turn active, as the attack is not covert anymore. This opens up the possibility to devise strategies for identifying active adversaries and design exclusion or punishment strategies to disincentivize attacks and protect users. However, it remains an open question how detectable active attacks against mixnets such as Nym really are and it is crucial to systematically evaluate this threat.

Evaluation of Nym Hardened by Deployed Defenses. Once Nym has deployed some of the countermeasures discussed in our work [7], a reevaluation of passive flow correlation is in order. While we tried to accurately simulate the effects that the proposed defenses would have on flows, their real effects once implemented and deployed on Nym may bear out differently and should be assessed. In particular, countermeasures such as splitting flows across multiple entries may be sensitive to the exact parameters chosen for them by the mixnet's developers, while splitting flows across multiple exits is impossible to realistically simulate at all by only post-processing collected flows. When conducting this reevaluation, it may be a good idea to evaluate defense configurations beyond the ones we covered along with it, e.g., which parameter for the number of consecutive packets to send through which entry optimally defends against the attack. Additionally, future work should investigate further the combinations of defenses, as this has shown promise in our work. The goal should be to identify the configuration of multiple defenses that leads to the largest aggregate protection effect.

Systematic Comparison of Flow Correlation Across Onion Routing, Continuous-Time, and Batched Mixnets. We now have deployed implementations of major AN concepts available and it would be useful to empirically compare their resistance to flow correlation. We have Tor [3] implementing onion routing, Nym [2] implementing continuous-time mixing, and, e.g., xx network based on cMix [1] implementing synchronous mixing. A future study should fix as many variables during data collection as possible (e.g., application use case, user behavior, geographical location of users) and collect a very large and representative dataset for each AN. All assignments for variables

influencing a data collection should be tested: idealized network (no propagation delay and no loss) versus deployed live network, individual anonymity components present or disabled (e.g., mixing delays set to zero, loop cover packets enabled or disabled in Nym, cover packets enabled or disabled in xx network) or differently configured (e.g., lower or higher mixing delay variance than default, longer packet paths, no traffic shaping at all, stricter traffic shaping). The goal would be to arrive at one table comparing the performance of adversaries of varying reach and capability at conducting passive and active flow correlation against these three frontier AN implementations.

Systematically Investigate Cover Traffic Schemes. Cover traffic plays an essential role in enabling an AN to realize its anonymity properties. However, the design space for cover traffic is huge. A future study could make an effort to comprehensively classify and analyze a broad range of cover traffic schemes under several adversary models and for varying AN designs. Some questions that arise are: which entities should send cover packets (endpoints, mixes, interface nodes), where should cover traffic terminate (endpoints, mixes, interface nodes), how much do they contribute to passive or active flow correlation protection, and how do we optimize cover traffic under external constraints such as bandwidth budget or limited knowledge of recipient addresses?

Continue Attacking and Defending ANs with New Methods. There is value in advancing the cat-and-mouse game between attacks on and defenses to traffic analysis attacks. We see this particularly pronounced for the website fingerprinting threat on Tor, where the back-and-forth between researchers and Tor developers as well as new attacks and adapted defenses contribute to the ever-evolving understanding of Tor's vulnerability to this type of attack. Such work is essential for real-world privacy: we need to understand where adversaries of various shapes and forms stand *today*, i.e., with the tools—such as continuously advancing machine learning methods—available today. Only if we somewhat accurately understand how well capable adversaries perform at breaking our systems' guarantees, can we take measures to improve them and thus their anonymity guarantees. Future research should continue assessing the current state of the art for both sides of traffic analysis, potentially guided by methods such as *neural architecture search* to take advantage of the full toolset of machine learning architectures [9].

Provide Foundations for Traffic Analysis Research by Collecting High-Quality Datasets. Datasets serve as the lower layers of traffic analysis attacks and defenses. If they are of bad quality, any result drawn from them is meaningless. Ensuring and advancing the quality of datasets by identifying quality criteria, closeness to real-world use cases, unifying data collection methods across ANs, and validating datasets is critical for the higher-level works that build on them. Efforts are already underway for website fingerprinting on Tor [4] and should also be undertaken for other domains and ANs.

Systematization of Knowledge: Attacks and Defenses on Mixnets with Guidelines for Future Mixnet Designers. The design space of mixnets is vast and much progress has

been made across all of its dimensions over the past twenty years. We know better today how best to design mixnets, be it for particular use cases or at desired levels of traffic analysis resistance or both. Attacks that used to be threats may not be threats anymore today (or may be particularly easy to defend against), e.g., replay and tagging attacks. And defenses that may have been promising once might not be anymore today. However, an up-to-date, coherent, and accessible understanding of the attacks and defenses on the various approaches for how to realize a mixnet is missing for (aspiring) mixnet designers in 2025. A future study could close this gap and provide such systematization of knowledge paper from the perspective of today's knowledge.

Bibliography

[1] D. Chaum, D. Das, F. Javani, A. Kate, A. Krasnova, J. De Ruiter, and A. T. Sherman. "cMix: Mixing with Minimal Real-Time Asymmetric Cryptographic Operations". In: *Applied Cryptography and Network Security*. Ed. by D. Gollmann, A. Miyaji, and H. Kikuchi. Cham: Springer International Publishing, 2017, pp. 557–578. ISBN: 978-3-319-61204-1 (p. 92).

[2] C. Diaz, H. Halpin, and A. Kiayias. *The Nym Network*. Tech. rep. Version 1.0 as of February 26, 2021. Nym Technologies SA, Feb. 2021. URL: https://web.archive.org/web/20250403044143/https://nym.com/nym-whitepaper.pdf (pp. 89, 90, 92).

[3] R. Dingledine, N. Mathewson, and P. Syverson. "Tor: The Second-Generation Onion Router". In: *13th USENIX Security Symposium (USENIX Security '04)*. USENIX Association, 2004 (pp. 89, 92).

[4] R. Jansen, R. Wails, and A. Johnson. *A Measurement of Genuine Tor Traces for Realistic Website Fingerprinting*. Tech. rep. 2024. DOI: 10.48550/arXiv.2404.07892 (p. 93).

[5] L. Oldenburg, G. Acar, and C. Diaz. "From "Onion Not Found" to Guard Discovery". In: *Proceedings of the 22nd Privacy Enhancing Technologies Symposium (PETS 2022)*. Vol. 2022. 2022, pp. 522–543. DOI: 10.2478/popets-2022-0026 (p. 89).

[6] L. Oldenburg, M. Juarez, E. Argones Rúa, and C. Diaz. "MixMatch: Flow Matching for Mixnet Traffic". In: *Proceedings of the 24th Privacy Enhancing Technologies Symposium (PETS 2024)*. Vol. 2024. 2024, pp. 276–294. DOI: 10.56553/popets-2024-0050 (p. 90).

[7] L. Oldenburg, M. Juarez, E. Argones Rúa, and C. Diaz. "Shift Your Shape: Correlating and Defending Mixnet Flows Based on Their Shapes". Under submission to *IEEE Transactions on Dependable and Secure Computing* since March 31, 2025. 2025 (pp. 91, 92).

[8] L. Oldenburg and F. Tschorsch. "Strong Anonymity is not Enough: Introducing Fault Tolerance to Planet-Scale Anonymous Communication Systems". In: *Proceedings of the 16th International Conference on Availability, Reliability and Security*. ARES '21. Association for Computing Machinery, 2021. DOI: 10.1145/3465481.3469189 (p. 88).

[9] P. Singh, S. Arun Naik, N. Malekghaini, D. Barradas, and N. Limam. "A First Look at Generating Website Fingerprinting Attacks via Neural Architecture Search". In: *Proceedings of the 22nd Workshop on Privacy in the Electronic Society*. WPES '23. Copenhagen, Denmark: Association for Computing Machinery, 2023, pp. 173–178. ISBN: 9798400702358. DOI: 10.1145/3603216.3624961. URL: https://doi.org/10.1145/3603216.3624961 (p. 93).

Part II

Publications

Chapter 7

Strong Anonymity is not Enough: Introducing Fault Tolerance to Planet-Scale Anonymous Communication Systems

Publication Data

L. Oldenburg and F. Tschorsch. “Strong Anonymity is not Enough: Introducing Fault Tolerance to Planet-Scale Anonymous Communication Systems”. In: *Proceedings of the 16th International Conference on Availability, Reliability and Security*. ARES ’21. Association for Computing Machinery, 2021. DOI: 10.1145/3465481.3469189

Contributions

Principal author.

Strong Anonymity is not Enough: Introducing Fault Tolerance to Planet-Scale Anonymous Communication Systems

Lennart Oldenburg,[1] Florian Tschorsch[2]

[1] *imec-COSIC KU Leuven, Belgium*
[2] *DSI, TU Berlin, Germany*

Abstract. Current Anonymous Communication Systems (ACS) lack fault tolerance and thus risk becoming unavailable when failures occur, forcing users offline or to less private messengers. In this work, we evaluate end-to-end message transmission latencies and resource demands of state-of-the-art mixnet Vuvuzela and CPIR system Pung under different network failure scenarios on an ACS test bed across four continents. We compare Vuvuzela and Pung to proof-of-concept mixnet FTMix that we equip with simple fault tolerance measures. Our analysis shows that FTMix maintains the smallest divergence of end-to-end latencies under failures from their respective baseline among all three ACS, while also achieving a balanced resource consumption trade-off. Thus, we consider fault tolerance effective in ensuring service availability and a crucial design principle for future ACS proposals.

7.1 Introduction

Messaging services with content confidentiality from *end-to-end encryption* already have more than 2 B users [47, 49]. Anonymous Communication Systems (ACS) that additionally protect *communication metadata*, however, are not yet widely adopted. ACS prevent powerful adversaries that collect and analyze metadata from linking senders to recipients. For this work, we focus on *message-oriented* ACS providing *strong anonymity*. To achieve scalability, ACS typically differentiate participants into clients that converse anonymously and servers that route their messages. Purely peer-to-peer ACS have proven incapable of reaching large network sizes [4, 51].

Benign or malicious failures in an ACS (e.g., crashed or slow processes, lost or delayed messages, broken network links) can have devastating impact on its *availability*. At worst, they render an ACS unable to deliver messages and force users offline or back to less private messengers. In *zerotrust* systems, all servers can be compromised and still guarantee anonymity. Computational Private Information Retrieval (CPIR) [30] systems such as Pung [3] work in this setting. However, this favorable trust property

comes at significant resource costs. *Anytrust* systems, where at least one server per path is assumed honest, attain lower and more balanced resource costs. This suffices to unlink senders from recipients in mixnets [11], the best-studied anytrust anonymity approach. Surprisingly, existing ACS rarely address unavailability issues.

In this work, we show the negative impact of network failures on the ability of message-oriented ACS to relay user messages according to expectation. We do this by evaluating the end-to-end message transmission latencies and resource demands of state-of-the-art mixnet Vuvuzela [26] and CPIR system Pung [3]. To demonstrate that simple fault tolerance measures improve user-experienced performance expressed as end-to-end latencies, we compare them to proof-of-concept mixnet FTMix that we equip with basic fault tolerance techniques. FTMix achieves 3.9× and 13.15× lower mean First Wasserstein distances between end-to-end latencies from the baseline and failure scenarios, and a balanced resource demand trade-off across clients and servers. This shows that simple measures promise to prevent ACS end-to-end latencies from becoming erratic or even unusable once failures occur.

We make the following contributions:

- We evaluate two state-of-the-art ACS, Vuvuzela and Pung, and a fault-tolerant proof-of-concept mixnet, FTMix, regarding their end-to-end latencies and resource demands in an ACS test bed spanning four continents.
- We find that FTMix achieves end-to-end latencies closest to expectation when failures occur and a promising resource demand balance that scales well with a growing user base.
- We raise the issue that neglecting fault tolerance when designing strong anonymity services leads to a negative impact on user-experienced performance, and encourage system designers to make fault tolerance a priority.

7.2 Anonymity Foundations

We introduce the foundations for strongly anonymous communication with a focus on their susceptibility to failures.

DC-nets. In DC-nets [10], one genuine message is transmitted per round in a full-mesh topology with information-theoretic anonymity through XOR of all broadcast messages. Each node broadcasts the XOR of a round's secret random values it has established with each other node. The designated sender additionally includes the genuine message. Modern DC-nets [4, 14, 15, 51] have mostly traded off the original zerotrust peer-to-peer setting for larger network sizes by adopting client-server architectures. The all-to-all communication requirement of the DC-net core makes systems based on them prone to

unintentional and intentional disruption, particularly at scale. We thus do not further consider DC-nets for our subsequent planet-scale evaluation.

Mixnets. In mixnets [11], users send their messages through a sequence of *mixes* that obfuscate the relationships between incoming and outgoing messages by reordering them. As long as at least one mix per taken path is honest and keeps its message shuffling secret, mixnets achieve their anonymity guarantee (anytrust setting). In *threshold* and *timed* mixes, mixes batch messages until a criterion is met, remove one layer of encryption, shuffle the batch, and forward the messages. In *continuous-time* mixes [29, 43], messages are delayed individually instead of in batches, enabling low-latency designs. To protect against long-term intersection attacks, clients may achieve constant-bandwidth traffic by sending *cover messages* when no genuine ones are available [6]. Selecting messages to forward from a *pool* across multiple rounds and additional cover traffic [20, 21, 44, 46] may help thwart *blending attacks*. Heartbeat messages [18, 43] and verifiable shuffles [5, 39, 50] are effective against active attacks. Finally, the topology in which mixes are arranged determines how well a mixnet scales, its potential for fault tolerance, but also how prone its users are to epistemic attacks. Commonly, mixnets either adopt *cascade*, *stratified*, or *free-route* topologies [7, 8, 17, 23]. In contrast to DC-nets, failures in a part of a mixnet do not necessarily prevent delivery of messages in the rest of the network, depending on the mixnet's design specifics.

PIR. Two variants of PIR exist that differ in the assumptions made to enable anonymous communication. In the *information-theoretic* variant (ITPIR [13]), clients are able to privately query a database maintained by two or more servers with information-theoretic privacy if at least one server is honest. An optimally Byzantine-robust scheme exists for ITPIR [19]. However, if *computational assumptions* are made, PIR can also be used if only a single, untrusted server is available (CPIR [30]). Highly optimized schemes exist in prototypes [1, 2, 3], but CPIR typically still incurs significant computation and communication overhead over ITPIR. With a single to a handful servers between sender and recipient in either PIR variant, failures have even less opportunities to disrupt correct service operation for users of a PIR-based ACS.

7.3 ACS Selection and Configuration

We now introduce the three ACS we evaluate in Section 7.4. We select mixnet Vuvuzela [26] and CPIR system Pung [3], because they represent state-of-the-art designs of their respective ACS foundation and publish their source code. We also outline the simple fault tolerance components in our proof-of-concept cascade mixnet, FTMix. We use FTMix as a benchmark for how ACS with fault tolerance at their core perform when failures occur. Our ACS test bed in Section 7.4 is built on the Google Cloud Platform (GCP) [25], machine types below are thus with regard to GCP's naming scheme.

Vuvuzela. Mixnet Vuvuzela [26] operates on a single cascade and achieves 68,000 relayed messages per second for one million users and 37 seconds of end-to-end latency. We use a fork of Vuvuzela with optimizations for experiment scripting and removed Alpenhorn dialing [36] for reduced resource demands. Vuvuzela's servers are four `n1-standard-4` instances (four threads, 15 GB RAM), one entry server accepting messages for 20 seconds each round and three mixes, creating a cascade of length seven.

Pung. CPIR system Pung [3] is a key-value store that provably hides accesses by users who deposit and retrieve messages. Grounded in CPIR, Pung upholds its privacy guarantees even if all servers are compromised, thus enabling single-server setups. We deploy a fork of Pung minimally adjusted for experiment scripting and configure it for one-to-one messaging with one message sent and retrieved per round. Pung's server runs as one process (`-n 1`) with ten worker threads (`-w 10`) handling equal numbers of message. To balance the multiple instances deployed for FTMix, Pung's server is of instance type `n1-highmem-16` (16 threads, 104 GB RAM). It is set to single retrievals (`-k 1`), explicit download of label-to-index mappings (`-t e`), and PIR depth of two (`-d 2`).

FTMix. We build simple fault tolerance measures into a cascade-based, pool-style mixnet basis in order to evaluate how effective and costly straightforward countermeasures to failures in distributed systems are. In Section 7.4, we deploy FTMix's servers as three cascades with seven mixes each (21 total). Each is of type `n1-standard-4` (four threads, 15 GB RAM). Entry mixes are configured to collect messages for 20 seconds each round. We achieve robustness against Byzantine failures through *redundancy in space* (e.g., operating backup nodes in case primaries fail) and *time* (e.g., masking failures through transparent retries). We note Forward Error Correction algorithms as a promising future avenue to integrate fault tolerance into ACS. We now briefly describe each fault tolerance building block in turn.

If any one mix on a cascade in a cascade-based mixnet *fails or becomes too slow to operate*, the whole cascade stops routing messages as they have to go through each mix on the cascade. Thus, we extend the typical single mix cascade to a cascades matrix with three independent cascades that work in parallel. Clients send genuine and cover messages in an appropriate cryptographic packet format such as Sphinx [16] to each cascade each round, using fresh key material in every instance. If process failures incapacitate all but one cascade, the remaining one transmits all messages—without degradations in end-to-end latency. These resources on redundant paths gain nothing towards throughput. Other topologies with multiple independent paths may improve throughput, but require all honest mixes to be in the same position across all paths.

In case a message is *lost or unusably delayed* in a cascade-based mixnet, much time will pass until an acknowledgment timeout fires and a retransmission is requested due to the already long transmission times. This inherent behavior might force users to divert to less anonymous services. We increase the probability for each genuine transmission

to succeed to counter this effect of network troubles. Each client sends each genuine message through all cascades for two consecutive rounds, without the protocol-layer application being aware of this. Across three independent cascades and two consecutive rounds, each genuine message is thus sent six times, of which FTMix can tolerate the loss of up to five attempts.

7.4 Evaluation

We evaluate the end-to-end latencies and resource demands in terms of CPU and RAM usage of Vuvuzela, Pung, and FTMix under three failure scenarios. We deploy the three ACS on GCP to capture the hardware and ecosystem influence on each ACS as realistically as possible. We analyze how well each ACS scales with more clients and whether an ACS is able to maintain service expectation when failures take hold. We make the ACS test bed [40] and evaluation results [41] publicly available.

7.4.1 Failure Scenarios

Each ACS is placed under the following three scenarios.

(1) No impediments. Baseline scenario to showcase performance of the evaluated ACS when no failures disturb its operation. Due to planet-scale deployment, we already capture real-world latency between nodes.

(2) High message delay. Using `netem tc`, we emulate the effects of worst-case network complications, e.g., corrupted or congested links. We delay packets from clients in three of the 17 used regions by millisecond durations sampled from $\mathcal{N}(400, 10{,}000)$ and drop 1 % of all packets with 25 % correlation on the previous drop decision to mimic loss bursts. On servers from one region, we delay packets according to $\mathcal{N}(100, 625)$ with no packet loss.

(3) High message loss. In this second worst-case scenario, we raise the packet drop probability to 3 % on affected clients (same correlation), while packet delays follow $\mathcal{N}(100, 2{,}500)$. Servers in the one affected region experience 2 % packet loss with 25 % correlation and no delay applied to outgoing packets.

7.4.2 Metrics

The following three metrics determine the performance of an ACS.

Traffic volume. By querying a node's `iptables` for the ACS ports every second, we collect the number of bytes transferred to and from each node. We establish the *goodput overhead* of an ACS, i.e., the traffic volume on top of the content bytes for a message to reach its destination. This enables us to compare network efficiency among ACS and allows nodes to decide whether ACS participation is within budget. We offset the measurements by the value observed at the end of a warm-up phase.

Computational and memory load. In order to determine the lower bound on hardware requirements for participation, we extract busy CPU percentage and in-use RAM from the `proc` filesystem each second. Additionally, CPU and RAM usage on server side indicate the maximum number of users an ACS should be able to support and how demand develops when user bases grow.

End-to-end message transmission latencies. We record end-to-end latencies of a configured number of messages between two clients after a warm-up phase with send and receive UNIX timestamps in nanoseconds. We use the divergence of end-to-end latencies of an ACS in scenarios (2) and (3) from its latencies in baseline scenario (1) as a measure of its tolerance to network failures and thus user-perceived performance.

7.4.3 Experimental Setup

For each ACS and each failure scenario, we have 1,000, 2,000, and 3,000 clients exchange messages while measuring our metrics of interest. We assign ten logical clients to one physical instance, and distribute the resulting 100, 200, and 300 machines uniformly at random across 17 GCP regions on four continents. Each instance is of type `n1-standard-4` (four threads, 15 GB RAM), runs Ubuntu 19.04 with Linux kernel 5.0.0, and has NTP-synchronized clocks. End-to-end latencies are measured between pairs of clients on the same physical machine to avoid comparing timestamps across different machines. Excluding a warm-up phase of three messages, all results are based on 20 consecutively sent and received messages of 256 B payload each. Each scenario is evaluated over three runs, except for two runs of Pung in scenario (2) with 3,000 clients that were unusable and could not be reattempted.

7.4.4 Results

We show evaluation results for traffic volume and server-side resource demands in Figure 7.1, and for end-to-end message transmission latencies in Figure 7.2 and Table 7.1. Both figures have common x-axes across their respective subplots, the three user bases for Figure 7.1 and end-to-end latencies in seconds for Figure 7.2.

Traffic volume. We depict the average highest traffic volume required to send and receive 20 consecutive messages in mebibytes (MiB) per logical client on a y-axis with

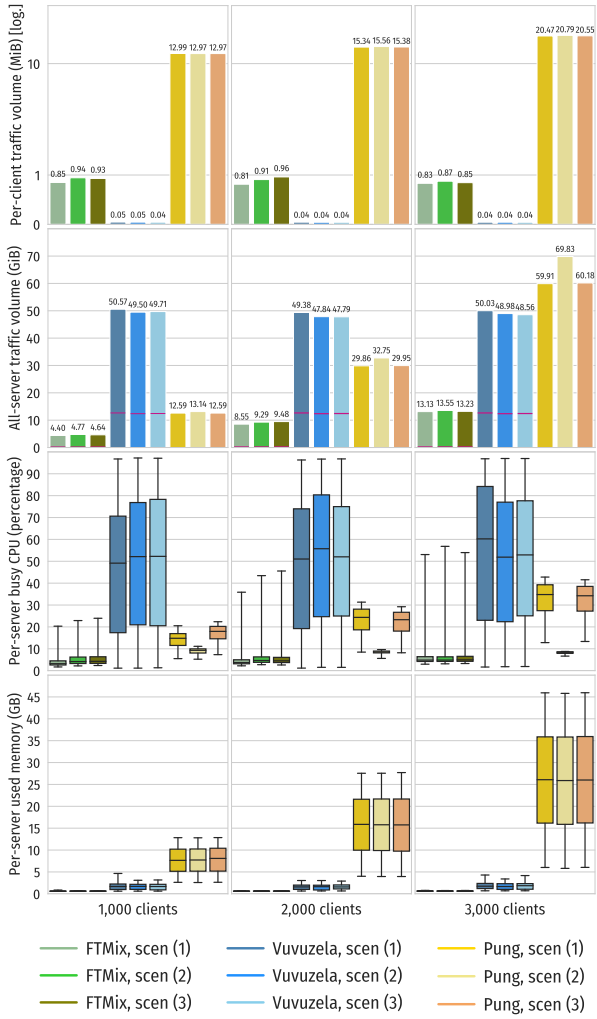


Figure 7.1: Sending and receiving 20 consecutive messages; from top to bottom: (a) average highest traffic volume per logical client, (b) all-server total average highest traffic volume, (c) per-server CPU usage, (d) per-server RAM usage.

logarithmic scale in Figure 7.1a and multiplied by the number of servers in each ACS in gibibytes (GiB) on a y-axis with linear scale in Figure 7.1b. As reference, the goodput of these transmissions amounts to $20 \cdot 2 \cdot 256\ \text{B} < 0.01$ MiB. On clients, Vuvuzela achieves impressively low values (0.04–0.05 MiB) due to lack of client-addressed cover traffic and

relatively small layered-encryption overhead. Clients in FTMix require up to 24× of Vuvuzela's traffic volume, while still performing about an order of magnitude better than Pung clients. We observe how network failures in scenarios (2) and (3) in FTMix lead to a slightly increased traffic volume over scenario (1). Vuvuzela's and Pung's client traffic volumes exhibit relatively small variance across their three scenarios. Interestingly, while FTMix's and Vuvuzela's client traffic volumes do not increase with the number of clients, Pung clients require further 2.45 and 5.18 MiB when scaling to 2,000 and 3,000 clients, respectively. If persistent, this trend might become a serious concern at real-world network sizes.

On server side, we show *total* server traffic volume per setting, i.e., the per-server average highest volume multiplied by the number of servers (per-server averages shown as red lines). Across all scenarios and user bases, FTMix achieves the lowest traffic volumes (averages: 0.21–0.65 GiB). For Vuvuzela, mix cover traffic sampled from `Laplace`(300,000, 13,800) to maintain the target Differential Privacy level over 250,000 rounds means that traffic volume always exceeds 47 GiB (averages: 11.95–12.64 GiB). However, this value is likely to remain at this level for many thousands of additional clients. Introducing packet delay or loss curiously causes traffic volume to decrease, likely due to timeouts and fewer send events. Pung's single server reaches the overall highest traffic volume when 3,000 clients participate (69.83 GiB). Judging from traffic volume growth when we increase user bases, traffic volumes at real-world network sizes again appear like a potential problem. Packet delay seems to impact Pung's server significantly stronger than packet loss, ever more so the more clients participate.

Resource demands on servers. Figure 7.1c depicts computational load per server as a percentage, while Figure 7.1d displays absolute memory usage per server in GB. Low and high whiskers comprise all observations. Vuvuzela's mixes utilize their CPUs close to capacity in some cases, due to considerable message volume from cover traffic (medians: 49–60 %). Pung's busy CPU percentage increases with the number of users (medians: 14–34 %), except for in scenario (2) when high packet delays are introduced. Then, we see 8.5–9.5 % median busy CPU, likely due to more time spent waiting for transmissions. FTMix's busy CPU medians remain fairly low (3.2–5.4 %), save for entry mixes that handle many client connections and in a few cases reach close to 58 %. Median memory usage on FTMix and Vuvuzela across all measurements is very low, 0.59–0.62 GB and 1.57–1.85 GB, respectively. Pung's median RAM usage increases steadily across the three user bases again (up to 26 GB). We do not show client-side CPU and RAM measurements here, as all ACS exhibited values that appear suitable for deployment on entry-level hardware.

End-to-end message transmission latencies. End-to-end latencies in seconds are shown as CDF plots in Figure 7.2. The x-axis range was limited to the overwhelming majority of all measurements. Latencies reach 113.64 s (Pung), 319.41 s (FTMix), and 378.52 s (Vuvuzela). Unfortunately, 0.00032 % of all Pung latencies are negative. While physically impossible, they are negligible in quantity and we accept this error. In order

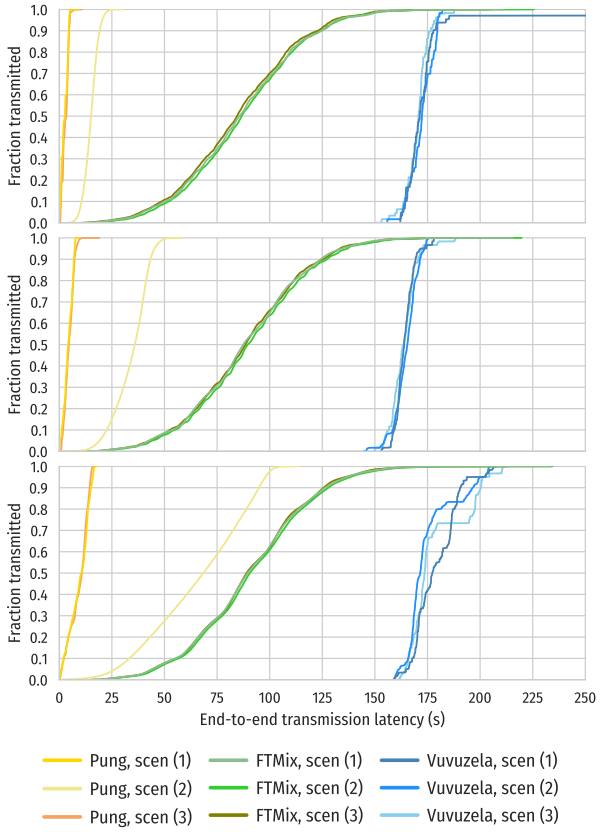


Figure 7.2: End-to-end latencies of sending and receiving 20 consecutive messages between pairs among 1,000 (top), 2,000 (middle), and 3,000 (bottom) clients.

to make deviation of end-to-end latencies from baseline once failures are introduced comparable across scenarios and systems, we list the First Wasserstein distances (W_1) between scenarios (1) and (2) and scenarios (1) and (3) in Table 7.1 per system and user base. Due to different numbers of observations across systems, scenarios, and user bases, we use each setting's respective probability density function values at the same number of bins of equal width and range to compute each W_1 distance. All distances are listed as factors of 10^{-5}, and we also include the per-system mean of its respective six distances to enable cross-system comparisons.

Pung achieves impressively low absolute latencies when no failures occur and few users participate—a clear advantage of single-server setups. However, Pung appears to struggle with packet delay, where latencies diverge from the unimpeded baseline significantly,

Table 7.1: First Wasserstein distances (W_1) between pairs of probability density function values at the bins of end-to-end latency histograms under scenarios (1) and (2) and scenarios (1) and (3), as factors of 10^{-5}. FTMix achieves the lowest mean W_1 across scenario pairs and user bases, indicating its fault tolerance measures succeed at maintaining consistent end-to-end latencies for client messages.

System	**Scenarios** (compared)	**Clients** (number)	$\mathbf{W_1}$ ($\times 10^{-5}$)	**Mean** ($\times 10^{-5}$)
FTMix	(1) and (2)	1,000	10.970	**13.037**
		2,000	16.226	
		3,000	7.860	
	(1) and (3)	1,000	8.545	
		2,000	15.847	
		3,000	18.777	
Vuvuzela	(1) and (2)	1,000	27.699	**50.877**
		2,000	65.110	
		3,000	38.139	
	(1) and (3)	1,000	68.814	
		2,000	54.780	
		3,000	50.723	
Pung	(1) and (2)	1,000	228.342	**171.488**
		2,000	304.130	
		3,000	383.321	
	(1) and (3)	1,000	34.336	
		2,000	36.938	
		3,000	41.861	

especially when the user base grows. This is evident from Pung's particularly high W_1 between scenarios (1) and (2), ranging from $228.342 \cdot 10^{-5}$ (1,000 clients) to $383.321 \cdot 10^{-5}$ (3,000 clients). Except for a long tail at 1,000 clients and distinct plateaus when failures occur at 3,000 clients (Figure 7.2), Vuvuzela achieves latencies with limited variance across settings. Nevertheless, Vuvuzela's W_1 distances for scenarios (1) and (2) are all noticeably higher than FTMix's, while even exceeding all of Pung's distances when we compare scenario (1) to scenario (3).

Due to pool-style delays over cascades of seven mixes, FTMix's latencies span the greatest range across all settings. Latency CDFs for failure scenarios (2) and (3) proceed almost identical to the baseline of scenario (1) across all user bases. This is confirmed by FTMix achieving the lowest W_1 in each pair of compared scenarios and user base size across all systems. FTMix clients experience the least degradations in end-to-end latencies when network failures occur. FTMix's mean W_1 of $13.037 \cdot 10^{-5}$ is $3.9\times$ less than Vuvuzela's

(mean W_1: $50.877 \cdot 10^{-5}$) and 13.15× less than Pung's (mean W_1: $171.488 \cdot 10^{-5}$). This indicates the success of FTMix's straightforward fault tolerance measures in providing users a reliable anonymous messaging service despite severe network failures.

7.5 Related Work

Vuvuzela [26] and its successors [34, 35, 36] unlink messages via a mixnet foundation and use Differential Privacy [24] to determine privacy loss over time. Service availability is acknowledged as important, but fault tolerance measures are not integrated. In Vuvuzela and Alpenhorn [36], a single failed mix halts the service. Karaoke [34] "requires all servers to be online", forfeiting any fault tolerance its free-route topology may afford. Yodel's [35] low latencies enable private US-EU voice calls, while remaining prone to unavailability attacks and more than 20 % malicious mixes. Stadium [48] also arranges mixes in multiple distinct cascades, but for increased scalability not fault tolerance.

Pung [3] reduces the high network and compute costs inherent to CPIR via multi-retrieval schemes while enabling group communication and asynchronous retrievals. CPIR generally promises fault tolerance potential, however, Pung does not provide redundancy and forfeits availability upon server failure. We show in Section 7.4 that operating Pung still requires enterprise-grade hardware, especially at realistic network sizes. An optimally Byzantine-robust (i.e., fault-tolerant) ITPIR scheme exists [19] that is general-purpose and significantly faster than prior work, but dedicated messaging components would first need to be constructed. Systems like XRD [33] and Riffle [32] use compute-intense cryptographic operations to withstand active attacks. This results in higher resource demands and operational costs, while transmission latencies degrade quickly with more users.

Atom [31] tolerates up to a configured amount of mix failures through threshold encryption, but message latencies increase proportionally. Many of Atom's components involve disruption-prone all-to-all communication in groups that might comprise adversarial nodes. Loopix [43] composes many state-of-the-art mixnet components into a "best of breed" design, however, without any further fault tolerance measures than its stratified topology. Tor [22] remains the largest deployed ACS (ca. 8 M daily users [37]), but as a low-latency anonymity network is prone to attacks [27, 28, 38] that the systems we evaluate in Section 7.4 are resilient against. TARANET [12] achieves low-latency anonymity at the network layer, but requires AS nodes to participate.

Borisov et al. show the devastating impact on anonymity when onion routing and mixnet systems ignore (selective) denial-of-service (DoS) attacks [9]. Security cannot be increased arbitrarily with longer paths. Architectures prone to DoS attacks are likely to be compromised entirely if honest nodes are the minority. Shirazi, Díaz, and Wright

propose a metric for resilience against optimally attacking adversaries [45] in Tor by determining nodes an adversary on a budget is likelier to attack through simulations.

7.6 Conclusions

In this work, we evaluate how network failures common to the environment that planet-scale ACS work in negatively impact user experience in the form of end-to-end latencies. In case failures of benign or malicious origin compromise the ability of an ACS to route messages according to expectation or at all, users might be forced offline or back to less private messengers. After discussing the technical foundations of ACS, we select state-of-the-art mixnet Vuvuzela and CPIR system Pung for evaluation. In a multi-continent ACS test bed, we measure end-to-end latencies as well as client and server resource demands of both systems and compare them to fault-tolerant proof-of-concept mixnet FTMix under various failure scenarios. Our analysis shows that FTMix is able to maintain end-to-end latencies that consistently diverge less from expectation than either Vuvuzela or Pung while at the same time achieving a balanced resource consumption trade-off. We conclude that a mixnet with fault tolerance as a core design principle presents a promising starting point for designing resilient and efficient ACS at planetary scale.

Acknowledgments

This work was supported by CyberSecurity Research Flanders with reference number VR20192203. Lennart Oldenburg is funded by an FWO fellowship. We thank Claudia Díaz and the reviewers for their feedback, and Google for a generous GCP credits grant.

Bibliography

[1] C. Aguilar-Melchor, J. Barrier, L. Fousse, and M.-O. Killijian. "XPIR : Private Information Retrieval for Everyone". In: *Proceedings on Privacy Enhancing Technologies* 2016.2 (2016) (p. 101).

[2] S. Angel, H. Chen, K. Laine, and S. Setty. "PIR with Compressed Queries and Amortized Query Processing". In: *2018 IEEE Symposium on Security and Privacy*. IEEE, 2018 (p. 101).

[3] S. Angel and S. Setty. "Unobservable Communication over Fully Untrusted Infrastructure". In: *OSDI '16*. USENIX Association, 2016 (pp. 99, 100, 101, 102, 109).

[4] L. Barman, M. Zamani, I. Dacosta, J. Feigenbaum, B. Ford, J. Hubaux, and D. Wolinsky. "PriFi: A Low-Latency and Tracking-Resistant Protocol for Local-Area Anonymous Communication". In: *WPES '16*. ACM, 2016 (pp. 99, 100).

[5] S. Bayer and J. Groth. "Efficient Zero-Knowledge Argument for Correctness of a Shuffle". In: EUROCRYPT 2012. Springer Berlin Heidelberg, 2012 (p. 101).

[6] O. Berthold and H. Langos. "Dummy Traffic against Long Term Intersection Attacks". In: *PET 2002*. Springer Berlin Heidelberg, 2003 (p. 101).

[7] O. Berthold, A. Pfitzmann, and R. Standtke. "The Disadvantages of Free MIX Routes and How to Overcome Them". In: *Designing Privacy Enhancing Technologies*. Springer Berlin Heidelberg, 2001 (p. 101).

[8] R. Böhme, G. Danezis, C. Díaz, S. Köpsell, and A. Pfitzmann. "On the PET Workshop Panel "Mix Cascades Versus Peer-to-Peer: Is One Concept Superior?"" In: *PET 2004*. Springer Berlin Heidelberg, 2005 (p. 101).

[9] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz. "Denial of Service or Denial of Security?" In: *CCS '07*. ACM, 2007 (p. 109).

[10] D. L. Chaum. "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability". In: *Journal of Cryptology* 1.1 (1988) (p. 100).

[11] D. L. Chaum. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms". In: *Commun. ACM* 24.2 (1981) (pp. 100, 101).

[12] C. Chen, D. E. Asoni, A. Perrig, D. Barrera, G. Danezis, and C. Troncoso. "TARANET: Traffic-Analysis Resistant Anonymity at the Network Layer". In: *2018 IEEE European Symposium on Security and Privacy*. IEEE, 2018 (p. 109).

[13] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. "Private Information Retrieval". In: *FOCS '95*. IEEE, 1995 (p. 101).

[14] H. Corrigan-Gibbs, D. Boneh, and D. Mazières. "Riposte: An Anonymous Messaging System Handling Millions of Users". In: *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015 (p. 100).

[15] H. Corrigan-Gibbs, D. I. Wolinsky, and B. Ford. "Proactively Accountable Anonymous Messaging in Verdict". In: *22nd USENIX Security Symposium*. USENIX Association, 2013 (p. 100).

[16] G. Danezis and I. Goldberg. "Sphinx: A Compact and Provably Secure Mix Format". In: *2009 IEEE Symposium on Security and Privacy*. IEEE, 2009 (p. 102).

[17] G. Danezis. "Mix-Networks with Restricted Routes". In: *PET 2003*. Springer Berlin Heidelberg, 2003 (p. 101).

[18] G. Danezis and L. Sassaman. "Heartbeat Traffic to Counter (N-1) Attacks: Red-green-black Mixes". In: *WPES '03*. ACM, 2003 (p. 101).

[19] C. Devet, I. Goldberg, and N. Heninger. "Optimally Robust Private Information Retrieval". In: *21st USENIX Security Symposium*. USENIX Association, 2012 (pp. 101, 109).

[20] C. Díaz and B. Preneel. "Reasoning About the Anonymity Provided by Pool Mixes That Generate Dummy Traffic". In: *IH 2004*. Springer Berlin Heidelberg, 2005 (p. 101).

[21] C. Díaz and A. Serjantov. "Generalising Mixes". In: *PET 2003*. Springer Berlin Heidelberg, 2003 (p. 101).

[22] R. Dingledine, N. Mathewson, and P. Syverson. "Tor: The Second-Generation Onion Router". In: *13th USENIX Security Symposium*. USENIX Association, 2004 (p. 109).

[23] R. Dingledine, V. Shmatikov, and P. Syverson. "Synchronous Batching: From Cascades to Free Routes". In: *PET 2004*. Springer Berlin Heidelberg, 2005 (p. 101).

[24] C. Dwork and A. Roth. "The Algorithmic Foundations of Differential Privacy". In: *Foundations and Trends® in Theoretical Computer Science* 9.3–4 (2014) (p. 109).

[25] Google, Inc. *Google Cloud: Cloud Computing Services*. URL: `https://cloud.google.com/` (p. 101).

[26] J. van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. "Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis". In: *SOSP '15*. ACM, 2015 (pp. 100, 101, 102, 109).

[27] R. Jansen, F. Tschorsch, A. Johnson, and B. Scheuermann. "The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network". In: *NDSS '14*. The Internet Society, 2014 (p. 109).

[28] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson. "Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries". In: *CCS '13*. ACM, 2013 (p. 109).

[29] D. Kesdogan, J. Egner, and R. Büschkes. "Stop-and-Go-MIXes Providing Probabilistic Anonymity in an Open System". In: *Information Hiding*. Springer Berlin Heidelberg, 1998 (p. 101).

[30] E. Kushilevitz and R. Ostrovsky. "Replication Is Not Needed: Single Database, Computationally-Private Information Retrieval". In: *FOCS '97*. IEEE, 1997 (pp. 99, 101).

[31] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford. "Atom: Horizontally Scaling Strong Anonymity". In: *SOSP '17*. ACM, 2017 (p. 109).

[32] A. Kwon, D. Lazar, S. Devadas, and B. Ford. "Riffle: An Efficient Communication System With Strong Anonymity". In: *Proceedings on Privacy Enhancing Technologies* 2016.2 (2016) (p. 109).

[33] A. Kwon, D. Lu, and S. Devadas. "XRD: Scalable Messaging System with Cryptographic Privacy". In: *NSDI '20*. USENIX Association, 2020 (p. 109).

[34] D. Lazar, Y. Gilad, and N. Zeldovich. "Karaoke: Distributed Private Messaging Immune to Passive Traffic Analysis". In: *OSDI '18*. USENIX Association, 2018 (p. 109).

[35] D. Lazar, Y. Gilad, and N. Zeldovich. "Yodel: Strong Metadata Security for Voice Calls". In: *SOSP '19*. ACM, 2019 (p. 109).

[36] D. Lazar and N. Zeldovich. "Alpenhorn: Bootstrapping Secure Communication without Leaking Metadata". In: *OSDI '16*. USENIX Association, 2016 (pp. 102, 109).

[37] A. Mani, T. Wilson-Brown, R. Jansen, A. Johnson, and M. Sherr. "Understanding Tor Usage with Privacy-Preserving Measurement". In: *IMC '18*. ACM, 2018 (p. 109).

[38] S. J. Murdoch and G. Danezis. "Low-Cost Traffic Analysis of Tor". In: *2005 IEEE Symposium on Security and Privacy*. IEEE, 2005 (p. 109).

[39] C. A. Neff. "A Verifiable Secret Shuffle and Its Application to E-Voting". In: CCS '01. ACM, 2001 (p. 101).

[40] L. Oldenburg. *Planet-Scale ACS Test Bed on Public Clouds*. URL: `https://github.com/numbleroot/acs-test-bed` (p. 103).

[41] L. Oldenburg. *Planet-Scale Public-Cloud Evaluation of zeno, Vuvuzela, and Pung*. URL: `https://github.com/numbleroot/acs-eval-2019` (p. 103).

[43] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis. "The Loopix Anonymity System". In: *26th USENIX Security Symposium*. USENIX Association, 2017 (pp. 101, 109).

[44] A. Serjantov, R. Dingledine, and P. Syverson. "From a Trickle to a Flood: Active Attacks on Several Mix Types". In: *IH 2002*. Springer Berlin Heidelberg, 2003 (p. 101).

[45] F. Shirazi, C. Díaz, and J. Wright. "Towards Measuring Resilience in Anonymous Communication Networks". In: *WPES '15*. ACM, 2015 (p. 110).

[46] F. Shirazi, M. Simeonovski, M. R. Asghar, M. Backes, and C. Díaz. "A Survey on Routing in Anonymous Communication Protocols". In: *ACM Comput. Surv.* 51.3 (2018) (p. 101).

[47] Signal Foundation. *Signal*. URL: `https://signal.org/` (p. 99).

[48] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich. "Stadium: A Distributed Metadata-Private Messaging System". In: *SOSP '17*. ACM, 2017 (p. 109).

[49] WhatsApp Inc. *Two Billion Users – Connecting the World Privately*. 2020. URL: `https://blog.whatsapp.com/two-billion-users-connecting-the-world-privately` (p. 99).

[50] D. Wikström. "A Sender Verifiable Mix-Net and a New Proof of a Shuffle". In: ASIACRYPT 2005. Springer Berlin Heidelberg, 2005 (p. 101).

[51] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. "Dissent in Numbers: Making Strong Anonymity Scale". In: *OSDI '12*. USENIX Association, 2012 (pp. 99, 100).

