Title
Scalable Techniques for Security and Anonymity in Distributed Systems

Permalink
https://escholarship.org/uc/item/03c0z1bh

Author
Akhoondi, Masoud

Publication Date
2015

Peer reviewed|Thesis/dissertation
UNIVERSITY OF CALIFORNIA
RIVERSIDE

Scalable Techniques for Security and Anonymity in Distributed Systems

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Masoud Akhoondi

December 2015

Dissertation Committee:

Dr. Harsha V. Madhyastha, Co-Chairperson
Dr. Srikanth V. Krishnamurthy, Co-Chairperson
Dr. Zhiyun Qian
Dr. Vagelis Hristidis
The Dissertation of Masoud Akhoondi is approved:

________________________________________

________________________________________

________________________________________

Committee Co-Chairperson

Committee Co-Chairperson

University of California, Riverside
Acknowledgments

It is an honor for me to have had the opportunity to work with my advisor Prof. Harsha V. Madhyastha. I am deeply grateful to Harsha for the invaluable guidance and his generous support during my PhD. Without his brilliant and insightful advice he has given me, this dissertation would not have been possible and this will continue to benefit my entire research life. Harsha is very enthusiastic and one of the smartest people that I know, he was very adequate in our meetings or when it came to those late night deadlines. I hope that I could be as helpful, energetic, and dedicated as Harsha is.

I am extremely fortunate to collaborate and work with Prof. Srikanth Krishnamurthy during second year of my PhD. I would like to express my deepest gratitude to him for his scientific advice, knowledge, many insightful discussions and suggestions.

I would like to express immense appreciation to Prof. Zhiyun Qian for his advice and mentoring not only when I was intern at NEC Laboratories, but also during fifth year of my PhD on the project that I started during my internship. I would also like to thank all the people at NEC Laboratories that I worked with.

A big shout-out to my labmates Dorian, Curtis, Zhe, Indrajeet and Ali at distributed systems lab with whom I spent hours understanding problems and providing great feedback on all the aspects of the work done on this dissertation. I would also like to thank all my friends at UCR for being part of this wonderful journey. I will forever cherish the memorable times at UCR with my friends Hamid, Mohammad, Moloud, Pamela, Shailendra and Steve.

I cannot find words to express my gratitude to my family, especially to my parents for their constant love, patience and support through my life. Without them, I could not have made this achievement.
To my parents for their constant support and unconditional love.
ABSTRACT OF THE DISSERTATION

Scalable Techniques for Security and Anonymity in Distributed Systems

by

Masoud Akhoondi

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, December 2015
Dr. Harsha V. Madhyastha, Co-Chairperson
Dr. Srikanth V. Krishnamurthy, Co-Chairperson

Security and privacy in distributed systems are long-standing hard problems. On the one hand, solutions for anonymous communications over the Internet are either vulnerable to traffic analysis or offer poor performance. On the other hand, compromises within enterprises remain hard to track down due to complex dependencies between hosts, applications, and their data.

In this thesis, I develop two solutions to improve the anonymity vs. performance trade-off for communications over the Internet. LASTor improves performance of Tor by modifying path selection algorithm and it also mitigate traffic analysis attack by detecting common autonomous system (AS) across the entry and exit segments of a circuit and avoiding using those paths. LASTor reduces median latencies of visiting top 200 websites by 25% while the false negative rate of not detecting a potential snooping AS from 57% to 11%. Next solution, Innominate, is a new framework for anonymous online communication that both offers traffic analysis resistant strong anonymity and scalable performance. Innominate adopts relay-based technique for low latency communication, however instead of a single client servers as a relay, group relay is used to provide strong anonymity.

As of security inside enterprises, I develop DeltaTrack, the first enterprise attack
forensics system that leverages differential dependency tracking to automate the pruning of irrelevant nodes and edges in the backtracking graph. DeltaTrack continuously monitors system call events from all hosts and summarizes their common execution behaviors in a reference model. Then, the reference model is leveraged to prune away frequently observed events across many hosts since they are unlikely to be relevant to the intrusion. DeltaTrack can reduce the number of nodes and edges of the backtracking graph by up to 131x and 512x, respectively, while maintaining its accuracy.
Contents

List of Figures xi
List of Tables xiv

1 Introduction 1

1.1 Anonymous Communications 2
1.2 Dependency Tracking for Attack Forensics in Enterprises 3
1.3 Thesis and Contributions 4

1.3.1 Privacy in Distributed Systems 4
1.3.2 LASTor 5
1.3.3 Innominate 6
1.3.4 Security in Distributed Systems 7
1.4 Organization 8

2 LASTor: A Low-Latency AS-Aware Tor Client 9

2.1 Background and motivation 10

2.1.1 Tor overview 11
2.1.2 Motivation 12

2.2 Overview 17

2.2.1 Problem statement 17
2.2.2 Measurement datasets 18

2.3 Path Selection 19

2.3.1 Preferential selection of low-latency paths 20
2.3.2 Clustering of relays 22
2.3.3 Accounting for distributed destinations 27
2.3.4 Latency versus anonymity tradeoff 29

2.4 AS Awareness 33

2.4.1 AS set estimation 33
2.4.2 Avoiding snooping ASes 35
2.4.3 Evaluation of AS-awareness 37
2.4.4 Impact of AS-awareness on path latency 39