Chapter 8

From “Onion Not Found” to Guard Discovery

Publication Data

L. Oldenburg, G. Acar, and C. Diaz. “From “Onion Not Found” to Guard Discovery”. In: *Proceedings of the 22nd Privacy Enhancing Technologies Symposium (PETS 2022)*. Vol. 2022. 2022, pp. 522–543. DOI: 10.2478/popets-2022-0026

Contributions

Principal author.

From "Onion Not Found" to Guard Discovery

Lennart Oldenburg,[1] Gunes Acar,[2] Claudia Diaz[1]

[1] *imec-COSIC KU Leuven*
[2] *Radboud University*

Abstract. We present a novel web-based attack that identifies a Tor user's guard in a matter of seconds. Our attack is low-cost, fast, and stealthy. It requires only a moderate amount of resources and can be deployed by website owners, third-party script providers, and malicious exits—if the website traffic is unencrypted. The attack works by injecting resources from non-existing onion service addresses into a webpage. Upon visiting the attack webpage with Tor Browser, the victim's Tor client creates many circuits to look up the non-existing addresses. This allows middle relays controlled by the adversary to detect the distinctive traffic pattern of the "404 Not Found" lookups and identify the victim's guard. We evaluate our attack with extensive simulations and live Tor network measurements, taking a range of victim machine, network, and geolocation configurations into account. We find that an adversary running a small number of HSDirs and providing 5 % of Tor's relay bandwidth needs 12.06 seconds to identify the guards of 50 % of the victims, while it takes 22.01 seconds to discover 90 % of the victims' guards. Finally, we evaluate a set of countermeasures against our attack including a defense that we develop based on a token bucket and the recently proposed Vanguards-lite defense in Tor.

8.1 Introduction

With an estimated eight million daily users [31], Tor [10] is the most popular anonymous communication network in use today. Tor enables users to browse privately, evade censorship, and access or run anonymous network services (*onion services*). In order to access the web via Tor, users install Tor Browser, a modified version of Firefox that routes all traffic through the Tor network. With its strong defenses against cross-site tracking and browser fingerprinting [41], Tor Browser also protects users from unwanted tracking at the application level.

When users visit a webpage with Tor Browser, their traffic is sent on a *circuit* routed through a sequence of Tor relays. When Tor Browser encounters a subresource (e.g., an image) located at an *onion address*, it first contacts a *hidden service directory* (HSDir) to resolve the address to its descriptor (`HS_DESC`). This resolution is made automatically

over a newly created circuit. Using the descriptor, Tor Browser can privately connect to the onion service and access the content.

Every time a new circuit is created, Tor freshly selects new relays for all circuit positions—except the first (called *guard*), which Tor users keep fixed for many months. Guards play a critical role among all relays as they typically know the IP addresses of the Tor users connected to them. By picking one guard and keeping it for a long time, Tor users avoid repeatedly exposing themselves to selecting malicious relays as the first hop in their circuits, which would happen after a short time if users selected new first-hop relays every time [61]. The Tor network has put strong requirements in place for relays that want to become eligible as guards [8].

Attacks that discover the guard of a Tor user (victim) are thus an important subject of research. Identifying a victim's guard may help expose the victim's identity or enable other attacks, including user geolocation [18], traffic confirmation [25, 36], website fingerprinting [38, 45, 58, 59], and selective denial-of-service attacks to force the victim to choose a malicious guard [7, 13, 23, 24]. Depending on bilateral agreements and the guard's jurisdiction, an adversary may seize [57] or subpoena [6, 11] the guard or its network provider to access traffic logs. Adversaries with offensive capabilities may try to compromise the guard server.

We present an attack that allows an adversary to identify the guard of a Tor user visiting a target webpage, using a novel method that is *low-cost*, *fast*, and *stealthy*. The adversary controls (parts of) the target webpage and runs a limited number of HSDirs and Tor relays. When a victim visits the target webpage, the adversary launches the attack by injecting a large number of subresources (e.g., images) on non-existing onion service addresses. To obtain the embedded resources, the victim first connects to the HSDirs responsible for the addresses' descriptors, providing many opportunities for adversarial relays to be selected as second hops. Scanning her second-hop relays for the distinctive traffic pattern of an `HS_DESC` lookup for a non-existing address, the adversary obtains the victim's guard as the predecessor on the circuit that matched the target pattern.

A large class of adversaries is able to deploy our attack: website owners, embedded third parties such as advertisers, administrators of online marketplaces, users of blogging platforms, and malicious Tor exit relays when outbound connections are unencrypted or not protected against SSL stripping attacks [19]. While we assume the adversary injects the subresources using JavaScript, a scriptless version of our attack proves effective against Tor users who have disabled JavaScript.

We run our attack against a large number of our own clients on the live Tor network to obtain a set of realistic victim profiles. We find that Tor users on faster computers and network connections are more vulnerable to our attack, while geographical proximity to the majority of Tor relays also makes our attack more effective. Through extensive experimentation, we determine that our attack identifies the guards of most victims

after they visit the attack webpage for a short amount of time. For instance, an adversary with a small number of HSDirs and 1 % share of Tor's relay bandwidth identifies half the victims' guards in less than 40 seconds. An adversary providing 5 % of Tor's relay bandwidth discovers more than 99 % of the victims' guards in under 50 seconds. In some instances, the adversary requires less than a second to learn a victim's guard.

Our attack is cheap and easy for the adversary to deploy. Of the Tor relays that the adversary requires for the attack, only a few HSDirs (e.g., ten) need to be launched at least 96 hours in advance of the attack. As our attack does not require to run relays in distinguished circuit positions (e.g., guards, exits), it attracts little attention to the adversary. An adversary controlling 5 % of Tor's relay bandwidth only needs a moderate budget of EUR 1,579.56 to run the attack for a full month. Precomputing onion service public keys that will be maintained by at least two adversarial HSDirs in an upcoming network status is perfectly feasible. An adversary with ten HSDirs in the network requires on average 10.3 minutes to generate the keys for a one-minute attack (52.3 minutes on average for a five-minute attack). The adversary has at least twelve hours for this precomputation before the first time a Tor client uses the new network status to download descriptors.

We make the following contributions:

- We present a novel, low-cost, fast, and stealthy guard discovery attack on Tor users. The attack exploits the unregulated creation of new circuits for `HS_DESC` lookups and the distinctive cell pattern of lookups for non-existing onion addresses.
- Through extensive simulations and experiments on the live Tor network, we find that the attack succeeds in identifying a victim's guard in seconds.
- We evaluate countermeasures: a token bucket per Tor Browser tab to limit the number of circuits, restricting the choice of second hops with the recent Vanguards-lite proposal, and a combination of both approaches. While Vanguards-lite prevents a majority of the attack attempts, over 18 % of Tor users remain vulnerable to our attack. We find the token bucket defense to be effective against our attack.

8.2 Background and Related Work

Tor. Tor [10] is an overlay network of volunteer-run servers (*relays*) that route user connections over multi-hop *circuits* to make them anonymous. While an open, generic network, Tor is primarily used to privately access the web with Tor Browser. This allows users to hide their IP addresses from web servers, and prevents linking of their visits by online trackers. When users visit public-Internet webpages with Tor Browser, their

traffic is onion-encrypted and sent through a sequence (circuit) of three relays: *guard*, *middle*, *exit*. In order to limit linkability across different webpage visits, Tor Browser scopes these public-web circuits to the URL bar origin (*first party isolation*) [41]. The same third-party resource requested as part of visits to two distinct first-party webpages is sent across two distinct circuits.

The circuit's relays are chosen by the client according to Tor's *routing policy* which randomly selects a fresh middle and exit relay for every circuit. The choice of relay is weighted by its bandwidth while at the same time accounting for constraints such as destination filters on exits, relay family affiliation, and subnet overlap. Guards, however, are updated only once every 2–3 months [30] and are used on every circuit during that period. Guards were introduced to protect users from adversaries that compromise a subset of Tor relays and are able to fully deanonymize a circuit when they control its first and last hop [62]. Without guards, such adversaries typically have a low probability of compromise for each individual circuit created by a user unless they control a very large fraction of the network. However, every new circuit is a new chance of adversarial success. If the user creates many circuits, it is only a matter of time until the adversary obtains the needed vantage position. The issue is mitigated by fixing the choice for the first relay (which becomes the *guard*). If the user has an honest guard, the adversary will have no chances to obtain the needed vantage position, no matter how many circuits are created.

When data is in transit, each hop of a circuit removes or adds one layer of encryption (*onion routing*), making input and output connections cryptographically unlinkable in terms of both cell headers and payloads. Users learn about all relays and their characteristics from the *consensus* document describing the network state once every hour. The unit of transmission on circuits is a *cell*, a single Tor packet with 509 bytes payload.

In addition to providing anonymous web browsing, Tor also enables services to hide their network identity (address and location) by running *onion services* (previously: *hidden services*). Onion services accept connections from within the Tor network via circuits of six hops (instead of three). Two versions of onion services currently exist, v2 [55] and v3 [56], with v2 nearing end-of-life due to its outdated cryptographic foundations [15, 16]. Onion services are accessed via Tor-internal domain names derived from the services' public keys. *Hidden service directories* (HSDirs) are Tor relays that collectively maintain the *hidden service descriptors* (`HS_DESC`s) of the onion services. An `HS_DESC` contains the information required by clients to establish private connections to the service. Each Tor relay carrying the `HSDir` flag serves a specific set of `HS_DESC`s to Tor clients.

`HS_DESC` lookups take place over dedicated four-hop circuits—which consist of the user's guard, two (instead of one) middles, and an HSDir that maintains the `HS_DESC` (instead of an exit relay). These `HS_DESC` lookup circuits are thus distinct from three-hop circuits used to browse the public web and from six-hop circuits used to connect to

onion services. A fresh HS_DESC lookup circuit is created per lookup attempt.

Guard Discovery. Unlike the second and third hops on a Tor circuit, guards typically know the IP addresses of their users and remain their guards for a long time. Thus, discovering a victim's guard is a stepping stone towards full deanonymization. An adversary successfully mounting a guard discovery attack may be able to seize [57], subpoena [6, 11], or compromise a non-cooperating guard. This opens up a wide array of devastating deanonymization attacks on the guard's users. The adversary may monitor the guard and employ website fingerprinting techniques to learn which websites a victim is browsing [38, 45, 58, 59], uncover a victim's geolocation [18], or perform full traffic confirmation [25, 36]. Some approaches to guard discovery exploit side channels. Mittal et al. show that it is possible to identify the guard(s) of Tor users by fingerprinting the guard's throughput [32]. Rochet and Pereira [42] exploit the public relay load measurements to discover the guards of onion services.

Other approaches (including ours) rely on forcing the creation of many circuits, each of which increases the chances of adversarial compromise. Biryukov et al. [5] present an attack that discovers the guards of an onion service by forcing it to open many rendezvous connections to an adversarial rendezvous point, while a middle relay fingerprints the established circuits. They identify several guards in less than an hour. In a follow-up study, Biryukov et al. [4] propose a similar attack to deanonymize onion service users. The attack works by sending a unique traffic signature from adversary-controlled HSDirs during HS_DESC lookups and expecting the traffic to be routed through the adversary-controlled guards. Compared to our attack, Biryukov et al.'s attack can target all users of an onion service, but their attack misses all targets that do not use the adversarial guards. Their attack can also be easily detected based on the distinctive signature the HSDir sends and would not work against v3 onion services. Compared to guard discovery attacks in prior works, our attack converges much faster and does not require invasive techniques like congesting Tor relays.

Circuit Fingerprinting. Kwon et al. use decision trees to determine circuit purpose and relay position, which they use in turn to identify onion service circuits [29]. Addressing a similar problem, Jansen et al. instead opt for random forests to improve the robustness of the purpose and position classifier [21]. Through a series of simulations, Elahi et al. show that short-term guards make it easier to profile Tor users [12]. Motivated in part by Elahi et al.'s results, Dingledine et al. propose using one entry guard for nine months in Tor [9]. These studies informed the current design of Tor's guard selection method, which uses a single guard for 2–3 months for each user [30].

Attacks Using JavaScript. Evans et al. [13] exploit the then missing limit on path lengths to build long circuits (24 hops) to congest Tor relays selectively. Their attack requires taking latency measurements from the victim, which is achieved by injecting malicious JavaScript code on an exit controlled by the adversary. Evans et al.'s attack requires iterating over all potential Tor guards to find the ones that belong to the victim.

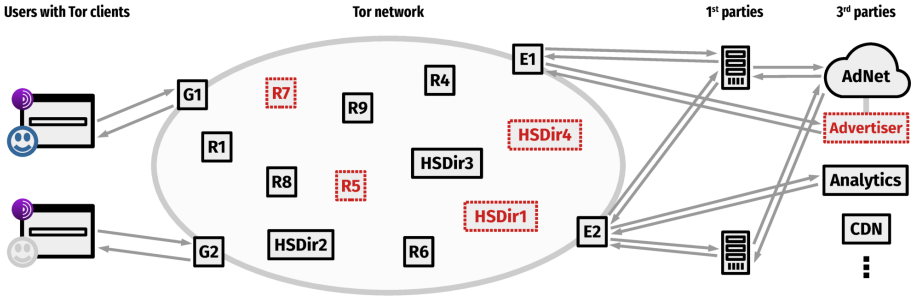


Figure 8.1: Tor network model including adversarial nodes in red with dashed borders. As one possibility for how the adversary may become part of the website visited by victims, we show the adversary as a third-party advertiser here.

Abbott et al. present an end-to-end correlation attack based on injecting a distinctive signal using JavaScript from a malicious exit [1]. Their attack requires the victim to pick a malicious exit and malicious guard at the same time, which is made impractical with Tor's current guard rotation policy.

8.3 Guard Discovery Attack

In this section, we describe the resources required by an adversary to conduct our guard discovery attack. We provide a step-by-step overview of the attack and investigate crucial properties of the target cell pattern that we exploit to identify guards.

8.3.1 Adversarial Objectives and Capabilities

In order to identify a Tor user's guard, the adversary requires three capabilities. First, the adversary has the ability to run scripts on (or otherwise manipulate) the target webpage. Second, the adversary operates one or more HSDirs responding to `HS_DESC` lookups in the Tor network. Third, the adversary runs a set of middle relays in the Tor network. We now detail these capabilities.

Website Manipulation. The adversary needs to be able to inject subresources on non-existing onion services into a webpage visited by the victims using Tor Browser. Clearly, this is the case if the adversary controls the webpage entirely (first party). However, it suffices for the adversary to have the chance of becoming a third party on the target website. This might be the case if the adversary is an advertiser in an advertisement network that the target webpage uses. We illustrate this scenario in Figure 8.1. Other

suitable settings include the adversary being part of a social network or marketplace that allows HTML as part of user-generated content (e.g., direct messages, posts, adverts). Going forward, we assume the adversary to be a third party with JavaScript capabilities on the target webpage. However, our attack also works when Tor Browser users have disabled JavaScript, which we discuss in Section 8.4.3.

Hidden Service Directories. When victims attempt to download the embedded resources at non-existing onion services, they first create a circuit to an HSDir to download the onion service's `HS_DESC`. Our attack relies on the adversary knowing if and when a specific target `HS_DESC` is requested. Thus, the adversary needs to operate at least one of the six responsible HSDirs for each attack onion address. Without HSDirs, the attack's success degrades significantly due to the high rate of false positives (see Section 8.5.1), as the adversary cannot observe the requested attack address and use this to distinguish victim lookups from noise.

Any relay is assigned the `HSDir` flag automatically if it requests it and meets a set of criteria. Specifically, the relay needs to be sufficiently reliable (`Stable` flag), capable of fast network operation (`Fast` flag), and have been running continuously for at least 96 hours [53]. In Section 8.C in the Appendix, we show that operating the required relays including the HSDirs can be achieved at a low cost. Note that the Tor developers are discussing a change to make it more difficult to obtain the `HSDir` flag [50]. If implemented, this change will likely increase the attack costs for the adversary.

Middle Relays. The adversary needs to run middle relays that users may choose as second hops when constructing a new circuit (successors to guards). These middle relays are crucial for the attack as only middle relays are ultimately able to reveal the victims' guards to the adversary. As relays are selected according to their bandwidth share during circuit construction, the adversary may run more middle relays with high bandwidth to speed up the attack.

Relays that are not intended for first or last hops on circuits do not require any flags that are hard or slow to obtain. This benefits the adversary in terms of the time it takes to position all nodes for the attack, limits the attack's cost, and incurs less scrutiny compared to relays in more critical positions (e.g., guards, exits). All adversarial Tor nodes run tor binaries appropriately modified for the attack, i.e., exposing circuit metadata required for the adversary to learn the guards. Additionally, all nodes run a time synchronization protocol, as the attack requires comparing timestamps across machines.

8.3.2 Attack Overview

We now provide a step-by-step description of our web-based guard discovery attack, as shown in Figure 8.2.

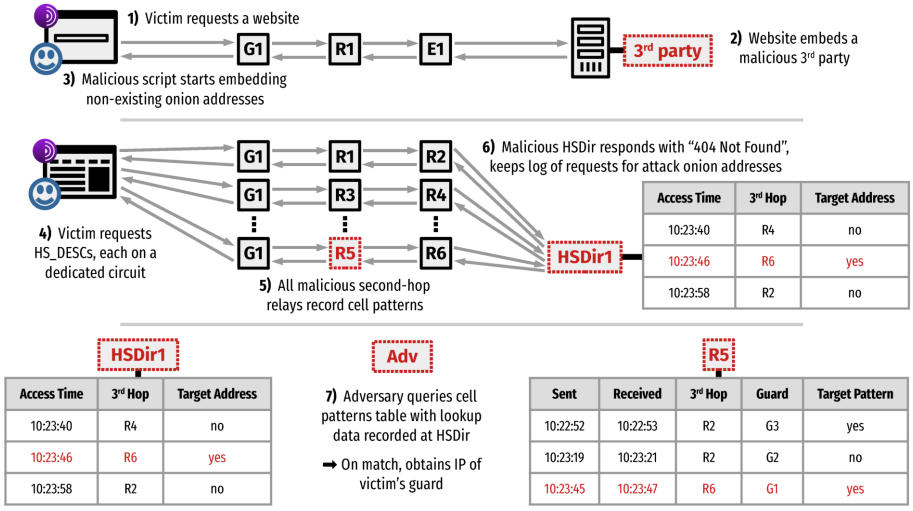


Figure 8.2: Step-wise overview of our guard discovery attack, where a malicious third-party JavaScript causes the victim's Tor Browser to rapidly create a large number of new lookup circuits for non-existing HS_DESCs. Adversarial components in red with dashed borders.

First, a victim requests the target webpage using Tor Browser (step 1). If this is a webpage in the public Internet, the request will by default be relayed through a three-hop circuit to the webserver. We assume the adversary to be a part of this target webpage in the form of either a first or a third party. Once the victim connects to the adversary to download the embedded JavaScript, the attack can be launched by serving the malicious script (step 2).

The malicious JavaScript injects into the webpage's DOM a number of subresources (e.g., scripts, images, fonts) located on onion addresses (step 3). Crucially, the onion services referenced by the injected subresources do not exist. The adversary intentionally crafts attack onion addresses so that the requests for the onion services' HS_DESCs are (partly) handled by her HSDirs. The malicious JavaScript takes care of hiding the injected non-existing resources visually from the victim (e.g., by injecting scripts or setting image dimensions to zero) to evade detection.

For each non-existing onion address on the webpage, the victim's Tor Browser sequentially creates six four-hop circuits to download the HS_DESCs (called *victim lookups*), one per responsible HSDir (step 4). Each newly created circuit represents an opportunity for the adversary's middle relays to be selected as the hop succeeding the victim's guard. Currently, there is no limit to the number of circuits a webpage can cause

to be created (a known issue since 2016 [2]). Through live Tor network measurements (Section 8.4) we find that the adversary can cause victims to perform a median of 12.4 HS_DESC lookups per second, while upwards of 23 lookups per second are also possible.

During the attack, adversarial middle relays continuously record cell metadata *circuit logs* (step 5). This metadata includes: predecessor and successor on the circuit, whether the circuit's cell pattern matches the target pattern, and if so, timestamp t_s of the cell carrying the HS_DESC lookup and timestamp t_r of the response cell. We discuss the target cell pattern in Section 8.3.3.

Adversarial HSDirs also record metadata (in *access logs*) when they respond to HS_DESC lookups (step 6). They keep track of the lookup's timestamp t_a, from which relay they received it, and whether it requested one of the specially crafted target addresses. Independent of the result of the latter check, malicious HSDirs always respond according to specification. That means that they respond with "404 Not Found" to lookups for which they do not have the descriptors, including to the ones for the attack addresses. This response always fits into the payload of a single cell, in contrast to the much larger responses that carry an HS_DESC for an existing resource. Upon receiving a negative lookup response, the victim's Tor client contacts the other HSDirs responsible for the address until all have been queried and the lookup is abandoned.

Our attack exploits the distinctiveness of HS_DESC lookups for non-existing onion addresses to identify victim circuits. The adversary compares entries of interest in the logs of the HSDirs and the middle relays (step 7). The adversary first selects those circuits from the middle relays that match the target cell pattern and those from the HSDir that correspond to the lookup of a target address. The adversary then compares the timestamps and third hops of the selected circuits. Two circuits match when they share the same third hop and the lookup timestamp t_a falls between t_s and t_r that the middle relay recorded. For each matching circuit the adversary records the identity of the relay at the first hop as candidate for being the victim's guard.

8.3.3 Characteristics of Target Cell Pattern

In order to determine the feasibility of our guard discovery attack, we investigate whether the cell pattern of an HS_DESC lookup with a "404 Not Found" response is deterministic and unique among a large number of Tor traffic patterns. Our attack relies on this target cell pattern being deterministic and distinctive enough so that adversarial middle relays can detect it among other unrelated Tor traffic. Evaluating the uniqueness of this pattern on the live Tor network would require recording the traffic of Tor users, which we avoid for ethical reasons. Instead, we opt for analysis in an isolated test network using Shadow [20] and TorNetTools [22, 44].

Shadow is a discrete-event network simulator that directly interfaces with a tor binary

via a plugin. Shadow can run on a single machine of sufficient hardware capabilities and accepts a network model, traffic patterns among the network's nodes, and a custom tor binary. Shadow executes a simulation of the specified traffic patterns, producing logs for further analysis. TorNetTools allows us to generate a scaled-down, representative version of the Tor network based on official metrics data from the Tor Project.

We conduct two simulations, which we describe in Section 8.A of the Appendix. In the first simulation, a single Tor client attempts to connect to a non-existing onion service multiple times over a minimal Tor network (Section 8.A.1). From the logs produced by Shadow, we extract the expected number of the target "Onion Not Found" cell pattern as seen at the second hop in the lookup circuit. We find the cell pattern to be deterministic and visualize the involved cells and commands in Figure 8.9 in the Appendix.

Second, we conduct a much larger experiment where clients generate a diverse set of traffic patterns that *exclude* the target cell pattern (Section 8.A.2). We run this experiment on a 2 %-scale Tor network generated with TorNetTools. We analyze the observed cell patterns to see if any of them match the target cell pattern, which would lead to false positives when used in this attack. However, among 1,866,782 circuits that clients generate as part of this simulation, we do not find the target cell pattern once in any circuit position. Based on the data we collect in these two simulations, we determine the cell pattern of an `HS_DESC` lookup for a non-existing onion address to be deterministic and unique, and thus usable for the adversary in our attack.

8.4 Tuning the Attack

The adversary aims to maximize the victim lookup rate as that speeds up discovery of the victim's guard. In this section, we perform two experiments to determine the parameters that trigger the highest number of victim lookups across different types of clients and thus optimize the effectiveness of our attack. We also present a version of the attack that works without JavaScript.

8.4.1 Attack Parameters

We investigate the impact of three attack parameters the adversary controls: type of the injected resource, onion service version of the injected address, and rate of resource injections per second. In addition, we study the influence of client machine and network characteristics. The adversary aims to adjust these parameters for triggering the highest rate of victim lookups per second in a broad range of clients. More lookups mean more opportunities for creating a circuit that includes the adversarial middle and HSDir relays and thus less time needed to compromise the victim.

Table 8.1: Median, minimum, and maximum number of victim lookups per second, grouped by resource injection rate and onion service version. Experiment described in Section 8.4.1.

Injections / s	Onion service	Median	Min	Max
1.0	**v2**	5.216	4.076	5.681
	v3	5.267	3.737	5.674
2.0	**v2**	10.416	3.083	11.225
	v3	10.405	4.277	11.221
3.0	**v2**	14.377	7.790	16.620
	v3	14.334	6.877	16.600
4.0	**v2**	14.298	6.402	21.894
	v3	14.180	4.513	21.985
5.0	**v2**	12.906	0.170	22.091
	v3	12.555	6.535	22.607
6.0	**v2**	11.611	6.070	17.660
	v3	10.811	4.868	16.718

We set up an attack webpage that we automatically visit with Tor Browser from victim clients under our control. We use the `tor-browser-selenium` [3] Python library to automate Tor Browser. We experiment with injecting six different resource types into the attack webpage: `script` elements (`js`), images, CSS stylesheets, fonts, `iframe` elements, and making asynchronous requests using JavaScript's `Fetch` API. Using `stem` [26], a Python library to interact with Tor, we collect Tor control port logs of victim lookups when visiting the attack webpage with a victim client. We use a desktop computer as victim Tor client that is configured to use a Tor relay we operate as its guard, and run the attack while varying onion version and injection rate for each resource type. In total, we obtain data from 242,004 `HS_DESC` lookups that result in a "404 Not Found" response. We find only very minor differences between resource types, and thus exclusively embed `script` elements going forward.

Next, we assess the impact of onion service version and injection rate on the achieved victim lookup rates over the live Tor network. To approximate Tor's distribution of users over guards, we sample 50 relays from all relays carrying the `Guard` flag in the most recent consensus, weighted by their respective `guard_probability`. For each injection rate in $\{1.0, 2.0, 3.0, 4.0, 5.0, 6.0\}$ resources per second and onion service version in {v2,v3}, we cycle three times through the guards list with three different clients and in each instance visit the attack webpage for 60 seconds.

In total, we collect traces for 3,532,291 victim lookups. In Table 8.1, we list the rate of

Table 8.2: Median number of victim lookups per second, grouped by resource injection rate and victim setting. Experiment described in Section 8.4.1.

Injections / s	Machine and network setting	Median
1.0	**Desktop, fiber, wired**	5.293
	Fast notebook, fiber, wired	5.250
	Slow notebook, VDSL, wireless	5.194
2.0	**Desktop, fiber, wired**	10.520
	Fast notebook, fiber, wired	10.495
	Slow notebook, VDSL, wireless	10.200
3.0	**Desktop, fiber, wired**	15.680
	Fast notebook, fiber, wired	14.292
	Slow notebook, VDSL, wireless	12.763
4.0	**Desktop, fiber, wired**	19.063
	Fast notebook, fiber, wired	14.592
	Slow notebook, VDSL, wireless	11.275
5.0	**Desktop, fiber, wired**	16.342
	Fast notebook, fiber, wired	13.030
	Slow notebook, VDSL, wireless	9.434
6.0	**Desktop, fiber, wired**	13.783
	Fast notebook, fiber, wired	11.496
	Slow notebook, VDSL, wireless	8.585

`HS_DESC` lookups obtained in the experiments, grouped by resource injection rate and onion service version. Based on minimal differences in median values, we consider both the v2 and v3 onion services to be equally vulnerable to our attack. In the remaining live-network experiments we use v2 addresses for simplicity.

In our experiments, we use three different client configurations as shown in Table 8.2. The variance in results across these three client configurations indicates that Tor users with faster computers and better network connections are more vulnerable to our attack, as this results in higher lookup rates. In order to better understand these client-dependent variations and determine the optimal resource injection rate, we perform a larger study taking into account various client machine and network settings, including a client's geolocation.

8.4.2 Maximizing the Victim Lookup Rate

We conduct a third live Tor network measurement to determine the injection rate that yields the highest rate of `HS_DESC` lookups across a diverse set of victim locations, machines, and network conditions. Based on results from the preceding experiments, we focus on covering the most relevant injection rates while fixing the onion service version to v2 and the embedded resource type to `script`. We define three victim client hardware and network profiles to represent three different types of Tor users. We deploy this experiment's clients across three geographically distributed vantage points of a public cloud provider in order to factor in the effects of physical distance along Tor circuits.

Our three victim profiles are characterized as follows. A *weak* Tor client has 2 virtual CPU cores, 2 GB of memory, and a *slow* network connection with 10 Mbit/s downstream and 1 Mbit/s upstream bandwidth, latency following a normal distribution with 30 ms mean and 100 ms variance, and 0.5% packet loss with 25% correlation with the prior drop decision. A *regular* client has 4 vCPUs, 8 GB RAM, and a *medium* connection (50 Mbit/s down, 10 Mbit/s up, latency following $\mathcal{N}(20, 25)$, 0.25% packet loss with 25% correlation). A *powerful* client has 8 vCPUs, 16 GB RAM, and a *fast* connection (150 Mbit/s down, 50 Mbit/s up, latency following $\mathcal{N}(10, 1)$, no packet loss). The network speeds, latency, and packet loss profiles are modeled with some abstraction after the FCC's annual consumer broadband measurements in the U.S. [14]. To also provide a realistic upper limit, we include our *Desktop* office machine (8 vCPUs, 64 GB RAM) connected with *no caps* to our university's high-speed network in Europe.

We sample a fresh list of 100 relays from all relays carrying the `Guard` flag in the most recent consensus, again weighted by their respective `guard_probability`. For each victim client, we visit the attack webpage three times per relay in the list of 100 guards, each time configuring the relay as the victim's guard for the duration of the attack. As before, we use `stem` to collect metadata (e.g., lookup start and end times) on 9,751,286 victim lookups from 17,163 completed out of 18,000 total runs. Failed runs are mainly due to the guard being unavailable during the experiment.

We find that injecting three resources per second yields the highest median lookup rate (12.4 per second) across all victim locations, machine, and network settings. Thus, three resource injections per second is the best choice for the adversary to be effective against the most victim configurations. Note that the adversary may further optimize the attack by first measuring the network latency of a victim [46] and adjusting the resource injection rate accordingly. We refer to Section 8.B in the Appendix for further discussion of the impact of geolocation, machine characteristics, and network conditions on the adversarial injection rate.

8.4.3 Scriptless Attack

In the attack described so far, the adversary relies on JavaScript to control the timing and injection rate of non-existing resources into the webpage viewed by the victim. While JavaScript is enabled by default in Tor Browser, users with high security requirements may disable it by setting Tor Browser's security level to *high*. Such users can still be targeted by the *scriptless* version of our attack. Instead of relying on JavaScript to inject resources at regular intervals, the scriptless attack embeds all non-existing resources in a `noscript` HTML tag [33]. This causes the resources to be inserted into the page only if JavaScript is disabled.

The main challenge we have to overcome in this version of the attack is to prevent the page being blocked from loading, which would cause the loading indicator to spin for a long time—potentially raising suspicion. Experimenting with different techniques to delay the loading of DOM elements beyond initial page load, we find that embedding image elements with the `loading="lazy"` attribute causes Tor Browser to load images in the background, after the initial page load is completed. This effectively gives the victim the impression that the page has loaded successfully.

To test the effectiveness of our attack in this setting, we modify the attack webpage to include a `noscript` tag with 180 image elements having non-existing onions service domains (`src`). We pick 180 as it corresponds to the number of resources injected by a rate of three per second over 60 seconds. Using this attack webpage, we repeat the experiment described in Section 8.4.2 to measure the rate of victim lookups. We only take measurements using the university desktop and the same sample of 100 guards used in Section 8.4.2. We visit the attack page while having the security slider in Tor Browser automatically set to *high*, using `tor-browser-selenium`. We repeat the visit three times with each guard, resulting in 300 visits—of which one third failed due to offline guards. Based on the remaining 200 visits to the scriptless attack webpage, we obtain a median rate of 13.6 `HS_DESC` lookups per second. This shows that the attack is feasible without JavaScript, albeit slightly slower.

8.5 Attack Evaluation

We now evaluate how accurately and quickly our attack identifies the guards of victims. In order to account for false positives that could impair the accuracy of our attack, we first establish the rate of background noise lookups in Tor. We then present simulations to compute the expected time and accuracy of identifying the victims' guards through our attack. Next, we compute the time it takes to generate a fresh set of attack onion service public keys. Finally, we discuss the monetary cost of the attack, considering various adversarial capabilities.

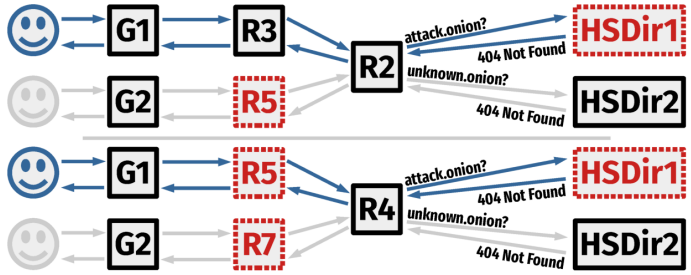


Figure 8.3: Lookups for non-existing, non-attack onion addresses (noise lookups) may result in false positives. Second-hop adversarial relays (e.g., R5, R7) see the target cell pattern, but cannot distinguish whether it is due to a victim or noise lookup.

8.5.1 Estimating Tor's Noise Lookup Rate

In Section 8.3.3, we show that the cell pattern of a victim lookup is unique among a large sample of other Tor traffic. However, as prior research [31] shows, Tor users still make `HS_DESC` requests that result in "404 Not Found" responses independently from our attack. We call these *noise lookups*. Noise lookups occur when Tor users try to connect to an offline onion service or mistype an onion address. Noise lookups may cause false positives when they go through a middle relay controlled by the adversary. Below we discuss in detail how we measure and address these false positives to prevent them from reducing the efficacy of our attack.

We show in Figure 8.3 two scenarios where a pattern match may be due to a false positive. In the first case (top figure), the victim lookup (blue arrows) uses an honest second hop (R3) and the noise lookup (grey arrows) uses an adversarial second hop (R5), while victim and noise lookups share the same (honest) third hop (R2). In this case, the adversary may mistakenly identify the noise lookup guard (G2) as the victim's guard (G1). In the second case (lower figure), the adversary detects two simultaneous attack cell patterns and can only deduce that one of the two candidate guards (G1, G2) belongs to the victim, but not which one.

In order to quantify the rate (N) of noise lookups naturally occurring in the Tor network, we take measurements on the Tor relay that we are running at our university, which already carries the `HSDir` flag. We modify the tor binary (version `0.4.4.5`) to count the number of times our relay responds to `HS_DESC` lookups with a specific response (HTTP code and message). We log these counts binned by hour for 31 days (744 hours) and we make sure to log nothing else. Prior to data collection, we discussed our plans with the Tor Research Safety Board and integrated their feedback (with an overall *no objections* conclusion from both reviewers). Data from these measurements are shown in

Table 8.3: Counts and percentages of responses to v2 and v3 `HS_DESC` lookups, observed over 31 days on a single HSDir. Adversarial second-hop relays might mistake single-cell responses from non-attack `HS_DESC` lookups as false positives (except for responses marked with an asterisk).

Onion service	HTTP code	Message	Response cells	Count (%)
v2	200	Found	multiple	47,331 (5.35)
	400	Decoding Failed	single	0 (0)
	400	Invalid Descriptor	single	10 (0)
	404	Not Encrypted	single*	0 (0)
	404	Not Found	single	446,826 (50.53)
v3	200	Found	multiple	31,564 (3.57)
	404	Decoding Failed	single	0 (0)
	404	Not Found	single	358,421 (40.53)
	503	Reject Single Hop	single*	108 (0.01)
			multiple	**78,895 (8.92)**
			single	**805,257 (91.07)**

Table 8.3, noting additionally whether a particular response requires a single or multiple cells. Responses marked with an asterisk may not confuse adversarial middle relays, because they either do not take place over four-hop circuits or are not encrypted (and thus distinguishable).

Of 884,260 responses over 31 days, more than 91 % were single-cell responses. Of those 805,365 single-cell responses, almost all are due to lookups that resulted in "404 Not Found", which is in line with prior results [31]. We take the number of single-cell responses (without asterisk) that the adversary might mistake for the target cell pattern as the basis for the noise lookup rate. Extrapolating this number from our single HSDir to all ca. 3,500 relays carrying the `HSDir` flag as of the time of writing, we obtain:

$$\mathrm{N} = \frac{805{,}257 \cdot 3{,}500}{744 \cdot 60 \cdot 60} = 1{,}052.27$$

as an estimate for the rate of background noise lookups per second across the entire Tor network.

8.5.2 Experimental Setup

We use simulations to evaluate the efficacy of our attack taking into account factors such as noise lookups, Tor's path selection algorithm, and a varying range of adversarial capabilities. The simulations have three main components as explained below. In order to realistically model these components, we use empirical data collected in the previous sections.

Victim. We simulate a single victim that visits the attack webpage for 300 seconds. For the first 60 seconds, we use the actual victim lookup timings that we record in our live-network experiments (see Section 8.4.2). Limiting ourselves to experiments with the optimal rate of three injections per second, we extract the start and end times of victim lookups from 3,000 experiments (10 machines $\times$ 100 guards $\times$ 3 visits), excluding 133 live-network experiments where tor failed to launch. Extracting the lookup timings allows us to *replay* the victim lookups from different client configurations. To prolong each 60-second experiment to 300 seconds, we generate cover victim lookups at the average lookup rate of the experiment's last 30 seconds. We use the average from the last 30 seconds to prevent a potential initial burst of lookups from skewing the stable-state rate.

We run each experiment 100 times to account for the variance introduced by Tor's path selection process. We simulate a range of adversarial capabilities: adversarial fraction $b = \{0.01, 0.02, 0.05\}$ of Tor's relay bandwidth and fraction $h = \{\frac{1}{6}, \frac{1}{3}, 1\}$ of responsible HSDirs for each attack address that are adversarial (i.e., either one, two, or all six responsible HSDirs). For v3 onion services, $h = \frac{1}{3}$ is the attack success lower bound when the adversary spends a few minutes every day (see Section 8.5.4) to compute a set of attack onion service keys that each map to *at least two* of her HSDirs. Similarly, $h = \frac{1}{6}$ is the attack success lower bound when she uses attack onion service keys that each map to *at least one* of her HSDirs. Setting $h = 1$ requires future HSDir hashring placement to be predictable, which is the case for v2 onion services but not v3. As v2 is deprecated and will soon stop being supported [15, 16], we consider $h = \frac{1}{3}$ as the most representative baseline for the attack impact.

For each victim lookup, we simulate building a four-hop circuit, taking into account relay weights, adversarial relay bandwidth share, and adversarial fraction of responsible HSDirs per attack address. First, we assign the same guard from the live-network experiment to the victim. Next, to select the second hop, we represent all adversarial relays as a single logical relay that we select with probability b. Accordingly, we represent all benign relays as a single logical relay with selection probability $1 - b$. This abstraction does not affect the results of our evaluation, as the adversary is assumed to coordinate among her set of middle relays. Then, we pick a third hop according to relay probabilities from the same consensus we used in Section 8.4.2. Finally, depending on h, we pick an HSDir as follows: if $h = 1$, every victim lookup is sent to the adversary's HSDirs. If

$h = \frac{1}{3}$, two out of six victim lookups are routed to the adversary's HSDirs on average. With $h = \frac{1}{6}$, on average only one of six victim lookups reaches an adversarial HSDir.

Noise Lookups. A realistic model of the noise lookups is crucial for our evaluation, as the overwhelming majority of `HS_DESC` lookups in the Tor network exhibits the target cell pattern that the adversary's middle nodes scan for. Using the measurements from Section 8.4.2, we simulate noise lookups at the rate we extrapolate for the Tor network from our HSDir measurements (Section 8.5.1, $\mathrm{N} = 1052.27$ noise lookups per second).

We build circuits for noise lookups following a similar method as for victim lookups. We assume each noise lookup comes from a different user. Thus, we sample a new guard for each noise lookup circuit from all relays marked as guards in the consensus obtained during Section 8.4.2, weighted by their `guard_probability`. The HSDirs for noise lookups are picked randomly from the ca. 3,500 HSDirs present in the Tor network.

Adversary. The simulated adversary follows the steps described in Figure 8.2 to identify the guard of the victim. The adversary runs middle relays with combined bandwidth share b and controls fraction h of responsible HSDirs. As the target cell pattern can be reliably detected by adversarial middle relays (see Section 8.3.3), we assume that the simulated malicious second hop detects all "404 Not Found" lookup circuits that it relays.

The adversary compares the lookup logs from her HSDirs with pattern matches recorded on her adversarial middle. The adversary sees a match when the HSDir and the middle are connected to the same third hop, and lookup timestamp t_a is between t_s and t_r that the middle relay recorded (Section 8.3.2). The adversary keeps track of the number of observed matches per guard. Once any of these counters reaches two (*double match*), the adversary concludes that the corresponding guard belongs to the victim.

8.5.3 Attack Success

We run 100 simulations for each of the 2,867 distinct experiments and nine different adversarial settings, obtaining a total of 2,580,300 simulation instances. To give an indication of the scale, we generate more than 814.8 billion noise circuits during the simulations, considering 300 seconds of simulation per experiment and 1,052.27 noise lookups per second. For each simulation instance, we record whether the guard detected by the adversary is correct or not, the time it takes to make the detection, and the adversarial parameters used in the simulation.

We use the circuits we constructed as ground truth to identify false positives in guard detection. We manually verify by checking a sample of false positives that they indeed correspond to the scenarios presented in Figure 8.3. Due to the high background rate of noise lookups, we observe that false positives for single matches range from 19.41 % to

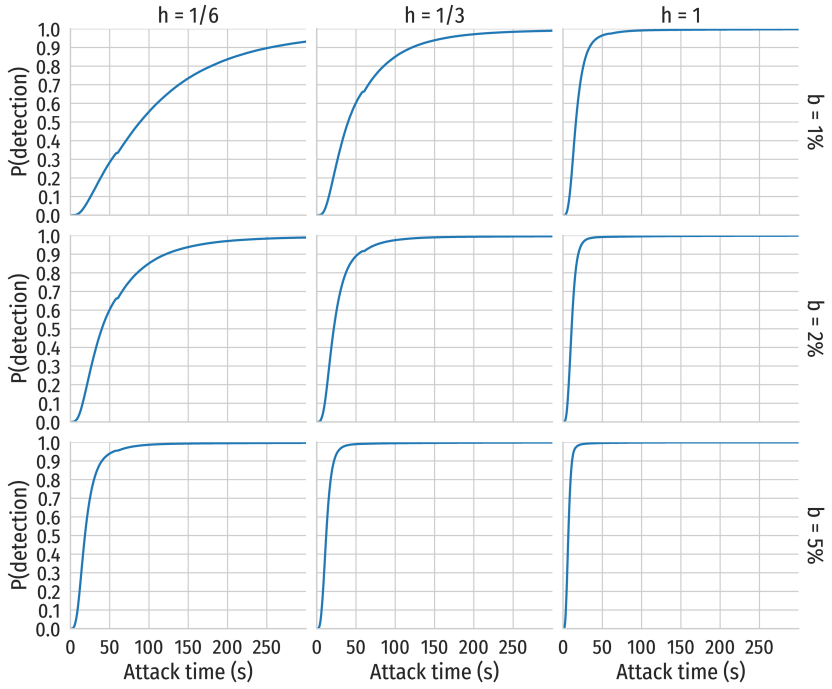


Figure 8.4: Probability of identifying the victim's guard by the attack time in seconds based on 286,700 simulations per setting. The probability increases with time spent on the attack webpage, adversarial fraction of responsible HSDirs for attack onion addresses (h), and adversarial relay bandwidth share (b).

20.22 % for different adversarial settings. While a true positive rate of ca. 80 % is already substantially better than randomly guessing a victim's guard from among the ca. 3,600 Tor guards, the attack's false positive rate drops almost by factor 100 when requiring a double (instead of a single) match. With double matching, the attack's false positive rate decreases to between 0.08 % and 0.37 % (Table 8.4). We use guard decisions from double matching in attack accuracy and duration results in Table 8.4 and Figure 8.4, as well as other metrics reported throughout the paper.

In some simulation instances the adversary does not observe enough matches for any guard to determine the victim's guard. This occurs more often for adversaries with a lower share of controlled HSDirs and relay bandwidth. Such cases fall under the "No Call" column in Table 8.4. The weakest adversary ($h = \frac{1}{6}, b = 1\,\%$) is unable to make a decision 6.82 % of the time even after five minutes of attack. However, the rate of "No Call" attacks drops to 0.09 % when the adversary has $b = 5\,\%$ of Tor's relay bandwidth

Table 8.4: Time in seconds it takes an adversary to identify a victim's guard (via double matching) based on 286,700 simulations per setting. For a $b = 5\,\%$ adversary controlling two of the six responsible HSDirs per v3 attack onion address ($h = \frac{1}{3}$), the attack takes 8.70 seconds for 25 % of the victims and 50 seconds for more than 99 % of the victims. Even a moderate $b = 1\,\%$ adversary ($h = \frac{1}{3}$) needs less than 40 seconds to discover 50 % of the guards. We include $h = 1$ for the end-of-life v2 onion services where HSDirs could be placed. The percentiles where our attack does not converge during 300 seconds of simulation ("No Call") are indicated as $\lfloor 300.0 \rfloor$.