2.5 Implementation 41

2.5.1 Client in action 41
2.5.2 Modification of default Tor client 42
2.5.3 Input datasets 43

2.6 Discussion 44
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.6</td>
<td>2.6.1</td>
<td>Accounting for dynamic load</td>
<td>44</td>
</tr>
<tr>
<td>2.6</td>
<td>2.6.2</td>
<td>Load balancing</td>
<td>46</td>
</tr>
<tr>
<td>2.7</td>
<td></td>
<td>Related Work</td>
<td>47</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Innominate: Strongly Anonymous, Yet Scalably Performant, Online Communications</td>
<td>52</td>
</tr>
<tr>
<td>3.1</td>
<td></td>
<td>Introduction</td>
<td>52</td>
</tr>
<tr>
<td>3.2</td>
<td></td>
<td>Background and Related Work</td>
<td>55</td>
</tr>
<tr>
<td>3.3</td>
<td></td>
<td>Overview</td>
<td>58</td>
</tr>
<tr>
<td>3.3.1</td>
<td></td>
<td>Challenges</td>
<td>61</td>
</tr>
<tr>
<td>3.3.2</td>
<td></td>
<td>Threat Model</td>
<td>62</td>
</tr>
<tr>
<td>3.4</td>
<td></td>
<td>Design</td>
<td>63</td>
</tr>
<tr>
<td>3.4.1</td>
<td></td>
<td>Distributed Anonymity Sets Creation</td>
<td>63</td>
</tr>
<tr>
<td>3.4.2</td>
<td></td>
<td>Distributed Group Creation</td>
<td>66</td>
</tr>
<tr>
<td>3.4.3</td>
<td></td>
<td>Group-based Traffic Routing</td>
<td>70</td>
</tr>
<tr>
<td>3.5</td>
<td></td>
<td>Implementation</td>
<td>71</td>
</tr>
<tr>
<td>3.5.1</td>
<td></td>
<td>Registration</td>
<td>71</td>
</tr>
<tr>
<td>3.5.2</td>
<td></td>
<td>Communication with destination</td>
<td>71</td>
</tr>
<tr>
<td>3.6</td>
<td></td>
<td>Anonymity Analysis</td>
<td>72</td>
</tr>
<tr>
<td>3.6.1</td>
<td></td>
<td>Control Plane</td>
<td>72</td>
</tr>
<tr>
<td>3.6.2</td>
<td></td>
<td>Data Plane</td>
<td>74</td>
</tr>
<tr>
<td>3.6.2.1</td>
<td></td>
<td>Group Anonymity</td>
<td>75</td>
</tr>
<tr>
<td>3.6.2.2</td>
<td></td>
<td>The originator Anonymity</td>
<td>75</td>
</tr>
<tr>
<td>3.7</td>
<td></td>
<td>Performance Analysis</td>
<td>78</td>
</tr>
<tr>
<td>3.7.1</td>
<td></td>
<td>Innominate Instantiation</td>
<td>78</td>
</tr>
<tr>
<td>3.7.1.1</td>
<td></td>
<td>Analysis of Anonymity Sets in Social Networks</td>
<td>79</td>
</tr>
<tr>
<td>3.7.1.2</td>
<td></td>
<td>Group</td>
<td>81</td>
</tr>
<tr>
<td>3.7.2</td>
<td></td>
<td>Facilitators</td>
<td>82</td>
</tr>
<tr>
<td>3.7.3</td>
<td></td>
<td>Throughput of Innominate</td>
<td>83</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>DeltaTrack: Differential Dependency Tracking for Attack Forensics</td>
<td>86</td>
</tr>
<tr>
<td>4.1</td>
<td></td>
<td>Motivation</td>
<td>90</td>
</tr>
<tr>
<td>4.1.1</td>
<td></td>
<td>Sources of Dependency Explosion</td>
<td>91</td>
</tr>
<tr>
<td>4.1.2</td>
<td></td>
<td>Limitations of Existing Heuristics</td>
<td>95</td>
</tr>
<tr>
<td>4.2</td>
<td></td>
<td>Challenges and Assumptions</td>
<td>97</td>
</tr>
<tr>
<td>4.2.1</td>
<td></td>
<td>Assumptions</td>
<td>97</td>
</tr>
<tr>
<td>4.2.2</td>
<td></td>
<td>Challenges</td>
<td>98</td>
</tr>
<tr>
<td>4.2.3</td>
<td></td>
<td>Threat Model</td>
<td>99</td>
</tr>
<tr>
<td>4.3</td>
<td></td>
<td>Design</td>
<td>100</td>
</tr>
<tr>
<td>4.3.1</td>
<td></td>
<td>Definitions</td>
<td>100</td>
</tr>
<tr>
<td>4.3.2</td>
<td></td>
<td>The Reference Model of Relevancy Scores</td>
<td>101</td>
</tr>
<tr>
<td>4.3.3</td>
<td></td>
<td>K-hop Backtracking Algorithm</td>
<td>104</td>
</tr>
<tr>
<td>4.3.4</td>
<td></td>
<td>Security Analysis of the K-hop Algorithm</td>
<td>105</td>
</tr>
<tr>
<td>4.4</td>
<td></td>
<td>Implementation</td>
<td>108</td>
</tr>
<tr>
<td>4.5</td>
<td></td>
<td>Evaluation</td>
<td>111</td>
</tr>
<tr>
<td>4.5.1</td>
<td></td>
<td>Analysis of Real Attacks</td>
<td>111</td>
</tr>
<tr>
<td>4.5.2</td>
<td></td>
<td>Reference Model Evaluation</td>
<td>117</td>
</tr>
<tr>