Setting	Attack Duration (Percentiles)						Accuracy		
(h, b)	Min	P25	P50	P75	P90	P99	TP	FP	"No Call"
$\frac{1}{6}$, 1%	1.56	45.09	88.62	155.95	254.07	$\lfloor 300.0 \rfloor$	93.10%	0.08%	6.82%
$\frac{1}{6}$, 2%	1.23	23.90	39.98	74.29	121.31	$\lfloor 300.0 \rfloor$	98.90%	0.08%	1.02%
$\frac{1}{6}$, 5%	0.59	12.76	18.47	27.13	40.03	112.92	99.63%	0.09%	0.28%
$\frac{1}{3}$, 1%	1.07	23.75	39.89	74.32	121.15	$\lfloor 300.0 \rfloor$	98.85%	0.15%	1.00%
$\frac{1}{3}$, 2%	0.71	14.59	21.69	33.13	52.49	144.84	99.50%	0.14%	0.37%
$\frac{1}{3}$, 5%	0.34	8.70	12.06	16.42	22.01	48.56	99.78%	0.13%	0.09%
1, 1%	0.76	11.52	16.37	23.49	33.59	91.49	99.42%	0.37%	0.21%
1, 2%	0.37	7.87	10.90	14.67	19.35	40.20	99.63%	0.30%	0.06%
1, 5%	0.42	4.91	6.71	8.98	11.52	24.07	99.75%	0.24%	0.01%

and generates attack addresses for which she controls $h = \frac{1}{3}$ of the responsible HSDirs. Due to "No Call" outcomes, some CDF subplots in Figure 8.4 do not reach 1.0 within the simulation time.

As shown in Figure 8.4, the probability of identifying a victim's guard increases with 1) the time spent on the attack page, 2) the adversarial bandwidth share (b), and 3) the fraction of adversarial HSDirs for the attack addresses (h). We particularly note the effect h has on the attack's speed. The adversary may substantially reduce the attack duration by spending a small amount of time each day (see Section 8.5.4) to precompute attack public keys that map to two of the six responsible HSDirs ($h = \frac{1}{3}$) instead of one ($h = \frac{1}{6}$).

The strongest adversary in our evaluation (providing $b = 5\,\%$ of Tor's relay bandwidth and controlling $h = \frac{1}{3}$ of the responsible HSDirs per attack address) needs 8.7 seconds for a quarter of the victims, 12.06 seconds for half of them, and 48.56 seconds for 99 % of the victims (false positive rate: 0.13 %). If this adversary controls only one of the six responsible HSDirs per v3 attack onion address ($h = \frac{1}{6}$), the attack times increase to 12.76 (P25), 18.47 (P50), and 112.92 (P99) seconds (false positive rate: 0.09 %). A 1 %

adversary ($h = \frac{1}{3}$) needs less than 40 seconds for 50 % and 74.32 seconds for 75 % of the victims (false positive rate: 0.15 %). Even the weakest adversary ($h = \frac{1}{6}, b = 1\,\%$) discovers the guards of more than 50 % of the victims within 90 seconds of the webpage visit (false positive rate: 0.08 %).

For v2 onion services, an adversary is able to generate multiple HSDir relay public keys offline that are placed in sequence on the HSDir hashring once they obtain the `HSDir` flag. Through offline precomputation, the adversary can then obtain v2 attack onion service keys that map exclusively to her HSDirs ($h = 1$). As shown in Figure 8.4 and Table 8.4, this leads to extremely short attack times. The strongest v2 adversary ($h = 1, b = 5\,\%$), learns the guards of 50 % of the victims in just 6.71 seconds, and in 24.07 seconds for 99 % of them. We include the results on v2 just for the record however, as v2 onion services are nearing end-of-life [15, 16].

We note that the small notch in some CDF plots around 60 seconds is an artifact of how we generate cover lookups to extend 60-second experiments to 300-second simulations. To avoid inflating the victim lookup rates we start the cover lookups after the 60 second mark, but never before. An actual attack deployed against real Tor users does not experience this issue and will thus be slightly more advantageous to the adversary.

8.5.4 Time to Generate Attack Keys

Figure 8.4 shows that the attack is much faster for $h = \frac{1}{3}$, i.e., when the adversary precomputes v3 attack onion addresses that map to two of her adversarial HSDirs. A hashring constructed from data about each HSDir in the network and public parameters determines which HSDir serves which part of the v3 onion services address space. To prevent HSDir placement on the hashring from being predictable too far into the future, the process involves a shared random value from Tor's consensus that changes daily [56]. A relay wanting to become an HSDir needs to be active in the network for at least 96 hours before that happens, so that the shared random value has changed multiple times when the relay joins the hashring.

We implement the algorithm to obtain a set of v3 attack onion service keys that map to *at least two* adversarial HSDirs for a set of public parameters (shared random value, current time period) from an actual Tor network state. We target a set size of 900 keys, which corresponds to five minutes of attack at three injections per second (see Section 8.4.2). We are interested in the average time it takes an adversary to generate keys that map to at least two adversarial HSDirs, given that she controls a number of HSDirs in the network. We detail the generation process in Section 8.D in the Appendix.

We run the key generation script ten times for each number of adversarial HSDirs to account for variance in their selection. We parallelize the script to run across 20 processes (using ten hyper-threaded cores). The average time to obtain 900 v3 attack

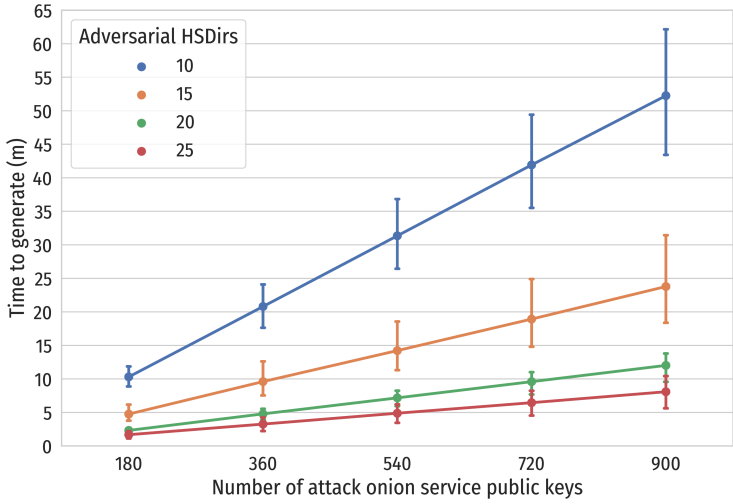


Figure 8.5: Average time in minutes it takes an adversary running 10, 15, 20, or 25 HSDirs to generate a set of up to 900 onion service public keys that map to at least two adversarial HSDirs. Due to daily rotation of the responsible HSDirs per v3 onion service, an adversary needs to regenerate these keys every day. Error bars mark the 95 % confidence interval from ten repetitions.

onion service keys that map to at least two adversarial HSDirs is shown in Figure 8.5. On average, an adversary running 25 HSDirs needs between 1.68 minutes (180 keys) and 8.1 minutes (900 keys). As the adversary has at least twelve hours between knowing the subsequent shared random value and this value's first use by Tor clients, it is easily feasible to precompute attack onion service keys to obtain $h = \frac{1}{3}$. For an adversary with ten HSDirs, this takes between 10.3 minutes (180 keys) and 52.3 minutes (900 keys), on average. Thus, the adversary makes a trade-off between running more HSDirs to minimize the daily precomputation and running fewer HSDirs that require longer—but feasible—daily precomputation efforts.

8.6 Countermeasures

We evaluate three countermeasures against our guard discovery attack: a **token bucket** that rate-limits the number of lookup circuits created by a Tor Browser tab, deploying **Vanguards-lite** [27] to introduce second-hop guards, and a **combination** of the two approaches. The token bucket restricts the number of lookup circuits a Tor Browser tab can create and thus reduces the adversary's chances to observe the lookup pattern.

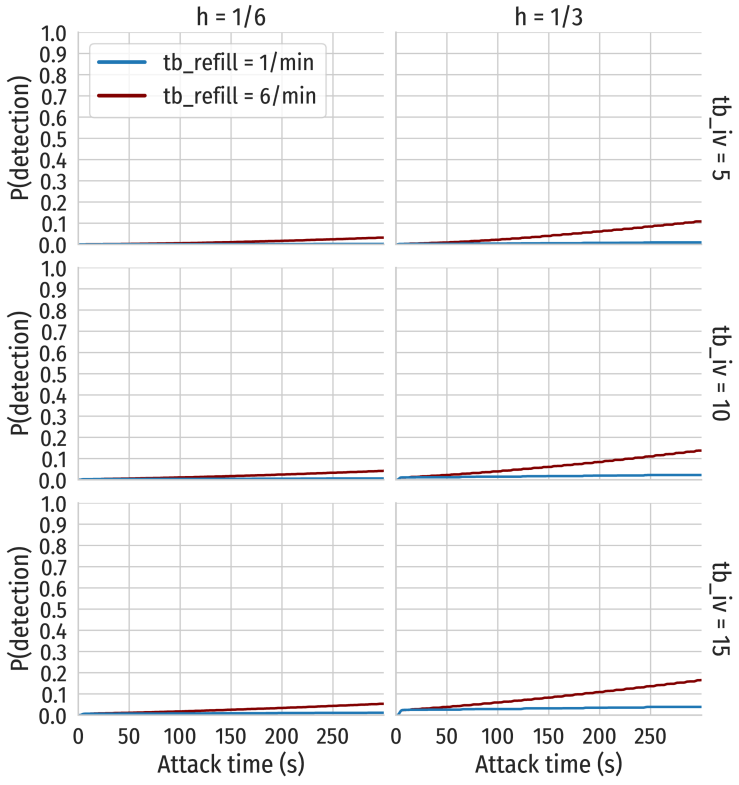


Figure 8.6: Probability of detection for a 5 % adversary, considering a token bucket with `tb_iv` initial tokens and refill rate `tb_refill`.

Vanguards-lite restricts the set of eligible second-hop relays to four per user (called *L2 guards*) and thus limits our attack to users with adversarial relays among their L2 guards. Combining Vanguards-lite with a specially configured token bucket allows to limit attack success in the presence of adversarial L2 guards.

We evaluate each countermeasure by integrating it into our attack simulation from Section 8.5.3. Each individual configuration below is based on 50 simulations for each of our 2,867 live-network lookup profiles from Section 8.4.2, leading to 143,350 simulations per displayed CDF. We focus on v3 onion services and a $b = 5\,\%$ adversary, and thus exclude the adversarial settings of $h = 1$ and $b = \{1\,\%, 2\,\%\}$.

Token Bucket. The token bucket regulates the initial (and burst) number `tb_iv` of lookup circuits that each tab in Tor Browser can cause the Tor client to create, as well as the rate `tb_refill` with which tokens are replenished as long as less than `tb_iv` are

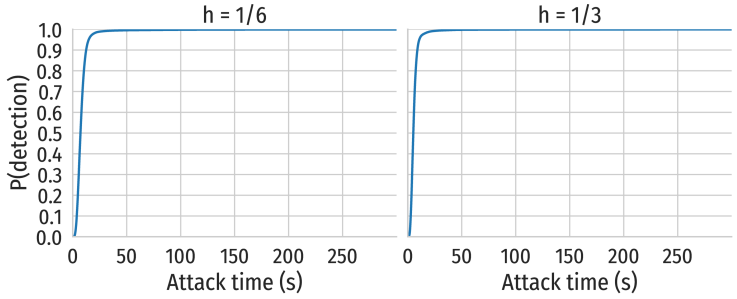


Figure 8.7: (Entry) Guard detection probability for a 5 % adversary when at least one of the victim's four L2 guards is adversarial.

available. When `tb_iv` and `tb_refill` are chosen appropriately, for most scenarios the adversary does not get enough chances to be selected as second hop before the rate limiting sets in. On the other hand, selecting too low `tb_iv` and `tb_refill` impairs the user experience of benign websites that embed more onion service resources than these limits allow. A special case represents a `tb_refill` of zero, where `tb_iv` acts as an upper limit on the total number of circuits a tab can create.

In order to estimate which `tb_ivs` do not impair most benign websites, we crawl 115 onion websites from a list of high-profile onion websites [35] with `tor-browser-selenium`. Using tor and Tor Button logs, we find three to be the maximum number of distinct onion addresses embedded on the crawled websites. We pick `tb_ivs` of five, ten, and fifteen to account for onion websites with a few more embedded onion addresses and cases where not all lookups are successful on first try.

We evaluate each `tb_iv` in { 5, 10, 15 } with each `tb_refill` rate in { 1/min, 6/min } and show the time it takes the adversary to compromise a victim's guard in Figure 8.6. Against an $h = \frac{1}{3}$ adversary, a refill rate of 6/min still leads to compromise probabilities above 10 % after five minutes, independently of the chosen `tb_iv`. Thus, we do not further consider `tb_refill` = 6/min. However, with a refill rate of 1/min, we see ca. 1 % (`tb_iv` = 5), 2 % (`tb_iv` = 10), and 4 % (`tb_iv` = 15) after five minutes against the $h = \frac{1}{3}$ adversary. Thus, at `tb_refill` = 1/min, even with `tb_iv` = 15 (five times the observed maximum of three distinct onion addresses) a victim's chance of compromise is reduced to ca. 2.5 % after 60 seconds on the attack webpage and ca. 4 % after 300 seconds.

Vanguards-lite. With the recent Vanguards-lite [27] proposal enabled, Tor clients and onion services sample four relays from the list of all active relays weighted by bandwidth and use them as second-hop guards (L2 guards) on every onion circuit. Instead of picking a second hop for a new onion circuit from all middle relays, this hop is now chosen

uniformly at random from the four L2 guards. In contrast to the existing Vanguards Tor add-on [28, 39], Vanguards-lite rotates L2 guards sooner (average retention: seven days), and does not write them to disk. Thus, a new set of four L2 guards is picked when the user restarts tor or requests a new identity [49]. Moreover, unlike Vanguards, Vanguards-lite does not change the default path lengths by adding an additional layer of (L3) guards.

Considering four L2 guards and a $b = 5\,\%$ adversary, on average about 18.5 % of Tor users will have at least one adversarial L2 guard. As our attack requires the adversary to control the second and fourth hop of a victim lookup circuit, Vanguards-lite reduces the share of vulnerable Tor users, from close to 100 % to about 18.5 % on average. The alternative of three L2 guards considered by the proposal authors further reduces the average share of vulnerable Tor users to about 14.3 %.

However, once an adversarial relay is part of a victim's L2 guards, our attack needs a much shorter time to complete. We implement Vanguards-lite in our attack simulation and evaluate the case of a Vanguards-lite user with at least one adversarial L2 guard against a $b = 5\,\%$ adversary with h in $\{\, \frac{1}{3}, \frac{1}{6} \,\}$. The simulation results (Figure 8.7) show that it takes 21.31 seconds for an $h = \frac{1}{3}$ adversary (or 26.76 seconds for $h = \frac{1}{6}$) to discover the entry guards of 99 % of the victims. Comparing the results to Table 8.4, the median attack time drops by more than half: from 12.06 to 5.13 seconds.

Vanguards-lite with Token Bucket. To address the vulnerability of Vanguards-lite users with an adversarial L2, we combine the two approaches presented above by applying a token bucket per Tor Browser tab on top of Vanguards-lite. Due to limited usefulness of low refill rates yet steadily increasing compromise probabilities over longer attack times, we set `tb_refill` = 0 for this defense. Thus, `tb_iv` effectively acts as an upper bound on the total number of lookup circuits any Tor Browser tab can create. We show a victim's guard discovery probability with at least one of four adversarial L2 guards and a token bucket with `tb_iv` in { 5, 10 } in Figure 8.8. For the stronger adversary ($h = \frac{1}{3}$), compromise probability is at 7 % (`tb_iv` = 5) and 22.5 % (`tb_iv` = 10), respectively. Combined with applying to only about 18.5 % of Tor users to begin with, we obtain on average about 1.3 % (`tb_iv` = 5) and 4.2 % (`tb_iv` = 10) compromised Tor users after five minutes of attack, down from almost 100 % without any defenses.

Discussion. Based on our simulations, a token bucket per Tor Browser tab with a relatively high `tb_iv` of 15 and a `tb_refill` of one (or lower) appears to be a simple and effective countermeasure to limit the risk from our guard discovery attack. However, it is a specific defense that only targets `HS_DESC` lookup circuits from Tor Browser tabs and may lead to user experience (UX) issues once a tab has exhausted its tokens. If a webpage embeds many subresources on non-existing onion addresses, the token bucket defense blocks all `HS_DESC` lookups once it is empty, preventing the webpage from loading fully or functioning properly. A challenging UX decision then is to decide whether and how to communicate the status *"no tokens left"* to users. On onion websites,

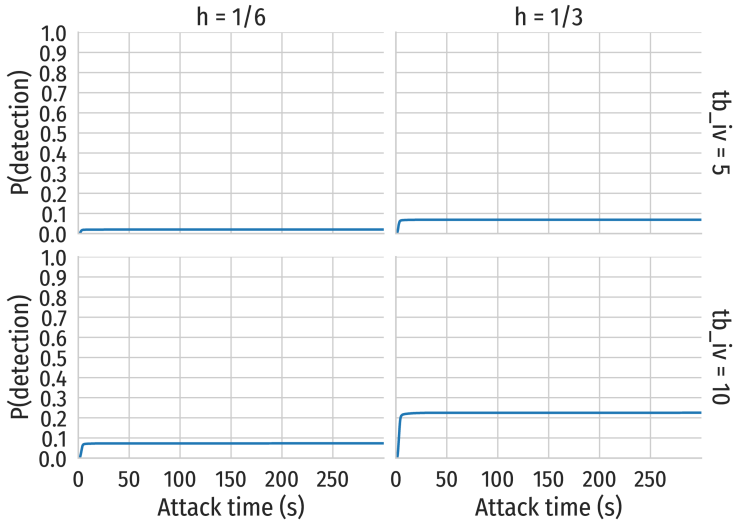


Figure 8.8: Guard detection probability for a 5 % adversary when at least one of the four second-hop guards on onion circuits (Vanguards-lite) chosen by the victim is adversarial and a token bucket with `tb_iv` initial tokens and `tb_refill` = 0 is used.

Tor Browser displays different icons in the address bar to warn users about potential security issues such as "The Onion Service is served with a mixed form over an insecure URL" [51]. We show the existing set of onion icons in Section 8.E in the Appendix. We believe adding a new icon that is displayed when no tokens are left to be the most effective and least intrusive solution. Further, research on token replenishment strategies based on user actions (e.g., clicks) may be an interesting avenue for future work to avoid unwanted token depletion.

Vanguards-lite is a more general-purpose countermeasure that does not impair website rendering. It is informed by and benefits from the Tor community's experience with the guard concept and is derived from the already used and tested Vanguards add-on. However, a substantial number of Tor users will have at least one adversarial L2 guard (ca. 18.5 % for $b = 5\,\%$) and thus remain vulnerable to our attack. Adding a cap on the number of lookup circuits to Vanguards-lite has a limited effect: there are only four choices for the second hop, so it does not take many circuits to select an adversarial second hop twice, allowing the adversary to succeed.

8.7 Discussion

Fundamental Problem and Countermeasures. The fundamental issue that enables our attack is the same that led Tor to introduce guards in the first place: a small probability of circuit compromise becomes problematic if the user can be tricked into creating many circuits (i.e., taking many fresh chances), as the adversary only needs to get lucky once to succeed. Our attack uncovers a new vector exploitable by adversaries who target users visiting a certain webpage. Known countermeasures aim to reduce the chances of compromise, either by *capping the number of attempts* (e.g., token bucket) or by *reusing previously made choices* (e.g., Vanguards-lite and guards themselves).

Capping attempts can have a usability impact, while reusing choices introduces linkability and thus may be exploitable for deanonymization. We find that capping circuit creations is a more effective protection in our attack scenario. Note however, that while a cap on the number of `HS_DESC` lookup circuits per Tor Browser tab is possible with a small impact on user experience, capping is not a viable solution in other contexts. For example, the "capping" alternative to reusing the first-hop relay (entry guard) on circuits would imply limiting the amount of pages browsed with Tor, completely defeating its purpose. The best solution (capping attempts vs. reusing choices) for each manifestation of the problem is therefore highly context-dependent.

Recurring attacks against Tor guards show that a useful future research direction may be to systematize the concept of guards, including attacks and defenses against them. Such research may attempt to consolidate the existing specialized defenses into a framework and inform future countermeasures about the fundamental problems and trade-offs.

Attack Impact and Feasibility. Typically, guard discovery serves as a stepping stone for further attacks such as traffic confirmation [25], website fingerprinting [38, 59], and selective denial of service [7]. Upon identifying a victim's guard, actors with offensive capabilities may go on to compromise, coerce, or subpoena the server. Past web-based guard discovery attacks [47] with a similar impact were flagged as *high priority* by Tor developers, while being roughly 60× slower than our attack. Furthermore, our attack only needs a modest budget and can thus be carried out by adversaries with limited budgets (Section 8.C, Appendix). The Tor relays that the adversary operates are of the types that receive the least scrutiny. The biggest constraint for the adversary is that she needs to run her designated HSDirs at least 96 hours in advance of the attack so that each obtains the `HSDir` flag.

By forcing a Tor client to create many circuits in a short time, our attack enables other attacks that can be deployed by an adversary with a limited view of the network (e.g., with a small bandwidth share). For instance, if an adversary deploys our attack while also running malicious guards, she may fully deanonymize Tor users instead of just identifying their guards.

Our attack can be deployed by a wide range of adversaries, including website owners and embedded third parties. If the adversary cannot get access to a website visited by the target, she may lure the victim into visiting a malicious website via a spear phishing email, text message, or direct message on an online platform (e.g., a dark-web marketplace). Moreover, a malicious exit may inject the attack code into websites visited by Tor users over unencrypted HTTP connections [60].

Ethical Considerations. We collected data with the potential to impact other Tor network participants on two occasions. First, when we performed our geographically distributed victim crawls (Section 8.4). Second, when we counted the number of times our HSDir responded with a specific code to `HS_DESC` requests (Section 8.5.1). For both experiments, we first discussed our plans with Tor's Research Safety Board and modified details based on the valuable feedback we received. We took great care of not logging any privacy-sensitive information about the requests we saw on our HSDir. We made sure not to overload guards by spreading our experimental traffic over time.

Limitations. Tor tries to use a single primary guard across all circuits of a user [30]. However, if for example exit and first primary guard are part of the same relay family or `/16` subnetwork, Tor needs to use one of the other two primary guards as the first hop [40]. In such unlikely cases, our attack may identify the second or third instead of the first primary guard of a victim.

Our attack works best when deployed against a single victim or small number of victims in a targeted manner. The adversary may rate-limit the attack by serving the malicious script selectively. When run against a large number of victims concurrently, our attack will output the set of guards used by all victims, without a mapping between an individual victim and its guard. However, due to the short time it takes to obtain a guard using our attack, the adversary may run it one-by-one against a large group of victims in rapid succession.

Our estimate of the Tor-wide noise lookup rate is extrapolated from data collected on a single HSDir. We collected the data over 31 days to account for variance and approximate our numbers to the average HSDir, but our view of the `HS_DESC` space was still limited. While fluctuations in the noise lookup rate may affect the efficacy of our attack, the adversary can use her HSDirs to continuously monitor the noise lookup rate and adjust her relay bandwidth share accordingly.

We demonstrate determinism and uniqueness of the target cell pattern with Shadow simulations. These simulations make assumptions about the network conditions they work in, which may not account for all possible traffic patterns occurring on the live Tor network and thus should be considered an abstraction.

Tor users may detect that they are being targeted by our attack, e.g., by checking their HTTP requests with Tor Browser's Network Monitor [34] and seeing the suspiciously high number of onion service requests.

Feedback from Tor Developers. We shared our findings with the Tor developers and have been in discussion with them about potential ways to address our attack. We provided them early results from our countermeasures evaluation, including on the recent Vanguards-lite proposal [27]. We made sure the evaluation we presented in this paper represents Vanguards-lite fairly and accurately. In addition, they provided very useful feedback on the potential side effects of countermeasures on performance and reliability, as well as their implementation difficulty. For instance, the current design of our token bucket defense has benefited from their feedback that countermeasures that only require client-side changes are easier to implement than those affecting relays. Concurrent to our work, the Tor developers integrated Vanguards-lite [27, 48] into Tor `0.4.7`.

8.8 Conclusions

We present a fast, stealthy, and low-cost attack that reveals a Tor user's guard through a web-based attack. Our attack exploits the fact that webpages can cause victim Tor clients to create an unlimited number of circuits to look up onion service descriptors as well as the distinctiveness of the cell pattern if the lookup is unsuccessful. The attack can be deployed for a few hundred euros per month by a diverse set of adversaries, including third-party script providers, website administrators, and malicious exits (when unencrypted HTTP is used). Through extensive simulations and live-network experiments, we evaluate our attack against Tor users with different geolocations, computer speeds, and network conditions. We find that a 5 % adversary controlling two HSDirs per attack onion address identifies a victim's guard in 12.06 seconds at the median case and under 50 seconds for 99 % of the users. The better a victim's network connection and computer, the more effective is our attack. While our attack works best when JavaScript is enabled, we develop a scriptless version that works against Tor users with stricter security settings. We show that a token bucket countermeasure is effective in mitigating our attack and that the use of Vanguards-lite can also reduce its impact.

8.9 Acknowledgements

We thank George Kadianakis and Mike Perry for valuable discussions, particularly about the countermeasures. We also thank the anonymous reviewers for their helpful feedback that improved this work. Lennart Oldenburg is funded by a PhD fellowship of the Fund for Scientific Research - Flanders (FWO). Gunes Acar was supported by a FWO postdoctoral fellowship, and was affiliated with imec-COSIC KU Leuven during the study. This research is partially supported by the Research Council KU Leuven under the grant C24/18/049, by CyberSecurity Research Flanders with reference

number VR20192203, and by DARPA FA8750-19-C-0502. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any of the funders.

Bibliography

[1] T. G. Abbott, K. J. Lai, M. R. Lieberman, and E. C. Price. "Browser-Based Attacks on Tor". In: *International Workshop on Privacy Enhancing Technologies*. Springer, 2007, pp. 184–199 (p. 121).

[2] G. Acar. *Tor can be forced to open too many circuits by embedding .onion resources (#20212) · Issues · The Tor Project / Core / Tor*. Sept. 2016. URL: https://gitlab.torproject.org/tpo/core/tor/-/issues/20212 (p. 124).

[3] G. Acar, M. Juarez, and individual contributors. *GitHub: tor-browser-selenium - Tor Browser automation with Selenium*. [Online, accessed 2021/09/14]. 2020. URL: https://github.com/webfp/tor-browser-selenium (p. 126).

[4] A. Biryukov, I. Pustogarov, F. Thill, and R.-P. Weinmann. "Content and Popularity Analysis of Tor Hidden Services". In: *2014 IEEE 34th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2014 (p. 120).

[5] A. Biryukov, I. Pustogarov, and R.-P. Weinmann. "Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization". In: *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013 (p. 120).

[6] Boing Boing. *What happened when we got subpoenaed over our Tor exit node*. [Online, accessed 2021/09/14]. Aug. 2015. URL: https://boingboing.net/2015/08/04/what-happened-when-the-fbi-sub.html (pp. 117, 120).

[7] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz. "Denial of Service or Denial of Security?" In: *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007 (pp. 117, 142).

[8] R. Dingledine. *The lifecycle of a new relay*. [Online, accessed 2021/09/14]. Sept. 2013. URL: https://blog.torproject.org/lifecycle-new-relay (p. 117).

[9] R. Dingledine, N. Hopper, G. Kadianakis, and N. Mathewson. "One Fast Guard for Life (or 9 months)". In: *7th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2014)*. 2014 (p. 120).

[10] R. Dingledine, N. Mathewson, and P. Syverson. "Tor: The Second-Generation Onion Router". In: *13th USENIX Security Symposium*. USENIX Association, 2004 (pp. 116, 118).

[11] T. Durden. *[tor-relays] Subpoena received*. [Online, accessed 2021/09/14]. Apr. 2015. URL: https://lists.torproject.org/pipermail/tor-relays/2015-April/006804.html (pp. 117, 120).

[12] T. Elahi, K. Bauer, M. AlSabah, R. Dingledine, and I. Goldberg. "Changing of the Guards: A Framework for Understanding and Improving Entry Guard Selection in Tor". In: *Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society*. ACM, 2012 (p. 120).

[13] N. S. Evans, R. Dingledine, and C. Grothoff. "A Practical Congestion Attack on Tor Using Long Paths". In: *USENIX Security Symposium*. 2009 (pp. 117, 120).

[14] Federal Communications Commission, Office of Engineering and Technology. *Tenth Measuring Broadband America Fixed Broadband Report*. [Online, accessed 2021/09/14]. Jan. 2021. URL: https://www.fcc.gov/reports-research/reports/measuring-broadband-america/measuring-fixed-broadband-tenth-report (p. 128).

[15] D. Goulet. *[tor-dev] Onion Service v2 Deprecation Timeline*. [Online, accessed 2021/09/14]. June 2020. URL: https://lists.torproject.org/pipermail/tor-dev/2020-June/014365.html (pp. 119, 132, 136).

[16] D. Goulet. *Onion Service version 2 deprecation timeline*. [Online, accessed 2021/09/14]. July 2020. URL: https://blog.torproject.org/v2-deprecation-timeline (pp. 119, 132, 136).

[17] Hetzner Online GmbH. *Hetzner Cloud: Pricing*. [Online, accessed 2021/09/14]. n.d. URL: https://www.hetzner.com/cloud?country=gb#pricing (p. 154).

[18] N. Hopper, E. Y. Vasserman, and E. Chan-Tin. "How Much Anonymity Does Network Latency Leak?" In: *ACM Transactions on Information and System Security (TISSEC)* 13.2 (2010) (pp. 117, 120).

[19] isabela. *Tor security advisory: exit relays running sslstrip in May and June 2020 | Tor Blog*. [Online, accessed 2021/09/14]. Aug. 2020. URL: https://blog.torproject.org/bad-exit-relays-may-june-2020 (p. 117).

[20] R. Jansen and N. Hopper. "Shadow: Running Tor in a Box for Accurate and Efficient Experimentation". In: *Network and Distributed Systems Security (NDSS) Symposium 2012*. Internet Society, Feb. 2012 (pp. 124, 151).

[21] R. Jansen, M. Juarez, R. Galvez, T. Elahi, and C. Diaz. "Inside Job: Applying Traffic Analysis to Measure Tor from Within". In: *Network and Distributed Systems Security (NDSS) Symposium 2018*. Internet Society, 2018 (p. 120).

[22] R. Jansen, J. Tracey, and I. Goldberg. "Once is Never Enough: Foundations for Sound Statistical Inference in Tor Network Experimentation". In: *30th USENIX Security Symposium*. USENIX Association, 2021 (pp. 124, 151).

[23] R. Jansen, F. Tschorsch, A. Johnson, and B. Scheuermann. "The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network". In: *Network and Distributed Systems Security (NDSS) Symposium 2014*. Internet Society, Feb. 2014 (p. 117).

[24] R. Jansen, T. Vaidya, and M. Sherr. "Point Break: A Study of Bandwidth Denial-of-Service Attacks against Tor". In: *28th {USENIX} Security Symposium ({USENIX} Security 19)*. USENIX Association, 2019 (p. 117).

[25] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson. "Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013 (pp. 117, 120, 142).

[26] D. Johnson. *Stem: a Python controller library for Tor*. [Online, accessed 2021/09/14]. n.d. URL: `https://stem.torproject.org/` (p. 126).

[27] G. Kadianakis. *[tor-dev] [RFC] Proposal 332: Vanguards lite*. [Online, accessed 2021/09/14]. June 2021. URL: `https://lists.torproject.org/pipermail/tor-dev/2021-June/014569.html` (pp. 137, 139, 144).

[28] G. Kadianakis and M. Perry. *Mesh-based vanguards*. [Online, accessed 2021/09/14]. May 2018. URL: `https://gitweb.torproject.org/torspec.git/tree/proposals/292-mesh-vanguards.txt` (p. 140).

[29] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas. "Circuit Fingerprinting Attacks: Passive Deanonymization of Tor Hidden Services". In: *24th USENIX Security Symposium*. USENIX Association, Aug. 2015, pp. 287–302 (p. 120).

[30] I. Lovecruft, G. Kadianakis, O. Bini, and N. Mathewson. *Tor Guard Specification*. [Online, accessed 2021/09/14]. n.d. URL: `https://gitweb.torproject.org/torspec.git/tree/guard-spec.txt` (pp. 119, 120, 143).

[31] A. Mani, T. Wilson-Brown, R. Jansen, A. Johnson, and M. Sherr. "Understanding Tor Usage with Privacy-Preserving Measurement". In: *Proceedings of the Internet Measurement Conference 2018*. ACM, 2018 (pp. 116, 130, 131).

[32] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov. "Stealthy Traffic Analysis of Low-Latency Anonymous Communication Using Throughput Fingerprinting". In: *Proceedings of the 18th ACM conference on Computer and Communications Security*. ACM, 2011 (p. 120).

[33] Mozilla and individual contributors. *<noscript> - HTML: HyperText Markup Language | MDN*. [Online, accessed 2021/09/14]. Feb. 2021. URL: `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/noscript` (p. 129).

[34] Mozilla and individual contributors. *Network Monitor - Firefox Developer Tools | MDN*. [Online, accessed 2021/09/14]. Feb. 2021. URL: `https://developer.mozilla.org/en-US/docs/Tools/Network_Monitor` (p. 143).

[35] A. Muffett. *GitHub: Real-World Onion Sites*. [Online, accessed 2021/09/14]. n.d. URL: `https://github.com/alecmuffett/real-world-onion-sites` (p. 139).

[36] M. Nasr, A. Bahramali, and A. Houmansadr. "DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning". In: *CCS '18*. ACM, 2018 (pp. 117, 120).

[38] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle. "Website Fingerprinting at Internet Scale". In: *Network and Distributed Systems Security (NDSS) Symposium 2016*. Internet Society, 2016 (pp. 117, 120, 142).

[39] M. Perry et al. *GitHub: The Vanguards Onion Service Addon*. [Online, accessed 2021/09/14]. n.d. URL: https://github.com/mikeperry-tor/vanguards (p. 140).

[40] M. Perry. *The move to two guard nodes*. [Online, accessed 2021/09/14]. Mar. 2018. URL: https://gitweb.torproject.org/torspec.git/tree/proposals/291-two-guard-nodes.txt (p. 143).

[41] M. Perry, E. Clark, S. Murdoch, and G. Koppen. *The Design and Implementation of the Tor Browser [DRAFT]*. [Online, accessed 2021/09/14]. June 2018. URL: https://www.torproject.org/projects/torbrowser/design/ (pp. 116, 119).

[42] F. Rochet and O. Pereira. "Dropping on the Edge: Flexibility and Traffic Confirmation in Onion Routing Protocols". In: *Proceedings on Privacy Enhancing Technologies* 2018.2 (2018) (p. 120).

[43] Shadow Team. *GitHub: TGen*. [Online, accessed 2021/09/14]. n.d. URL: https://github.com/shadow/tgen (p. 153).

[44] Shadow Team. *GitHub: tornettools*. [Online, accessed 2021/09/14]. n.d. URL: https://github.com/shadow/tornettools (pp. 124, 151).

[45] P. Sirinam, M. Imani, M. Juarez, and M. Wright. "Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018 (pp. 117, 120).

[46] P. Tellis. *Analyzing Network Characteristics Using JavaScript And The DOM, Part 1*. [Online, accessed 2021/09/14]. Nov. 2011. URL: https://www.smashingmagazine.com/2011/11/analyzing-network-characteristics-using-javascript-and-the-dom-part-1 (p. 128).

[47] The Tor Project. *#9063 enables Guard discovery in about an hour by websites (#9072) · Issues · Legacy / Trac*. [Online, accessed 2021/09/14]. Feb. 2021. URL: https://gitlab.torproject.org/legacy/trac/-/issues/9072 (p. 142).

[48] The Tor Project. *Build vanguards lite into little-t-tor (#40363) · Issues · The Tor Project / Core / Tor*. [Online, accessed 2021/09/14]. Apr. 2021. URL: https://gitlab.torproject.org/tpo/core/tor/-/issues/40363 (p. 144).

[49] The Tor Project. *Introduce vanguards-lite (!408) · Merge requests · The Tor Project / Core / Tor*. [Online, accessed 2021/09/14]. July 2021. URL: https://gitlab.torproject.org/tpo/core/tor/-/merge_requests/408/diffs#fb72b9489ffeb300a7d2e454d0407f5947ecfdc4_3933_4144 (p. 140).

[50] The Tor Project. *Make it even harder to become HSDir (#19162) · Issues · The Tor Project / Core / Tor*. [Online, accessed 2021/09/14]. May 2016. URL: https://gitlab.torproject.org/tpo/core/tor/-/issues/19162 (p. 122).

[51] The Tor Project. *Onion Services | Tor Project | Support*. [Online, accessed 2021/09/14]. n.d. URL: https://support.torproject.org/onionservices/#onionservices-5 (p. 141).

[52] The Tor Project. *Relay requirements*. [Online, accessed 2021/09/14]. n.d. URL: https://community.torproject.org/relay/relays-requirements/ (p. 155).

[53] The Tor Project. *Tor directory protocol, version 3*. [Online, accessed 2021/09/14]. n.d. URL: https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt (p. 122).

[54] The Tor Project. *Tor Metrics: Traffic*. [Online, accessed 2021/09/14]. n.d. URL: https://metrics.torproject.org/bandwidth-flags.html (p. 154).

[55] The Tor Project. *Tor Rendezvous Specification*. [Online, accessed 2021/09/14]. n.d. URL: https://gitweb.torproject.org/torspec.git/tree/rend-spec-v2.txt (p. 119).

[56] The Tor Project. *Tor Rendezvous Specification - Version 3*. [Online, accessed 2021/09/14]. n.d. URL: https://gitweb.torproject.org/torspec.git/tree/rend-spec-v3.txt (pp. 119, 136, 155).

[57] torservers.net. *Coordinated raids of Zwiebelfreunde at various locations in Germany*. [Online, accessed 2021/09/14]. July 2018. URL: https://blog.torservers.net/20180704/coordinated-raids-of-zwiebelfreunde-at-various-locations-in-germany.html (pp. 117, 120).

[58] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg. "Effective Attacks and Provable Defenses for Website Fingerprinting". In: *23rd {USENIX} Security Symposium ({USENIX} Security 14)*. USENIX Association, 2014 (pp. 117, 120).

[59] T. Wang and I. Goldberg. "On Realistically Attacking Tor with Website Fingerprinting". In: *Proceedings on Privacy Enhancing Technologies* 2016.4 (2016) (pp. 117, 120, 142).

[60] P. Winter, R. Köwer, M. Mulazzani, M. Huber, S. Schrittwieser, S. Lindskog, and E. Weippl. "Spoiled Onions: Exposing Malicious Tor Exit Relays". In: *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2014 (p. 143).

[61] M. Wright, M. Adler, B. N. Levine, and C. Shields. "Defending Anonymous Communications Against Passive Logging Attacks". In: *2003 Symposium on Security and Privacy*. IEEE, 2003, pp. 28–41 (p. 117).

[62] M. K. Wright, M. Adler, B. N. Levine, and C. Shields. "The Predecessor Attack: An Analysis of a Threat to Anonymous Communications Systems". In: *ACM Transactions on Information and System Security (TISSEC)* 7.4 (2004), pp. 489–522 (p. 119).

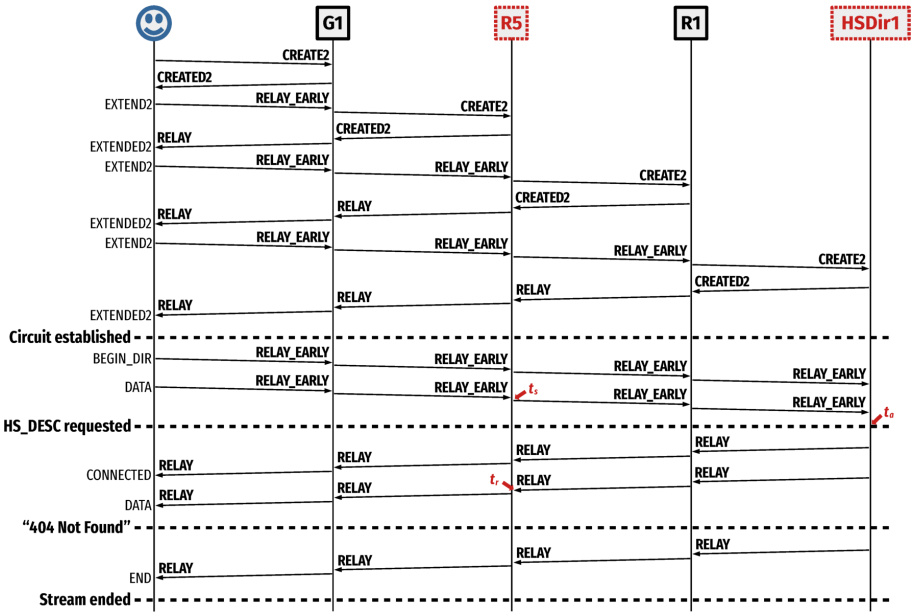


Figure 8.9: Target cell pattern of a victim lookup that the adversarial middle relays (e.g., R5) scan for. Timestamps t_s, t_a, and t_r correspond to columns `Sent` (R5), `Access Time` (HSDir1), and `Received` (R5) in the tables in Figure 8.2, respectively.

8.A Adversarial Cell Pattern Characteristics with Shadow

For our cell pattern investigations in Section 8.3.3, we conduct two experiments that analyze the target "404 Not Found" cell pattern using Shadow [20] and TorNetTools [22, 44]. In all experiments, we run a tor binary patched at release `0.4.4.5` to export metadata about routed cells. We log a timestamp, the cell's channel-circuit assignment and purpose of the channel, whether cells are incoming or outgoing of the relay, their direction (away from or towards the circuit origin), and their command (`CREATE`, `CREATED`, `RELAY`, and `DESTROY` cells). For cells with command `RELAY`, we additionally extract the relayed command, if visible to the node.

8.A.1 Is the Target Cell Pattern Deterministic?

For our first experiment we construct a minimal Tor network in Shadow, consisting of five Tor nodes: a client that connects via a guard, a fixed second-hop relay, and a

fixed third-hop relay to an HSDir. The client uses the circuit to request the HS_DESC of a non-existing onion address. We use this simple setup to extract the second hop's view on the cell pattern exhibited by the victim lookups. As we are interested primarily in the cell pattern, we configure a low, fixed latency of 10 ms per edge and no packet loss or jitter. We note that this is an abstraction from network conditions in the live Tor network that may require adaptations to the matching routine when deployed on live networks. The experiment simulates one hour of operation, during which the client attempts to connect to the non-existing onion address a fixed number of times (leading to a fixed number of lookups). Among other things, the long simulation time allows us to account for cells that are sent due to inactivity and are still part of the target cell pattern.

Analyzing the cell traces and metadata collected for the second-hop relay in the simulation, we find that the adversarial cell pattern is indeed deterministic. Figure 8.9 shows how the pattern is viewed at each circuit hop. Specifically, we are interested in the circuit's second-hop position, which is next to a victim's guard (relay "R5" in Figure 8.9). The pattern always features three parts, possibly completed by a final closure part. First, the telescoping circuit setup iterates from guard "G1" to middle "R5" to middle "R1" and finally "HSDir1". Next, the client opens a directory connection and requests the (non-existing) HS_DESC. Third, the HSDir sends back confirmation of directory connection establishment, the "404 Not Found" response, and a cell ending the stream. We do not depict the circuit teardown behavior after a certain period has elapsed.

The target cell pattern being deterministic is advantageous for our attack. The adversary is able to distinguish whether an adversarial middle is positioned at the second or third hop of the victim lookup circuit. This allows to exclude all circuits where an adversarial middle relay has only been chosen into the third circuit position and does not have visibility over the guard. Further, the "404 Not Found" response payload is the same across both onion service versions, v2 and v3, and fits into a single cell in both scenarios. Finally, a deterministic cell pattern removes the need for more elaborate classification techniques such as machine learning.

8.A.2 Is the Target Cell Pattern Unique?

In order for the attack to work, the HS_DESC lookup cell pattern also needs to be unique: in case the cell pattern is otherwise frequently occurring in all sorts of unrelated circuits, its observation likely corresponds to a false positive. We conduct a second experiment using Shadow to investigate whether the target pattern is spuriously observed. In contrast to the preceding experiment, here we generate a larger, diverse, and more realistic network using TorNetTools.