<td>4.6</td>
<td></td>
<td>Discussion</td>
<td>120</td>
</tr>
<tr>
<td>4.7</td>
<td></td>
<td>Related Work</td>
<td>122</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Random relay selection can inflate end-to-end latencies due to circuitous routing</td>
<td>12</td>
</tr>
<tr>
<td>2.2</td>
<td>Comparison of latencies on the direct Internet path, with Shortest Path routing on Tor, and with the default Tor client</td>
<td>13</td>
</tr>
<tr>
<td>2.3</td>
<td>An example in which an AS (AS2) can subvert the client’s anonymity by correlating traffic across the entry and exit segments</td>
<td>14</td>
</tr>
<tr>
<td>2.4</td>
<td>False negatives in detecting snooping ASes with default Tor client</td>
<td>15</td>
</tr>
<tr>
<td>2.5</td>
<td>Distribution of relays across ASes in PL-BGP-Rand and PL-Tor-Web datasets</td>
<td>18</td>
</tr>
<tr>
<td>2.6</td>
<td>Comparison of end-to-end latency when using geographical distance versus the use of path latency in the Weighted Shortest Path algorithm</td>
<td>20</td>
</tr>
<tr>
<td>2.7</td>
<td>WSP results in greater preference for paths through relays located close to the direct line between the client and the destination</td>
<td>22</td>
</tr>
<tr>
<td>2.8</td>
<td>Clustering with higher cell sizes provides better resilience</td>
<td>24</td>
</tr>
<tr>
<td>2.9</td>
<td>Clustering of relays reduces the probability of an adversary compromising a large fraction of paths</td>
<td>25</td>
</tr>
<tr>
<td>2.10</td>
<td>Clustering of relays increases the length of the chosen path</td>
<td>26</td>
</tr>
<tr>
<td>2.11</td>
<td>WSP yields latencies lower than those obtained with the default Tor client</td>
<td>27</td>
</tr>
<tr>
<td>2.12</td>
<td>Lower latencies obtained with WSP when accounting for distributed destinations</td>
<td>28</td>
</tr>
<tr>
<td>2.13</td>
<td>Increasing the value of $\alpha$ when using WSP results in (a) higher latencies and (b) greater entropy of path selection</td>
<td>30</td>
</tr>
<tr>
<td>2.14</td>
<td>End-to-end distances on paths chosen with WSP when using $\alpha$ to tailor the set of relays from which we select entry guards</td>
<td>31</td>
</tr>
<tr>
<td>2.15</td>
<td>Relative changes in AS path length data across days</td>
<td>36</td>
</tr>
<tr>
<td>2.16</td>
<td>The probability of existence of snooping ASes across (src, dst) pairs in the PL-BGP-Rand dataset</td>
<td>37</td>
</tr>
<tr>
<td>2.17</td>
<td>(a) Distribution of predicted AS set sizes, (b) accuracy of predicted AS sets encompassing actual AS paths</td>
<td>38</td>
</tr>
<tr>
<td>2.18</td>
<td>Distribution of (c) false negative and (d) false positive rates in predicting the existence of snooping ASes</td>
<td>39</td>
</tr>
<tr>
<td>2.19</td>
<td>Comparison of (a) latencies and (b) normalized geographical distance along paths chosen with WSP ($\alpha = 0$) with and without AS-awareness</td>
<td>40</td>
</tr>
<tr>
<td>2.20</td>
<td>(a) Distribution of bandwidth across Tor relays, and (b) comparison of end-to-end latencies with and without taking relay bandwidth into account; median latency across 5 paths are shown</td>
<td>44</td>
</tr>
<tr>
<td>2.21</td>
<td>Variation across time of latencies on paths spanning similar end-to-end geographical distances</td>
<td>45</td>
</tr>
</tbody>
</table>
3.1 High level design of \textit{Innominate}. (a) Data plane is responsible for transferring traffic through multiple groups to the destination. (b) Clients share their information about their buddies via DHT to build ASes which happens infrequently. Inside of each AS, groups are built frequently. A series of facilitators speed up group building process. (c) All clients are partitioned into ASes and ASes are split into groups.