Based on data published by the Tor Metrics team for May 2020, we have TorNetTools produce a Tor test network that is representative of a 2%-scale sampled version of

the live Tor network. Note that downscaling does not have a significant impact on the experiment, since it is concerned with cell patterns that appear within a circuit, which are insensitive to total network size. The simulated network includes 133 Tor relays, 16 `tgen` [43] clients instructed to measure download performance, 158 `tgen` clients modelled to emulate actual user behavior, and 16 `tgen` servers. The entities are mapped to locations in a network model that aims to realistically capture geographical distribution and network conditions of the Tor network. Thus, upstream and downstream bandwidth as well as latency are adjusted according to the nodes' geographical locations.

In order to have `HS_DESC` lookups (of *existing* addresses) as part of the results, we configure four of the 16 `tgen` servers to operate as onion services, two as v2 and two as v3 onion services. The remaining TorNetTools parameters stay at their defaults (`process_scale = 0.01`, `server_scale = 0.1`, `torperf_scale = 0.001`, and `load_scale = 1.0`). This means that each `tgen` client process emulates 100 actual Tor users. The total simulated time is set to one hour, of which the initial five minutes are dedicated to bootstrapping. Once bootstrapping is completed, `tgen` traffic generation begins and runs until the end of the experiment.

In total, clients construct and use 1,866,782 circuits over the course of the experiment. For each hop on each circuit, we check whether any of the observed cell patterns matches the attack cell pattern, as observed by R5 in Figure 8.9. We do not find a single occurrence of the adversarial pattern. Based on our observations from these two simulations using Shadow and TorNetTools, we find the cell pattern of an `HS_DESC` lookup with a "404 Not Found" response to be deterministic and unique, and thus exploitable as part of our attack.

8.B Establishing the Most Effective Set of Attack Parameters

Regarding our experiments in Section 8.4.2 that determine the most effective attack parameters for the adversary on the live Tor network, we detail how injection rates and geolocation affect victim lookup rates in Figure 8.10. Each boxplot shows the distribution of the average number of lookups per second over 60 seconds of attack, per victim setting and configured injection rate. We observe that geographical distance between the victim and the region where most Tor relays are located (Europe) appears to have an effect on lookup rates. Lookup rates for victims in Frankfurt are highest in each machine-network class, followed by victims in New York City, and finally victims in Singapore (exceptions are the weakest and slowest clients at four, five, and six injections per second).

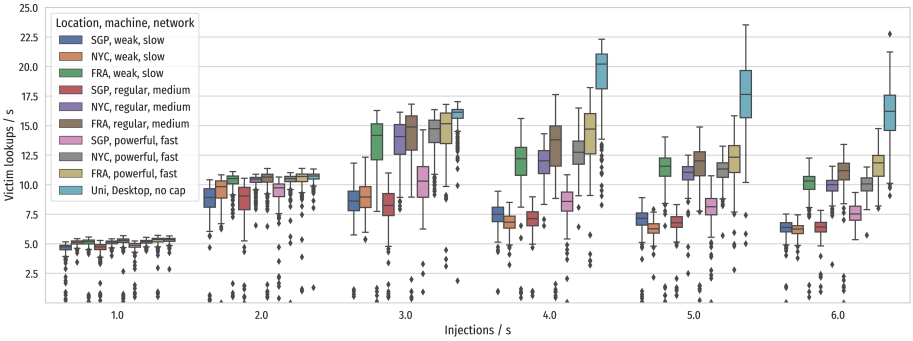


Figure 8.10: Victim lookups per second across different rates of injected non-existing resources and victim settings. Experiment and victim settings described in Section 8.4.2. The four geographical locations represent: Singapore (SGP), New York City (NYC), Frankfurt (FRA), and our university (Uni).

At injection rates higher than three per second, results begin to diverge for more and less powerful clients. While more powerful machines generate more lookups with a higher injection rate, less powerful machines become "saturated" and generate fewer lookups than if only three resources per second are injected. Our office Desktop, which we include as an upper limit, reaches a peak rate of 23.5 lookups per second at five injected resources per second.

8.C Monetary Cost of the Attack

We give an exemplary cost calculation of our attack to demonstrate its affordability using prices from Hetzner, a low-cost cloud provider [17]. While the adversary's expenses may also include the cost of manipulating the webpage (e.g., as an advertising third party), we focus on HSDir and relay operation costs, as these likely represent the bulk of the budget.

According to Tor Metrics [54], the total advertised relay bandwidth is around 600 Gbit/s. However, Tor users actually consume only about half of it. Thus, the adversary may advertise twice as much bandwidth as officially available with each instance. Hetzner includes 20 TB traffic per month per instance, thus we advertise $2 \cdot \frac{160{,}000}{30 \cdot 24 \cdot 60 \cdot 60} \approx 0.123$ Gbit/s per middle relay. To cumulatively offer 5 % of Tor's relay bandwidth, the adversary needs to advertise $\frac{1}{19} \cdot 600 \approx 31.58$ Gbit/s and thus requires $\frac{31.58}{0.123} < 257$ instances. For 2 % cumulative relay bandwidth, she needs to run $\frac{12.25}{0.123} < 100$ instances, while she only requires $\frac{6.06}{0.123} < 50$ instances for 1 %.

Based on the Tor Project's information on relay requirements [52], Hetzner's `CX21` instances (2 vCPUs, 4 GB RAM, 5.88 €/month) appear adequate. Assuming ten HSDirs, the adversary needs to spend EUR 1,579.56 (5 %), EUR 656.40 (2 %), and EUR 362.40 (1 %) for one month of attack. We deem these costs to be affordable even for adversaries with limited financial resources.

8.D Attack Key Precomputation

Our attack key precomputation script works as follows. We load the current shared random value from a Tor consensus file and calculate a time period involving the `valid-after` field. We load the list of identity public keys of all relays with the `HSDir` flag in the server descriptors document downloaded with the consensus file and build the sorted list of each relay's hash of key information and public parameters (i.e., the hashring). For each number of adversarial HSDirs in { 10, 15, 20, 25 }, we sample this many unique HSDirs uniformly at random from the list of all HSDirs as the current set of adversarial HSDirs. Now we take the time it takes to generate fresh onion service key pairs, blind them according to v3 specification [56], calculate their hashring indices and respective sets of responsible HSDirs, and select the ones that map to at least two of the sampled adversarial HSDirs until the list of selected attack onion service keys has size 900. Reaching intermediate sizes of multiples of 180 is marked as dedicated timestamps.

8.E Onion Security Icons

We show the existing set of onion icons used by Tor Browser in Figure 8.11.

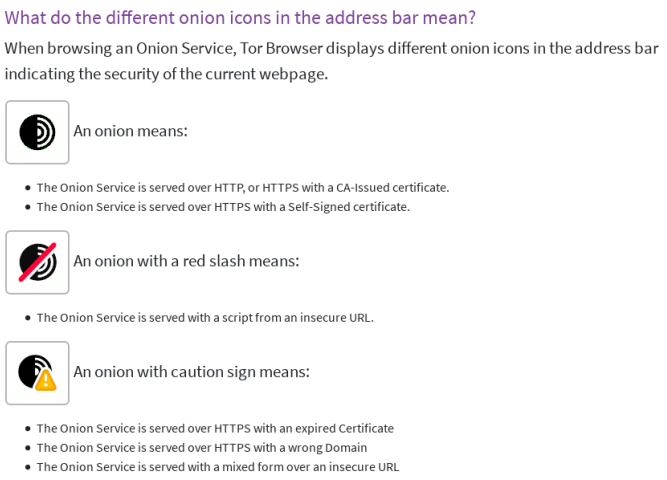


Figure 8.11: The existing set of onion icons used by Tor Browser to convey the security of an onion website.

Chapter 9

MixMatch: Flow Matching for Mixnet Traffic

Publication Data

L. Oldenburg, M. Juarez, E. Argones Rúa, and C. Diaz. "MixMatch: Flow Matching for Mixnet Traffic". In: *Proceedings of the 24th Privacy Enhancing Technologies Symposium (PETS 2024)*. Vol. 2024. 2024, pp. 276–294. DOI: 10.56553/popets-2024-0050

Contributions

Principal author.

MixMatch: Flow Matching for Mixnet Traffic

Lennart Oldenburg,[1] Marc Juarez,[2]
Enrique Argones Rúa,[1] Claudia Diaz[1,3]

[1] *COSIC, KU Leuven*
[2] *School of Informatics, University of Edinburgh*
[3] *Nym Technologies, SA*

Abstract. Mixnets provide communication anonymity against network adversaries by routing packets independently via multiple hops, delaying them artificially at each hop, and introducing cover traffic. We show that these features (particularly the use of cover traffic) significantly diminish the effectiveness of state-of-the-art flow correlation techniques developed to link the two ends of a Tor connection. In this work, we propose novel methods to determine whether a set of endpoints exchanges packets via a mixnet and demonstrate their effectiveness by applying them to the Nym mixnet. We consider Nym in both an *idealized lab setup* and the official *live network*, and propose and compare three classifiers to conduct *flow matching* on it. Our *statistical* classifier tests whether egress packet timestamps are consistent with ingress timestamps and the (known) routing delay characteristic of the mixnet. In contrast, our two *deep learning* (DL) classifiers learn to distinguish matched from unmatched flow pairs from collected datasets directly, rather than relying on priors that describe the delay distribution. All three classifiers use our *flow merging* technique, which enables testing a match for sets of communicating endpoints of any cardinality. Considering a use case where two observed endpoints communicate exclusively to exchange a file through Nym, we find that flow matching is fast and accurate in the idealized lab setup. If flow pairs are aligned using all network observations in a download, we achieve a TPR of circa 0.6 (DL) and 0.47 (statistical) at an FPR of 10^{-2} after only processing 100 observations. We evaluate classifier performance under key variations of this setup: the absence of loop cover traffic, an increased or decreased average per-mix delay, larger communicating sets (three endpoints) with faster responders, and the presence of realistic network effects (live network). The classifiers' matching performance diminishes on the live network where packet losses and variable propagation delays exist, reducing DL TPR to circa 0.26 and statistical TPR to circa 0.28 at an FPR of 10^{-2}. Informed by the insights of our analyses, we outline countermeasures that can be deployed in mixnets such as Nym to mitigate flow matching threats.

9.1 Introduction

With every packet we send or receive over the Internet, we generate *metadata*—data *about* the packets, such as timestamp, size, and direction. If left unprotected, an adversary able to collect and analyze these metadata can determine who is talking to whom, enabling privacy-invasive insights into our increasingly online lives. *Anonymous communication networks* such as *mixnets* [6] aim to protect metadata even against adversaries observing all network links, e.g., by artificially delaying packets and shaping traffic to be of constant bandwidth. Mixnets provide strong protection to individual packets, which are hard to trace as they are routed through the network. Once two endpoints exchange a large number of packets in a short period of time, however, an adversary may be able to match the flows of packets entering and leaving the network. While this issue has long been researched in the context of onion routing networks such as Tor, the extent to which mixnets may be vulnerable to this threat remains an open question.

Two recent developments bring renewed urgency to this question. First, the recent deployment of *Nym* [10, 23], a mixnet that aims to provide a general-purpose anonymous routing layer. The Nym mixnet is based on Loopix [28] and as such relies on source routing, continuous-time mixes, and a packet sending schedule following a Poisson process and where clients send "loop" cover packets to themselves in addition to the payload packets transmitted to others. Second, researchers have recently shown that adversaries employing *deep learning methods* are effective at linking the endpoints engaged in packet exchanges through Tor [11], the longest-running and most popular anonymity network. DeepCoFFEA [24] and its predecessor DeepCorr [20] strikingly demonstrate how well supervised learning methods are able to correlate network flows despite obfuscation from having been routed through Tor. While Tor is not designed to prevent the correlation of flows by an adversary observing both ends of a connection [38], this attack is squarely within the threat model of mixnets. Additionally, when mixnets like Nym use fixed packet sizes that enable general-purpose use, modern application workloads (e.g., email, messaging, web browsing), their data-intensive protocol parts (e.g., authentication, encryption), and interactivity can quickly produce large flows—prompting us to revisit the threat of matching flows on mixnets.

In this work, we develop novel methods for evaluating flow matching in mixnets and empirically investigate their effectiveness on the deployed Nym mixnet. Flow matching is a special case of *flow correlation* where sets of endpoints communicate *exclusively* among themselves (*endpoints closed set condition*), such that any packet sent by an endpoint to the mixnet is received by itself or another endpoint from its set. We design three classifiers for this flow matching setting that consider an adversary who observes communications between the endpoints and the mixnet (in the case of Nym, this can be achieved by running *gateways*). Through our *flow merging* operation (Section 9.3.3), we accommodate Nym's loop cover traffic in our model. Our *statistical classifier* (Section 9.4) uses the fact that the drift between ingress and egress timestamps follows a stationary distribution

only for matched flows (but deviates for unmatched flows). We also propose two *deep learning classifiers* (Section 9.5), called *drift* and *shape*, that automatically extract the relevant information from the training data. Our drift network exploits the temporal drift between egress and ingress packets similar to how our statistical classifier does, while the shape network follows DeepCoFFEA's approach in learning embeddings where matched inter-arrival sequences are located closer to each other than unmatched sequences.

We evaluate our classifiers' performance on six datasets that represent distinct setups (Section 9.6). This allows us to identify and assess key factors in the effectiveness of Nym's metadata protection (Section 9.7). Dataset `baseline` captures file download trace pairs across default Nym over an idealized network (near-instant and loss-free transmissions). After aligning using all observations of a completed download and processing the first 100 observations per flow, our statistical and our drift classifier are fast and accurate in matching flows here (circa 0.47 and 0.6 TPR at 10^{-2} FPR, respectively), while our shape model lags behind at about 0.26 TPR. We examine flow matching in the absence of loop cover traffic (dataset `no-cover`, idealized network) and find that the effectiveness increases further to an FPR slightly higher than 10^{-6} for the same TPR. Reducing the average delay μ that packets stay at a mix for (`low-delay`, idealized network) increases TPR to 1.0 at 10^{-2} FPR, while increasing μ (`high-delay`, idealized network) leads TPR to fall to circa 0.13. Our simulated dataset `two-to-one` (idealized network) shows that our classifiers retain utility even when positive and negative samples share endpoints in larger sets. The sixth dataset (`live-nym`, real-world network) evaluates the threat in the deployed mainnet Nym in default configuration. Network effects like packet losses violate our classifiers' assumptions and cause performance to degrade significantly: roughly 0.28 (statistical), 0.26 (drift), and 0.03 (shape) TPR at 10^{-2} FPR. We outline countermeasures to our attacks and highlight our approach's shortcomings (Section 9.8). We summarize our main contributions as follows:

- We propose a *flow merging* technique that enables evaluating flow matching in mixnets, given a pair (or closed set) of endpoints communicating exclusively with each other.
- We empirically assess flow matching in different settings of state-of-the-art mixnet Nym. We examine three classifiers (statistical, drift, shape) that can be used after applying our flow merging technique. We find that all three classifiers perform well in a variety of evaluated Nym configurations, but degrade once communication is not loss-free or exclusive. We propose countermeasures to mitigate flow matching threats based on the insights derived from the analysis.
- We develop a method to collect metadata on Nym in an ethical way as well as six datasets of the same file download scenario in different Nym and network configurations. We make this work's source code and datasets available [1].

9.2 Background and Related Work

9.2.1 Flow Correlation Against Tor

Fixed-sized data packets and per-hop cryptographic transformations prevent adversaries observing *both ends* of a packet exchange over connection-based anonymity systems like Tor (we give a brief overview of relevant aspects of Tor in Section 9.A) from trivially deanonymizing it based on packet content or size. There remain, however, observable features of traffic flows that can be exploited: the number, direction, and timings of transmitted packets. *Flow correlation* attacks consider a passive adversary that observes traffic in the links between the endpoints and the anonymity network, without having visibility of traffic between network intermediaries. The adversary records traffic features observed in the available flows and tries to determine which endpoints are communicating with each other by analyzing correlations between flows. The susceptibility of connection-based anonymity systems to this type of attack has long been known [4, 12, 29, 30, 35] and, in fact, Tor is explicitly not designed to withstand such an attack [11]. Nonetheless, flow correlation represents a relevant threat to users of these systems and has thus been studied extensively.

We review early flow correlation attacks that rely on traditional statistical methods to identify correlated flows in Section 9.B. State-of-the-art methods rely on deep learning techniques, first introduced by DeepCorr [20]. In contrast to other approaches, deep learning methods do not require feature engineering by human experts, but instead extract the dataset's features on their own. This allows to identify useful features potentially indiscernible to humans. DeepCorr processes packet timestamps (as *inter-packet delays*) and packet sizes from both connection ends in a *convolutional neural network* (CNN) architecture. Impressively, for a *false positive rate* (FPR) of 10^{-2}, DeepCorr achieves a *true positive rate* (TPR) of roughly 0.9 compared to circa 0.3 of RAPTOR [34], the previous state-of-the-art attack based on traditional statistical methods.

Oh et al. [24] introduced flow pair correlator DeepCoFFEA, which surpasses DeepCorr's performance and practicality by way of two main improvements. First, DeepCoFFEA transforms both traces of a candidate pair into a low-dimensional space where correlated traces are located closer together (according to a distance metric) than uncorrelated traces, and comparison is fast. This is learned by minimizing a *triplet loss* [15] between embeddings of an initiator trace (*anchor*), a correlated responder trace (*positive*), and an uncorrelated responder trace (*negative*). Second, DeepCoFFEA partitions each trace of a candidate pair into windows and arrives at the pair's correlation decision by aggregating window-by-window embedding distances in order to lower the FPR. As a result, DeepCoFFEA outperforms DeepCorr significantly, with a TPR of roughly 0.98 compared to DeepCorr's circa 0.3 at an FPR close to 10^{-3}.

9.2.2 Mixnets and Nym

In contrast to onion routing networks like Tor, mixnets [6] aim to provide anonymity against a *global passive adversary* who observes *all* network links, including both ends of a communication, and is thus capable of conducting flow correlation attacks. Mixnets differ in two key ways from onion routing networks. First, mixnets route each packet independently, using fresh cryptographic key material and a randomly chosen path of intermediaries through the network each time. They are thus *packet-oriented* rather than *connection-oriented* (where packets are explicitly associated to a circuit for all intermediaries in the path), meaning that an adversarial intermediary cannot tell whether or not two packets belong to the same flow. Second, packets are artificially delayed at each intermediary (called *mix node*) before being passed on. The delays alter the flow of packets routed by mix nodes so that their inputs and outputs cannot be linked based on packet order, and may also cause packets within a flow to be received out of order. Per-node packet reordering can be achieved in different ways. Many mixnet designs have nodes assemble incoming packets into *batches* before shuffling and forwarding them [5, 6, 7, 14, 16, 19]. On the other hand, *continuous-time mixes* delay packets individually, typically by a duration randomly drawn from an exponential distribution [17, 28], to achieve a similar effect.

The Nym Network. Nym [10, 23] is a recently deployed system based on continuous-time mixnet Loopix [28] intended for general-purpose traffic relaying (e.g., messaging, cryptocurrency access). We use Nym as the reference mixnet for this work as it is a state-of-the-art design and a deployed system with available source code [22]. We analyze Nym in git repository versions `nym-binaries-1.0.2` and `nym-binaries-v1.1.13`, with all described Nym behavior in this work as of the latter version unless otherwise noted. Endpoints communicating via Nym are associated to *gateways* that act as interfaces with the mixnet. Application data sent through Nym is fragmented and padded into one or more Sphinx [9] packets of fixed size (2413 bytes). Packet paths start at the sender's gateway, traverse three randomly selected mixes, and end in the receiver's gateway. For each intermediate mix, the sending endpoint samples a delay from an exponential distribution with mean $\mu = 50$ ms and encodes the value in the packet header, such that the mix will retain the packet for that amount of time before forwarding it along the path. Data packets contain an *acknowledgment* packet (386 bytes in size) that is extracted by the receiving gateway and sent to the mixnet, which routes it back to the sender via a dedicated fresh path independent of its data packet's path. Acknowledgments are delayed at each intermediate mix also following an exponential distribution with mean μ (i.e., 50 ms). Given that the sender selects all the per-mix delays for the packet and the acknowledgment, it can estimate when the acknowledgment should be received. If either the packet or the acknowledgment is lost in transit, the sending endpoint retransmits the data after a timeout.

With the goal of achieving unobservability [27], Nym endpoints regulate the times at

which they send packets to their gateways. The emission schedule is modeled as a Poisson process that on average sends 50 packets per second, with inter-packet delay drawn from an exponential distribution with average $\lambda_R = 20$ ms. Each time the emission schedule indicates that a packet should be sent, the endpoint checks if an application-level packet is available for sending, and if so, sends it out. If no application-level packet is ready, the endpoint instead generates a *loop cover packet* with throwaway payload addressed to itself. Loop cover packets are routed like regular packets and are indistinguishable for all entities except the sender. This way, an adversary monitoring the link between endpoint and gateway is unable to tell whether the endpoint is sending any "real", application-level packets or not. In addition to these "gap-filling" loop cover packets, endpoints also send loop cover packets at all times (independent of application traffic), according to a second Poisson process with average inter-packet delay $\lambda_C = 200$ ms.

9.2.3 Flow Correlation Against Mixnets

Danezis [8] presents an attack on the anonymity provided by a simulated network of continuous-time mixes with exponential delay, assuming that packets are associated to flows as they traverse the mixnet. The problem tackled in this work considers a more general and harder attack setting where all packets are independently routed and not associated to flows. We discuss how our results extend prior work in Section 9.4. Zhu et al. [40] evaluate flow correlation for batching mix strategies considering a single mix, four endpoints, and no cover traffic—a very simplistic setup compared to deployed systems such as Nym. Thus, while prior work on flow correlation against connection-based anonymity systems is plentiful, we currently lack methods to evaluate this threat in realistic mixnet setups that are packet-oriented.

9.3 Flow Matching on Mixnets

9.3.1 System Model

We consider a set of *endpoints* (clients or network services) that communicate through the Nym mixnet via a *gateway* of their choice. We focus on pairwise communications between two endpoints, but note that our methods apply to closed communication groups that involve any number of endpoints (Section 9.3.3). As convention to distinguish the two endpoints, we denote the sender of the first packet in the exchange as *initiator* I, while the receiver of that first packet is *responder* R. The packets from I to R are considered to flow *upstream*, while the packets from R to I travel *downstream*. Figure 9.1 illustrates our abstract system model, where endpoints I and R communicate with each other via their respective gateways G_I and G_R. We call *ingress* the traffic sent from a

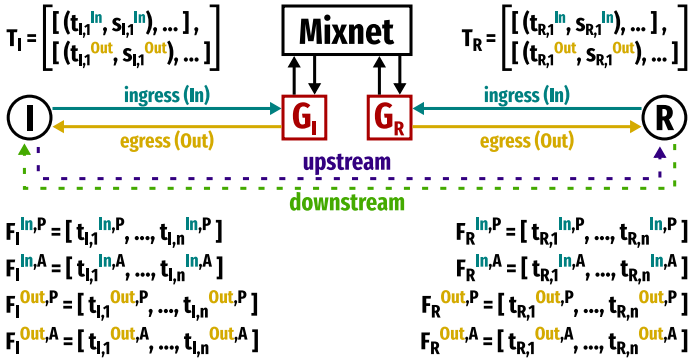


Figure 9.1: Traces recorded by an adversary that controls gateways G_{I} and G_{R} and wants to determine whether endpoints I and R are communicating.

gateway to the mixnet and *egress* the traffic received by a gateway from the mixnet. We assume that I and R are communicating exclusively with each other and that the same gateways (G_{I} and G_{R}) are used during their entire communication session.

In practice, endpoints in Nym connect to gateways via WebSocket, a bidirectional, ordered, message-oriented protocol. We call the bidirectional packet exchange of the WebSocket session between endpoint $E \in \{\mathrm{I}, \mathrm{R}\}$ and its gateway G_E a *trace* $T_E = (T_E^{\mathrm{In}}, T_E^{\mathrm{Out}})$, where T_E^{In} and T_E^{Out} denote the sequences of ingress and egress Sphinx packets, respectively. For each packet in these directional sequences, the trace records its timestamp and size (normal packet or acknowledgment). The entire trace T_E observable by G_E as the result of acting as network interface for E is split into four flows. For endpoint E, packet type $M \in \{\mathrm{P}, \mathrm{A}\}$, and packet direction $D \in \{\mathrm{In}, \mathrm{Out}\}$, let flow $F_E^{D,M}$ denote the sequence of packet timings in T_E of type M and direction D.

9.3.2 Threat Model

Adversarial Capabilities. We consider an adversary that runs one or more Nym gateways. An adversarial gateway logs the observed traces T_E for all of the endpoints E it interacts with and makes these traces available for adversarial analysis. The attack is *passive*, i.e., malicious gateways do not delay, modify, inject, or drop packets. They follow the packet transmission protocols perfectly and only deviate from honest behavior by logging packet information. We assume the mixnet nodes and endpoints to be *honest*, i.e., they follow the protocols and do not collude with the adversary.

Depending on how endpoints select gateways, it may be easier or harder for the adversary to become the gateway of target endpoints of interest. The adversary may have to gain

a target's trust if the gateway selection is manual, wait until it "gets lucky" if gateways are regularly reassigned in a randomized way, or optimize its parameters in case of automated selection that follows criteria such as proximity to the target, uptime, or server capacity. Regardless of the ease or difficulty of targeting specific endpoints, we consider that the adversarial gateways are used by at least two endpoints, allowing the adversary to assess whether any pair of observable endpoints are communicating with each other. Note that we require the adversary to simultaneously observe the traces of *both* candidate endpoints I and R to deploy the attack. There is, however, no requirement for the adversary to observe the traces of *other* endpoints that may also be communicating with each other through the mixnet at the same time as I and R.

Compared to an alternative adversary that records packet metadata at the communication link level, our gateway adversary obtains packet metadata at the Sphinx layer with much less noise and thus less necessary processing before analysis. This and the fact that becoming a Nym gateway is relatively accessible to a wide range of actors, leads us to focus on this type of adversary for this work.

Flow Correlation Versus Flow Matching. In Tor, a malicious guard and exit (first and last intermediaries) colluding to determine if they are part of the same circuit can check whether the observed flow of packets $F_{\mathrm{I}}^{\mathrm{In}}$ matches the received flow $F_{\mathrm{R}}^{\mathrm{Out}}$, as *every* packet in the input flow *must* appear in the output after a small delay. As circuits are bidirectional, the adversary similarly checks whether $F_{\mathrm{R}}^{\mathrm{In}}$ matches $F_{\mathrm{I}}^{\mathrm{Out}}$. The timing of circuit establishment and destruction at both ends is an additional feature to determine that the flows correspond to the same circuit. Furthermore, if an endpoint is simultaneously engaged in multiple connections routed through multiple circuits, the traffic sent or received via other circuits does not obfuscate the observed flows F_E^D with spurious packets that may hinder flow correlation. Thus, a pair of traces $T_{\mathrm{I}} = (F_{\mathrm{I}}^{\mathrm{In}}, F_{\mathrm{I}}^{\mathrm{Out}})$ and $T_{\mathrm{R}} = (F_{\mathrm{R}}^{\mathrm{In}}, F_{\mathrm{R}}^{\mathrm{Out}})$ are either a match or not a match, and there is no such thing as a "partial" match in this setting and threat model.

In mixnets, on the other hand, the traffic flows observed by gateways aggregate independently routed packets that may belong to different connections and it is not possible to distinguish which packet belongs to which end-to-end connection. In Nym, even if endpoints I and R are communicating exclusively with each other, the loops of cover traffic sent by clients ensure that some of the packets seen in input flow $F_{\mathrm{I}}^{\mathrm{In}}$ will appear in the same endpoint's output flow $F_{\mathrm{I}}^{\mathrm{Out}}$ and *not* be received at the other endpoint as part of $F_{\mathrm{R}}^{\mathrm{Out}}$. Attempting to match $F_{\mathrm{I}}^{\mathrm{In}}$ to $F_{\mathrm{R}}^{\mathrm{Out}}$ and $F_{\mathrm{R}}^{\mathrm{In}}$ to $F_{\mathrm{I}}^{\mathrm{Out}}$—the approach to evaluate this attack in Tor—is thus not adequate for the Nym mixnet.

When considering endpoints that are communicating exclusively with each other, all the packets in their ingress flows will appear in *one* of their egress flows (unless they are lost in transit), either to themselves in the case of loop cover packets or the counter-party in the case of data transmissions. We call this problem—where all ingress packets are represented as egress packets—*flow matching*. We leave the more complex problem of

partial flow correlation for future work, where some fraction of packets in the observed ingress flows has no corresponding output in any of the observed egress flows and vice versa, as some packets received in egress flows were sent by endpoints that are not under observation by the adversary. With this more complex problem in mind, flow matching can also be understood as *complete flow correlation*.

9.3.3 Endpoints Closed Sets and Flow Merging

As shown in Figure 9.1, the adversary collects a bidirectional trace $T_E = \{T_E^{\text{In}}, T_E^{\text{Out}}\}$ for each endpoint $E \in \{\text{I}, \text{R}\}$. Each directional trace (In, Out) is a sequence of packets characterized by their timestamp and size, which depends on whether it is a *payload* (P) or an *acknowledgment* (A) packet. The egress flows $F_E^{\text{Out,P}}$ and $F_E^{\text{Out,A}}$ are obtained by splitting T_E^{Out} into two vectors according to packet type $\{\text{P}, \text{A}\}$. The adversarial gateway obtains the ingress payload flow $F_E^{\text{In,P}}$ from T_E^{In}, which contains all the packets sent by E. Ingress acknowledgments are sent by the gateway itself immediately upon receiving and processing a payload packet, rather than received from endpoint E. Thus, the ingress acknowledgment flow is simply a copy of the egress payload flow, i.e., $F_E^{\text{In,A}} = F_E^{\text{Out,P}}$.

In flow matching, the goal is to determine whether a set of ingress flows matches a set of egress flows. Considering a single endpoint E, a match between $\{F_E^{\text{In,P}}, F_E^{\text{In,A}}\}$ and $\{F_E^{\text{Out,P}}, F_E^{\text{Out,A}}\}$ would reveal that E is not communicating with anyone, but just sending loop cover traffic to itself, which we call a *loner*. Given a pair of endpoints I and R, let $F_{\{\text{I,R}\}}^{\text{In}}$ be the combination of the ingress flows for both parties (separately for each type P and A, which we omit in the following for convenience) and $F_{\{\text{I,R}\}}^{\text{Out}}$ be the combination of the egress flows. A positive match between $F_{\{\text{I,R}\}}^{\text{In}}$ and $F_{\{\text{I,R}\}}^{\text{Out}}$ indicates that I and R are communicating with each other.

Generalizing to any number of parties, let $\mathscr{E} = \{E_1, \ldots, E_N\}$ be a set of endpoints communicating via the mixnet. Flow matching can be applied whenever the *endpoints closed set condition* is satisfied, i.e., all endpoints $E_i \in \mathscr{E}$ communicate with other endpoints in the set (including themselves), but do not exchange (send or receive) packets with endpoints not in $\mathscr{E}$. If this is the case, then the combination of egress flows must contain the same packets included in the combination of ingress flows. We define the combined flows as:

$$F_{\mathscr{E}}^{\text{In}} = \biguplus_{E_i \in \mathscr{E}} F_{E_i}^{\text{In}}\,; \quad \text{and} \quad F_{\mathscr{E}}^{\text{Out}} = \biguplus_{E_i \in \mathscr{E}} F_{E_i}^{\text{Out}}, \tag{9.1}$$

where $\uplus$ stands for *multiset union and reordering*, which we call *merging* for convenience and illustrate in Figure 9.2. In order to test whether $F_{\mathscr{E}}^{\text{In}}$ and $F_{\mathscr{E}}^{\text{Out}}$ match, the adversary needs to build a detector that is able to distinguish matched from unmatched ingress

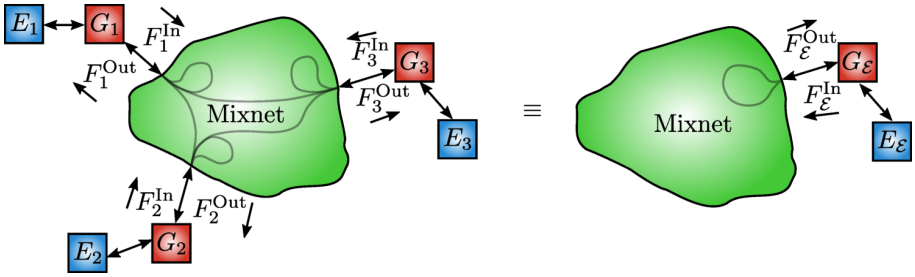


Figure 9.2: *Flow merging* in a closed set $\mathscr{E} = \{E_1, E_2, E_3\}$. Then, $F_{\mathscr{E}}^D = F_1^D \biguplus F_2^D \biguplus F_3^D$, with $D \in \{\text{In}, \text{Out}\}$.

and egress flows. Formally, this detector can be defined as follows:

$$d_{\text{Ideal}}\left(F_{\mathscr{E}}^{\text{Out}}, F_{\mathscr{E}}^{\text{In}}\right) = \begin{cases} 1 & \text{if} \quad F_{\mathscr{E}}^{\text{Out}} \parallel F_{\mathscr{E}}^{\text{In}} \\ 0 & \text{if} \quad F_{\mathscr{E}}^{\text{Out}} \nparallel F_{\mathscr{E}}^{\text{In}} \end{cases}, \tag{9.2}$$

where $\parallel$ stands for *matches* and $\nparallel$ stands for *does not match*.

Focusing on the concrete case where two endpoints I and R communicate with each other, it only holds if:

$$d_{\text{Ideal}}\left(F_{\mathscr{E}}^{\text{Out}}, F_{\mathscr{E}}^{\text{In}}\right) = 1, \tag{9.3}$$

$$d_{\text{Ideal}}\left(F_{\text{I}}^{\text{Out}}, F_{\text{I}}^{\text{In}}\right) = 0, \text{ and} \tag{9.4}$$

$$d_{\text{Ideal}}\left(F_{\text{R}}^{\text{Out}}, F_{\text{R}}^{\text{In}}\right) = 0, \tag{9.5}$$

where $\mathscr{E} = \{\text{I}, \text{R}\}$. In other words, the adversary determines that endpoints I and R form a closed set and are not loners.

Given multiple packet types, the endpoints closed set condition must hold for all types. Thus, the detector is applied separately to both payload and acknowledgment packets. The ideal detector shown in Equation (9.2) provides the true answer considering that both flows are matching. Practical detectors built by the adversary thus must combine the evidence provided by the separate matching of both payload and acknowledgment flows. There are different methods an adversary may use to approximate the ideal detector of Equation (9.2). In this paper, we explore three relevant ones: a *statistical classifier* (Section 9.4), based on the adversary's a priori knowledge regarding the statistical nature of the delays introduced by mixnets in matched ingress and egress flows, and two more general *deep learning classifiers* (Section 9.5), called drift and shape, which learn this knowledge directly from collected data.

9.3.4 Packet Alignment

As a practical consideration, when the adversary analyzes ingress and egress flows, it is important to properly align them in order to ensure that the distribution of the drift exhibited by matched pairs is always the same. If both ingress and egress flow capture packets starting at the same global time, the first packets in the egress flow may correspond to packets sent before the capture process started, hence not included in the observed ingress flow. These packets should be discarded. In practice, the adversary can align both time series to ensure that the average difference between same-index egress and ingress timestamps approximates the average delay introduced by the mixes plus other processing and transmission delays. Once this is done, the non-overlapping parts of the aligned flows are cropped to ensure same length. In all our experiments, this alignment is done considering the full flows captured during the experiment duration.

9.4 Statistical Classifier

The anonymity of continuous-time mixes has been modeled before by Danezis [8]. Assuming that the incoming traffic is a Poisson process, the anonymity set size of packets traversing a continuous-time mix is given by the differential entropy of its inverse delay characteristic function plus the logarithm of the incoming traffic average inter-packet delay λ^{In}, i.e.:

$$\mathcal{A} = d\mathcal{E}\left\{f'\left(t^{\text{In}} \mid t^{\text{Out}}\right)\right\} + \log \lambda^{\text{In}}, \tag{9.6}$$

where $d\mathcal{E}\{\cdot\}$ is the differential entropy operator, and $f'\left(t^{\text{In}} \mid t^{\text{Out}}\right)$ is the inverse delay characteristic function, which is the probability density describing the likelihood a message being ejected at time t^{Out} was injected into the mix at time t^{In}. In the case of a multi-hop mixnet, if we assume for simplicity that network propagation delays are negligible compared to mixing delays, then the inverse delay characteristic is the probability density function (pdf) of an Erlang random variable with shape r and rate ψ, denoted as $E(r, \psi)$, where r is the number of mix nodes in a packet's route through the mixnet and $\psi = \mu^{-1}$ is the inverse of the average of the exponential delay distribution introduced at each mix node. Rather than characterizing anonymity, our goal is to build effective statistical tests that distinguish matched from unmatched sets of ingress and egress flows corresponding to endpoints in $\mathcal{E}$, thus approximating the ideal detector defined in Equation (9.2) as much as possible.

9.4.1 Statistical Detector

Let us denote the pdf of $F_{\mathcal{E}}^{\text{Out}}$ given $F_{\mathcal{E}}^{\text{In}}$ and $F_{\mathcal{E}}^{\text{Out}} \parallel F_{\mathcal{E}}^{\text{In}}$ as:

$$f^{+}\left(F_{\mathcal{E}}^{\text{Out}} \middle| F_{\mathcal{E}}^{\text{In}}\right) = f_{F_{\mathcal{E}}^{\text{Out}}}\left(F_{\mathcal{E}}^{\text{Out}} \middle| F_{\mathcal{E}}^{\text{In}}, F_{\mathcal{E}}^{\text{Out}} \parallel F_{\mathcal{E}}^{\text{In}}\right),$$

and its corresponding likelihood as:

$$L\left(F_{\mathscr{E}}^{\text{Out}} \,\middle|\, F_{\mathscr{E}}^{\text{In}}, F_{\mathscr{E}}^{\text{Out}} \parallel F_{\mathscr{E}}^{\text{In}} ; \Theta\right) = L^{+}\left(F_{\mathscr{E}}^{\text{Out}} \,\middle|\, F_{\mathscr{E}}^{\text{In}} ; \Theta\right),$$

where Θ is the set of parameters of f^{+} and packet type has been suppressed for convenience.

The different possible scenarios for the non-matched case are too many, so the adversary can focus on modeling $f^{+}_{F^{\text{Out}}}$ to build its statistical test and ignore the cases where $F_{\mathscr{E}}^{\text{Out}} \nparallel F_{\mathscr{E}}^{\text{In}}$. For matched flows, each timestamp in $F_{\mathscr{E}}^{\text{Out}}$ matches another timestamp in $F_{\mathscr{E}}^{\text{In}}$, as egress packets are just ingress packets delayed by the mixnet. Therefore, if we assume there are no propagation or processing delays and no packet losses, in mixnets with exponential delays, there exists a permutation of the ingress packet indexes $\mathcal{P}(n_{\mathscr{E}}^{\text{In}}) = \left(p_1, \ldots, p_{n_{\mathscr{E}}^{\text{In}}}\right)$ such that $t_{\mathscr{E},p_k}^{\text{Out}} - t_{\mathscr{E},k}^{\text{In}} \sim E\left(r, \psi\right) \; \forall k$.

Under these assumptions, it is possible to compute the exact likelihood of $F_{\mathscr{E}}^{\text{Out}}$ given $F_{\mathscr{E}}^{\text{In}}$ and $F_{\mathscr{E}}^{\text{Out}} \parallel F_{\mathscr{E}}^{\text{In}}$ (i.e., the evaluation of $f^{+}_{F^{\text{Out}}}$ for some given parameters of the mixnet) by using a recursive procedure based on the following:

$$L^{+}\left(F_{\mathscr{E}}^{\text{Out}} \,\middle|\, F_{\mathscr{E}}^{\text{In}} ; r, \psi\right) = \sum_{l \mid t_{\mathscr{E},l}^{\text{In}} < t_{\mathscr{E},k}^{\text{Out}}} f_{E(r,\psi)}\left(t_{\mathscr{E},k}^{\text{Out}} - t_{\mathscr{E},l}^{\text{In}}\right) L^{+}\left(F_{\mathscr{E}}^{\text{Out}-k} \,\middle|\, F_{\mathscr{E}}^{\text{In}-l} ; r, \psi\right), \quad (9.7)$$

where $f_{E(r,\psi)}$ is the pdf of an $E\left(r, \psi\right)$ random variable with $r = 3$ and $\psi = 1/50$ms in the case of Nym, and $F_{\mathscr{E}}^{D-q}$ is the flow resulting from removing $t_{\mathscr{E},q}^{D}$ from $F_{\mathscr{E}}^{D}$ with $D \in \{\text{In}, \text{Out}\}$. Although this exact computation removes the impossible correspondences, it still requires impractical computational power for even modest flow lengths as it only evaluates permutations where $t_{\mathscr{E},p_k}^{\text{Out}} > t_{\mathscr{E},k}^{\text{In}} \; \forall k$. Therefore, the adversary is forced to rely on a practical approximation. Also, this computation would differ if we introduced network delays. Instead of evaluating all possible orders of packets, the adversary can use a strategy robust against packet reordering by paying attention to the drift of output and input flows. If ingress and egress flows are aligned, the difference of egress and ingress timestamps with the same index follows a stationary distribution for matched flows, but randomly drifts away from this distribution for unmatched flows. In an idealized case without reordering between the ingress and egress flows, i.e., $p_k = k$ in the index permutation, this strategy would be equivalent to the exact computation.

For convenience, let us define the logarithmic scoring function $f_{\text{stat}}\left(F_{\mathscr{E}}^{\text{Out}}, F_{\mathscr{E}}^{\text{In}}\right)$ as:

$$f_{\text{stat}}\left(F_{\mathscr{E}}^{\text{Out}}, F_{\mathscr{E}}^{\text{In}}\right) = \ell^{+}\left(F_{\mathscr{E}}^{\text{Out}} \,\middle|\, F_{\mathscr{E}}^{\text{In}} ; r, \psi\right),$$

where ℓ^{+} is the loglikelihood function of matched flows and stat is short for statistical. The adversary can use the matched flow stationary drift distribution to approximate

this scoring function as:

$$f_{\text{stat}}\left(F_{\mathscr{C}}^{\text{Out}}, F_{\mathscr{C}}^{\text{In}}\right) \approx \sum_{l} \log\left[f_{\Delta(\Gamma)}\left(t_{\mathscr{C},l}^{\text{Out}} - t_{\mathscr{C},l}^{\text{In}}\right)\right], \tag{9.8}$$

where $f_{\Delta(\Gamma)}$ is the pdf of the drift (delay of packets with the same index) in aligned matched flows and Γ is the set of parameters of this pdf. The distribution of the drift, given that reordering happens, differs from the $E(r, \psi)$ mixnet delay random variable regardless of the propagation delays. This distribution has a much lighter right tail than the Erlang delay introduced by the mixnet in individual packets, due to packet reordering that happens in mixnets. The adversary must approximate the actual $f_{\Delta(\Gamma)}$ from its own collection of matched flows. In this study, we use a Gaussian approximation, i.e., $\Delta(\Gamma) \sim \mathcal{N}(\eta_\Delta, \sigma_\Delta)$, where η_Δ and σ_Δ are the sample mean and standard deviation of the delays between packets with the same index in matched flows measured by the adversary.

As this score (which is also a loglikelihood) yields higher values for matched than for unmatched flows and the target value also depends on the adversary's objective and the flows' length, the adversary builds their Ingress/Egress Flow Matching hypothesis test as follows:

Egress flow $F_{\mathscr{C}}^{\text{Out}}$ matches ingress flow $F_{\mathscr{C}}^{\text{In}}$ if, and only if, the logarithmic score is above a threshold that depends on the length of the flows and the target working point Ω, i.e.:

$$F_{\mathscr{C}}^{\text{Out}} \parallel F_{\mathscr{C}}^{\text{In}} \iff f_{\text{stat}}\left(F_{\mathscr{C}}^{\text{Out}}, F_{\mathscr{C}}^{\text{In}}\right) \geq \theta\left(n, \Omega\right),$$

or equivalently:

$$d_{\text{stat}}\left(F_{\mathscr{C}}^{\text{Out}}, F_{\mathscr{C}}^{\text{In}}\right) = \begin{cases} 1 & \text{if} \quad f_{\text{stat}}\left(F_{\mathscr{C}}^{\text{Out}}, F_{\mathscr{C}}^{\text{In}}\right) \geq \theta\left(n, \Omega\right) \\ 0 & \text{if} \quad f_{\text{stat}}\left(F_{\mathscr{C}}^{\text{Out}}, F_{\mathscr{C}}^{\text{In}}\right) < \theta\left(n, \Omega\right) \end{cases} \tag{9.9}$$

9.4.2 Combination of Payload and Acknowledgment Flow Scores

As a fresh mixnet route is sampled for each acknowledgment packet attached to a payload packet (Section 9.2.2), the scores for payload and acknowledgment flows obtained by the adversary are statistically independent. Together with the fact that these scores are loglikelihoods, the adversary can compute the combined score $f_{\text{stat}}^{\text{comb}}$ as:

$$f_{\text{stat}}^{\text{comb}}\left(F_{\mathscr{C}}^{\text{Out}}, F_{\mathscr{C}}^{\text{In}}\right) = \sum_{M \in \{\text{P,A}\}} f_{\text{stat}}\left(F_{\mathscr{C}}^{\text{Out},M}, F_{\mathscr{C}}^{\text{In},M}\right), \tag{9.10}$$

with each packet-type-specific score computed using Equation (9.9).