3.2 A path with three groups, which first and last groups are compromised. The attacker can know the client who forwards the message to middle group and the one that forwards the message in third group are in same group.

3.3 Sharing temporary keys among clients. Every facilitator decrypts one layer from received messages, shuffles them randomly and finally forwards them to the next facilitator. The keys are published on public storage, bulletin.

3.4 First, a new random key $k_{new}$ is obtained, then this keys is used to map the keys to a new order.

3.5 Probability that the originator is exposed to the destination, $R = 1$ in all cases.

3.6 Cumulative Distribution Function (CDF) of ASes vs their sizes (b) CDF of clients vs number of their ASes.

3.7 (a) Cumulative Distribution Function (CDF) of ASes vs their sizes (b) CDF of clients vs number of their ASes.

3.8 Time to run DC-net protocol for various file sizes in groups of different sizes.

3.9 Time to deliver the keys to the facilitators, (b) time to verify a message.

3.10 Downloading home page of “top 200” web sites.

3.11 Performance of \textit{Innominate} for upload and download files to Dropbox.

4.1 DepGraph (a) and its backtracking graph (b), where rectangles are OS processes and ovals are files.

4.2 Our \textit{DeltaTrack} enterprise attack forensics system.

4.3 Examples of sources of dependency explosion in the backtracking graph. (a) \texttt{bash}_{n} depends on all previous bashes. (b) installation of a package depends on all previous package installations. (c) \texttt{cupsd} depends on all applications that use the printer. (d) all processes sharing a pipe depend on each other.

4.4 Inflation in number of (a) nodes and (b) edges of the backtracking graph.

4.5 Removing pipe nodes may lead to wrong results. (a) the original \textit{DepGraph for cat passwd | grep USER}, (b) the \textit{DepGraph} after removing the pipe node.

4.6 Cumulative number of files written to vs. analysis duration. For any fixed period of data, files written to prior to that period can mistakenly appear as read-only.

4.7 (a) Patterns required to create a longer data dependency chain, (b) control dependency edge creates shortcuts in the dependency graph (c) generating a rare event resets the $k$ and revealing all attacks, (d) an example of an attack which happens via a long data dependency chain and independent processes. Red bold arrows trigger attacks, $P_{V}$ and $P_{A}$ are victim and attack process.

4.8 The backtracking graph for an intra-host attack (CVE-2008-5377 \cite{7}), where the attacker escalates privilege by exploiting a vulnerability in CUPS to add a user. The detection point is the unauthorized access to passwd.

4.9 Overview of the inter-host attack, where the attacker compromises a Nginx server, gains a reverse shell, and then compromises a git server.
| 4.10 | The backtracking graph for the inter-host attack in Figure 4.9. The external attacker gains a shell to an internal version control server through an enterprise web server. | 115 |
| 4.11 | False positive rate decreases as more data is used to build the reference model. | 118 |
| 4.12 | Reduction ratio in the number of (a) nodes and (b) edges. A reference model of one week data is used. | 120 |
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Overview of techniques developed to build LASTor.</td>
<td>16</td>
</tr>
<tr>
<td>2.2</td>
<td>Summary of datasets.</td>
<td>18</td>
</tr>
<tr>
<td>3.1</td>
<td>Comparison of different anonymity networks where A, S and P stand for Anytrust, centralized Servers and Peer respectively</td>
<td>54</td>
</tr>
<tr>
<td>3.2</td>
<td>Summary of techniques and their goals in Innominate’s design.</td>
<td>64</td>
</tr>
<tr>
<td>3.3</td>
<td>Parameter definition.</td>
<td>74</td>
</tr>
<tr>
<td>3.4</td>
<td>Social networks used in our analysis for ASes.</td>
<td>80</td>
</tr>
<tr>
<td>4.1</td>
<td>Sources of dependency explosion that we identified together with frequency, whether they can be handled by existing techniques, and why, based on one month of data from 50 hosts (* was based on one week of data from 10 hosts).</td>
<td>91</td>
</tr>
<tr>
<td>4.2</td>
<td>The system calls being monitored (partial list).</td>
<td>109</td>
</tr>
<tr>
<td>4.3</td>
<td>Comparing performance for the intra-host attack</td>
<td>113</td>
</tr>
<tr>
<td>4.4</td>
<td>Comparing performance for the inter-host attack</td>
<td>116</td>
</tr>
<tr>
<td>4.5</td>
<td>Experimental results on six additional attacks.</td>
<td>117</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Due to growing reliance of computer systems (e.g., a quarter of the world’s population has a smartphone [14]), security and privacy of distributed systems have become more important. On one hand, a recent survey [25] shows that a lot of users are still concerned about their anonymity, meaning that they want to protect their privacy against network surveillance, including enterprises. Based on this survey, “55% of internet users have taken steps to avoid observation by specific people, organizations, or the government” and “86% of internet users have taken steps online to remove or mask their digital footprints.” Although a lot of anonymity-preserving communication systems have been deployed over the last few years, they are either vulnerable to traffic analysis or offer poor performance. The work in this dissertation is done over three different systems. The first two systems focus on users privacy over the Internet. LASTor improves latency and anonymity of Tor, while Innominite is a new design that offers traffic analysis resilient communications at scale.

On the other hand, one of the challenging tasks within enterprises is to track down security breaches due to complex dependencies between hosts, applications, and their data [54, 104, 74]. The third system is about backtracking attack graphs to perform root-cause analysis.
after an attack is detected. Backtracking attack graph aims to help enterprise administrator to find all vulnerabilities leading to the attack.

1.1 Anonymous Communications

For decades, online users has been concern about their privacy. The ability to anonymously conduct interactive communications on the Internet is important in a variety of scenarios. Dissidents may seek to anonymously exchange messages with each other, despite government surveillance of their network traffic. Users who want to provide sensitive information as input to journalists may also wish to do so anonymously. Furthermore, in some services (e.g., cloud storage), anonymity can be the only way to ensure user privacy since encrypting data may degrade functions desirable either by the service provider (e.g., deduplication) or by users (e.g., collaborative recommendations).

However, user privacy is undefended against network surveillance in current design of the Internet. Though several solutions have been developed to cater to these various needs for anonymity, all existing solutions suffer from one of two drawbacks. Some anonymity networks such as Tor [35] that are optimized to provide low latency are susceptible to a variety of traffic analysis attacks [119, 65, 98, 45, 95, 101]. For example, a recent study [49] showed that more than 81% of Tor clients can be de-anonymized via traffic analysis. Some other anonymity networks such as Dissent [57, 117] designed to guarantee traffic analysis resilient suffer from poor performance at scale, i.e., they impose overhead that requires high bandwidth or computation power.

In this dissertation, We first present LASTor, a Tor client that improves performance of the Tor network while making it more resilient against traffic analysis attacks. LASTor reduces latency perceived by users via changing the path selection algorithm. It also detects paths with
common autonomous system (AS) across the entry and exit segments of a circuit and it avoids using them at the time of path selection. A common AS is able to collect and evaluate packets in order to identify the source and destination of the communication.

Then, we present next anonymous system, Innominate. Innominate is an anonymity framework that guarantees traffic analysis resilient anonymity while offering good performance even at scale. Innominate relies on relay-based forwarding [106] for low latency communications and it takes advantages of DC-net [53] to offer traffic analysis resilient communications. In Innominate, rather than individual machines serving as relays, every relay comprises a small group of users who shuffle received data among themselves via a DC-net before forwarding. The membership of any group of users who together serve as a relay is not visible to those outside the group. Together, these features of Innominate ensure that despite its use of relay-based forwarding, it is resilient to traffic analysis attack.

1.2 Dependency Tracking for Attack Forensics in Enterprises

Not only protecting an enterprise against attacks is important, but also it is critical that after the attacks is detected, all vulnerabilities get fixed. Traditionally, enterprises protect themselves from attackers by deploying perimeter defenses such as firewalls and intrusion prevention systems. However, given the sophistication of modern attacks such as drive-by downloads, phishing emails, contaminated mobile devices, and insider attacks, successful intrusions and compromises are practically unavoidable. Examples of recent attacks and high-profile data breaches include Home Depot [18], Target [33], Sony Inc. [31], eBay [13], and Anthem [3]. In such cases, it is important for enterprises to conduct intrusion forensics, to gain security intelligence and situational awareness [32], backtrack the attack origin [75], and estimate the impact [76].
In *DeltaTrack*, we focus on the problem of backtracking intrusions, which is to trace back the actions of intruders to identify how they entered the system. This can help system administrators identify and patch the root causes of the intrusion and strengthen the security of the enterprise system. However, the increasing complexity of modern enterprise systems makes backtracking hard. Due to a plethora of dependencies among different components and applications across the enterprise, even simple attacks can lead to thousands of nodes and edges in the backtracking graph. In this graph, the root is the detected intrusion and the edges are other resources upon which the compromised resource is dependent. The large size of the backtracking graph often prohibits human administrators from conducting attack forensics.