9.5 Deep Learning Classifiers

Our machine learning approach to the mixnet flow matching problem is inspired by DeepCoFFEA, the state-of-the-art traffic correlation attack on Tor [24]. Although some of the choices behind DeepCoFFEA's design are also applicable to flow matching in mixnets, we make substantial modifications to the original design to effectively tackle the novel setting. In this section, we describe our approach highlighting the points where it diverges from DeepCoFFEA's.

9.5.1 Deep Learning

Similar to DeepCoFFEA and in contrast to the statistical classifier introduced above, this approach is based on a deep learning (DL) classifier. Instead of explicitly modeling the mixnet's delay characteristic and making decisions based on how well the traffic observations fit the hypothesis that the flows *match*, a DL algorithm aims to find a model f_{DL} with parameters W able to discriminate between *matched* and *unmatched* ingress and egress flows.

To obtain f_{DL}, DL algorithms search for the values of W that minimize a *loss function* l, which penalizes incorrect predictions of f_{DL} on a sample of correctly labeled data. This optimization process is referred to as *training* and hence the dataset used for training is known as the *training set*. The ability of DL algorithms to produce models that generalize well (i.e., perform well on samples that were not included in the training set) indicates good learning capability.

The resulting f_{DL} is a possibly non-linear model that, unlike the statistical classifier, does not assume a specific delay distribution for ingress and egress packets with the same index. Therefore, a DL classifier has the potential to overcome some of the limitations of the statistical classifier—albeit at a higher demand for training data and computational resources.

9.5.2 Input Representation

As discussed in previous sections, we tackle a flow matching problem that is slightly different from the one that DeepCoFFEA was designed to address. Rather than matching a pair of ingress and egress flows for each upstream and downstream direction, here the problem concerns the matching of pairs of merged ingress and egress flows for a type of packet and pair of endpoints communicating exclusively with each other. Following the notation introduced in previous sections, for an endpoint pair $\mathscr{C} = \{I, R\}$, we denote its associated merged ingress and egress flow pair as $F^{\mathrm{In}}_{\mathscr{C}} = F^{\mathrm{In}}_{I} \uplus F^{\mathrm{In}}_{R}$ and $F^{\mathrm{Out}}_{\mathscr{C}} = F^{\mathrm{Out}}_{I} \uplus F^{\mathrm{Out}}_{R}$, omitting packet type for convenience. Just like the statistical

classifier incorporates different packet types into the test (Section 9.4.2), we consider acknowledgments and payload packets as independent samples to train the DL classifiers.

Note that our merged flow representation is especially well-suited to account for Nym's loop cover traffic, which does not degrade matching accuracy when the ingress and egress flows of the various endpoints are merged. As we demonstrate, DeepCoFFEA's performance is severely impaired when Nym endpoints send loop cover traffic (see Figure 9.15 in Section 9.D). Our input representation is robust to loops, as merging ensures that loop cover packets appear in both the ingress and egress flows being matched by the classifier.

9.5.3 Window Partitioning

In line with DeepCoFFEA, we partition the flow pairs into windows to take advantage of the augmentation effect of multiple independent tests [24]. Our windows differ from DeepCoFFEA's in that they are packet-based rather than time-based. A packet-based windowing approach is better suited for flow matching, as for matched flows it is more likely that ingress and egress packets will fall in windows with the same index. To prevent a possible misalignment between the ingress and egress windows, it is crucial that we apply the technique described in Section 9.3.4 per each window index.

We consider windows that may overlap a constant number of packets. We denote by n_w the number of packets per window and by δ the fraction of packets by which each two consecutive windows overlap. For example, the window partitioning of the flow pair $F^{\text{In}}_{\mathscr{E}}$ and $F^{\text{Out}}_{\mathscr{E}}$ for $\delta = 0.5$ and $n_w = 10$, would return the packet indices within the intervals $[1, 10], [6, 15], [11, 20]$, etc., for each flow.

The final representation of a window comprises the inter-arrival times of the window's time sequence, and we treat it as an independent data point to train the DL-based models. Thus, the number of windows is a multiplicative factor of the size of the training set. The large training datasets collected for our experiments allow us to discard the last window of the flows, as it usually contains less than n_w packets. We do so for convenience, but, alternatively, we could have padded them with zeros [24]. To be able to pair every ingress window with an egress window, we also ensure the same number of windows across all training flows by discarding windows. Below, we describe the two types of DL classifiers that we have designed for flow matching: the *drift* network and the *shape* network.

9.5.4 Drift Network

The drift network exploits the same information as the statistical classifier. It takes in flow pair $F^{\text{In}}_{\mathscr{E}}, F^{\text{Out}}_{\mathscr{E}}$ and computes the sequence $t^{\text{Out}}_{\mathscr{E},k} - t^{\text{In}}_{\mathscr{E},k}$ for $k = 1, \ldots, n$. This

sequence of time differences between egress and ingress flows captures their *temporal drift*, which correlates with whether the flows are matched or not.

Network Architecture. The architecture of the drift network differs significantly from the one of DeepCoFFEA: instead of learning intermediate representations of the flows through metric learning, it learns to classify them directly. Our tests show that such a simple network already performs remarkably well, indicating that metric learning and other dimensionality reduction techniques may only provide marginal improvements.

The drift network is similar to DeepCoFFEA in that it also has several hidden convolutional layers, but it is much smaller, as traffic matching is a simpler learning problem than traffic correlation. In addition, we use average pooling instead of max-pooling due to the importance of the localization of the drift: when the flows are unmatched, as time passes, the ingress and egress flows drift further apart, thus providing more signal to the classifier. We expand on the drift network's architecture and its parameters in Section 9.E.1.

Testing. After training, the resulting drift model f_{DL} returns a score for each ingress–egress window pair that quantifies how confident the model is that the windows match. The adversary sets a threshold on these scores to make a final decision. To aggregate the decisions across all the windows in a flow, we average the window scores and then set the threshold on the average. This is in contrast to Oh et al.'s approach to set the threshold on each individual window score, count the number of windows that match, and then make a decision based on a majority-vote rule. We argue that the majority-vote rule loses useful information at window level, as it does not capture how far (or close) each vote was from the threshold value. Figure 9.14 in Section 9.D shows that DeepCoFFEA performs better with our average aggregation rule. When acknowledgments are available, we aggregate the averaged scores for acknowledgments and payloads before applying the threshold, as two independent inputs for the same test data point.

9.5.5 Shape Network

Instead of exploiting drift information, the shape network attempts to match the ingress and egress inter-arrival timing sequences. Similar to DeepCoFFEA, the shape model is obtained by training a *triplet network* [15], a DL algorithm that we adapted to obtain new representations of the traffic flows that are more efficient for solving the flow matching problem.

Typically, triplet networks take three data points as input: a reference point known as *anchor*, a matching point to the anchor called *positive*, and a non-matching point called *negative*. We denote them as a, x^+, and x^-, respectively. Given a similarity metric S, the training objective of the triplet network is to find the parameters W of a model f_{DL}

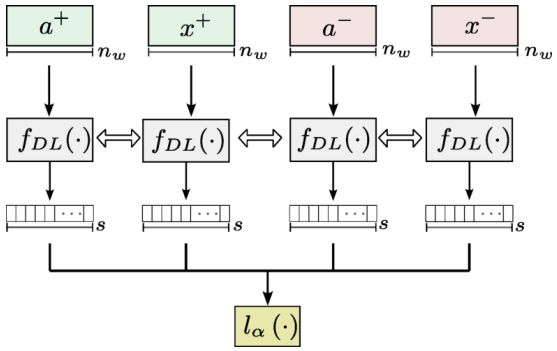


Figure 9.3: Adapted triplet network for the shape classifier. The inputs are the anchor-positive, positive, anchor-negative, and negative windows. The horizontal arrows between the f_{DL} boxes indicate the parameters W are shared among them. f_{DL}'s outputs are the embeddings which are passed to the modified l_α. The sizes of the windows and the embeddings are denoted as n_w and s, respectively.

such that:

$$S(f_{\text{DL}}(a), f_{\text{DL}}(x^+)) > S(f_{\text{DL}}(a), f_{\text{DL}}(x^-)) + \alpha, \tag{9.11}$$

where the model f_{DL} is typically a convolutional neural network with parameters W and α is a slack variable called *margin*. The outputs of f_{DL} are low-dimensional vectors of fixed length, also known as *embeddings*. If f_{DL} satisfies Equation (9.11), its embeddings for matched flows are more similar to each other than those for unmatched flows by at least α, thus allowing to distinguish between them by just comparing their similarities. In practice, it might not be possible to always satisfy Equation (9.11), so we find W that minimizes the hinge loss function l_α, defined as:

$$l_\alpha := \max\left(S(f_{\text{DL}}(a), f_{\text{DL}}(x^-)) - S(f_{\text{DL}}(a), f_{\text{DL}}(x^+)) + \alpha, 0\right).$$

Our novel input representation requires us to modify the objective and the loss functions above. Flows are defined relative to $\mathscr{E}$, creating a dependency in the definition of an "anchor flow" to $\mathscr{E}$. For example, for an endpoint E_1, its matched pair is $\mathscr{E}^+ = \{E_1, E_2\}$, and one of its unmatched pairs may be $\mathscr{E}^- = \{E_1, E_3\}$, resulting in an anchor flow for each $\mathscr{E}^+$ and $\mathscr{E}^-$. We call the anchor flow for $\mathscr{E}^+$ the *anchor-positive* flow, defined as $a^+ := F^{\text{In}}_{\mathscr{E}^+}$, and its corresponding positive flow is defined as $x^+ = F^{\text{Out}}_{\mathscr{E}^+}$. Analogously, we define the *anchor-negative* flow as $a^- := F^{\text{In}}_{\mathscr{E}^-}$ and its negative flow as $x^- = F^{\text{Out}}_{\mathscr{E}^-}$. Therefore, the triplet network objective becomes:

$$S(f_{\text{DL}}(a^+), f_{\text{DL}}(x^+)) > S(f_{\text{DL}}(a^-), f_{\text{DL}}(x^-)) + \alpha, \tag{9.12}$$

and l_α changes accordingly. This is in contrast to DeepCoFFEA's triplet network, which follows the usual single-anchor approach.

In Figure 9.3, we show a high-level view of our modified triplet network, abstracting the internal model f_{DL}. Recall that after window partitioning, the training data points are not flows but windows of size n_w. Note that although the input size of f_{DL} is n_w, the training results in one single f_{DL} model. We denote by s the embedding size, which is a parameter of the network. We follow DeepCoFFEA and use cosine similarity as S and embedding size $s = 64$.

Triplet Mining. Traditional triplet networks improve convergence by *triplet mining*, the search for negative instances that lie within the margin [39]. These negative instances are known as *semi-hard negatives*, as opposed to *hard* negatives, which already satisfy Equation (9.11). Semi-hard negatives are closer to positives and thus help the learning algorithm to focus on differences that are more relevant to satisfying the margin imposed by Equation (9.11).

For $\mathscr{E}^+$ in the previous example, we could could have picked any other non-communicating endpoint to define $\mathscr{E}^-$, indicating that there is room for triplet mining. Similarly to DeepCoFFEA, we compute the pair-wise similarities for all possible negative flows and select those that are within the margin of Equation (9.12). While DeepCoFFEA trains with one semi-hard negative per batch, we feed them all to the classifier. The computational complexity of triplet mining increases quadratically on the number of matched flows. Thus, in our implementation, we distribute the triplet mining computations across batches, calculating similarities for only a small subset of the negative flows at a time.

Network Architecture. The architecture of the network that fits the embeddings in the triplet network and produces the model f_{DL} is almost identical to DeepCoFFEA's. We depict the network architecture and its parameters in Figure 9.17a in Section 9.E.1.

Testing. A shape model also returns a score for whether or not two windows match. More precisely, in the case of a shape model, this score is the cosine similarity between two window embeddings. Unlike DeepCoFFEA, we set a global threshold instead of a local one. For each anchor, Oh et al. take the maximum similarity score in the k-nearest negatives to the anchor as the threshold to discriminate instances relative to that anchor. The pool of negatives from which the final negative is picked includes all possible negatives, thus implicitly assuming that the adversary has visibility over all the egress links of the Tor network. In our threat model, the adversary does not necessarily control all the Nym gateways and therefore may not be able to observe all negative flows. Thus, we use a global threshold in all our evaluations, even though the results of our experiments indicate that a local threshold would provide superior performance (see Figure 9.13 in Section 9.D). More details on the parameters of the network architectures and hyperparameter tuning for the learning algorithms can be found in Section 9.E.

9.6 Data Collection

We consider a file download as application use case for our empirical evaluation of Nym in various setups. We implement a `curl` process (triggered by the *initiator* I) that downloads a file of 1 MiB size via HTTP over Nym from a (default) `python3` HTTP server (the *responder* R). By fixing variables such as file size and endpoint send rate (except responder send rate in the case of dataset `two-to-one`, see below) across experiments, we aim to assess the effect of isolated mixnet features. For this work, we assume the adversary knows when the HTTP request starts and ends within a trace.

We obtain six trace pair datasets to evaluate flow matching in Nym. We collect the first four datasets (called `baseline`, `no-cover`, `low-delay`, and `high-delay`), using a single-host experimental setup based on Docker, with Nym in version `nym-binaries-1.0.2`. We call this the *isolated setup* and depict it in Figure 9.9 in Section 9.C. We simulate a fifth dataset (called `two-to-one`) on the isolated setup by crafting it from `baseline`. The final dataset (called `live-nym`), we collect from the live Nym mainnet using instrumented Nym components in version `nym-binaries-v1.1.13`. This is the *live-network setup* and we show it in Figure 9.10 in Section 9.C.

Isolated Setup. We encapsulate the components of a minimal Nym deployment using Docker containers and orchestrate them via Docker Compose on a single instance running at a public cloud provider, based on a single-host Kubernetes setup shared with us by the Nym development team. Nym in version `nym-binaries-1.0.2` requires at least the following processes to be running for endpoints (*initiator*, *responder*) to exchange packets: a gateway (*gateway*), three mixes arranged in three layers with one mix per layer (*mixnode*), a process maintaining Nym's network state in a blockchain (*validator*), a process providing a network state interface to Nym processes (*validator-api*), and a process initializing Nym's network state (*busybox*). When building the Docker containers for an experiment, we include patches to enable data collection and possibly modify Nym in one aspect to evaluate this aspect's contribution to anonymity in isolation. In all experiments, we patch the gateway to track Sphinx packet metadata (timestamp, size) in both directions (ingress, egress) via the Nym address of the respective endpoint. This models our adversary's metadata gathering capabilities (Section 9.3.2). Endpoint Nym components are compiled with Rust in version `1.63.0` across all isolated setup experiments, while the remaining Nym-compiling Docker images use Rust in version `1.65.0` (`baseline`, `no-cover`) or `1.66.1` (`low-delay`, `high-delay`).

Next, we generate a random file that (appended by a fixed-width and download-specific identifier) is of 1 MiB size. Once all but the endpoint components have started up correctly and formed a functional Nym network, we collect our trace pair dataset. Until we reach a target number of successful HTTP-based downloads of the random file, we spawn a fresh initiator-responder pair (including fresh Nym identities for both Nym endpoints) and connect them to each other via Nym. Our modified gateway will track

their packet metadata via their Nym addresses and we are able to establish the correct endpoint pairs via our experiment orchestration. Right before and right after the actual `curl` request on the initiator, we take the current time to delineate *start* and *end* of the download. We include HTTP header `` `Accept-Encoding: identity' `` to avoid compression distorting the size of the transferred file. We only mark a download as successful (and thus increment our counter towards reaching our target number) if the downloaded file at the initiator is byte-by-byte equal to the served file at the responder.

`baseline`. As our baseline dataset, we collect trace pairs over the isolated setup (i.e., no network effects like propagation delay or packet loss) without any changes to how Nym endpoints send messages. Thus, the default artificial delay introduced by the setup's mixes independently delaying the packets they process is the only influence on packet timings and reorderings. Also applying to all other isolated setup experiments, we backport a patch from the official Nym code base that fixes a buffer reordering bug in Nym endpoints. Additionally for these experiments, we patch the epoch time to be three minutes instead of one hour and apply the before-explained gateway patch to enable Sphinx-level metadata recording.

`no-cover`. In order to assess the contribution of cover traffic towards Nym's claimed privacy protections, we disable "gap-filling" and regular loop cover traffic for this experiment. The only packets in this dataset are thus "real", application-level packets. The remaining additional patches are functionally the same as for `baseline`.

`low-delay` (`high-delay`). Next to applying the same patches as in `baseline`, we reduce (increase) the endpoints' average per-mix delay parameter (μ) for any message sent out, from 50 ms to 20 ms (200 ms). As we keep the packet emission rates (λ_R, λ_C) the same as before, this reduces (increases) the set of other packets that a particular packet can be reordered with when being delayed at a mix. These two experiments thus change the ratio $\mu/\lambda_{T\in\{R,C\}}$, i.e., how long packets dwell at mixes on average in relation to how quickly endpoints send them.

`two-to-one`. In order to evaluate how our classifiers perform once positive and negative flows share endpoints and how flow merging works with larger endpoints sets, we model the setting of two initiators downloading simultaneously from a single (simulated) responder (`two-to-one`). We are interested in two cases: classifier performance when either both (positive) or exactly one (negative) of the two initiators are matched with the responder in the set (*semi-matched*) and either both (positive) or none (negative) of the two initiators are matched with the responder in the set (*unmatched*). Without collecting new data, we simulate these flow pairs by pairing always two responders from `baseline` to become one merged, logical responder and selecting initiators such that we arrive at above specified cases. Mind that we align each initiator-responder pair independently before merging both responders into a single logical responder. Finally, merging the two initiators and the single logical responder created this way, we obtain endpoints sets of size three, and are thus able to gauge

classifier performance on larger endpoints sets that may overlap in endpoints. Note that the merged logical responders in `two-to-one` send messages at twice the default rate of Nym endpoints. This sketches a more realistic "Nym server" deployment where endpoints that only proxy between services and mixnet adjust their send rate linearly with the number of connected Nym endpoints (but this deviates from default Nym).

Live-Network Setup. In order to evaluate the attack in the presence of real-world network effects like network propagation delays and packet loss, we collect the dataset `live-nym` on the available Nym mainnet. This experimental setup only requires running a private gateway modified for capturing traces and an endpoint instance for each simultaneous initiator-responder pair (we run five endpoint pair instances in parallel). We are able to keep our gateway private by not bonding the gateway after initializing it. Bonding is required for any gateway intended for use by regular endpoints. With a small patch to our experiment Nym endpoints, we hard-code our private gateway as the only gateway in the network available to our experiment endpoints. By following this process, we obtain data for our own endpoints and avoid routing traffic for other Nym endpoints. As the traffic we generate in below experiment is indistinguishable from regular Nym traffic, we collect data in an ethical way and avoid negatively impacting the anonymity of unrelated Nym endpoints. As we collect packet timestamps at the gateway and thus exclude the propagation delay of the last hop between the endpoint and gateway, we run our gateway and endpoints instances in the same location. Due to collecting with only five simultaneous endpoint pairs, the additional load caused by our experiment on the Nym network is limited to ten additional Nym clients.

`live-nym`**.** Similarly to the `baseline` dataset, Nym endpoints behave as specified in the Nym system without any modifications. This dataset was collected on the deployed Nym network and thus corresponds to a more recent Nym version (`nym-binaries-v1.1.13`) than the one used in the isolated setup. A difference between both versions is the recently introduced adjustment of the packet emission rate based on backpressure from the gateway. We disable this feature to maintain consistency and better comparability with our isolated setup experiments. Since the Nym version used in our isolated setup, the previously separate `nym-client` functionality was added to the `nym-network-requester` process directly, eliminating the need to run it separately. All Nym components are compiled with Rust in version `1.68.2`.

Post-Processing. Once collected, we run three post-processing steps on each dataset. The first checks the integrity of an experiment's setup and execution. The second produces dataset versions from the initially collected "raw" datasets that are directly usable in either of our classifiers. In line with our assumption that the adversary knows when the HTTP request within a trace pair starts and ends, this step also restricts each trace pair to the duration of the `curl` request (stopwatch symbol in Figure 9.9 in Section 9.C). We plot the duration of each successful file download (i.e., of all collected trace pairs) across datasets in Figure 9.11 in Section 9.C. The third post-processing step partitions each collected dataset into trace pair subsets to be used exclusively for training,

validation, and testing by the classifiers. Each collected dataset contains a little more than 35,000 trace pairs. For consistency across experiments, we first take the top-35,000 trace pairs from the sorted list of trace pair identifiers in lexicographic order. Then, we select the first 24,500 trace pairs for training, the next 5,250 for validation, and the final 5,250 for testing. We make all instrumentation code and all collected datasets publicly available [1].

9.7 Experimental Results

Evaluation Details and Metrics. We make the source code of our statistical classifier (implemented in Octave) and our DL classifiers (implemented in Python using TensorFlow/Keras) publicly available [1]. We split the datasets into training, validation, and testing parts as explained in Section 9.6. The training set is used to obtain classifier parameters, i.e., estimation of the delay probability density function parameters as well as the actual training of our DL classifiers. We use the validation set to tune the hyperparameters of the DL algorithms (more details in Section 9.E.2) and to track how well the models generalize on the validation set during training. Finally, the test set is used to assess the final performance of our classifiers on data excluded from the training and validation sets.

We primarily use the *receiver operating characteristic* (ROC) curves to represent a classifier's performance in terms of its *true positive rate* (TPR) in relation to its *false positive rate* (FPR). An adversarial *observation* refers to the metadata of one payload packet and its corresponding acknowledgment packet. Unless otherwise specified, this section's figures show results for the first 100 observations (i.e., the first 100 payload packets and their corresponding acknowledgments) per flow, after flow pairs have been aligned using all observations from the entire download. By plotting the x-axis in log scale, we focus on the operating points of the curve with low FPRs, as a low FPR is a requirement for the adversary to have high confidence in positive predictions. We also fix the window size of our deep learning classifiers to 100 observations. The classifiers are responsible for converting traces to flows and aligning flow pairs (Section 9.3.4). We report time and hardware requirements for training and evaluating each classifier in Section 9.G.

Impact of Classifier Choice. For Figures 9.4 to 9.7 here and Figures 9.12 and 9.16 in the appendix, we only report the performance of our statistical and our drift classifier. While we include the shape network in this study as it is the most direct adaptation of DeepCoFFEA [24] to the setting of flow matching on mixnets, we do not evaluate it for the above mentioned settings, as its poor performance became clear early on (see Figure 9.8). At an FPR of 10^{-2}, our shape model performs on dataset `baseline` at the level that the other two classifiers reach on the much more challenging dataset `live-nym`. Thus, we focus exclusively on the results for our statistical and our drift

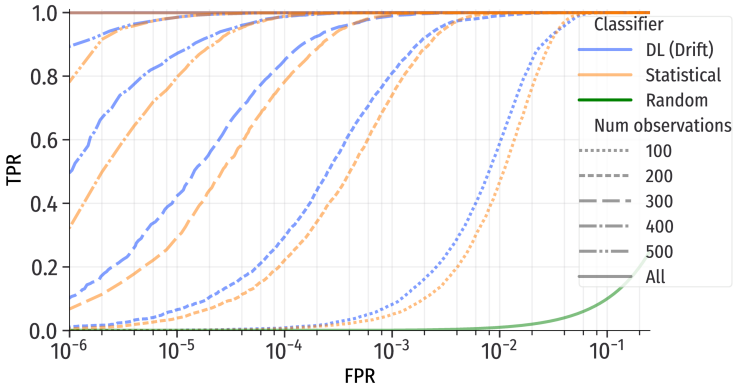


Figure 9.4: Flow matching performance (ROC curves) based on number of observations considered per candidate flow pair of our statistical and our drift classifier on dataset `baseline`.

classifier for most of the below discussed results.

Impact of Distinguishable Packet Type. We start by evaluating the adversarial advantage derived from having two types of packets (payload, acknowledgment) distinguishable by size. We show in Figure 9.12 in Section 9.D that the best results are obtained when both packet types are taken into account, compared to using only either payload or acknowledgment packets. This is due to acknowledgments constituting independent observations of the drift, providing additional useful information to the classifiers. Thus, we always use both packet types going forward.

Impact of Number of Observations. Next, we consider how flow matching performance increases when the adversary has more observations available for matching decisions on dataset `baseline` (i.e., default Nym configuration and no real-world network effects). As Figure 9.4 clearly shows, each increment of 100 additional observations raises performance of either of our two classifiers significantly. Our drift model maintains a lead over the statistical classifier for most operating points on the curves, until they reach TPRs greater than circa 0.9, after which both classifiers either perform equally or the statistical classifier has a slight lead. Overall, flow matching performance on `baseline` is high, roughly 0.6 TPR (drift model) and 0.47 TPR (statistical classifier) at 10^{-2} FPR after 100 observations of flow pairs aligned using the entire download.

Impact of Base Rate (Simultaneous Communications). In Figure 9.16 in Section 9.F, we show how the required number of observations to achieve high classifier performance (TPR ≥ 0.95) with high confidence (precision ≥ 0.95) on `baseline` increases

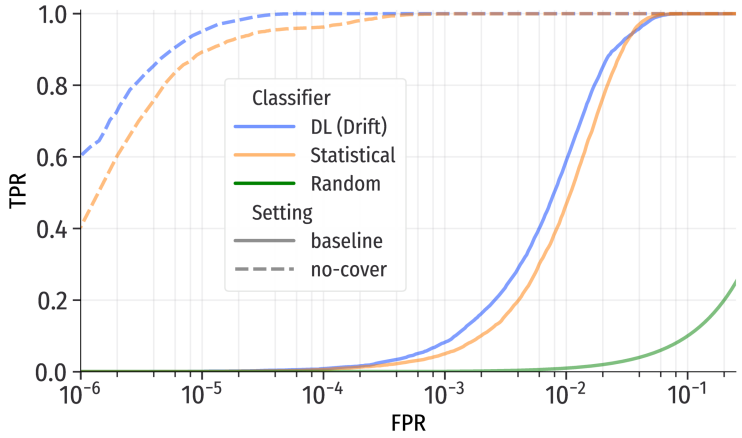


Figure 9.5: Flow matching ROC curves of our statistical and our drift classifiers on the `baseline` and `no-cover` settings.

approximately logarithmically with the number of concurrent communication pairs. This indicates that our attacks scale remarkably well: we only need an additional 100 observations in order to maintain high classification performance at the next higher order of magnitude number of simultaneous one-to-one communications. Both classifiers need to process at least four windows of 100 observations each (circa 314.65 KB downloaded) to achieve the target performance level when faced with 1,000 simultaneous one-to-one packet exchanges.

Impact of Loop Cover Traffic. Nym endpoints send packets following a Poisson process. If no application-level packet is waiting to be sent at the scheduled timeout, a loop cover packet is sent instead to fill the gap. Additionally, endpoints send loop cover packets via a separate slow-rate background Poisson process, regardless of application-level traffic that may be ready. We assess the contribution of loop cover packets to Nym's anonymity by comparing classifier performance on datasets `baseline` and `no-cover`, which differ in the presence of loop cover packets, but are otherwise identical setups. The results are shown in Figure 9.5, where we observe that loop cover packets contribute significantly to thwarting flow matching attacks. As discussed above, `baseline` flow matching performance is high considering the heavy countermeasures Nym employs, but omitting cover traffic has devastating consequences for anonymity: both classifiers reach their `baseline`-level performance at FPRs close to four orders of magnitude lower (slightly higher than 10^{-6} instead of at 10^{-2}).

Impact of Average Per-Mix Delay. By changing the average per-mix delay (μ) that Nym endpoints use when building packets while keeping their inter-emission delays

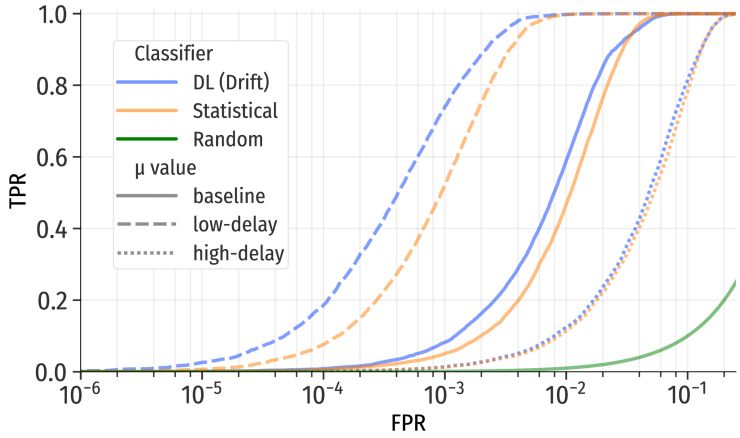


Figure 9.6: Flow matching ROC curves of statistical and drift classifier under $\mu = 50$ ms (`baseline`), $\mu = 20$ ms (`low-delay`), and $\mu = 200$ ms (`high-delay`) average per-mix packet delay.

(λ_R, λ_C) fixed, we modify the number of other packets that a packet at a mix can be swapped with in datasets `low-delay` and `high-delay`. The curves in Figure 9.6 show that both classifiers' performances increase significantly when μ is dropped from 50 ms to 20 ms (reaching 1.0 TPR at 10^{-2} FPR), while their performance diminishes when we increase μ from 50 ms to 200 ms (dropping to circa 0.13 TPR at 10^{-2} FPR). This demonstrates the importance of μ for the threat of flow matching.

Impact of Larger Endpoints Sets. We aim to estimate how classifier performance changes when negative flows share endpoints with positive flows. In Figure 9.7, we show the performance of both classifiers trained on `baseline` but evaluated on the simulated dataset `two-to-one`, i.e., performing flow matching on endpoints sets with three logical endpoints (two initiators, one merged double-rate responder). Flow matching performance remains high in the unmatched setting (without overlap, but larger sets), with circa 0.57 TPR (drift) and circa 0.62 TPR (statistical) at 10^{-2} FPR. In the semi-matched case (where roughly half the packets in merged negative flows overlap with a merged positive flow), both classifiers' performance drops to circa 0.15 TPR at 10^{-2} FPR. However, our classifiers have only analyzed 100 observations at this point and outperform random guessing by a wide margin. Thus, while reduced, utility in this more difficult setting remains significant. We also prove our flow merging technique to handle larger endpoints sets well.

Impact of Internet-Level Network Effects. Finally, we evaluate our flow matching attacks in the presence of real-world network effects (e.g., propagation delays, packet

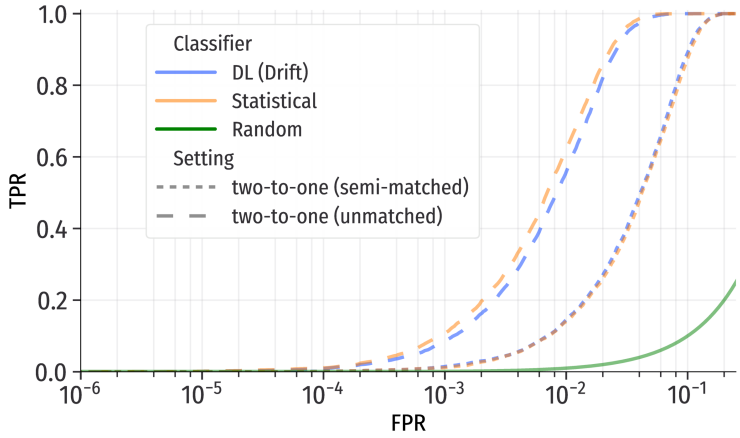


Figure 9.7: Flow matching ROC curves of the statistical classifier and drift model on `two-to-one`, i.e., on endpoints sets of size three simulated from `baseline`, with two initiators and one (faster) responder. Positive samples consist of two matched initiator-responder flow pairs for both cases, while negatives contain either one unmatched initiator (*semi-matched*) or two unmatched initiators (*unmatched*).

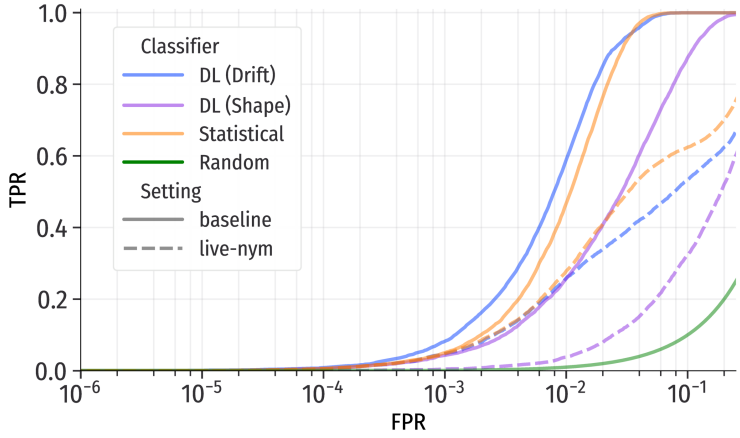


Figure 9.8: Flow matching ROC curves of our statistical classifier, and our drift and shape classifiers without (`baseline`) and with (`live-nym`) real-world network effects.

losses). We do so by comparing results for the `live-nym` dataset collected on the deployed Nym mainnet to `baseline` collected on the isolated setup. Given that propagation time is not negligible anymore in `live-nym`, we increase the average

propagation time in our packet alignment procedure to account for the average aggregate propagation time of four Internet-level links (from the gateway through three mixes back to the gateway). We estimate the mean propagation time of one link by analyzing a 1-hour RIPE Atlas Ping dataset [31] from January 20, 2023. We filter out incomplete measurements, exclude outliers greater than the 99th percentile assuming a well-connected server-to-server network, and use three *round-trip time* (RTT) measurements per probe and destination. We obtain 3,339,429 RTT values that we halve as an estimate for their logical link's propagation time and average to arrive at the final value of 51.103 ms.

As we can see in Figure 9.8, the performance of all our classifiers is significantly degraded under real-world network effects. While significant, this is also unsurprising, as packet losses severely interfere with our classifiers' assumption that all ingress packets must eventually appear in the egress flow. At an FPR of 10^{-2}, our drift model drops from about 0.6 to circa 0.26 TPR, our statistical classifier goes from roughly 0.47 down to circa 0.28 TPR, and our shape model diminishes from circa 0.26 to close to 0.03 TPR.

9.8 Discussion

9.8.1 Flow Matching Effectiveness Against Nym

Under Nym's default configuration (`baseline`) and considering idealized network conditions that exclude propagation delays and packet losses, both our statistical and drift classifiers are able to determine accurately and quickly whether a flow pair is matched or unmatched. Both classifiers retain some utility even on larger endpoints sets with endpoints overlap in positive and negative samples (`two-to-one`). While these idealized network conditions are clearly unrealistic, they enable us to evaluate the role of the foundational anonymity techniques implemented in Nym's mixnet. By isolating the effects of the mixnet's Poisson sending and per-hop exponential mixing delays from random failures and network-related effects (present in `live-nym`), we can assess the distinct contribution that these techniques make to the overall anonymity that Nym offers to packet flows. We disclosed an intermediate as well as the final version of our results to the Nym team so that possible countermeasures (e.g., modifications to the cover traffic scheme) can begin to be considered as early as possible.

Our statistical and drift models outperform random guessing by wide margins. On average, Nym endpoints emit more than 50 packets per second, which means the adversary gathers the required number of observations after only a few seconds. With a few more seconds of eavesdropping on dataset `baseline`, at 500 observations, flow matching performance increases to circa 0.98 TPR at 10^{-5} FPR for both classifiers, while ensuring high confidence in their predictions even when there are 10,000 concurrent

connections. After 14 seconds of observation (slightly more than the `baseline`'s median download duration, Figure 9.11 in Section 9.C), our statistical and drift models essentially exhibit perfect classification.

One of Nym's strengths is its use of loop cover traffic. Without it, flow matching would be much easier and quicker. The choice of average per-mix packet delay parameter μ also impacts flow matching performance strongly: raising it to 200 ms reduces performance significantly. While such level of end-to-end latency increases download times only moderately (15.58 s median, Figure 9.11 in Section 9.C), it might be prohibitive for some intended use cases of Nym. The largest drop in classifier performance, however, is due to real-world network effects that an adversary would encounter when deploying the attack in the wild, rather than an idealized lab setup. Propagation delays between nodes in Nym's overlay network are highly variable and of a magnitude comparable to the randomized mixing delays introduced for anonymity purposes. In practice, these delays introduce a large variability in end-to-end transmission times compared to the single-machine lab setting. Moreover, some packets are lost in transit in the real network. Packet loss is not considered in the statistical classifier's design, the DL classifiers' windowing strategy, or the packet alignment phase (which may be challenging to implement in realistic conditions).

Our statistical and drift classifiers track each other's performance quite closely across the evaluated configurations, with the drift model typically performing slightly better. Both designs rely on the same ingress to egress packet alignment and thus share the same weakness in real-world conditions, where packet losses occur. Our shape model—the classifier architecture closest to state-of-the-art Tor flow correlator DeepCoFFEA [24]—lags far behind in performance compared to our statistical and drift classifiers.

Finally, when contrasting our DL approaches with our statistical approach, we see that the right choice of DL classifier typically achieves slightly higher performance. Our DL approaches require more training data and computational resources than our statistical one, but in comparison are able to incorporate the drift distribution of unmatched flows and generalize better than the statistical classifier. Despite better generality of the drift classifier, the statistical classifier performs slightly better in the `two-to-one` semi-matched setting. This slight advantage of the statistical over the drift classifier may indicate the former has better transferability, i.e., robustness to a shift between the training and test distributions.

9.8.2 Limitations and Assumptions of our Attack

The endpoints closed set condition is the main assumption of our attacks. It allows the adversary to simplify the attack by merging ingress and egress flows, but it can only be applied by an adversary that observes *all* gateways used by *all* endpoints in the

set. The attack may become infeasible if endpoints are concurrently communicating with many other endpoints, some of which may be outside the adversary's observation. Mind, however, that our classifiers do not rely on observing *other* simultaneous but unrelated packet exchanges for making matching decisions, but consider each candidate flow pair independently. Furthermore, the efficacy of the attack decreases if only a low number of packets are available in the exchange, becoming essentially ineffective when the packet exchange between endpoints is very short. From our results here, it is clear that even modest packet loss significantly degrades the performance of our attacks, as the classifiers are implicitly designed for scenarios where all the packets in the flows match. While non-trivial, it seems feasible to include these real-world network effects in classifiers and still rely on the endpoints closed set condition.

Packet alignment also plays an important role for our results. It improves discrimination capability by reducing the variance of the scores of matched flow pairs. We always align flows using all observations from their entire duration. This makes the results shown for lower numbers of packets slightly optimistic. Also, direct comparison between `baseline` and `two-to-one` is cumbersome, as the influence of alignment of both flow pairs accumulates in the latter, increasing the impact of this optimistic effect.

While some of our assumptions give an advantage to the adversary, others make flow matching more difficult than it could be in reality. Clearly, our base setting of each endpoint communicating exclusively with one other endpoint only captures some ways in which Nym may be used, with more sophisticated use cases posing a greater challenge to the adversary. The assumption that the adversary knows exactly when each file download starts and ends within a trace pair also reduces the attack's difficulty. At the same time, we consider a setting where equal-sized files are downloaded at the same time, making the experiment more challenging than in realistic use cases where endpoints transfer payloads of varying sizes and at different times.

9.8.3 Countermeasures

When the endpoints closed set condition holds, our classifiers are able to discriminate between pairs of endpoints communicating exclusively with each other and pairs of endpoints not communicating at all. As shown in Figure 9.15, loop cover traffic is effective in thwarting existing approaches such as DeepCoFFEA, which assume that all packets in a flow travel in either the upstream or the downstream direction—an assumption that no longer holds with loop cover packets. Loop cover packets, however, fail to provide effective protection against our approach, where they are accounted for by merging the sets of ingress and egress flows of the endpoints in the closed communication set.

An objective for an effective countermeasure is to invalidate the endpoints closed set condition. Our `live-nym` results demonstrate that even the minor deviations from

this condition caused by a small fraction of packets being lost in transit significantly decrease flow matching accuracy. Thus, a logical countermeasure to flow matching is *non-loop* cover traffic, i.e., cover packets addressed to Nym entities other than the sender and present on the merged ingress flows but not the merged egress flows of an endpoints set. This can be achieved either by sending cover packets addressed to a large set of endpoints or by having them dropped at intermediate mixes on their way through the mixnet. Levine et al. proposed a countermeasure in this style called *defensive dropping* [18].

In Nym's current configuration, the adversary is able to tell apart payload and acknowledgment packets based on size. Removing this immediate traffic distinguisher by making acknowledgment packets the same size as payload packets would increase protection against flow matching attacks, at the cost of additional bandwidth. Also, our attacks rely on knowing the average per-mix delay of packets. Should this parameter become less certain (e.g., by adding multiple latency classes [13]) or be entirely unknown to the adversary, flow matching would become much harder.

9.9 Conclusions

In this work, we tackle the problem of *flow matching* in mixnets, where an adversary observing both communication ends tries to determine whether two endpoints are communicating via the anonymous network. We first show that existing techniques, developed for connection-oriented networks like Tor, are not well suited for packet-oriented mixnets that include packet reordering and cover traffic. We thus propose novel *flow matching* methods that are applicable to mixnet communications. We demonstrate the effectiveness of our techniques with an empirical analysis of the Nym mixnet.

As part of our solution, we propose and examine three classifiers that the adversary could use for the attack: a statistical classifier that relies on knowledge about the distribution of network delay and two deep learning classifiers that learn to discriminate matched from unmatched flow pairs directly by training on large amounts of network data. We collect six datasets of a file download scenario across different Nym configurations. In a lab setup without real-world network effects, our statistical and drift classifiers quickly deanonymize Nym communications. Under live network conditions, our classifiers' performances are significantly degraded due to packet losses and more variable routing delays, indicating the importance of flow gaps (intended or unintended) for designing countermeasures to this threat.

Acknowledgments

We thank the anonymous reviewers for their helpful feedback that greatly improved this work. We thank Reyhane Attarian for her involvement in the early stages of this research project. Also thanks to Piyush Kumar and Devashish Gosain for discussions and help with understanding RIPE Atlas data.

Multiple people from Nym Technologies SA helped us out with this work. Thanks to Tommy Verrall for providing a version of Nym's Kubernetes-based deployment framework and for answering questions around it. Thanks to Simon Wicky and Ania Piotrowska for meeting with us to discuss results and provide feedback. And thanks to Mark Sinclair for meeting with us to discuss how private Nym gateways can be realized.

Lennart Oldenburg is funded by an FWO fellowship. This research is partially supported by the Research Council KU Leuven under the grant C24/18/049, by VLAIO through the CS ICON project "Cyber Security Artificial Intelligence" (CSAI), and by CyberSecurity Research Flanders with reference number VR20192203. Some of the resources and services used to conduct this work were provided by the Edinburgh Compute and Data Facility (ECDF) at the University of Edinburgh, and the VSC (Flemish Supercomputer Center), funded by the Research Foundation - Flanders (FWO) and the Flemish Government.

Bibliography

[1] Authors of this paper. *Main repository listing this work's source code and datasets*. 2023. URL: https://github.com/mixnet-correlation/mixmatch-flow-matching-for-mixnet-traffic_popets-2024-2 (pp. 161, 180).

[2] A. Back, U. Möller, and A. Stiglic. "Traffic Analysis Attacks and Trade-Offs in Anonymity Providing Systems". In: *Information Hiding*. Springer Berlin Heidelberg, 2001, pp. 245–257 (p. 194).

[3] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker. "Low-Resource Routing Attacks against Tor". In: *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society*. WPES '07. New York, NY, USA: Association for Computing Machinery, 2007, pp. 11–20 (p. 194).

[4] O. Berthold, H. Federrath, and M. Köhntopp. "Project "Anonymity and Unobservability in the Internet"". In: *Proceedings of the Tenth Conference on Computers, Freedom and Privacy: Challenging the Assumptions*. CFP '00. Association for Computing Machinery, 2000, pp. 57–65 (p. 162).