*DeltaTrack* is a new differential dependency tracking method to drastically improve the performance of intrusion backtracking by automatically pruning irrelevant resource dependencies while maintaining the precision of the backtracking graph.

### 1.3 Thesis and Contributions

Contributions in this dissertation are divided up into two parts: 1) protecting users privacy over the Internet, and 2) securing enterprise systems by identifying vulnerabilities leading to an attack.

#### 1.3.1 Privacy in Distributed Systems

As of privacy of online users, we designed and deployed two systems, *LASTor* and *Innominate*. 


1.3.2 LASTor

LASTor aims to improve both latency of and anonymity of Tor. In LASTor, we make three contributions. First, we show that significant latency gains are possible by solely accounting for the inferred geographic locations of relays, rather than needing up-to-date latency information of Internet paths (e.g., from network coordinates). We implement the Weighted Shortest Path (WSP) algorithm that probabilistically chooses paths with a preference for shorter paths. However, with a naive implementation of WSP, an adversary can increase the probability of a relay under his control being on the chosen path by simply setting up a large number of relays in the same location, which is close to the direct line between the client and the destination. To preempt this attack, we implement LASTor to execute WSP on a graph of the Tor network where nearby relays are clustered together; this increases the onus on an adversary to establish relays in several locations in order to ensure a high probability for the chosen path traversing a relay under his control. A side-effect of clustering relays is that WSP’s runtime is significantly reduced.

Second, we make LASTor resilient to the attack where an AS can correlate traffic on the entry and exit segments of the chosen path by explicitly avoiding such paths. To do so, we need to equip LASTor with the ability to predict Internet routing between relays and end-hosts; we cannot simply measure routes from every relay since we seek a solution that only requires client-side modifications. The use of existing approaches for predicting Internet routes is however impractical since they either require clients to download gigabytes of data daily [89] or have significantly high runtimes [88], which would override the benefits of selecting a low latency path. Therefore, we instead develop a computationally lightweight technique that has a low false-negative rate in failing to identify paths that permit the possibility of “snooping” ASes. Our key insight here is to predict the set of ASes through which the Internet may route
traffic between a pair of IP addresses, rather than predicting the *precise route* between them. Importantly, in order to run this AS set prediction algorithm, clients need download only 13 MB of data initially and 1.5 MB every week thereafter.

Finally, *LASTor* makes path selection tunable. Probabilistic selection of paths with a preference for shorter paths reduces the entropy of path selection, and all users may not wish to trade-off the resulting reduction in anonymity for reduced latency. Therefore, *LASTor* enables a user to choose an appropriate tradeoff between latency and anonymity. By choosing a value between 0 (lowest latency) and 1 (highest anonymity) for a single parameter, a user can configure *LASTor* to appropriately tailor path selection.

We demonstrate *LASTor*’s benefits in improving latency by using it to visit the top 200 websites from 50 geographically distributed PlanetLab nodes. We see that even without any modification to the rest of Tor, *LASTor* provides a median latency improvement of 25% over the default Tor client. We also use measurements of AS-level routes on over 200K Internet paths to evaluate *LASTor*’s ability to preempt the possibility of snooping ASes jeopardizing the anonymity of clients. We see that for the median (client, destination) pair, *LASTor* fails to identify only 11% of the instances in which a snooping AS can exist; in comparison, we observe a false-negative rate of 57% with the default Tor client.