[5] D. Chaum, D. Das, F. Javani, A. Kate, A. Krasnova, J. De Ruiter, and A. T. Sherman. "cMix: Mixing with Minimal Real-Time Asymmetric Cryptographic Operations". In: *Applied Cryptography and Network Security*. Springer International Publishing, 2017, pp. 557–578 (p. 163).

[6] D. L. Chaum. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms". In: *Commun. ACM* 24.2 (1981), pp. 84–90 (pp. 160, 163).

[7] G. Danezis, R. Dingledine, and N. Mathewson. "Mixminion: Design of a Type III Anonymous Remailer Protocol". In: *2003 Symposium on Security and Privacy, 2003.* 2003, pp. 2–15 (p. 163).

[8] G. Danezis. "The Traffic Analysis of Continuous-Time Mixes". In: *Privacy Enhancing Technologies*. Ed. by D. Martin and A. Serjantov. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 35–50 (pp. 164, 169).

[9] G. Danezis and I. Goldberg. "Sphinx: A Compact and Provably Secure Mix Format". In: *2009 30th IEEE Symposium on Security and Privacy*. IEEE, 2009, pp. 269–282 (p. 163).

[10] C. Diaz, H. Halpin, and A. Kiayias. *The Nym Network*. Feb. 2021. URL: https://nymtech.net/nym-whitepaper.pdf (pp. 160, 163).

[11] R. Dingledine, N. Mathewson, and P. Syverson. "Tor: The Second-Generation Onion Router". In: *13th USENIX Security Symposium*. USENIX Association, 2004 (pp. 160, 162, 194).

[12] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. "Hiding Routing information". In: *Information Hiding*. Springer Berlin Heidelberg, 1996, pp. 137–150 (pp. 162, 194).

[13] I. B. Guirat, D. Das, and C. Diaz. "Blending Different Latency Traffic With Beta Mixing". In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2024 (2 2024) (p. 188).

[14] C. Gülcü and G. Tsudik. "Mixing E-mail with Babel". In: *Proceedings of Internet Society Symposium on Network and Distributed Systems Security*. 1996, pp. 2–16 (p. 163).

[15] E. Hoffer and N. Ailon. "Deep metric learning using triplet network". In: *Similarity-Based Pattern Recognition: Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings 3*. Springer. 2015, pp. 84–92 (pp. 162, 174).

[16] J. van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. "Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis". In: *Proceedings of the 25th Symposium on Operating Systems Principles*. SOSP '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 137–152 (p. 163).

[17] D. Kesdogan, J. Egner, and R. Büschkes. "Stop- and- Go-MIXes Providing Probabilistic Anonymity in an Open System". In: *Information Hiding*. Ed. by D. Aucsmith. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 83–98 (p. 163).

[18] B. N. Levine, M. K. Reiter, C. Wang, and M. Wright. "Timing Attacks in Low-Latency Mix Systems". In: *Financial Cryptography*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 251–265 (pp. 188, 194).

[19] U. Moeller, L. Cottrell, P. Palfrader, and L. Sassaman. *Mixmaster Protocol Version 2*. Internet-Draft draft-sassaman-mixmaster-03. Work in Progress. Internet Engineering Task Force, Dec. 2004. URL: `https://datatracker.ietf.org/doc/draft-sassaman-mixmaster/03/` (p. 163).

[20] M. Nasr, A. Bahramali, and A. Houmansadr. "DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 1962–1976 (pp. 160, 162).

[21] M. Nasr, A. Houmansadr, and A. Mazumdar. "Compressive Traffic Analysis: A New Paradigm for Scalable Traffic Analysis". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Association for Computing Machinery, 2017, pp. 2053–2069 (p. 195).

[22] Nym Technologies SA. *GitHub - nymtech/nym*. n.d. URL: `https://github.com/nymtech/nym` (p. 163).

[23] Nym Technologies SA. *Nym website*. n.d. URL: `https://nymtech.net/` (pp. 160, 163).

[24] S. E. Oh, T. Yang, N. Mathews, J. K. Holland, M. S. Rahman, N. Hopper, and M. Wright. "DeepCoFFEA: Improved Flow Correlation Attacks on Tor via Metric Learning and Amplification". In: *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1915–1932 (pp. 160, 162, 172, 173, 180, 186, 197).

[26] L. Øverlier and P. Syverson. "Locating Hidden Servers". In: *2006 IEEE Symposium on Security and Privacy (S&P'06)*. IEEE, 2006, pp. 100–114 (p. 194).

[27] A. Pfitzmann and M. Hansen. *A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management*. 2010. URL: `https://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf` (p. 163).

[28] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis. "The Loopix Anonymity System". In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 1199–1216 (pp. 160, 163).

[29] J.-F. Raymond. "Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems". In: *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability Berkeley, CA, USA, July 25–26, 2000 Proceedings*. Springer Berlin Heidelberg, 2001, pp. 10–29 (p. 162).

[30] M. Reed, P. Syverson, and D. Goldschlag. "Anonymous Connections and Onion Routing". In: *IEEE Journal on Selected Areas in Communications* 16.4 (1998), pp. 482–494 (pp. 162, 194).

[31] RIPE NCC. *RIPE Atlas Daily Dumps*. 2023. URL: `https://data-store.ripe.net/datasets/atlas-daily-dumps` (p. 185).

[32] A. Serjantov and P. Sewell. "Passive Attack Analysis for Connection-Based Anonymity Systems". In: *Computer Security – ESORICS 2003*. Springer Berlin Heidelberg, 2003, pp. 116–131 (p. 194).

[33] V. Shmatikov and M.-H. Wang. "Timing Analysis in Low-Latency Mix Networks: Attacks and Defenses". In: *Computer Security – ESORICS 2006*. Springer Berlin Heidelberg, 2006, pp. 18–33 (p. 194).

[34] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal. "RAPTOR: Routing Attacks on Privacy in Tor". In: *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, 2015, pp. 271–286 (pp. 162, 195).

[35] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr. "Towards an Analysis of Onion Routing Security". In: *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability Berkeley, CA, USA, July 25–26, 2000 Proceedings*. Springer Berlin Heidelberg, 2001, pp. 96–114 (p. 162).

[36] The Tor Project. *Tor Metrics*. n.d. URL: `https://metrics.torproject.org/` (p. 194).

[37] The Tor Project. *Tor Project website*. n.d. URL: `https://www.torproject.org/` (p. 194).

[38] The Tor Project. *What attacks remain against onion routing?* n.d. URL: https://support.torproject.org/about/attacks-on-onion-routing/ (p. 160).

[39] H. Xuan, A. Stylianou, and R. Pless. "Improved embeddings with easy positive triplet mining". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2020, pp. 2474–2482 (p. 176).

[40] Y. Zhu, X. Fu, B. Graham, R. Bettati, and W. Zhao. "On Flow Correlation Attacks and Countermeasures in Mix Networks". In: *Privacy Enhancing Technologies*. Springer Berlin Heidelberg, 2005, pp. 207–225 (p. 164).

9.A Brief Introduction to Tor

With close to three million daily users and 7,000 servers (called *relays*) [36], Tor [11, 37] is the largest and most popular anonymity network currently available. Tor offers bidirectional low-latency *circuits* for clients to anonymously connect to Internet resources using Tor relays as intermediaries. While in principle any TCP-based traffic can be tunneled through Tor, web browsing is the network's primary application. Tor circuits rely on *onion routing* [12, 30] to achieve their privacy properties. They traverse multiple (typically three) Tor relays selected by the Tor client, such that each relay only knows its immediate predecessor and successor in the circuit, but not the full path. To complete the circuit setup, the Tor client shares keys with each of the selected relays. Application payloads exchanged through a Tor circuit are fragmented and padded into fixed-length *cells* and then encrypted multiple times using a key per intermediary relay. Each intermediary relay uses the shared key associated with the circuit to strip (or add) a layer of encryption before forwarding the resulting packet along the circuit. These per-hop cryptographic transformations ensure that cell headers and payloads observed at different points in the network cannot be identified as being the same packet.

9.B Flow Correlation Attacks Against Connection-Based Anonymous Communication Systems

Back, Möller, and Stiglic [2] describe generic flow correlation attacks (packet counting, latency measurements) against low-latency anonymity system Freedom. Focusing on link-level passive adversaries mainly attacking hop-by-hop, Serjantov and Sewell [32] develop two packet counting attacks against a connection-based anonymity system model that covers many proposed systems at that time. Levine et al. [18] partition observed flows at the two connection ends into non-overlapping time windows for binning packet arrivals. They design the effective countermeasure *defensive dropping*, where packets are lost in the mixnet on purpose to confuse such timing adversary. Shmatikov and Wang [33] also count packets in bins on a set of entry and exit links and compute correlations between bin count vectors for each possible pair. When no defenses are used, their attack achieves an equal error rate close to zero. When traffic is defended with their *adaptive padding* scheme, the equal error rate is close to 0.5, the optimal (defended) value.

Øverlier and Syverson [26] utilize flow correlation to demonstrate how quickly and cheaply specific Tor relays can be identified by an adversary operating a single relay (in 2006). In response to this threat, Tor adopted *entry guards*, i.e., a restricted, small set of eligible first-hop relays for each circuit of a user. Bauer et al. [3] reduce the observations required for flow correlation on Tor to only the cells from circuit establishment collected

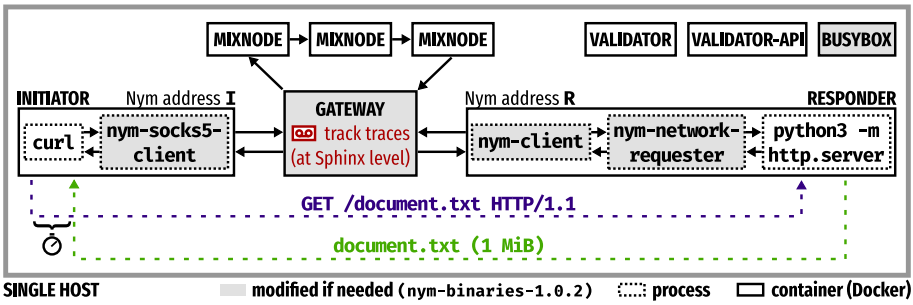


Figure 9.9: Single-host setup (called *isolated setup*) to collect our datasets over a minimal Nym deployment. Until a target number of succeeded downloads is reached, a freshly created initiator downloads a 1 MiB-sized file via HTTP over Nym from a freshly created responder. The remaining containers persist across all downloads. We track the packet exchange at the Sphinx packets level on the patched gateway via the endpoints' Nym addresses, and log start and end time of the download on the initiator.

on compromised first and last circuit positions. Sun et al. [34] exploit specifics of the BGP protocol that is used for Autonomous-System-level routing for Tor flow correlation. They develop a suite of routing attacks called *RAPTOR*, among which their asymmetric traffic analysis achieves 95 % correlation accuracy (using Spearman's rank correlation coefficient) after five minutes of analysis. Nasr, Houmansadr, and Mazumdar [21] improve the scalability of previous flow correlation attacks using techniques from compressed sensing.

9.C Data Collection Setups

We visualize the isolated setup we use to collect datasets `baseline` (and thus, `two-to-one`), `no-cover`, `low-delay`, and `high-delay` in Figure 9.9 and the live-network setup we use to collect dataset `live-nym` in Figure 9.10. We depict each file download duration per dataset collected in Figure 9.11 (excluding for dataset `two-to-one`, as it is crafted from `baseline`).

9.D Additional Experimental Results

To develop our shape and drift classifiers, we conducted several experiments to evaluate the impact of our design decisions, particularly when they deviated from

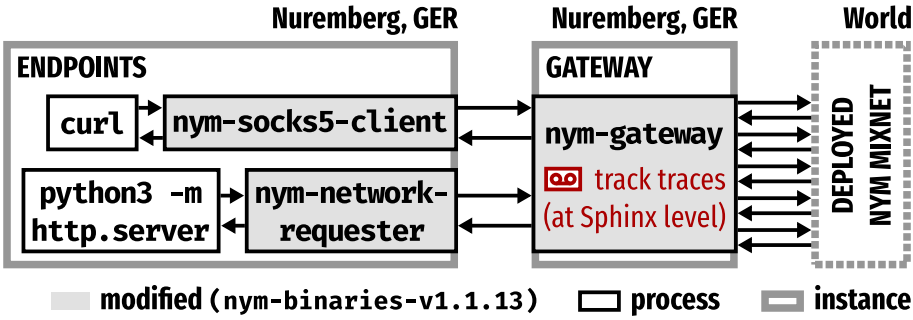


Figure 9.10: Deployed Nym mainnet experiment setup (called *live-network setup*). We inherit the entity spawning and file download behavior from the isolated setup (Figure 9.9).

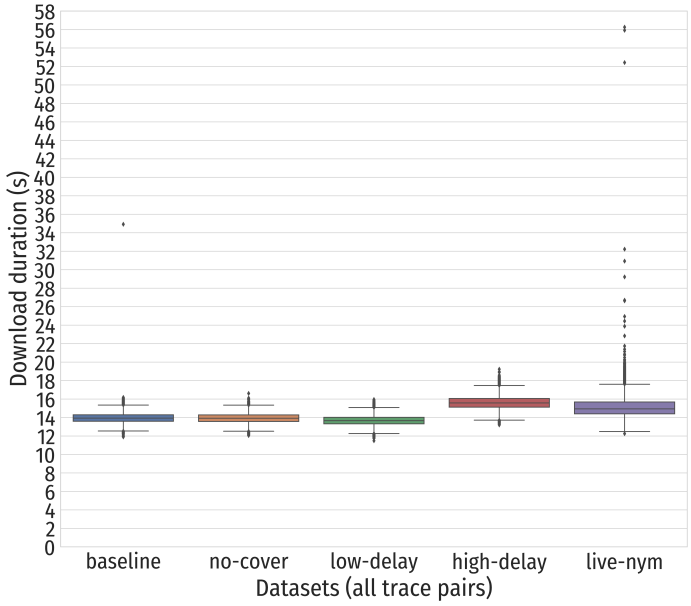


Figure 9.11: Duration of each successful 1 MiB file download as part of its respective dataset in this study. Mind that all collected trace pairs of each respective dataset are included, not only the first 35,000 we use for analysis in our classifiers.

DeepCoFFEA's design. We measured the effect of incorporating acknowledgments to the test (Figure 9.12), setting a global threshold (Figure 9.13), and averaging the

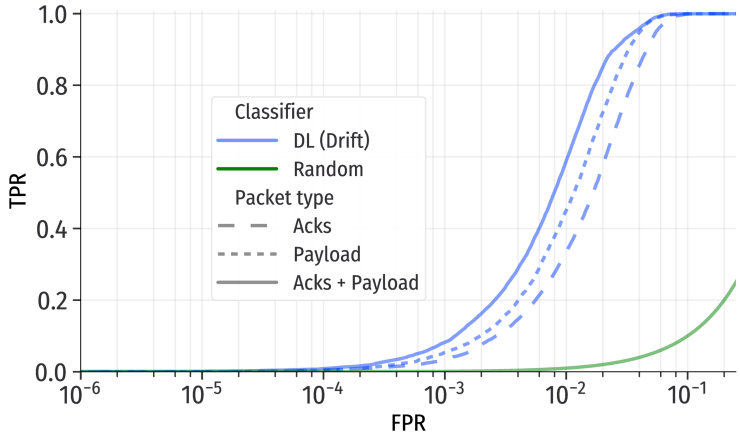


Figure 9.12: Flow matching performance (ROC curves) of our drift model on dataset `baseline` depending on which packet type is used: only payload packets, only acknowledgment packets, or both.

scores as an aggregation rule (Figure 9.14) in the evaluation. In addition, we compare the performance of the new input representation and the representation used by DeepCoFFEA (Figure 9.15) at coping with loop cover traffic.

Figure 9.12 shows the ROC curves of the drift classifier trained on dataset `baseline` for tests that include different combinations of packet types. As observed in the figure, including the acknowledgments into the test significantly improves the performance of the resulting model. This result is expected as—in contrast to Tor—acknowledgments in Nym do in fact provide useful information to the classifiers.

Before measuring the impact of any change in DeepCoFFEA's original design, we reproduced the results reported by the authors at a 0.002 loss on their dataset [24]. The ROC curve of our implementation of DeepCoFFEA, shown in Figure 9.13, is practically identical to the one reported in the DeepCoFFEA paper.

In Figure 9.13, we also plot the ROC curve when we change the threshold from local (default) to global. The gap between the curves indicates a decrease in performance, which we attribute to the fact that the local threshold is fitted to each individual decision by considering the relative distances to all potential negative examples. As discussed in Section 9.5, we opt for a global threshold because the adversary that we consider does not have the required visibility into the network.

Building upon these changes, in Figure 9.14, we again plot the ROC curve of our implementation of DeepCoFFEA with a global threshold that is applying a majority-vote

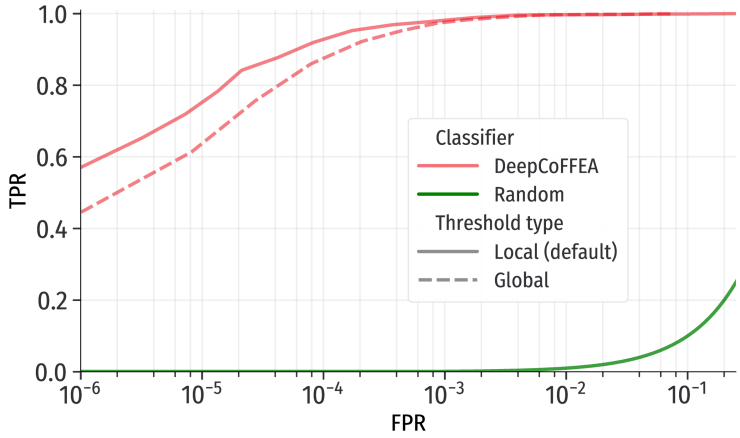


Figure 9.13: ROC curves of our implementation of DeepCoFFEA for a local and a global threshold. The model was trained and evaluated on DeepCoFFEA's dataset. To reproduce the results in DeepCoFFEA's paper, we trained the models until reaching a loss value of 0.002.

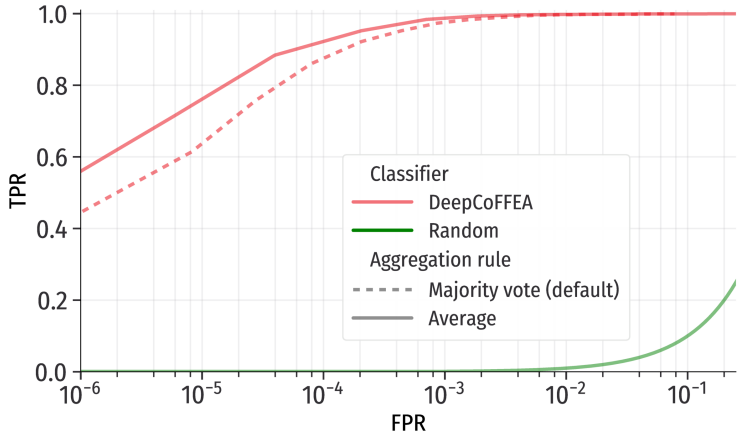


Figure 9.14: ROC curves of our implementation of DeepCoFFEA for different aggregation rules (average vs. maximum hit count) on DeepCoFFEA's dataset.

rule. The other line in the graph represents the ROC curve from evaluating that version of the classifier with the average aggregate rule instead. In this case, the performance of the classifier improved after our change, as the average aggregation rule captures fine-grained information about the scores that is overlooked by the majority vote rule.

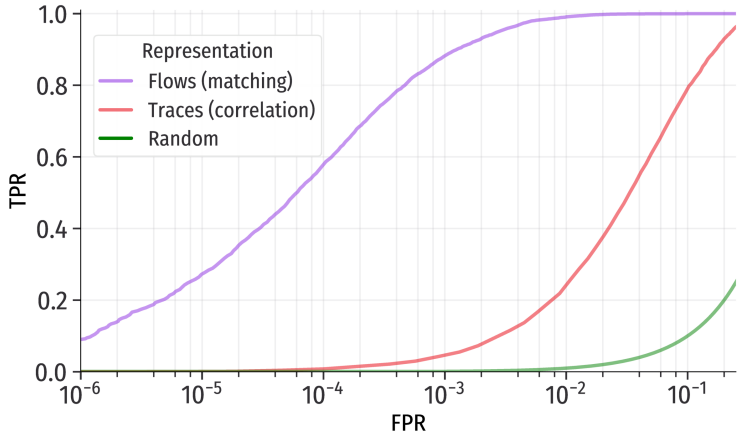


Figure 9.15: ROC curves of the baseline shape model for two different input representations: Traces and Flows. The Flows representation is the packet-based window representation of flows that we used in the rest of the paper. The Traces representation is the time-based window representation of traffic traces used by DeepCoFFEA. For a fair comparison, we take the first 500 packets for the Flows representation, as the first window of the Traces representation has approximately 500 packets on average.

We also assess the effectiveness of our new representation in addressing the flow matching problem in the presence of loop cover traffic and compare it to DeepCoFFEA's representation. In Figure 9.15, we show the ROC curves for the two representations, Flows (ours) and Traces (DeepCoFFEA's). A caveat in this comparison is that the windows in the Traces representation are time-based, which result in a different number of observations per window, thus not allowing us to directly compare the two results. For a fairer comparison between the two we take the number of observations in Flows as the average number of observations in Traces. The average number of observations in the first window of Traces is 500, so we consider the first 500 observations to plot the Flows ROC curve.

9.E Details on the DL Classifiers

For reproducibility, we fixed the PRNG seed and disabled all non-deterministic behavior as much as the GPU libraries allowed.

9.E.1 Network Architecture

In Figure 9.17, we depict the architecture of the shape (Figure 9.17a) and the drift (Figure 9.17b) networks, including the hyperparameter values that we used for our experiments, such as the number and size of the convolutional kernels and the activation function.

Drift Network. The architecture of the drift network is somewhat similar to the one of DeepCoFFEA. As shown in Figure 9.17b, the drift network also has multiple hidden convolutional layers with a ReLU activation function followed by a pooling layer. However, the drift network is much smaller than DeepCoFFEA's, as detecting changes in drift is an easier problem than the problem DeepCoFFEA is designed to tackle. In addition, it uses average pooling instead of max pooling due to the importance of the localization of the drift.

Shape Network. In Figure 9.17a, we depict the architecture of the shape network. It is a deep convolutional neural network with four blocks. Each block has a double convolutional layer with the ReLU activation function, a max pooling layer, and a dropout layer, except for the last block that does not have a dropout layer. The output of the last block is then flattened and densely connected to the output of the network, which holds the embedding. The figure also details the number and sizes of the kernels.

In the design of the shape classifier, we did not tune for the number of kernels and kernel sizes and kept DeepCoFFEA's values, as the networks have similar architectures and input sizes, and hence require similar receptive fields.

The main difference with DeepCoFFEA's architecture is the input, as we represent our input as flows instead of traces. In addition, we use a ReLU activation function throughout the network, whereas DeepCoFFEA uses ELU in the first block and ReLU elsewhere. This change is because DeepCoFFEA may take negative inputs—direction is encoded in the sign—while our inputs are always positive. Thus, there is no clear justification for using an ELU only in the first block. Finally, for increased regularization, the dropout probability increases over the layers of our network, instead of remaining constant.

9.E.2 Hyperparameter Tuning

We use Bayesian optimization to search the space of hyperparameters and find the ones that provide the best results. Bayesian optimization is often used for tuning the hyperparameters of deep learning algorithms as they can find minimum points of the loss function faster than grid-search techniques. In Table 9.1, we list all the hyperparameters of the learning algorithms and the search spaces that we have considered during hyperparameter tuning.

Table 9.1: Deep learning algorithm hyperparameters, the search space for tuning, and the final value that we chose to present the results in Section 9.7.

Hyperparameter	**Search space**	**Chosen value**
Batch size	{8, 16, 32}	16
Training epochs	$\{10, 50, 100\}$	100
Learning rate	$\{10^{-1}, 10^{-2}, 10^{-3}\}$	10^{-3}
Optimizer	SGD, NAdam	-
Window size (n_w)	$\{100, 200, 250, 300, 500\}$	100
Window overlap (δ)	$\{0.00, 0.25, 0.5, 0.75\}$	0.00
Margin (α)	{0.05, 0.10, 0.15}	0.10
Embedding size (s)	64	64

By comparing the learning curves on the training and validation sets, we evaluated the convergence rates for a set of hyperparameters and detected cases of under- and overfitting. In this evaluation of the hyperparameters, we did not always consider all the hyperparameters together, but also conducted a more in-depth analysis for some important hyperparameters, such as window size and overlap.

The last column of Table 9.1 lists the best values of the hyperparameters in our evaluation. We found that many of the hyperparameters did not significantly contribute to improving the convergence of the learning algorithms. For example, for a fixed window size, a $\delta > 0$ overlap does not significantly improve convergence. For a fixed overlap value, smaller window sizes increase the computational demand, but do not seem to either improve the performance—at least for the window sizes that we tested.

The last two rows of the table show hyperparameters that are exclusive for the shape network. We observe that different margins have an impact on convergence: looser margins relax the condition in Equation (9.12) allowing to find more semi-hard negatives, but the semi-hard negatives provide less information and the convergence is slower, and vice versa. Like with DeepCoFFEA, the best value in our tests was $\alpha = 0.1$.

Finally, we observed that the NAdam optimizer would converge faster than SGD in most experiments. However, for the same set of hyperparameters, NAdam would result in vanishing gradients in the experiments conducted on the Nym live network, so we used the SGD optimizer when training on the `live-nym` dataset.

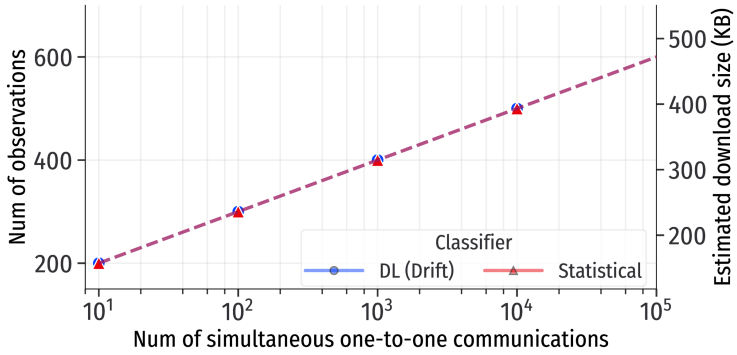


Figure 9.16: Number of `baseline` observations needed for a TPR of over 0.95 with at least 0.95 precision and a given number of simultaneous one-to-one communications (base rate).

9.F Impact of Base Rate (Simultaneous Communications)

In Figure 9.16, we show how the required number of observations to achieve high classifier performance (TPR $\geq$ 0.95) with high confidence (precision $\geq$ 0.95) on `baseline` evolves when the number of simultaneous communications increases.

9.G Hardware Specifications

Computing time is proportional to the number of packets in the traces and inversely proportional to the number of employed CPUs. Thus, it is approximately the same for all experiments, though slightly shorter for the `no-cover` experiment due to the absence of loop cover packets in its traces.

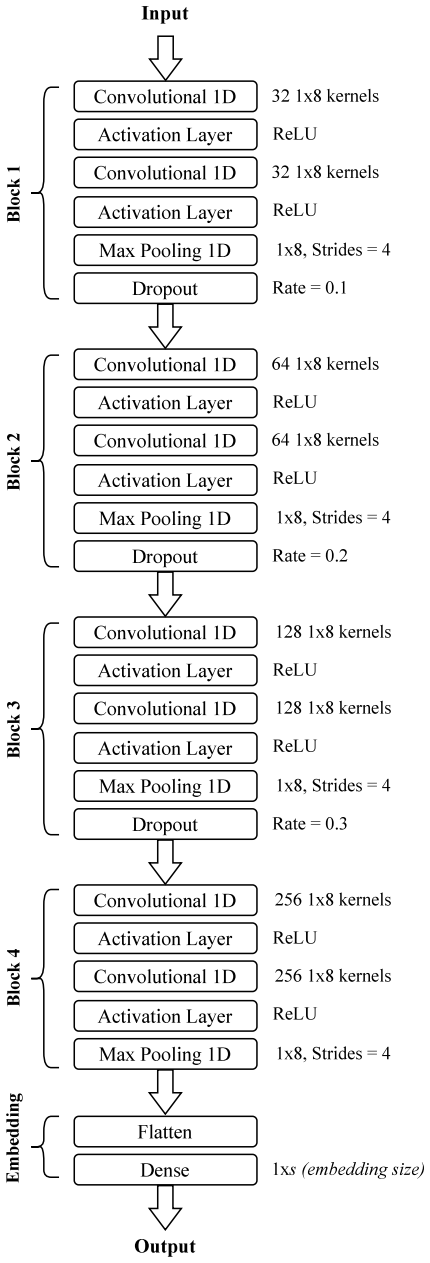
We describe the hardware specifications that we used for evaluating each classifier below.

Statistical Classifier. We ran the evaluation of the statistical classifier in parallel by dividing the score matrix into a grid and processing each submatrix using `Octave` scripts. As an illustrative figure, the experiment on the `live-nym` dataset took approximately three days on a machine with 48 CPUs and 192GB of RAM.

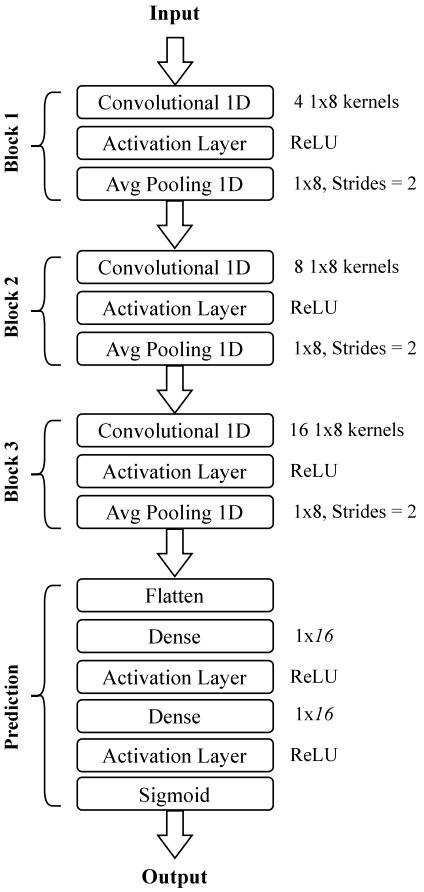
DL Classifiers. For the training and evaluation of the neural-network-based classifiers we used a machine with 64 CPUs (two sockets with 16 cores per sockets and two threads

per core) and 192GB of RAM. In particular, most of the training was performed on an NVIDIA GeForce RTX 3080 with 10GB of memory.

Training a model on the `baseline` dataset for 100 epochs takes approximately five days for the shape and two days for the drift classifier. The evaluation is divided into batches and takes approximately five hours for each model.



(a) Shape network.



(b) Drift network.

Figure 9.17: Network architectures of our DL classifiers.

Chapter 10

Shift Your Shape: Correlating and Defending Mixnet Flows Based on Their Shapes

Manuscript Data (Under Submission)

L. Oldenburg, M. Juarez, E. Argones Rúa, and C. Diaz. "Shift Your Shape: Correlating and Defending Mixnet Flows Based on Their Shapes". Under submission to *IEEE Transactions on Dependable and Secure Computing* since March 31, 2025. 2025

Contributions

Principal author.

Shift Your Shape: Correlating and Defending Mixnet Flows Based on Their Shapes

Lennart Oldenburg,[1] Marc Juarez,[2]
Enrique Argones Rúa,[1] Claudia Diaz[1,3]

[1] *COSIC, KU Leuven*
[2] *School of Informatics, University of Edinburgh*
[3] *Nym Technologies, SA*

Abstract. When the packet rate of flows in a mixnet is variable, it becomes possible to identify which flow entering is which flow exiting the mixnet based on their shapes. We present a passive shape-based flow correlation attack against state-of-the-art mixnet Nym and a systematic evaluation of countermeasures. Assuming an adversary controlling both the entry and exit gateway-requesters selected by users to access the public Internet through Nym, our attack's artificial neural network assigns correlation scores to flow pairs based on traffic distribution similarities in order to accurately distinguish paired from unpaired flow tuples. From data we collected on the live Nym mixnet, we generate 45 datasets and 85 testing scenarios for different defense configurations. After one minute of attacking flow pairs on default Nym, we achieve a PR-AUC of 0.9998 at a base rate of 1.9×10^{-4} paired flow tuples. However, (combinations of) the five evaluated defense strategies indicate that the right choice and scale of countermeasure(s) can offer meaningful protection. Our evaluation also informs on the resources overhead spent on defenses. We discuss steps a mixnet like Nym can take to make our attack both less likely and less accurate.

10.1 Introduction

Thanks to the widespread deployment of end-to-end encryption in recent years, the contents of most of our Internet activities are hidden from network intermediaries, thus strengthening our privacy when we communicate online. Yet, each time we access information or communicate on the Internet, we exchange data packets with visible *metadata*, such as sender and receiver network locations, timestamp, and packet size. Most communications require exchanging multiple packets in *packet flows* and thus aggregated notions like the amount of exchanged packets and flow patterns are also visible at the network level. If collected and analyzed by a capable adversary, metadata

carry the risk of compromising our online privacy through the threat of *traffic analysis*. Traffic analysis concerns the learning of insights about packet flows purely based on metadata and—if these metadata are left unprotected—has the potential to reveal enough to an adversary to gain insights into what we do online and monitor who we talk to, when, and for how long.

A *decryption mixnet* [2] is a type of *anonymous communication network* [26] that aims to make traffic analysis more difficult by combining *layered encryption* of independent packets, *artificial delays* along their paths through an overlay network of proxies, and *additional cover traffic*. When an application uses a mixnet as network transport, its data flows are converted into equal-sized mixnet packets, each with fresh cryptographic key material and its own independent mixnet path (i.e., a series of intermediaries to traverse). Each intermediary (called *mix*) is only able to decrypt the layer intended for it by the packet's sender, yielding the routing information of the next hop and the transformed packet payload to pass on. The packet's cryptographic transformation makes it impossible to know which packet forwarded by the mix corresponds to which input packet based on the packet's wire representation. Additionally, mixes forward packets in a different order than they received them to impede timing analysis. This can be done either by batching packets [1, 2, 11] or by delaying them independently by a random (e.g., exponentially distributed) amount of time, as is done in *continuous-time* mixnets [12, 23]. The combination of cryptographic transformation and packet reordering prevents even a *global network adversary* (who has visibility over all links between every pair of entities in the network but is unable to see the internal operations of mixes) from tracing packets from source to destination. This threat model has historically served as the benchmark adversary for evaluating mixnets.

Launched in 2021, the *Nym network* [8][1] is designed as a general-purpose network privacy layer based on a continuous-time mixnet made of best-in-class components.[2] Endpoints like users and Nym-native applications direct their network flows to be sent through Nym's mixnet for anonymity. Instead of connecting to mixes directly, each endpoint establishes a session with a *gateway*, a special type of node acting as an interface between endpoints and the mixnet. Additionally, endpoints have a packet emission scheduler to ensure that their network activity is of *constant bandwidth on average*, which requires sending *cover packets* when idling to hide genuine activity. Nym thus aims to provide its users *unobservability* [22] towards global network adversaries, while striving to fit the latency requirements of as many applications as possible.

Recently, Nym made significant changes to its mixnet configuration. Key among them is the removal of the packet emission scheduler for a particular kind of Nym application, the *network-requester*, which acts as a proxy for users to anonymously access the public Internet. Discarding the rigid emissions regulation on network-requesters helps Nym

[1] `https://nym.com`

[2] When we say Nym, we refer to Nym's five-hop mixnet variant in version *nym-binaries-v2023.5-rolo*, not Nym's more recent two-hop VPN variant.

be more usable for applications expecting low-latency, high-volume *downstream* data transmissions (i.e., towards their users), e.g., web browsing and streaming. However necessary, this marks a stark departure from Nym's previous strategy of having both endpoints' schedulers shape packet emissions (we call this a *bidirectionally regulated mixnet*) to the very different setting where network-requesters forward packets as fast as possible to the mixnet (a *downstream-unregulated mixnet*). With this change, a previously non-existent *signal* now becomes a threat for deanonymizing Nym users and forms the basis of this work. In previous work against bidirectionally regulated Nym [19], we had to exploit the *drift* between candidate flow pairs as the signal to conduct a *flow matching* attack with strict assumptions on how users communicate (exclusively within a group of endpoints all monitored by the adversary). In contrast, in this work we show how, due to the variable packet sending rate, we can leverage the *shape* (distribution of packets over time) of candidate flow pairs to mount a more general and effective *flow correlation* attack.

The attack can only be deployed against flow pairs for which the adversary controls *both* the user-side gateway and the remote network-requester selected by the user as proxy to the public Internet. This threat model provides the adversary with a noise-free view on downstream packet flows at the network-requester position, which is key for the attack's success. Based on the example use case of browsing the web through Nym, we evaluate the extent to which webpage downloads routed through *default Nym* (i.e., Nym in its current configuration) are vulnerable to shape-based flow correlation. Our flow correlator uses *convolutional neural networks (CNNs)* and is trained on large datasets of flow pairs that we collected on the deployed Nym mixnet. We find that if an adversary is selected for both the entry and exit position, shape-based flow correlation against default Nym is very effective in distinguishing paired from unpaired flows. After one minute of attack time, we achieve a PR-AUC of 0.9998 at a base rate of 1.9×10^{-4} for truly paired flows. Even at a two orders of magnitude lower (subsampled) base rate of 1.9×10^{-6}, we still correlate flow pairs with a PR-AUC of 0.9954.

Given our attack's potential, we examine five independent defense approaches, suitable for deployment in a mixnet like Nym: adding unrelated traffic to flows, splitting flows among multiple entries, increasing the per-mix delay of packets, increasing the size of payload packets, and removing acknowledgment packets. To determine if combining countermeasures offers greater protection than the countermeasures deployed in isolation indicate, we additionally evaluate a defense grid combining the addition of unrelated traffic with the splitting of flows. Our results show that the defenses are indeed able to offer meaningful protection, if deployed in combination or at high overhead cost, or both. We also include tests of a flow correlator trained on a different configuration than it is tested on. The dramatic reduction of adversarial success on most defense scenarios indicates the potential for an additional countermeasure, where flow parameters such as per-mix delay, amount of cover traffic, and splitting of flows are changed by users in a personalized or randomized manner. Adapting the attack to

such settings requires considering (i.e., training for) all possible combinations of user parameter choices, which may be infeasible depending on the parameter space.

While it may be impractical to completely neutralize this threat, our key takeaway for mixnet designers is that it can be significantly mitigated by: (i) making the vantage point conditions our threat model relies on (even) less likely to occur and (ii) implementing the defenses proposed in this work: removing mixnet-level acknowledgments, increasing the packet payload size, allowing for flow splitting over multiple entries (and exits, if possible), enabling users to customize mixing latency and levels of cover traffic for their connections, and regulating or at least rate-limiting downstream flows to avoid large traffic peaks. In summary, we contribute:

- We propose a passive flow correlation attack for mixnets with unregulated traffic in at least one direction. Given a packet flow entering the mixnet and another flow exiting the mixnet, our attack can leverage their variable packet rates to find correlations and determine whether or not the two flows are paired and thus part of the same connection.
- We propose five countermeasure strategies that can be deployed in isolation or in combination, namely: adding unrelated traffic, splitting flows, increasing the per-mix delay of packets, utilizing larger packets, and removing acknowledgment packets. We furthermore discuss strategies for diminishing the likelihood that a user connection is susceptible to the attack, as well as user-side parameter personalization and its usefulness for attack mitigation.
- We obtain 45 datasets from packet flows captured on the live Nym network and systematically assess the impact of our attack and effectiveness of the proposed defensive strategies in 85 scenarios. Based on these results, we provide actionable recommendations to reduce the threat of our flow correlation attack. Our work thus also contributes to improving privacy in real-world systems as the Nym project plans to implement several of our proposed defenses in the deployed network.

10.2 Problem Statement

10.2.1 System Model and the Nym Network

We consider the setting depicted in Figure 10.1, where *users* access public Internet resources through the Nym mixnet. Users run Nym's *socks5-client*, [3] which relays traffic

[3] The specific version of Nym we consider in our model and in our experiments is *nym-binaries-v2023.5-rolo*. At the time of submission, the deployed Nym network has been updated multiple times with respect to this

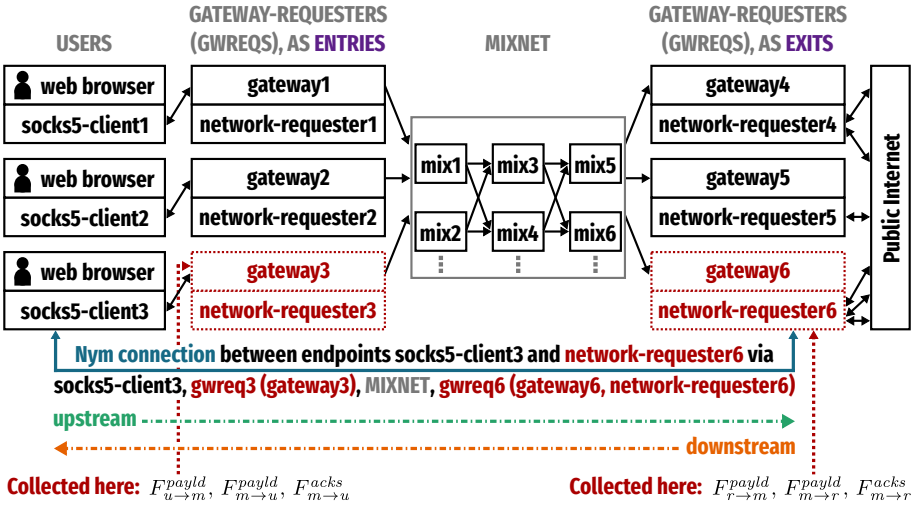


Figure 10.1: System model with adversarial entities in red with dotted borders.

between a web browser running on the same machine and the Nym network, and choose a pair of *gateway-requesters* (*gwreqs*) to act as their *entry* and *exit* points for anonymously accessing a public Internet resource (e.g., a webpage) through Nym.

Gateway-Requesters

Gwreqs are Nym nodes that act as interfaces between users and mixes (entry role) and between mixes and public Internet servers (exit role). Each gwreq has two components: a *gateway* and a *network-requester*. While Nym allows for gateways and network-requesters to be operated as distinct network entities, they can also follow a unified model where gateways run a network-requester process internally and thus offer Internet proxy functionality in addition to their gateway functionality. We adopt this configuration in our work. We use the terms *gateway* and *network-requester* only when referring specifically to the respective component.

In Nym, users can freely choose gwreqs to act as entry and exit for their public Internet accesses. When a gwreq is chosen as entry (e.g., socks5-client1 choosing gwreq1 in Figure 10.1), only the gateway component is active. This component forwards the (upstream) packets received from the socks5-client to the mixnet and receives the (downstream) packets addressed to the user from the mixnet. For simplicity, the

version, but none of the updates modifies the packet sending rates, packet sizes, per-mix delay distribution, acknowledgments, or cover traffic strategy, which are the relevant parameters for our setting.

arrows between gwreqs and the mixnet in Figure 10.1 show only the upstream direction. In reality, every gwreq services both directions (upstream and downstream) of each connection as well as both entry and exit roles for different connections.

In contrast, when a gwreq is chosen as exit (e.g., gwreq4 in Figure 10.1), both the gateway and network-requester components are active. The gateway behaves as described above while the network-requester acts as its client, i.e., the gateway forwards the network-requester's (downstream) traffic to mixes and receives (upstream) packets from mixes that it hands to the network-requester. The network-requester functions as a proxy for a user towards the public Internet. It receives and decodes (upstream) mixnet packets, reassembles their payloads into application packets, and forwards them to a public Internet destination (e.g., a DNS or web server). In the downstream direction, the network-requester encodes the responses from the public Internet destinations into mixnet packets and passes them to the gateway component of its own gwreq, which relays the packets to the mixes that route them to the user's entry.

From the perspective of an application running on top of the mixnet, the application client (e.g., web browser) and the accessed public Internet resource (e.g., web server) are the communication endpoints. End-to-end encryption implemented at this level (e.g., HTTPS) prevents malicious network-requesters from reading the application data. In this work, however, we are concerned with the metadata of mixnet packet flows and thus call the socks5-clients and the network-requesters the *endpoints*, as they represent the two ends of *mixnet connections*.

Mixnet and Packet Routes

Nym relays packets via five intermediate routing hops. The first and fifth hops are the entry and exit gwreqs chosen by users for their connections. In Nym's default behavior, the same gwreq pair persists throughout a user's flow. For all packets of a bidirectional connection, the entry is always the first hop of upstream traffic and the fifth hop of downstream traffic, while the exit is always the first hop of downstream traffic and fifth hop of upstream traffic. The second, third, and fourth routing hops correspond to the three layers of *mixes* that form the Nym mixnet. Contrary to gwreqs, mixes have no notion of a *connection*. Mixes process and relay every packet independently, using information in each packet's header. Three different mixes are randomly chosen for each packet in a flow, even as the same entry and exit are used throughout the flow.

Source Routing and Packet Processing

Nym is a source-routed decryption mixnet that relies on the *Sphinx* packet format [6]. The endpoint preparing a packet's header determines the hops of the packet's route by

using the public keys of the selected destination and intermediate hops to encrypt the packet headers in reverse routing order. A gateway receiving a Sphinx packet from an endpoint simply forwards the packet to the mix indicated as its next destination. When decrypting a received Sphinx packet, a mix reveals the identity of only the next hop on the route, a cryptographically transformed packet to forward, and a *delay* value. Each per-mix delay value is randomly sampled by the endpoint who prepared the header from an exponential distribution with average $\mu = 50$ [ms]. Mixes hold each transformed packet back for the specified delay before sending it to its next hop. The third and last mix on the route sends the packet to the receiving gwreq. The gateway decrypts the packet to reveal the final recipient of the Sphinx packet payload (either a socks5-client or a network-requester). Finally, the gateway also extracts the acknowledgment included in the packet and sends it back through the mixnet, as described next.

Packet Types in Nym

There are different types of Sphinx packets: *payload packets* to anonymously send data and *acknowledgment packets (ACKs)* to ensure reliable transport. Furthermore, *SURB packet headers* enable both anonymous acknowledgments and replies (downstream traffic).

When a Nym endpoint is tasked with sending an application network flow, it packages the contents into fixed-size payload packets of $2{,}413$ [bytes]. Thus, application packets are either chunked into multiple payload packets (if too large) or padded to reach the prescribed size (if too small). Included in each payload packet is an acknowledgment packet (386 [bytes]) created by the payload sender. An ACK is a self-contained packet that the recipient's gateway extracts from the payload packet and immediately sends back through the mixnet to the sender to confirm packet delivery. If a payload packet is not acknowledged after a predefined timeout, the sender retransmits it. For a payload sender to avoid revealing their *Nym address* (which includes their gateway) to the recipient's gateway, ACKs rely on *single-use reply blocks (SURBs)* [5].

A SURB is a packet header prepared by an endpoint that includes a path through the mixnet back to itself. The recipient of a SURB can optionally attach a payload (processed with keys associated with the SURB) and send the resulting packet through the mixnet back to the sender. In Nym, endpoints can use SURBs to achieve *receiver anonymity* for both ACKs and downstream traffic. A gateway receiving a payload packet with an enclosed ACK simply sends back the ACK as received to confirm delivery. The receiving gateway (whether part of an entry or exit) neither learns the sender's identity (user or network-requester) nor the sender's gwreq.

The same mechanism can be used to enable a network-requester to send payload packets downstream to the user, with no knowledge of the socks5-client's identity or the address of its entry. The socks5-client must provide the network-requester with enough SURBs via upstream payload packets so that the network-requester is able to send the entire

response downstream. The network-requester attaches encrypted payloads to the received SURBs to form the final packets sent to the user. To know which SURBs to use for which responses, network-requesters group received SURBs by *anonymous sender tag*, a randomly generated pseudonym for a user during a session.

Packet Emission Regulation

When a socks5-client creates a payload packet to be sent upstream via the mixnet, the packet is not sent immediately to the entry. Instead, it is placed onto the endpoint's *foreground packet emission queue*. A local *Poisson process* determines when packets are sent from the queue to the entry. *Inter-emission delays* are sampled from an *exponential distribution* with average $\lambda_{\mathbb{F}} = 20$ [ms] in default Nym. If the foreground queue is empty when the Poisson process signals to emit a packet (i.e., no application packet is ready), a *loop cover packet* is generated and sent instead. Loop cover packets travel through three randomly chosen mixes before circling back to user's entry, which sees them as a received packet for the user. The user's entry cannot distinguish loop cover packets from other payload packets from or to the user. In addition to the foreground queue, endpoints operate a second Poisson-scheduled *background queue* exclusively emitting loop cover packets, once every $\lambda_{\mathbb{B}} = 200$ [ms] on average by default.

The purpose of these Poisson processes is to send packets with a constant average rate to achieve unobservability, i.e., to make the sending activity of users look always the same, regardless of whether they are communicating or only idling (i.e., sending loop cover packets). In earlier versions of the Nym network (including the version we evaluated in MixMatch [19]), network-requesters also sent packets following the same foreground Poisson process as users. Crucially for this work, however, Nym introduced changes to reduce the latency of responses: network-requesters still operate the same Poisson-scheduled background queue (*regulated*), but their foreground queue is no longer governed by a Poisson process. Instead, they send downstream payload packets as soon as they are created without artificially delaying them (*unregulated*). No loop cover packets are sent to fill gaps in the foreground schedule (as Poisson-determined gaps no longer exist).

10.2.2 Adversarial Model

The adversary's objective is to link a user to the public Internet resource the user is accessing via the mixnet. If successful, the attack negates the unlinkability properties that the mixnet is meant to provide. Any content exchanged between the user and the public Internet resource remains inaccessible to the adversary as long as end-to-end encryption is used.

In order to deploy the attack, the adversary must control at least two gateway-requesters, while no mixes need to be compromised. The adversary is only able to attack users who choose a pair of colluding gwreqs as entry and exit of their Nym connection to a public Internet resource. We illustrate the adversarial resources in Figure 10.1, gwreq3 and gwreq6, in red with dotted borders. If socks5-client3 selects gwreq3 as entry and gwreq6 as exit, the adversary can deploy the attack and may learn which public Internet resource this user is accessing. However, the adversary is unable to attack socks5-client1 and socks5-client2 (even if they select the adversary's gwreq3 or gwreq6 as exit), as they connect via non-adversarial entries. Likewise, socks5-client3 is safe from our attack should they choose a non-adversarial gwreq as exit (e.g., gwreq4). Clearly, the higher the chances that two colluding gwreqs are chosen as both entry and exit of a connection, the higher the risk of compromise. We discuss the probability of compromise and mitigation strategies in Section 10.7, and assume from here on that *some* users are susceptible to our attack.

We recall that the network-requester component of the exit acts as the endpoint of the mixnet packet flows exchanged with the user. An adversarial exit thus has access to noise-free (i.e., reassembled, cover-packet-free, multiplexing-free) Sphinx packet flows associated with the anonymous requests it serves: $F^{payld}_{r\to m}$ represents the sequence of timestamps of (downstream) payload packets (containing application responses for the user) sent from the network-requester component to the mixnet, $F^{payld}_{m\to r}$ is the sequence of timestamps of (upstream) payload packets (containing user requests) received from the mixnet for the network-requester, and $F^{acks}_{m\to r}$ is the sequence of timestamps of received ACKs corresponding to the payload packets sent as part of $F^{payld}_{r\to m}$. Note that the transmission timestamps of the ACKs for payload packets coincide with the receipt timestamps of those packets and we therefore use the equivalence $F^{payld}_{m\to r} \equiv F^{acks}_{r\to m}$.

At the entry side, the adversary records the three flows $F^{payld}_{u\to m}$, $F^{payld}_{m\to u}$ ($\equiv F^{acks}_{u\to m}$), and $F^{acks}_{m\to u}$ for each socks5-client connection at the gateway (rather than the endpoint) level. The adversary is not able to tell packets apart at the gateway level, only the socks5-client knows which packets serve which purpose or belong to which application-level connection. This implies that the entry-side flows are not free of noise: they may include indistinguishable loop cover packets as well as packets from any other mixnet connection a socks5-client may be maintaining simultaneously with other endpoints.

Our attack method (detailed in Section 10.3) involves testing if pairs of recorded entry-side and exit-side flows show a high degree of correlation, indicating that the two underlying flows could be the two ends of a mixnet connection. We assume the adversary is able to synchronize the clocks of the colluding gwreqs, so that all collected timestamps are consistent with a global clock. The adversary then takes candidate pairs of entry-side and exit-side flows collected at any of the adversarial gwreqs and evaluates their level of correlation. We consider the attack successful if the adversary is able to accurately distinguish paired from unpaired flows.

We explicitly do not assume the adversary to have global visibility over all flows entering and exiting the mixnet. Our attack ascribes correlation scores to candidate pairs considered on their own and does not reason about the *best match* for an exit-side flow from among all entry-side flows it could be paired with. The attack requires only passive capabilities (making it hard to detect), as the adversary only records packet timestamps and never needs to tamper with packet flows. Finally, note that although we use Nym as the stand-in for a state-of-the-art mixnet, our methods can be adapted to other mixnet designs as long as our system and threat model apply. We further discuss the considered threat model in Section 10.7.

10.3 Shape-Based Correlation Attack and Defenses

10.3.1 Correlating Mixnet Flows Based on Their Shapes

The flow correlation attack we propose exploits the fact that traffic flows entering the mixnet may contain features (e.g., packet bursts) that persist at the other side. An adversary observing a flow entering the mixnet and another one exiting it can compare their local rates to find similarities. Considering bidirectional communication and the use of acknowledgments, there are four pairs of flows available to the adversary:

1. Upstream payload packets: $F^{payld}_{u\to m}$ and $F^{payld}_{m\to r}$,
2. ACKs to upstream payload packets: $F^{acks}_{r\to m} \equiv F^{payld}_{m\to r}$ and $F^{acks}_{m\to u}$,
3. Downstream payload packets: $F^{payld}_{r\to m}$ and $F^{payld}_{m\to u}$,
4. ACKs to downstream payload packets: $F^{acks}_{u\to m} \equiv F^{payld}_{m\to u}$ and $F^{acks}_{m\to r}$.

In Nym, pairs (1) and (2) are regulated flows, where packet rates are constant on average and the flows follow a Poisson process. We illustrate these in the top three flows of Figure 10.2. As expected, no distinctive traffic peaks are visible. We previously investigated the correlation of regulated mixnet flows in MixMatch [19] (see Section 10.8.1), where we matched flows by comparing the drift between entry and exit timestamps. Above pairs (3) and (4), however, are currently unregulated flows in Nym that do carry identifiable information in their local density, understood as the number of packet timestamps located in a temporal neighborhood. This is visible in the bottom three flows of Figure 10.2, where we see distinctive traffic peaks. Furthermore, we can observe apparent similarities between the peaks in $F^{payld}_{r\to m}$ and $F^{payld}_{m\to u}$ (downstream payload packets) as well as between the peaks in $F^{payld}_{m\to u}$ and $F^{acks}_{m\to r}$ (their corresponding ACKs). Our attack focuses on the pairs (3) and (4) of unregulated flows. We use an *artificial neural network (ANN)* to optimize the flow comparison strategy, as the timing

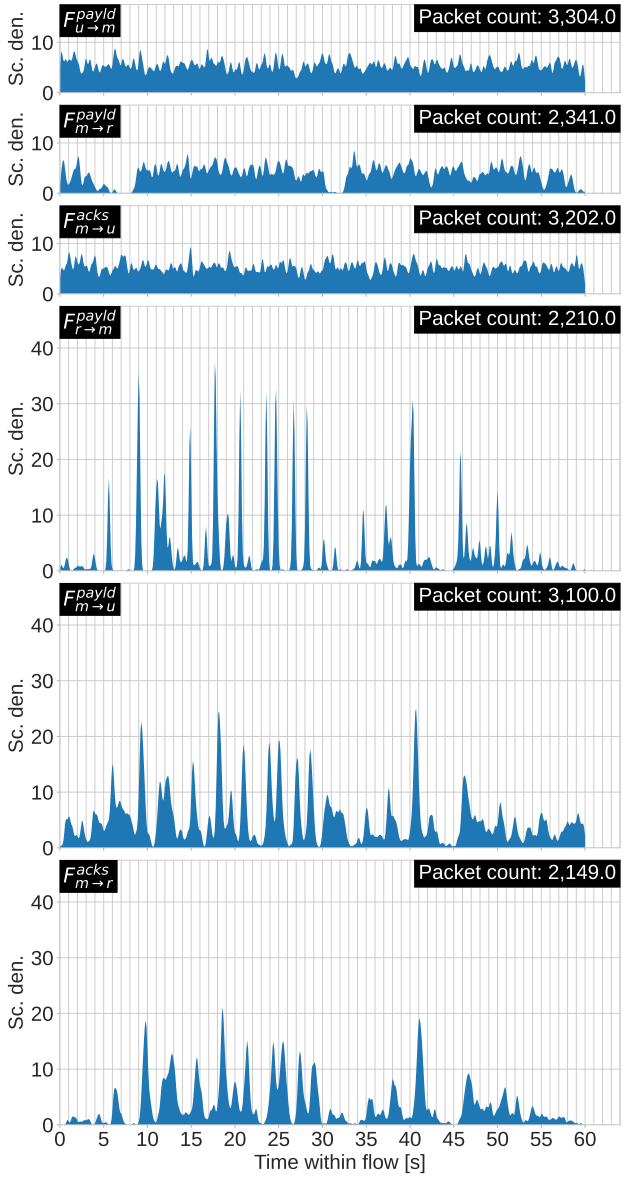


Figure 10.2: Plot of the scaled densities ("Sc. den.", y-axes) obtained by processing visit 70 to *theverge.com* in *default*'s testing split following our KDE process.

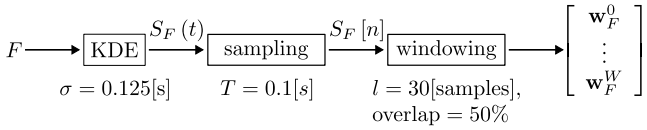


Figure 10.3: Processing of a flow from timestamp vector F to windowed representation $\left[\mathbf{w}_F^0, \ldots, \mathbf{w}_F^W\right]$.

distortion introduced by a mixnet operating under real network conditions is otherwise non-trivial to model.

Flow Shape Signal Computation

A flow F with constant packet size is fully characterized by an ordered sequence of timestamps, or equivalently, a signal consisting of a train of Dirac deltas located at these timestamps:

$$F = \left[t_1^F, \ldots, t_{N_F}^F\right] , \quad F(t) = \textstyle\sum_{k=1}^{N_F} \delta\left(t - t_k^F\right) , \tag{10.1}$$

where N_F stands for the number of packets in flow F. Although this representation is complete and exact, we would like to map the density of the flow to a discrete representation, where each sample describes the local density around each equally spaced sampling time. This representation, which we call the *shape* of a flow, simplifies feeding it to a discrete-time classifier. The shape of a flow can be obtained using *kernel density estimation (KDE)*, which formally computes the convolution of the flow and a given kernel $K_\sigma(t)$:

$$S_F(t) = F(t) * K_\sigma(t) = \sum_{k=1}^{N_F} K_\sigma(t - t_k^F) . \tag{10.2}$$

We can now define our shape signal as the discrete time version of $S_F(t)$, normalized such that its sum equals the total number of packets N_F:

$$S_F[n] = \frac{N_F}{\sum_k S_F(kT)} S_F(nT) , \tag{10.3}$$

where the sampling period is $T = 0.1$ [s]. The kernel width σ encodes its time resolution: it must be narrow enough to make $S_F[n]$ a good estimate of the density of timestamps around $t = nT$, but wide enough to consider timestamps in between neighboring sampling times. We use a Gaussian kernel with $\sigma = 0.125$ [s], which provides a good balance between smoothing and time resolution. In our experiments, we analyze the first

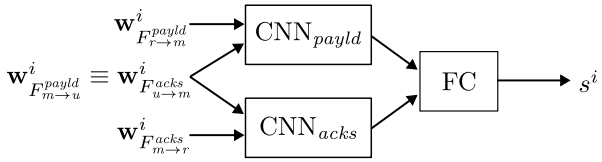


Figure 10.4: Window-wise comparison of flows based on their shapes.

$d = 60$ [s] of each flow pair (all timestamps not in $[0, d]$ are discarded from F). Following an approach similar to DeepCoFFEA [18], for each flow F, we transform $S_F[n]$ into a sequence of windows of width $l = 30$ [samples] with a 50 % overlap, obtaining $\mathbf{w}^i_F = \left[S_F[\frac{i \cdot l}{2}], \ldots, S_F[\frac{i \cdot l}{2} + l - 1]\right]$, for $i = 0, \ldots, W$, where $W + 1 = \left\lceil \frac{2d}{lT} \right\rceil + 1 = 41$ windows in total. If necessary, we pad $S_F[n]$ with zeros prior to windowing, such that its size is a multiple of l. We illustrate the full process of mapping flows to their windowed shape signals in Figure 10.3.

Shape Comparison

Our classifier compares an entry-side with an exit-side flow to decide whether they are *paired* or *unpaired*. We visualize this process in Figure 10.4. The adversary utilizes the two unregulated pairs of flows in the attack: the one of the downstream payload packets ($F^{payld}_{r\to m}$ and $F^{payld}_{m\to u}$) and the one of the acknowledgments sent back for each of the payload packets ($F^{acks}_{u\to m} \equiv F^{payld}_{m\to u}$ and $F^{acks}_{m\to r}$). Each pairing is fed through its own dedicated *convolutional neural network (CNN)* and we thus name them CNN_{payld} and CNN_{acks}, respectively. The two CNNs have the same structure but do not share weights. The outputs of the two CNNs are concatenated and fed into a fully connected stage (called FC) with a single sigmoid neuron in its output layer, giving us a window-wise comparison score s^i, as shown in Figure 10.4. The final correlation score s for a candidate flow pair is computed as the average of all its window correlation scores. Ideally, this score should approximate the probability of flows $F^{payld}_{r\to m}$, $F^{payld}_{m\to u}$, $F^{acks}_{m\to r}$ being paired. For this, we train this model using the *binary cross-entropy loss*. In case acknowledgments are not available (e.g., if not used in a particular mixnet), the adversary can use the same approach but remove CNN_{acks}. We will make the source code and instrumentation scripts of our flow correlator implemented with PyTorch[4] and NumPy[5] publicly available upon acceptance of this work. We are happy to make them available to reviewers during the review process upon request.

[4] `https://pytorch.org`
[5] `https://numpy.org`

10.3.2 Countermeasures Against Flow Correlation

If the adversary is able to identify with high confidence which flow observed at their exits (network-requesters) corresponds to which flow at their entries (gateways) based on the flows' shapes, then defending against the attack necessarily requires altering these flow shapes to cause misclassifications.

To keep our defense evaluation practical, we exclude countermeasures that require changing the exit-side behavior of flows, e.g., fetching different webpage resources via different exits. Instead, we focus on defenses whose effect we can simulate on the entry-side unregulated $F^{payld}_{m\to u}$ flows or defenses grounded in modifying mixnet parameters. We identify the following strategies against shape-based flow correlation:

Adding Unrelated Traffic

Placing additional packets onto $F^{payld}_{m\to u}$ effectively adds noise and makes correlating the flow to its exit-side counterpart more challenging for the adversary. This addition of noise can be achieved either by multiplexing multiple simultaneous connections (to different exits) via the same entry (*flow merging*) or by introducing additional cover traffic (*packet injection*). One way to do so is for users to distribute payload packets with SURBs to many different gwreqs (of which we assume only a small fraction to be compromised), which send them back at times requested by the users with cover (i.e., throwaway) payloads attached. Given that users also determine the mix delays of these packets, they have a considerable degree of control over the time when the packets will appear on their $F^{payld}_{m\to u}$.

Splitting Flows

An alternative strategy is to partially hide the $F^{payld}_{m\to u}$ from the adversary by distributing the flow's packets across several entry gateways. Users *split* the stream of payload packets they expect to receive from network-requesters between two or more entries by first creating connections with them and then constructing SURBs such that the packets arrive via the desired entry gateway. Assuming that some but not all gwreqs are compromised, splitting a user's $F^{payld}_{m\to u}$ flow across multiple entries reduces the adversary's visibility, creating uncertainty for the attack's classifier.

Combining Packet Injection with Flow Splitting

A defense may combine packet injection with flow splitting. We evaluate composites of both strategies to assess whether their use in combination is more effective than

indicated in isolation.

Increasing Per-Mix Delay

A higher per-mix delay μ not only increases the average end-to-end packet transmission latency but also latency *variance*. This leads to more packets being received out of order, diffusing traffic peaks and distorting flow shapes. This countermeasure thus increases μ, which determines the mean per-mix delay of packets. Given that socks5-clients determine the per-mix delays of the downstream packets they expect from network-requesters via the SURBs they prepare, this defense can be unilaterally deployed by users without relying on network-requesters.

Increasing Payload Packet Size

Finally, for the same number of exchanged payload bytes, the larger the size of payload packets, the fewer packets are needed to complete a transmission. This leads to shorter packet sequences and fewer network events the adversary can exploit to correlate flows.

Removing ACKs

In our attack, the adversary exploits two pairs of flows to determine the correlation score for the entire candidate, namely, $F^{payld}_{r\to m}$ and $F^{payld}_{m\to u}$ as well as $F^{payld}_{m\to u}$ and $F^{acks}_{m\to r}$. This countermeasure eliminates the second pair, $F^{payld}_{m\to u}$ and $F^{acks}_{m\to r}$, from correlation by removing mixnet-level ACKs entirely. In the version of Nym we analyzed, ACKs are distinguishable from payload packets due to being of a different size. Removing them leads to less evidence available to the adversary for correlation. We assume that reliable transmission is achieved through other means, e.g., by having acknowledgments at the application level.

10.4 Data Collection Infrastructure

To evaluate the success of the attacks and defenses under realistic conditions, we design a data collection infrastructure for the live Nym network. This setup enables us to collect Sphinx-level packet metadata at both entry and exit positions for the flows exchanged by our simulated users, without posing a privacy risk to other users of the live Nym network.

We rely on the deployed Nym infrastructure, particularly its mixnet, for most aspects of the system and thus only need to run user and gwreq instances to collect our datasets.

Our users make connections to public Internet web servers through a local socks5-client, the gateway component of the entry, the Nym mixnet, the gateway of the exit, and finally the network-requester component of the same exit. We deploy three gwreq instances and twelve user instances for each dataset. We pair them in such a way that four user instances use one gwreq instance exclusively as both entry and exit. The packet metadata logs collected by our gwreq instances for the simulated users serve as the flows used in the experiments.

All Nym processes (socks5-clients on user instances and gateway-requesters on gwreq instances) used for each data collection run Nym in version nym-binaries-v2023.5-rolo with minor patches applied to facilitate that collection. Our patches ensure that the spawned users choose their designated gwreq as both entry and exit, ignoring Nym's publicly available gwreqs. Additionally, we modify gwreqs to log Sphinx-level packet metadata (packet size and timestamp) to file. We prevent other Nym users from connecting to our modified gwreqs by not bonding them and thus not making them available publicly.

Once the gwreq and user instances are provisioned, we commence the actual data collection. Each user instance receives a separate list of 45 randomly chosen target webpages (URLs) that it must download 80 times each. The webpages provided to our simulated users are drawn from a curated set of 540 distinct URLs sourced from the Tranco list of the top-most popular sites [24] as of February 2024. We construct this subset by descending from the top of the ranked list (most popular) and manually reviewing each URL to skip those that are not reachable with Tor Browser (to increase our chances of successful visits through Nym) or contain content that may be considered disreputable or is unintelligible to us.

We initialize a fresh socks5-client for each attempted visit to a URL and thus ensure we use a fresh Nym address for each visit. We further configure socks5-clients to only be reachable by exit network-requesters through SURBs, preventing users from trivially disclosing their Nym addresses to the adversary. The socks5-clients always select the assigned gwreq as both entry and exit. Once the socks5-client has started and accepts application streams via its SOCKS5 proxy, we run a Python script that uses Playwright[6] (version 1.41.0) to launch a Firefox browser instance (version 121.0) and load a public Internet webpage from the list through the socks5-client until the *load* event fires. At that point, the script takes a screenshot of the loaded webpage and cleanly shuts down the browser. We export the state specific to the visit before moving to the next URL. By using a full-fledged browser instead of command-line tools like curl, we reproduce the actual client setup of Nym users more closely and thus generate more realistic traffic.

Our gwreqs create two log files for each socks5-client instance (identified by Nym address) connecting to them in their role as entry gateways. These files track the metadata (timestamp and size) of the socks5-client's Sphinx packets: one for packets sent by the

[6] `https://playwright.dev`

socks5-client towards the mixnet, one for packets arriving from the mixnet towards the socks5-client. At the exits, each gwreq's network-requester component creates three files per webpage download (identified by anonymous sender tag): one for received payload packets, one for sent payload packets, and one for received acknowledgments.

We use Hetzner[7] as the public cloud provider for our data collections and deploy each gwreq and user instance as a CCX33 machine with 8 exclusive vCPUs, 32 GB of memory, and 240 GB of SSD storage at Hetzner's Nuremberg data center location. Instances run Ubuntu 22.04 and compile their Nym processes with Rust in version 1.75.0.

10.5 Undefended and Defended Datasets

We evaluate the shape-based flow correlation attack on Nym and the effectiveness of potential countermeasures using 45 datasets in 85 scenarios. Of the 45 datasets, one captures undefended (default) flows, while each of the remaining 44 datasets represents one particular countermeasure configuration. We collect three of the datasets on the live Nym mixnet using the previously described infrastructure and generate the rest ad-hoc in a reproducible manner, as described in this section.

10.5.1 Undefended Dataset: *default*

We collect dataset *default* to evaluate the impact of our flow correlation attack on the analyzed Nym implementation (i.e., without modifications or specific countermeasures). Except for the patches described above to enable data collection, we do not modify parameters like packet sending rates $\lambda_{\mathbb{F}}$ and $\lambda_{\mathbb{B}}$, per-mix packet delay μ, or packet size. This allows us to treat *default* as the baseline regarding flow correlation susceptibility, with which to compare defended flows in order to assess the effectiveness of countermeasures.

We collected 42,902 flow pairs on the live Nym mixnet in April 2024, each representing a successful visit to one of the 540 selected URLs. Out of all flow pairs, we choose 70 for each of 500 (out of the initial 540) selected URLs to form our final dataset *default* of 35,000 flow pairs (the same number of flow pairs per dataset we collected in our previous work MixMatch [19]). We discard some flow pairs to account for failed visits, i.e., we ensure an equal number of visits (70) per URL from a set of 500 URLs intersected across all three datasets collected on the live Nym mixnet. Following a 70-15-15 split, we designate the 70 visits to each of the first 350 sorted URLs as the training set for our flow correlator (see Section 10.3), the 70 visits to each of the next 75 URLs as the validation set, and the 70 visits to each of the final 75 URLs as our testing set.

[7] `https://www.hetzner.com/cloud/`

Each flow pair is then processed into the form our classifier expects as input, which separates packets into the six sent and received payload and acknowledgment packet sequences shown in Figure 10.1. To align the flows collected at the entry and exit, the sequences' absolute timestamps are converted to be relative to the timestamp of the first downstream payload packet (first value in $F^{payld}_{r \to m}$). We visualize the six sequences of an example flow pair from *default* in KDE form in Figure 10.2.

10.5.2 Defended Datasets

We systematically evaluate countermeasure configurations against our flow correlation attack using 44 datasets, each representing a defense strategy listed in Section 10.3.2. Of these 44 datasets, 34 explore the effects of combinations of defense (1), injecting additional packets, and defense (2), splitting $F^{payld}_{m \to u}$ flows, i.e., one or both defenses being active to varying degrees, thus implementing strategy (3). Dataset *default* is the combination grid's thirty-fifth dataset, where both strategies are inactive. Four additional datasets investigate flow merging, the presented alternative to implementing strategy (1), where multiple webpage visits occur simultaneously. One dataset captures the effects of applying strategy (4), i.e., increasing the packets' per-mix delay μ, while another dataset represents the implementation of strategy (5), increasing the size of payload packets. In the final four datasets, we withhold ACKs from the attack classifier, thus implementing strategy (6).

Combined Splitting and Injection Datasets *splitS_injI*

We construct dataset grid *splitS_injI* along two dimensions: S represents the "number of entry gateways across which to split the downstream data flow $F^{payld}_{m \to u}$", while I represents the "amount of cover traffic added to each $F^{payld}_{m \to u}$". I is defined as a multiplying factor of the average number of packets per $F^{payld}_{m \to u}$, which is $2{,}541$ packets on the training subset of *default*. Thus, $I = 1.0$ results in $2{,}541$ additional packets, while $I = 2.0$ adds twice that amount. We consider 35 parameter combinations, given by $S \in \mathcal{S} = \{1, 2, 4, 8, 16\}$ and $I \in \mathcal{I} = \{0.0, 0.5, 1.0, 2.0, 4.0, 8.0, 16.0\}$. Note that $S = 1$ and $I = 0.0$, i.e., *split1_inj0.0*, is the *default* dataset.

We do not need to collect new data on the live Nym network for *splitS_injI*, as we can simulate the effects of both splitting flows as well as adding traffic by post-processing the *default* dataset. We consider that when both defenses are applied in combination, splitting is applied first to distribute actual payload packets among the S entries. Each of the resulting split sequences is then independently padded.

For $S > 1$, we need an algorithm to assign each downstream payload packet to one of the S entries. We avoid choosing the entry for each packet uniformly at random,

as this would only lead to shapes that resemble downscaled versions of their originals at each of the S entries. Instead, we assign entries using a Markov chain. We start by picking a gateway $0 < s \leq S$ uniformly at random for the first packet in each $F^{payld}_{m \to u}$. The next packet is also assigned to s with probability p or switched to gateway $j \neq s$ with probability $(1 - p) / (S - 1)$. Based on early experiments, we use $p = 0.9$ in our implementation, which makes the sequences of consecutive packets sent through a gateway to be of length 10 on average. This choice for p aims to avoid leaking information about single peaks across too many gateways, while also avoiding to give information about multiple consecutive peaks to any one particular gateway.

Without loss of generality, we assume the adversary operates entry gateway $s = 1$. All packets assigned to other entries are thus unavailable to the adversary and excluded from this point onward for the ongoing experiment. If the adversary controls more than one of the entries the user is using to split their $F^{payld}_{m \to u}$ packets between, the adversary could first merge the partial flows obtained at each adversarial entry and thus recover a larger portion of the original sequence.

The defense's second component adds cover packets to the $F^{payld}_{m \to u}$ obtained by the adversarial gateway after splitting, to simulate the packet injection defense. Rather than designing a specific cover traffic generation scheme, our aim is to establish an upper bound in terms of protection that such a defense can offer. We thus train a separate ANN that decides how to distribute the cover traffic budget I in each flow to optimally degrade the attack's effectiveness. We use a simple network comprising two fully connected layers for generating the injections. This approach is inspired by *generative adversarial networks* [9] and prior work [10]: for each batch, we train the flow correlator model based on injections proposed by the fixed packet injector model, immediately followed by training that packet injector against the now fixed flow correlator. We thus call our defensive packet injector the *generator* and our adversarial flow correlator the *discriminator*.

The generator takes as input a noise vector $\mathbf{z} = (z_1, \ldots, z_n)^{\mathrm{T}}$ (its latent variable), where n is the number of input features and each z_i is drawn i.i.d. from $\mathcal{N}(0, 1)$. The generator learns to generate cover traffic vectors from $\mathbf{z}$ that, when added to a $F^{payld}_{m \to u}$, cause the flow correlator to misclassify the candidate flow pair. For the sake of generality, we only feed noise to the generator to ensure its output is agnostic of the specific flow it is defending. The size of the generator output vector is set to the length of the flows in the dataset. We normalize the generated injection vectors to sum to 1.0 before scaling them to the configured factor I. If flow splitting is active as well (i.e., $S > 1$), we divide the injection budget of $I \cdot 2{,}541$ packets by S to arrive at the actual number of packets we can add to the adversarially observed flow. In Figure 10.5, we visualize the shapes of an example $F^{payld}_{m \to u}$ when only splitting between $S = 4$ gateways and when additionally adding cover traffic of factor $I = 4.0$.

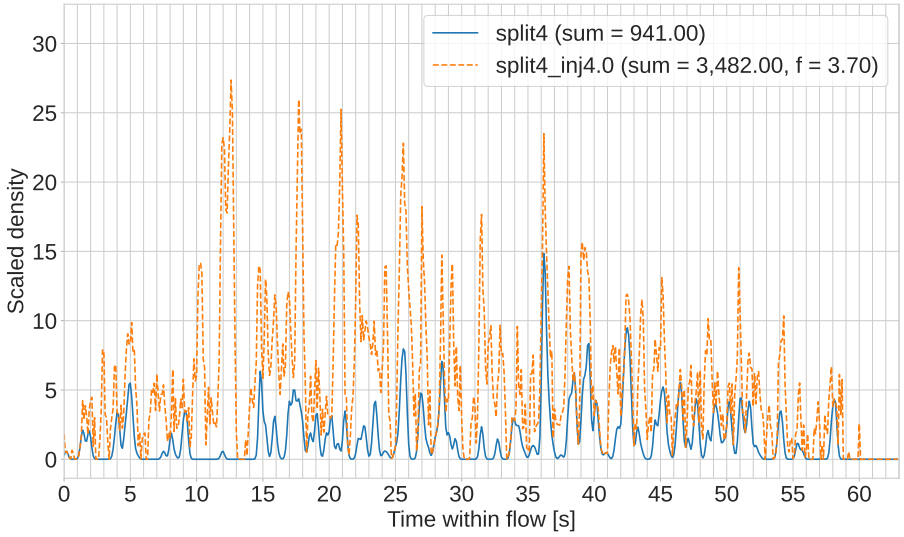


Figure 10.5: One of the $F^{payld}_{m\to u}$ flows in the test set of *split4_inj4.0* in two defended states: only defended by splitting across four entry gateways (solid blue line) and defended by both splitting across four entry gateways while also injecting $2{,}541.0$ packets into it (dashed orange line).

Concurrent Visits Datasets *mergeM*

A different approach to adding unrelated traffic to the entry gateway's view on $F^{payld}_{m\to u}$ involves users engaging in multiple concurrent proxy sessions with different network-requesters (of which some are non-adversarial). We call this scenario *flow merging*. In our setting, this translates to users visiting multiple distinct webpages at the same time in independent sessions, such that their respective $F^{payld}_{m\to u}$ flow includes traffic from various different $F^{payld}_{r\to m}$ flows. We assume for simplicity that each concurrent webpage visit has its own dedicated Poisson process at the users' socks5-clients and that all concurrent visits start at the same time. We simulate and evaluate this countermeasure on the fly, creating the four composite datasets *mergeM* with $M \in \mathcal{M} = \{2, 3, 4, 5\}$ by combining multiple *default* flows. Dataset *merge2* thus contains $F^{payld}_{m\to u}$ flows that are the addition of two independent visits, while *merge5* is the result of merging five $F^{payld}_{m\to u}$. As we follow a supervised learning process, we pick both positive and negative flow pairs from dataset *default* at training and testing time. The composite $F^{payld}_{m\to u}$ flows in the positive class are created by merging M randomly chosen flows, which include the one corresponding to the $F^{payld}_{r\to m}$ flow observed by the adversary at the adversarial exit, while this flow is never one of the M flows merged into the composite $F^{payld}_{m\to u}$ for

the negative class.

Higher Per-Mix Delay Dataset *high_delay*

This dataset corresponds to a configuration that implements defense (4) with higher average per-mix delay μ, where μ is increased from the default 50 [ms] to 200 [ms]. We collected flow pair dataset *high_delay* on the live Nym network, for which we modified the configuration file parameters *average_packet_delay* and *average_ack_delay* on all involved socks5-clients and network-requesters. This data collection took place in April 2024 and resulted in a total of $42{,}401$ flow pairs for *high_delay*, to the same 540 URLs we visited when collecting *default*. In order to ensure comparability across our collected datasets, we select $35{,}000$ of these $42{,}401$ flow pairs such that the list of the 500 visited URLs (each visited 70 times) is the same as it is for *default*, and thus the same in the training, validation, and testing subsets for both datasets.

Larger Packet Size Dataset *large_pkts*

We evaluate defense (5) in dataset *large_pkts*, where we increase the payload packet size by factor eight. In contrast to packet per-mix delay, packet size is a network-wide parameter and typically not a configuration variable that clients unilaterally change (not least because packets of a different size stand out and are trivial to distinguish). The experiment was possible because mixes did not drop our packets due to having too large payloads. For this experiment, we replaced the default packet size *PacketSize::RegularPacket* ($2{,}413$ [bytes]) on socks5-clients and network-requesters with the available larger *PacketSize::ExtendedPacket16* ($16{,}749$ [bytes]), which makes the packets' encapsulated payload eight times larger ($16{,}384$ versus $2{,}048$ [bytes]). Our goal with *large_pkts* is to evaluate whether using fewer but larger packets results in changes to the flows that are significant enough to degrade the attack's performance. We collected $42{,}987$ flow pairs on the live Nym network in late April and early May 2024, and selected $35{,}000$ of them so that the lists of (70 times) visited URLs for the training, validation, and testing splits of *large_pkts* are equal to their counterparts in *default* and *high_delay*.

"No ACKs" Datasets *splitS_injI_noacks*

Finally, the *splitS_injI_noacks* datasets capture the effect of countermeasure (6), i.e., the removal of acknowledgment packets at the mixnet level. Removing ACKs leads to less evidence available for the adversary to correlate flow pairs. We can easily approximate the effect of this design change by not considering the flows of ACKs. Instead of two CNNs, the adversarial flow correlator consists of only one CNN for this dataset, namely, the one called CNN_{payld} in Figure 10.4, convolving $F^{payld}_{r \to m}$ with $F^{payld}_{m \to u}$. We only

aim to understand the effect of this change on the *splitS_injI* grid (not to explore it exhaustively) and thus evaluate it at $S = 1$ and $I = 0.0$ (*default_noacks*), $S = 2$ and $I = 2.0$ (*split2_inj2.0_noacks*), $S = 4$ and $I = 1.0$ (*split4_inj1.0_noacks*), and $S = 4$ and $I = 8.0$ (*split4_inj8.0_noacks*). Note that due to simulating this defense by post-processing *splitS_injI*, we do not capture changes to the packet flows that could result from sending acknowledgments at the application level, e.g., as part of the payloads of packets (piggy-backing on exchanged application data instead of being dedicated Sphinx packets).

10.6 Empirical Evaluation

10.6.1 Experimental Setup

We empirically evaluate the proposed flow correlation attack on a downstream-unregulated mixnet such as Nym and investigate the effectiveness of possible countermeasures. Our method proceeds in three phases: (i) finding the optimal hyperparameters for the involved networks, (ii) training and validating the models for each of our 45 datasets, and (iii) assessing adversarial flow correlation performance across 85 scenarios on previously unseen test data.

Hyperparameter Search and Final Models

Balancing compute time and comparability across all results against the optimal performance of model parameters tuned per dataset, we select a single flow correlator and generator model pairing from the available 45 to tune on. We then extensively tune the models' hyperparameters on this scenario and use the resulting hyperparameters for our entire empirical evaluation. This leads to a single flow correlator model to be trained on any dataset and—if applicable—a single packet-injecting generator model. In our setup, we select *split4_inj2.0* as the dataset to tune on and obtain the globally used hyperparameters. Here, users split their $F^{payld}_{m \to u}$ between four entry gateways while injecting an additional $(2 \cdot 2{,}541.0)/4 = 1{,}270.5$ cover packets into each of their four partial $F^{payld}_{m \to u}$ flows. This dataset and defense scenario is neither the most advantageous nor the worst for either of the two models involved and, importantly, the models initialized with the hyperparameters obtained on this pairing perform relatively well when trained on the other datasets.

Due to the imbalanced nature of our datasets (unpaired flow tuples heavily outnumbering paired ones), the primary measure we use to assess adversarial flow correlation performance in our experiments is *area under the precision-recall curve (PR-AUC)*. The flow correlator model is set to maximize this measure while the packet-injecting generator

Table 10.1: Hyperparameter search spaces for our flow correlator (discriminator) and packet injector (generator). The final parameters determined through tuning are marked in bold.

Network	Variable	Details	Search Space
discriminator	C_1	no. channels out	$\{8, \mathbf{16}\}$
discriminator	K_1	kernel size	$\{\mathbf{12}, 24\}$
discriminator	K_2	kernel size	$\{\mathbf{2}, 4, 8\}$
discriminator	S_1	stride	$\{\mathbf{1}, 2\}$
discriminator	C_2	no. channels out	$\{8, \mathbf{16}\}$
discriminator	K_3	kernel size	$\{\mathbf{6}, 12, 24\}$
discriminator	K_4	kernel size	$\{\mathbf{2}, 4\}$
discriminator	S_2	stride	$\{1, \mathbf{2}\}$
discriminator	D_1	dropout	$\{\mathbf{0.2}, 0.35, 0.5\}$
discriminator	FC_1	no. features out	$\{\mathbf{128}, 256\}$
discriminator	FC_2	no. features out	$\{\mathbf{FC_1/2}, FC_1\}$
discriminator	LR_1	learning rate	$\{0.0002, \mathbf{0.002}\}$
generator	FC_3	no. features in	$\{\mathbf{75}, 150, 300, 600\}$
generator	FC_4	no. features out	$\{75, 150, \mathbf{300}, 600\}$
generator	D_2	dropout	$\{0.2, 0.35, \mathbf{0.5}\}$
generator	LR_2	learning rate	$\{\mathbf{0.0002}, 0.002\}$

model aims to minimize it. This presents us with a challenge during tuning, as we cannot optimize both models' hyperparameters simultaneously on their diametrically opposed objectives. Additionally, changing either model's hyperparameters while also updating the one's it is pitched against leads to a constantly shifting reference point and thus difficulty in assessing if tuning progress is made at all. For this reason, we devise a hyperparameter tuning procedure for opposing networks. We search the hyperparameter space of learning generator models considering one fixed trained flow correlator model to obtain the optimal generator configuration (i.e., with minimum PR-AUC) thus far. Then, we swap positions and train flow correlator models from their hyperparameter space (optimizing for maximum PR-AUC) against the best trained generator model obtained in the previous step. We repeat this process three times before saving the final best hyperparameters for both models. To bootstrap, we use a trained flow correlator with hyperparameters tuned during earlier analyses.

We implement the above tuning procedure in the Ray Tune[8] hyperparameter tuning framework (version 2.23.0). Utilizing Bayesian optimization, we sample the next

[8] `https://docs.ray.io/en/releases-2.23.0/tune/`

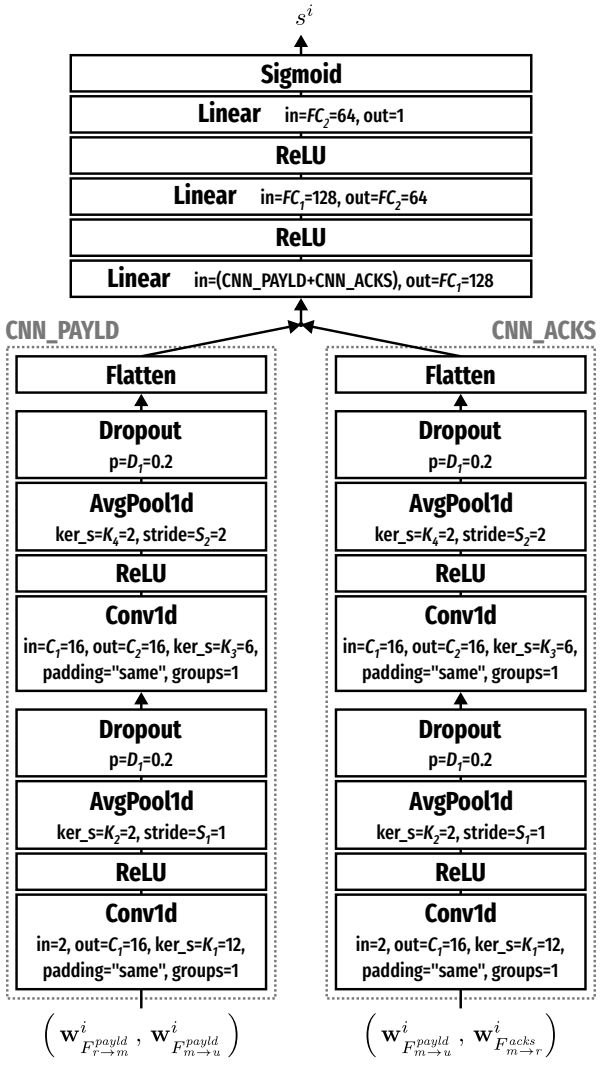


Figure 10.6: Our final flow correlator model (discriminator) after parameter tuning. The hyperparameter variable names correspond to the ones in Table 10.1.

candidate hyperparameters for the currently tuned network (flow correlator or generator) with the Gaussian Process sampler of the Optuna[9] hyperparameter search framework (version 3.6.1) and train the resulting discriminator-generator pair for ten

[9] `https://optuna.org/`

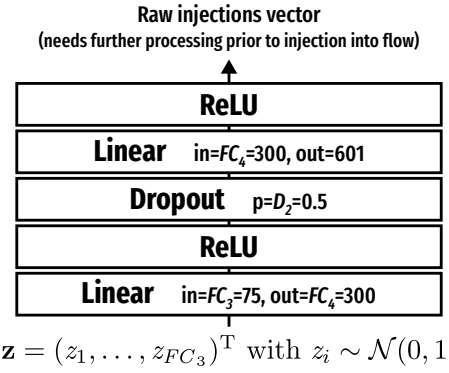


Figure 10.7: Our final packet injector model (generator) after parameter tuning. The hyperparameter variable names correspond to the ones in Table 10.1.

epochs before concluding on its (attack or defense) performance. Table 10.1 details the hyperparameters and search spaces we explored during tuning and the ultimately chosen hyperparameters.

In order to increase the search space we can explore during tuning, we utilize only 30 of the available 70 visits per URL. We also assess more flow correlator than generator hyperparameter configurations, as the flow correlator's overall search space is much larger. Over one week's time, we evaluate 900 flow correlator and 96 generator hyperparameter configurations using four NVIDIA Tesla P100-SXM2-16GB GPUs in parallel. We show the final flow correlator model in Figure 10.6 and the final packet-injecting generator model in Figure 10.7. Mind that for the "no ACKs" experiments, the flow correlator model consists only of the CNN for payload packets, not both CNNs. These hyperparameters determine the model configurations used in the experiments presented below.

Training and Validation

For each of our 45 datasets, we train a flow correlator model for 100 epochs on the dataset's training split and validate its performance after each epoch on the dataset's validation split. For the 33 *splitS_injI* and *splitS_injI_noacks* datasets where $I \geq 0.5$, we co-evolve the flow correlator (discriminator) with a packet-injecting generator model: considering a batch in an epoch, we first fix the generator and have the discriminator learn from the loss in that batch, before switching and training the generator on the same batch while keeping the discriminator static. At the end, we obtain 45 trained

flow correlators and 33 trained generators (trained in tandem with their respective flow correlator).

We load all $350 \cdot 70 = 24{,}500$ paired flow tuples from a dataset's training split. We separate the training, validation, and testing splits of a dataset by URL to ensure that there is no overlap between the flow pairs used to train our models and those used to test a model's performance. For each of the $24{,}500$ positive samples, we construct additional ten negative samples by randomly selecting unpaired $F^{payld}_{m \to u}$ flows from the training split and forming a candidate with the gwreq side from the positive sample. We select five of the ten negatives from the set of visits to the same URL as the positive sample (excluding the actual true positive from the selection) and call these candidates *hard negatives*. The other five negative samples, called *soft negatives*, are drawn from the set of visits to all other URLs. In the edge case of an empty $F^{payld}_{m \to u}$ due to the splitting defense not assigning any packets to the adversarial entry gateway, we skip the respective flow pair entirely during training. To increase reproducibility, we fix as many seeds in our Python environment as possible and instruct PyTorch to, if available, use the deterministic versions of algorithms. When no flow pair is skipped due to being empty, we end up with a training set of $24{,}500 \cdot 11 = 269{,}500$ flow pairs.

We follow a similar process for the dataset's validation split. However, we do not skip flow pairs with empty $F^{payld}_{m \to u}$, as this is a feature of the defense that the flow correlator has to cope with. Based on the validation split's initial $75 \cdot 70 = 5{,}250$ paired flow tuples, we construct a validation dataset with $57{,}750$ flow pairs, again by generating and adding ten negative samples for each positive one. We iterate over both training and validation splits in batches of 110 flow pairs (ten positives and their respective 100 negatives) and shuffle the respective splits at each epoch. Experiments with only a flow correlator model take circa 16–25 hours to train and validate on one NVIDIA Tesla P100-SXM2-16GB GPU, while experiments training and validating both a flow correlator and a generator take circa 26–34 hours on the same GPU model.

Testing

Based on our 45 datasets, we test 85 scenarios. For each, we load the initial $75 \cdot 70 = 5{,}250$ paired flow tuples of the underlying dataset's testing split. However, instead of crafting only ten unpaired flow tuples for each paired one, we form one candidate for each of the available unpaired ones with the gwreq side of the paired flow tuple. Thus, we obtain a testing split of $5{,}250 \cdot 5{,}250 = 27{,}562{,}500$ flow pairs with a base rate of paired flow tuples of $5{,}250 \,/\, 27{,}562{,}500 \approx 1.9 \times 10^{-4}$. We do not skip any flow pairs that might have empty partial flows resulting from the splitting defense.

Representing an adversary aware of the precise system configuration, we evaluate each flow correlator on the testing split of the dataset on which it was trained. In the *splitS_injI* scenarios, this means that flows are modified by the generator model (trained on the same

setting) before being classified by the respective flow correlator. This "aware adversary" setting accounts for 45 of the 85 testing scenarios. The remaining 40 scenarios assess the performance of a naive adversary that is unaware of all countermeasures, i.e., we test a *default*-trained flow correlator on flows from defended datasets.

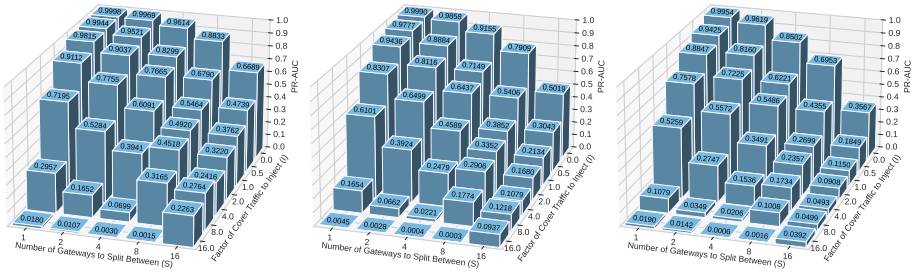
For each of the 85 testing scenarios, we save three flow correlation performance tables that underlie the results we present below. Adopting each of the $1{,}001$ equidistantly spaced values between 0.0 and 1.0 (inclusive) as the threshold for classification, the first table lists the confusion matrix and precision at that threshold when considering all $5{,}250$ paired flow tuples. To gauge whether our flow correlation attack is sensitive to the base rate of paired flow tuples, the second and third tables list the same metrics over the same thresholds vector but considering different base rates: instead of $5{,}250$ paired flow tuples, we select 525 and 53 for the two other performance tables, respectively. This allows us to estimate flow correlation performance at three different orders of magnitude in terms of base rate: 1.9×10^{-4}, 1.9×10^{-5}, and 1.9×10^{-6}. For fairness, we select the 525 or 53 paired scores that are equidistantly spaced on the indices of the sorted list of paired flow tuple scores. On an NVIDIA Tesla P100-SXM2-16GB, each of the 85 testing scenarios takes at most three hours to calculate the correlation score for the $27{,}562{,}500$ flow pairs and to export the performance tables.

10.6.2 Results

This section reports flow correlation performance across the 85 tested scenarios, with the results expressed as PR-AUC. The adversary is interested in maximizing PR-AUC while the objective of deploying defenses is to minimize PR-AUC. Ideally, the adversary would like to achieve a PR-AUC of 1.0, i.e., a perfect classifier that makes no mistakes when guessing whether two candidate flows are paired or unpaired, based on their correlation. The countermeasures, on the other hand, aim to lead the adversary's flow correlator to misclassify, ideally lowering PR-AUC to be no better than random guessing.

default and *splitS_injI*

We show in Figure 10.8 the performance of an adversarial flow correlator aware of the respective countermeasure on the 35 *splitS_injI* datasets (which include *default*), considering three base rates for paired flows (true positives). We see that the flow correlator achieves almost perfect classification (PR-AUCs above 0.995) against the current (undefended) Nym configuration (*default*, i.e., $S = 1$, $I = 0.0$), for any of the three considered base rates. When only applying the splitting defense ($S \in \{ 2, 4, 8, 16 \}$, $I = 0.0$), the greatest protection across all base rates is achieved when $S = 16$ (PR-AUCs of 0.6689, 0.5019, and 0.3567, depending on the base rate). Splitting $F_{m \to u}^{payld}$ flows between just $S = 2$ gateways offers little protection on its own, regardless of base



(a) Base rate of 1.9×10^{-4} (b) Base rate of 1.9×10^{-5} (c) Base rate of 1.9×10^{-6}

Figure 10.8: Adversarial flow correlation performance expressed as PR-AUC over datasets *splitS_injI*. The adversarial flow correlator is aware of and trained in tandem with the respective countermeasure configuration. Subfigures (b) and (c) show how our flow correlation attack scales to orders of magnitude lower base rates of paired flow tuples (obtained by sampling fewer paired scores while retaining all unpaired scores).

rate. If we consider the generator-based packet injection defense in isolation ($S = 1$, $I \in \{ 0.5, 1.0, 2.0, 4.0, 8.0, 16.0 \}$), we also observe that low values for I lead to only small PR-AUC reductions versus *default* results. For higher values of I, however, the decline in PR-AUC becomes significant. The highest protection level is reached at $I = 16.0$, resulting in PR-AUCs of 0.0180, 0.0045, and 0.0190 across the base rates.

The results for *splitS_injI* where $S \in \{ 2, 4, 8, 16 \}$ and $I \in \{ 0.5, 1.0, 2.0, 4.0, 8.0, 16.0 \}$ enable us to evaluate whether combining the two defenses offers higher protection than only the sum of their independent results. In terms of overhead, increasing S or I comes with either increased complexity or increased network burden, which is why we assume that mixnet designers and users choose the lowest values for the two dimensions that fit their trade-off and achieve the desired protection level. When looking at *split4_inj4.0* at base rate 1.9×10^{-4}, we find the reduction in PR-AUC performance this configuration achieves over *default* to be $0.9998 - 0.3941 = 0.6057$. Meanwhile, the sum of the independent protection effects of *split1_inj4.0* ($0.9998 - 0.7195 = 0.2803$) and *split4_inj0.0* ($0.9998 - 0.9614 = 0.0384$) amounts to only 0.3187. This holds for most configurations in *splitS_injI*, indicating that applying multiple defenses in combination provides a compounded protection effect that is larger than the sum of the two defenses applied in isolation.

When we consider the diagonals in each subfigure of Figure 10.8 (along which the amount of injected traffic per partial flow is constant), the combined defense's efficacy increases without exception for higher values of S. However, counterintuitively, our results also show that increasing either S or I only always leads to higher protection as long as either $S \leq 4$ or $I \leq 2.0$. The results where $S \in \{ 8, 16 \}$ and $I \in \{ 4.0, 8.0, 16.0 \}$

do not follow this pattern. For example in Figure 10.8a, *split16_inj16.0* achieves a PR-AUC of 0.2263, while quartering S to 4 and halving I to 8.0 leads to a much lower PR-AUC of 0.0699, an effect observable across all three base rates.

We provide two hypotheses that both may be part of an explanation for these counterintuitive results. First, recall that we conducted our hyperparameter tuning (Section 10.6.1) on configuration $S = 4$ and $I = 2.0$. If the generator model is particularly sensitive to the flow splitting parameter S that it is tuned for, this could (partially) explain why flow correlation performance for $S \geq 8$ increases at $I \geq 8.0$. Second, recall that we evaluate all flow splitting scenarios using a Markov chain with the same fixed remain probability $p = 0.9$. Given that p determines the average length of packet bursts distributed to each of the S entry gateways and thus heavily influences the excerpts of the original flows that the adversarial gwreq gets to see, it may well be the case that a different choice for p would be more effective once $S \geq 8$. We also note that the six counterintuitive results represent defense configurations that are very costly and thus less likely to be deployed in practice. If these defense configurations were indeed considered for deployment, additional experiments exploring different values for p and possibly a new hyperparameter tuning would be needed for a systematic evaluation of such configurations.

Regarding trading off S versus I, our results show opportunities to reduce the adversary's ability to correlate flows while spending different (amounts of) resources to achieve a desired level of protection. For example, halving the overall bandwidth overhead I from $= 2.0$ to $I = 1.0$ while doubling the number of gateways from $S = 2$ to $S = 4$ achieves comparable flow correlation PR-AUCs at any of the three base rates.

default, *mergeM*, and *split1_injI*

The results for datasets *mergeM* versus *default* are shown in Table 10.2. As *mergeM* represent an alternative implementation to the packet injection defense (see Section 10.3.2), we also include *split1_injI* results in the table (where available) to compare the two approaches in terms of effectiveness, given they have similar overheads when $I = M - 1$. The results show that a significant number of flows need to be merged into $F^{payld}_{m \to u}$ in order to meaningfully degrade the adversary's flow correlation performance. While *merge5* is able to lower flow correlation PR-AUC to 0.8321, 0.7393, and 0.6697 across the three respective base rates, merging two flows (*merge2*) barely improves protection over *default*. Furthermore, the results show that the generator-based packet injection approach of *split1_injI* consistently offers better protection than the equivalent flow merging approach *mergeM*. For example, at base rate 1.9×10^{-4}, *split1_inj4.0* achieves a PR-AUC that is 0.1126 lower than *merge5* while using fewer injected packets.

Given a bandwidth budget, adding traffic by injecting cover packets at target times into $F^{payld}_{m \to u}$ flows is clearly a more effective defense in terms of protection than merging

Table 10.2: PR-AUC performance and overhead across datasets and base rates against a tandem-trained discriminator. Packet overhead is established over our 60-seconds analysis window. "OF" stands for "overhead factor".

Dataset	PR-AUC at base rate $1.9\times$			Avg. no.	OF
	10^{-4}	10^{-5}	10^{-6}	packets	
default	0.9998	0.9990	0.9954	2,799.04	0.00
merge2	0.9883	0.9662	0.9292	5,598.03	1.00
split1_inj1.0	0.9815	0.9436	0.8847	5,340.04	0.91
merge3	0.9524	0.9021	0.8457	8,396.98	2.00
split1_inj2.0	0.9112	0.8307	0.7578	7,881.04	1.82
merge4	0.8922	0.8134	0.7414	11,195.83	3.00
merge5	0.8321	0.7393	0.6697	13,994.87	4.00
split1_inj4.0	0.7195	0.6101	0.5259	12,963.04	3.63

flows. Note, however, that flow merging may occur naturally and without wasting any bandwidth when users have simultaneous connections open to multiple public Internet resources through different exits, including long-standing webpage visits featuring frequent background requests ("webapps").

default, *high_delay*, and *large_pkts*

Table 10.3 compares the *default* results to scenarios *high_delay* and *large_pkts*, where, respectively, we increase the average per-mix delay of packets from 50 to 200 [ms] and increase the payload packet size from $2{,}413$ to $16{,}749$ [bytes]. The results show that quadrupling the delay does not significantly prevent the correlation of flow pairs compared to *default*. The largest improvement of *high_delay* over *default* occurs at base rate 1.9×10^{-6} with a PR-AUC 0.0086 lower while causing flows to take $26\,\%$ longer and needing $15\,\%$ more packets and transmitted data. Increasing the payload packet size reduces the number of packets needed to complete a data transmission. In our experiments, flow pairs in *large_pkts* complete $31\,\%$ faster and require $45\,\%$ fewer packets than for *default*, but the overall number of bytes transferred also almost triples due to padding small payloads to a much larger packet size. While increasing the payload packet size does not fully prevent the attack, it becomes increasingly effective at lower base rates: PR-AUC is 0.1266 lower at base rate 1.9×10^{-6} versus *default*.

Table 10.3: PR-AUC performance and overhead across datasets and base rates against a tandem-trained discriminator. The three overhead metrics are established over the full flows (not only their initial 60 seconds). The bytes overhead results from multiplying the number of payload and acknowledgment packets sent towards the mixnet from either endpoint with their respective size. "OF" stands for "overhead factor".

Dataset	PR-AUC at base rate 1.9×			Overhead, full flow					
	10^{-4}	10^{-5}	10^{-6}	Avg. no. seconds	OF	Avg. no. packets	OF	Avg. no. bytes	OF
default	0.9998	0.9990	0.9954	64.78	0.00	12,617.01	0.00	15,693,286.89	0.00
high_delay	0.9996	0.9973	0.9868	81.60	0.26	14,456.93	0.15	18,025,453.37	0.15
large_pkts	0.9757	0.9374	0.8688	44.65	−0.31	6,967.89	−0.45	61,448,898.12	2.92

Table 10.4: PR-AUC performance and overhead across datasets and base rates against a tandem-trained discriminator not using acknowledgment (only payload) packets for flow correlation.

Dataset	PR-AUC at base rate $1.9\times$		
	10^{-4}	10^{-5}	10^{-6}
default	0.9998	0.9990	0.9954
default_noacks	0.9998	0.9986	0.9932
split2_inj2.0	0.7755	0.6499	0.5572
split2_inj2.0_noacks	0.7387	0.6261	0.5384
split4_inj1.0	0.7665	0.6437	0.5486
split4_inj1.0_noacks	0.7266	0.5900	0.4861
split4_inj8.0	0.0699	0.0221	0.0206
split4_inj8.0_noacks	0.0542	0.0176	0.0201

splitS_injI and *splitS_injI_noacks*

Table 10.4 shows the impact of removing acknowledgments as distinct mixnet packets. For the four evaluated "no ACKs" datasets we also include the results of their counterpart datasets (in which the adversary does make use of ACKs to correlate flows). We see the largest effect of removing ACKs compared to the counterpart dataset on dataset *split4_inj1.0_noacks*, with a $6.25\,\%$ decrease in PR-AUC at base rate 1.9×10^{-6}. All other "no ACKs" settings see a smaller to vanishing protective effect, with almost no protection gained from removing ACKs in *default_noacks*.

Unaware Adversary

Finally, we present flow correlation results considering an adversary that is unaware of any deployed countermeasures, i.e., we evaluate a flow correlator that is trained on *default* but tested on datasets that include a countermeasure. Note that users could individually and unilaterally deploy countermeasures that include adding unrelated traffic, splitting flows, and increasing the per-mix delay of packets. Changing the size of packets and discarding acknowledgments, on the other hand, are system-wide configurations that users should not deviate from in order to be part of the anonymity set. In the cases where users unilaterally deploy countermeasures with custom parameters, the adversary is not necessarily aware of the exact user configuration and may thus use a model trained

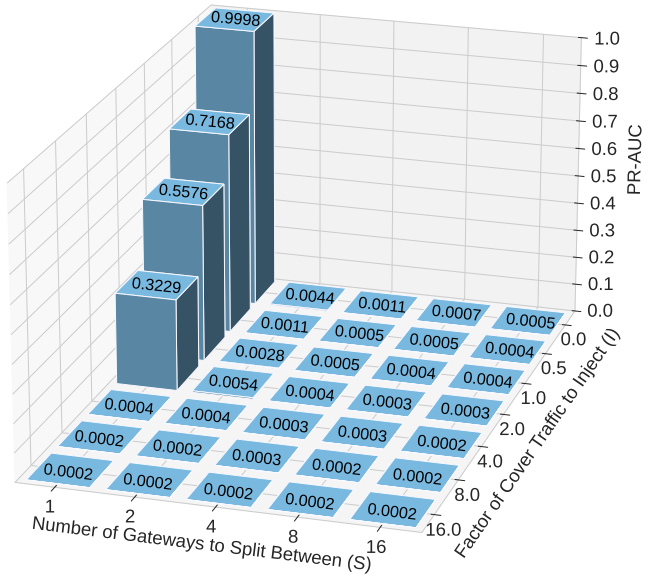


Figure 10.9: Adversarial flow correlation performance expressed as PR-AUC at base rate 1.9×10^{-4} on datasets *splitS_injI*. The adversary is unaware of the respective countermeasure configuration, i.e., the flow correlator is trained on *default* but tested on a dataset modified by the respective defense.

on the *default* configuration. We explore below the success of an adversary in exactly this situation.

In Figure 10.9, we show results for scenarios with flow splitting and packet injections. As we can see, the adversary is able to correlate flows as long as $S = 1$ (i.e., no splitting) and $I \leq 2.0$, but fails to correlate flow pairs in any other configuration. This result is similar across all three base rates and we thus leave out the figures for base rates 1.9×10^{-5} and 1.9×10^{-6}. Table 10.5 shows additional results for countermeasure scenarios with increased per-mix packet delay and flow merging, compared with relevant *splitS_injI* settings. Contrary to previous results where the discriminator is optimized for each countermeasure, in this case, merging up to three flows more effectively mitigates flow correlation than injecting an equivalent amount of packets with a generator. Increasing per-mix packet delays (*high_delay*) is also very effective here.

Table 10.5: PR-AUC performance across datasets and base rates against a countermeasure-unaware (default-trained) discriminator.

Dataset	PR-AUC at base rate $1.9\times$		
	10^{-4}	10^{-5}	10^{-6}
default	0.9998	0.9990	0.9954
high_delay	0.0761	0.0229	0.0209
merge2	0.2733	0.2316	0.2042
split1_inj1.0	0.5576	0.5238	0.4949
merge3	0.0738	0.0600	0.0589
split1_inj2.0	0.3229	0.2581	0.1932
merge4	0.0251	0.0208	0.0231
merge5	0.0090	0.0076	0.0189
split1_inj4.0	0.0004	0.0002	0.0016

10.7 Discussion

10.7.1 Likelihood of the Attack

The presented flow correlation attack relies on the adversary having access to mixnet packet flows on both the entry and exit positions of a connection. The attack is opportunistic rather than targeted: the adversary needs to "get lucky" and be selected by a user as both the entry and exit for the connection to be susceptible to the attack. Assuming the adversary controls C gwreqs out of N total gwreqs active in the Nym mixnet and that all gwreqs are equally likely to be selected, the probability that the adversary can attack a user's connection is $P_c = \frac{C \cdot (C-1)}{N \cdot (N-1)}$. At the time of writing, N hovered around 240 in the Nym network.[10] An adversary controlling the minimum number of gwreqs required to launch our attack, $C = 2$, can expect to attack $\frac{2 \cdot 1}{240 \cdot 239} \approx 0.0035\,\%$ of the connections to public Internet resources. For a higher $C = 10$, the adversary would be able to attack $\frac{10 \cdot 9}{240 \cdot 239} \approx 0.1569\,\%$ of the requests.

Above calculations assume that users choose their gwreqs randomly every time they build a connection. In practice, however, users may reuse gwreqs that they prefer due to their location, being more trusted, or offering better service quality. As is the case in Tor with the use of *entry guards* [21],[11] users who repeatedly pick an honest gwreq in at least

[10] `https://nym.com/explorer/table`
[11] `https://support.torproject.org/about/entry-guards/`

one of the (entry or exit) positions are immune to our attacks, while a significant fraction of connections may be compromised for users who unwittingly reuse adversarial gwreqs.

The attack likelihood can be lowered in at least two ways. First, increasing N requires the adversary to own or compromise a larger number of gwreqs C to be able to attack the same fraction of connections. Clearly, this only works as a mitigation strategy if mechanisms are in place that make it difficult (e.g., expensive) for any entity to operate a large number of gwreqs in the network. Nym requires operators to obtain a certain amount of NYM tokens (a cryptocurrency) as stake to be eligible to operate in the network and receive rewards for their work. These tokens are a finite resource that operators compete for. Thus, as long as it is difficult (or costly) for any entity to accumulate a large share of tokens, increasing N can make the preconditions of our attack less likely.

Second, Nym can reduce the likelihood of our attack by fostering trust building in the community (e.g., creating opportunities for users to engage with operators) and by incentivizing operator transparency (e.g., disclosing which gwreqs are operated by the same entity and building public reputations for operators). This enables users to judge for themselves how trustworthy they find different operators as well as avoid choosing entry and exit from the same operator. Even if only honest operators faithfully disclosed their gwreq sets while adversaries ran multiple allegedly "independent" gwreqs, users could already avoid susceptibility to our attack by selecting one gwreq from a well-known, trusted, and large operator and the other gwreq from some other operator's set.

10.7.2 Mitigation Strategies

Once an adversary does get lucky and occupies the entry and exit positions of a Nym connection, our work demonstrates that flow correlation attacks are effective at linking the two ends of the connection in under one minute. Our results also evaluate a number of defenses that can be deployed to mitigate the attack: additional cover traffic, splitting flows over multiple entries, increasing the packets' per-mix delay, increasing the payload packet size, and removing Sphinx-level ACKs. Further mitigation strategies are possible, such as splitting flows over multiple exits, e.g., fetching each webpage resource via a different exit in web browsing scenarios.

Our results indicate that combining multiple defenses has a positive compounding protection effect. Similar to the scenarios we captured as dataset *splitS_injI*, future work may find it worth exploring the deployment of multiple different defenses at the same time to find the most effective combinations of countermeasures. For example, considering settings with larger packet sizes, no acknowledgments, variable average per-mix delay (per user and connection), and frequent splitting of flows over two or more entries and exits, future work may evaluate what the required level

of cover traffic is to neutralize the attack. Such a "defense in depth" approach also enables optimizing the resource overhead spent on these countermeasures, by finding countermeasure combinations that provide the highest "bang for the buck", i.e., high levels of protection for the lowest possible overhead in terms of latency, (wasted) computation and bandwidth, and complexity.

Given that our attack is enabled by the variable rate of packets in flows due to unregulated sending in the downstream direction, an effective mitigation would be to revert to regulated (i.e., Poisson-based) sending patterns for downstream packets. This would remove the flow features used by our attack and thus defend against it. It would also, however, increase the latency of transmissions, with a potentially large impact on usability. An intermediate approach would be to allow sending with a variable rate as long as it is below a threshold, so that large traffic peaks are avoided. Note that sending downstream packets requires SURB headers prepared by the user, who is thus able to limit the amount of unused SURBs available to an exit at any time, enabling some degree of user control on the downstream traffic rate.

10.7.3 Threat Model

The considered adversary has an extraordinary vantage point with respect to the flows susceptible to the attack: it acts simultaneously as mixnet *endpoint* receiving and sending noise-free flows of application-relevant packets (no cover traffic can be sent to or received from public Internet resources) and as *interface* between the user endpoint (the attack's target) and the mixnet. It may not be practically possible to completely protect flows of variable, arbitrary length with limited latency and overhead (e.g., the amount of added cover traffic) against such a threat. The countermeasures we examined become rather costly at the overhead levels required to neutralize the attack. Furthermore, even with heavy countermeasures, longer flows consisting of even more packets than those in our experiments may still be vulnerable. Additionally, the attack is entirely *passive* (and thus covert), as the adversary is simply recording metadata of packet flows but not taking any action (e.g., dropping or artificially delaying packets) that could alter the flows. An adversary with the considered capabilities (control over entry and exit) can easily become *active*, e.g., by modulating the rate of sent packets to induce a pattern that facilitates (even easier) correlation with the flow received at the other end. While active versions of the attack are practically impossible to protect against, they may be detectable, as the targeted users may notice alterations in their received flows.

The results for the *unaware adversary* (Section 10.6.2) point to a possible limitation for adversaries that deploy the attack in practice. Many of the relevant packet flow parameters can be changed unilaterally by users, including: the per-mix packet delay, the rate of sending packets and SURBs, the amount and shape of cover traffic sent, the splitting of flows over multiple entries (and exits, if possible), and the multiplexing of

sessions (e.g., multi-tab browsing). Except for the sending rate, all other parameters can be modified privately by users without the adversary being immediately aware of the changes. While the flow correlation attack is very successful against users that employ the exact same parameters that the attack is trained for, attack performance degrades rapidly on flows with different parameters. Thus, the attack can be considered parameter-specific. The adversary would need to collect a training dataset representative of all possible parameter choices to generalize the attack, which may be prohibitively expensive and impractical to scale. Also note that in our threat model, a user deploying any of the above defenses benefits from increased protection against flow correlation—even if no other user is deploying them as well. Thus, a changing per-user parameter personalization (or periodic randomization of parameters) may significantly counter the attack in practice.

Finally, an open question remains how well *network adversaries* (observing packet flows at the link level only, without control over or insight into mixnet entities) could pair the two flows of a mixnet connection. Such adversary would need to correlate an application-level flow (e.g., a TLS stream) between an exit and a public Internet resource with a Sphinx-level flow between a user and an entry. The view on a flow between an exit and a public Internet resource would still be noise-free, but the conversion between Sphinx packets and application packets may significantly distort flow features and make correlation more challenging. The network adversary's view on a flow between a user and an entry is similar to what the entry sees. However, any caching done by the entry or link-level cover traffic exchanged between a user and the entry can significantly increase the flow's obfuscation towards adversaries in the underlying network. An evaluation of this threat requires a different data collection approach (collecting data as seen by the underlying network) and training the classifier to match mixnet packet flows to application flows in other packet formats, as well as to account for any link-level cover traffic exchanged between users and their entries.

10.7.4 Limitations

Our experiments simulate the effects of potential deployments of countermeasures on the live Nym network. These simulations make simplifications and therefore do not capture live network effects that may affect defense effectiveness. For example, our generator injects cover packets directly into static traces and thus does not model complex behaviors of a live network such as those that result from queuing and congestion control mechanisms of a full-stack network implementation. These simplifications may thus have resulted in flow shapes that differ from a real-world deployment of these countermeasures (e.g., in burst length or density distribution). However, we expect the overall effects of these simplifications to be small and the general trends we observed (Figure 10.8) to hold.

As a minor point, note that we avoid synchronization issues in our experimental setup by having the entry and exit be the same machine. In practice, the adversarial entry and exit will be running on different machines, requiring synchronization of their clocks. Any drift between timestamps due to imperfect synchronization may reduce the accuracy of the attack.

10.7.5 Responsible Disclosure

From the first stages of this work we maintained an open and ongoing collaboration with the Nym team (which one of the authors is part of), as this work addresses practically relevant questions for evaluating and improving the system design. The Nym team has thus been aware of preliminary results early on and has provided feedback. In response to this work's findings, the Nym project is considering the deployment of a set of countermeasures aimed at mitigating the impact of the described attack. Implementation planning is already underway for several of these measures, namely: increasing the packet payload size, removing mixnet-level acknowledgments, and providing users with customizable settings for (i) variable mixing delays, (ii) flow splitting over multiple entries and exits, and (iii) variable amounts of cover traffic.

10.8 Related Work

10.8.1 Comparison With MixMatch

We previously examined the problem of linking entities that exchange packet flows via a mixnet like Nym in MixMatch [19]. While there are similarities in overall objective and adversarial model between MixMatch and this work, there are also important differences. MixMatch requires assuming that endpoints communicating through the mixnet do so in *closed sets*, i.e., not a single packet is sent to or received from endpoints outside the set. A closed set can be of size one (an endpoint exclusively sending cover traffic to themself) or involve multiple entities that exchange packets among them. Additionally, the MixMatch adversary needs to control the gateways of *all* endpoints within a closed set to be able to successfully *match* that group of endpoints and determine that they are exchanging packets. MixMatch's flow matching performance degrades quickly when the closed set assumption is violated and can thus be mitigated with cover traffic that leaves the targeted set.

In contrast, this work does not require the closed set assumption. This is primarily due to considering a different threat model: MixMatch assumes that some gateways are compromised but the adversary does not control any endpoints. Gateways cannot tell which packet belongs to which flow at a connected endpoint or whether packets

are cover or genuine traffic. However, we assume a different threat model for this work, where users connect through the mixnet to (potentially compromised) network-requester endpoints, allowing the adversary to distinguish payload packets belonging to different flows as well as the network-requester's own cover traffic.

Furthermore, this work accounts for changes since introduced by Nym and aimed at speeding up flows from network-requesters to users that are key to enabling our new attack. The version of Nym evaluated in MixMatch worked as a bidirectionally regulated mixnet, in which endpoints (socks5-clients as well as network-requesters) shape their packet emissions following a Poisson process. Due to this, the only effective measure for deciding whether a candidate flow pair is a match was the temporal *drift* of the difference between the times of packets leaving and entering the mixnet. If a candidate pair's timestamp difference followed the distribution we knew held for matched flows, it was matched, otherwise it was unmatched. The version of Nym analyzed in this work removed the packet emission regulation on network-requesters, which now send packets as fast as possible to the mixnet, significantly reducing download times. This change introduces rate variability into flows that were previously of constant rate on average. We find that the *shape* of a flow (featuring packet bursts due to a changing packet rate) persists as it traverses the mixnet, allowing us to classify candidate flow pairs into *paired* or *unpaired* based on their shapes (Section 10.3).

10.8.2 Traffic Analysis Attacks on Anonymity Systems

We focus below on reviewing traffic analysis works based on *deep learning (DL)* methods, which have become predominant in recent years. We refer to MixMatch [19] for a more comprehensive overview of the literature, including statistical techniques and earlier anonymity systems.

Flow Correlation on Tor

Even though Tor was not designed to protect against the adversaries typically assumed in flow correlation, its popularity and sensitive use cases have highlighted the need to consider these attacks within its threat model. Consequently, research on flow correlation in Tor has grown significantly in recent years. Nasr et al. [16] were the first to apply DL techniques to flow correlation on Tor. While their DeepCorr attack shares the same goal as ours (i.e., determine whether entry and exit flows are paired), they assume that the adversary has full visibility over all pairs, reducing flow correlation to a bipartite graph matching problem between entry and exit flows. We consider a more challenging setting, where the adversary only sees a small fraction of all flows. Oh et al. [18] improved on DeepCorr by using contrastive learning and amplification techniques to reduce false positives. Similar to us, they assume an adversary with limited visibility and argue

that reducing false positives is critical in such a setting. We followed their windowed flow representation because an aggregate decision over multiple windows *amplifies* differences between flows, raising the bar for a positive decision and thus reducing false positives. Chawla et al. [3] proposed a transformer architecture that allows incorporating information of the whole flow within each window, further increasing amplification. Although the features of Tor flow correlation attacks also capture the shape of the traffic, they are specifically designed for Tor. In contrast, our attack uses a custom flow representation and ANN architecture designed to exploit Nym's unregulated downstream flows. Our model architecture is significantly simpler than those designed for Tor traffic, as our KDE representation already extracts useful features for flow correlation, reducing the complexity of the learning process.

Website Fingerprinting Attacks

Unlike in flow correlation, a website fingerprinting adversary is local, typically sitting near the client, and their goal is to learn which website a user visits through an anonymity system. Despite this difference, website fingerprinting techniques have inspired modern DL-based flow correlation attacks. In particular, some of the first DL-based flow correlation attacks [3, 16, 18] share the same underlying architecture as the first DL-based website fingerprinting attacks [25, 27, 28]. While the vast majority of research on website fingerprinting focuses on Tor, it has also been explored for mixnets. Wicky et al. [29] have evaluated the vulnerability of Nym to website fingerprinting attacks, demonstrating that Nym is susceptible to them, but significantly less so than Tor. The authors prioritized interpretable features and thus favored attacks based on traditional learning algorithms (e.g., random forests) over DL-based ones.

Countermeasures

Traffic analysis countermeasures follow various strategies, including defensive dropping, traffic splitting, packet padding, artificial delays, and cover traffic, with the aim to perturb the traffic features that can be exploited by attacks. Nym already pads packets to a fixed size and adds artificial delays as well as loop cover traffic, leaving room for additional countermeasures based on flow splitting, multiplexing, and adding unrelated cover traffic, such as the ones we have evaluated. Removing acknowledgments and increasing the packet payload size also help mitigate the attack, as well as adding variability to average mixing delays.

An alternative to the cover packet injection defense that we have evaluated is to drop the cover packets at an intermediate mix along their paths [4, 14]. However, dropping packets at intermediate mixes would require modifications to how Nym works that we have not explored and leave for future work. Finding the optimal cover traffic strategy

is a difficult problem. In the website fingerprinting literature [15], this problem has been approached from an adversarial learning angle, where the defender uses the loss of the attacker's classifier as feedback to optimize the performance of a cover traffic generator model [10, 17]. We have followed a similar approach to train the generator model that implements our packet injection defense.

Another defense mechanism we investigated is traffic splitting, where portions of the traffic are routed through communication channels that the adversary cannot observe. Traffic splitting has been evaluated as a defense against website fingerprinting [7], where DL-based flow correlation attacks have been applied to merge the resulting traffic splits and reconstruct the original flow [13].

10.9 Conclusions

We have evaluated the threat of flow correlation on mixnets that send packets with a variable rate in at least one direction, such as the Nym mixnet, as well as possible countermeasures. Our threat model assumes the adversary to operate at least two gateway-requesters, which users have to choose for both the entry and exit positions of their packet routes to be vulnerable. The attack looks at the correlation between two flows, one between a user and its entry and another between an exit and a public Internet resource, to decide whether they are part of the same flow pair. Due to the exit acting as an endpoint at the mixnet level, our defenses focus on distorting the user-side flow shapes. Our experimental results on data collected from the live Nym network show that if the attack's conditions are met, linking entry to exit flows is fast and accurate in the current configuration. By simulating 84 countermeasure scenarios, we evaluated five mitigation strategies deployed in isolation to counter the attack as well as a combined defense (cover traffic injection and flow splitting). Our results point to the synergetic benefits of layering multiple defenses, which also facilitates trading off the required overheads. In particular, the attack can be mitigated by removing mixnet-level acknowledgments, increasing the mixnet packet payload size, splitting flows across entries and exits, allowing users to personalize parameters such as mixing delays and cover traffic levels, and limiting the send rate of downstream traffic to prevent large traffic peaks to the extent possible while remaining compatible with the mixnet's usability requirements. These recommendations have been taken up by the Nym team and are planned for deployment to mitigate the attack, underscoring the practical impact and real-world contributions of our work. Finally, future research should investigate the two adjacent threats of active and network-level flow correlation in order to arrive at a comprehensive understanding of the risk that flow correlation poses to state-of-the-art mixnets like Nym.

Acknowledgments

We did not use generative AI in the creation of this work. We thank the Nym team and in particular Ania Piotrowska for meeting with us early on and providing valuable feedback. This work was supported by the Flemish Government through the Cybersecurity Research Program with grant number: VOEWICS02. Lennart Oldenburg was funded by FWO fellowship 11G3923N. The resources and services used to train and test our flow correlator and defenses were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation - Flanders (FWO) and the Flemish Government.

Bibliography

[1] D. Chaum, D. Das, F. Javani, A. Kate, A. Krasnova, J. De Ruiter, and A. T. Sherman. "cMix: Mixing with Minimal Real-Time Asymmetric Cryptographic Operations". In: *Applied Cryptography and Network Security*. Springer International Publishing, 2017, pp. 557–578 (p. 207).

[2] D. L. Chaum. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms". In: *Commun. ACM* 24.2 (1981), pp. 84–90 (p. 207).

[3] T. Chawla, S. Mittal, N. Mathews, and M. Wright. "ESPRESSO: Advanced End-to-End Flow Correlation Attacks on Tor". In: *Proceedings of the 8th Asia-Pacific Workshop on Networking*. 2024, pp. 219–220 (p. 245).

[4] H. Daginawala and M. Wright. "Studying Timing Analysis on the Internet with SubRosa". In: *Proceedings on Privacy Enhancing Technologies (PoPETs)*. Vol. 2008. Springer Berlin Heidelberg, 2008, pp. 133–150 (p. 245).

[5] G. Danezis, R. Dingledine, and N. Mathewson. "Mixminion: Design of a Type III Anonymous Remailer Protocol". In: *2003 Symposium on Security and Privacy*. 2003, pp. 2–15 (p. 212).

[6] G. Danezis and I. Goldberg. "Sphinx: A Compact and Provably Secure Mix Format". In: *2009 30th IEEE Symposium on Security and Privacy (SP)*. IEEE, 2009, pp. 269–282 (p. 211).

[7] W. De la Cadena, A. Mitseva, J. Hiller, J. Pennekamp, S. Reuter, J. Filter, T. Engel, K. Wehrle, and A. Panchenko. "Trafficsliver: Fighting website fingerprinting attacks with traffic splitting". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020, pp. 1971–1985 (p. 246).

[8] C. Diaz, H. Halpin, and A. Kiayias. *The Nym Network*. Version 1.0. Feb. 2021. URL: `https://nymtech.net/nym-whitepaper.pdf` (p. 207).

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*. Vol. 27. Curran Associates, Inc., 2014, pp. 2672–2680 (p. 224).

[10] J. K. Holland, J. Carpenter, S. E. Oh, and N. Hopper. "DeTorrent: An Adversarial Padding-only Traffic Analysis Defense". In: *Proceedings on Privacy Enhancing Technologies (PoPETs)*. Vol. 2024. 2024, pp. 98–115 (pp. 224, 246).

[11] J. van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. "Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis". In: *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP '15)*. Association for Computing Machinery, 2015, pp. 137–152 (p. 207).

[12] D. Kesdogan, J. Egner, and R. Büschkes. "Stop- and- Go-MIXes Providing Probabilistic Anonymity in an Open System". In: *Information Hiding*. Ed. by D. Aucsmith. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 83–98 (p. 207).

[13] G. Kim, H. Kwak, S. Kim, Y. Park, J. Park, and S. E. Oh. "DeepCoAST: Unveiling Split Trace Correlation to Counter Traffic Splitting Defenses". In: *IEEE Access* 12 (2024), pp. 158266–158281 (p. 246).

[14] B. N. Levine, M. K. Reiter, C. Wang, and M. Wright. "Timing Attacks in Low-Latency Mix Systems". In: *Financial Cryptography*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 251–265 (p. 245).

[15] N. Mathews, J. K. Holland, S. E. Oh, M. S. Rahman, N. Hopper, and M. Wright. "SoK: A Critical Evaluation of Efficient Website Fingerprinting Defenses". In: *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 969–986 (p. 246).

[16] M. Nasr, A. Bahramali, and A. Houmansadr. "DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning". In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Association for Computing Machinery, 2018, pp. 1962–1976 (pp. 244, 245).

[17] M. Nasr, A. Bahramali, and A. Houmansadr. "Defeating DNN-Based Traffic Analysis Systems in Real-Time With Blind Adversarial Perturbations". In: *30th USENIX Security Symposium (USENIX Security '21)*. USENIX Association, 2021, pp. 2705–2722 (p. 246).

[18] S. E. Oh, T. Yang, N. Mathews, J. K. Holland, M. S. Rahman, N. Hopper, and M. Wright. "DeepCoFFEA: Improved Flow Correlation Attacks on Tor via Metric Learning and Amplification". In: *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1915–1932 (pp. 218, 244, 245).

[19] L. Oldenburg, M. Juarez, E. Argones Rúa, and C. Diaz. "MixMatch: Flow Matching for Mixnet Traffic". In: *Proceedings of the 24th Privacy Enhancing Technologies Symposium (PETS 2024)*. Vol. 2024. 2024, pp. 276–294. DOI: 10.56553/popets-2024-0050 (pp. 208, 213, 215, 222, 243, 244).

[21] L. Overlier and P. Syverson. "Locating Hidden Servers". In: *2006 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2006 (p. 239).

[22] A. Pfitzmann and M. Hansen. *A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management*. v0.34. 2010 (p. 207).

[23] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis. "The Loopix Anonymity System". In: *26th USENIX Security Symposium (USENIX Security '17)*. USENIX Association, 2017, pp. 1199–1216 (p. 207).

[24] V. L. Pochat, T. V. Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen. "TRANCO: A Research-Oriented Top Sites Ranking Hardened Against Manipulation". In: *26th Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2019 (p. 221).

[25] V. Rimmer, D. Preuveneers, M. Juarez, T. van Goethem, and W. Joosen. "Automated Website Fingerprinting through Deep Learning". In: *25th Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2018 (p. 245).

[26] F. Shirazi, M. Simeonovski, M. R. Asghar, M. Backes, and C. Diaz. "A Survey on Routing in Anonymous Communication Protocols". In: *ACM Comput. Surv.* 51.3 (June 2018). ISSN: 0360-0300. DOI: 10.1145/3182658. URL: https://doi.org/10.1145/3182658 (p. 207).

[27] P. Sirinam, M. Imani, M. Juarez, and M. Wright. "Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning". In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Association for Computing Machinery, 2018, pp. 1928–1943 (p. 245).

[28] P. Sirinam, N. Mathews, M. S. Rahman, and M. Wright. "Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 1131–1148 (p. 245).

[29] S. Wicky, R. F. Ribeiro, A. M. Piotrowska, and C. Troncoso. *Empirical evaluation of traffic fingerprinting against the Nym mixnet*. Tech. rep. Nym Technologies, SA, and EPFL, 2024 (p. 245).

Statement on the use of Generative AI

Except for translating the Popularized Abstract and the Abstract into their Dutch versions with the help of DeepL Pro, I did not use generative AI assistance tools during the research and writing process of my thesis.
The text, code, and images in this thesis are my own (unless otherwise specified). Generative AI has only been used in accordance with the KU Leuven guidelines and appropriate references have been added. I have reviewed and edited the content as needed and I take full responsibility for the content of the thesis.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING
Kasteelpark Arenberg 10, box 2452, Leuven
B-3001 Leuven
lennart.oldenburg@esat.kuleuven.be
https://www.esat.kuleuven.be/cosic/