### 1.3.3 *Innominate*

*Innominate* provides scalable traffic analysis resilient anonymous network. In *Innominate*, we make the following contributions. First, *Innominate* uses a new group based relaying technique to forward the traffic, which has the following advantages: (1) it supports low latency communications in analogy to relay based approaches, (2) it is resilient to traffic analysis similar to strong anonymity approaches, and (3) with hidden group membership the set of clients who might be the source extends to all benign clients. However, designing *Innominate* based
on this approach is hard for several reasons. First, in order to keep group membership hidden from those outside of a group, the formation of groups must be decentralized, yet be resistant to Sybil attacks (i.e., a set of adversarial nodes colluding to form a group with a benign node). Second, the membership of any group should not be revealed during the process of the group forwarding traffic for others or for one of its members. Lastly, despite users routinely going offline, the process of forming and maintaining groups should ensure high availability and impose low overhead.

Our second contribution is separating control and data plane in Innominate. In the design of control plane, Innominate forces all control messages have exactly the same size in order to enable a fast group creation algorithm which guarantees hidden group memberships to those outside the group. In the control plane, Innominate combines some techniques to address the above challenges. First, we apply a distributed version of the dominating set algorithm \cite{71,79,80} on the social graph (that connects users to their friends and friends-of-friends) to scalably enable users to self-partition themselves into anonymity sets, each comprising tens of thousands of users. Second, for use within each anonymity set, we develop an efficient two-round protocol that enables users within a set to partition themselves into small groups of users, such that the membership of any group is not visible to those outside the group.

As of third contribution, we design, build, and deploy Innominate on Emulab \cite{15}. We empirically determine how many users should be each in group and how many groups each user should be part of, so as to maximize anonymity and availability at minimum overhead.

1.3.4 Security in Distributed Systems

In DeltaTrack, we make the following contributions:

- We re-formulate intrusion backtracking as a delta dependency tracking problem, which allows us to leverage the difference between attack and normal system behaviors to reduce
the complexity of the analysis.

- We implement a new intrusion backtracking system which instantiates the generic approach of *delta dependency tracking*.
- We evaluate *DeltaTrack* using real attacks on popular applications and data collected over a year to demonstrate the effectiveness of our proposed techniques.

*DeltaTrack* was deployed on 50 hosts, including both desktops and servers, in an enterprise system during the course of a year. The results show that *DeltaTrack* can significantly decrease the size of the backtracking graph while maintaining its accuracy. Specifically, the number of nodes and edges in the backtracking graph can be reduced by 3–131x and 4–512x, respectively, compared to prior approaches. The results also show that *DeltaTrack* has modest performance overhead: the average CPU overhead of monitoring agent on a host is 1–14.8%, while the storage overhead on the centralized back-end (which gathers all collected data) is 130 MB per host per day.

### 1.4 Organization

The rest of the dissertation is organized as follows. Chapter 2 and 3 describe *LASTor* and *Innominate*, our two anonymous systems. Chapter 4 presents *DeltaTrack* and finally we conclude the dissertation in Chapter 5.
Chapter 2

LASTor: A Low-Latency AS-Aware Tor Client

Tor [59] is a widely used and deployed network for anonymous communication on the Internet. Unlike other systems that facilitate anonymous communication [58, 97], Tor distinguishes itself by enabling low-latency communication. Indeed, a vast majority of users—accounting for over 90% of TCP connections [91] on Tor—use Tor for interactive traffic.

However, several measures for increasing client anonymity in Tor fundamentally inflate communication latencies. For example, the default Tor client sets up a tunnel between itself and a destination via three relays selected at random, with some preference for relay stability and access link bandwidth. This random selection of relays can lead to circuitous routing of tunnels around the globe, resulting in high latencies. Previous solutions for improving performance on Tor have either focused on increasing throughput [110], or those that focused on improving latencies mandate a revamp of the Tor network, e.g., by having all Tor relays participate in a network coordinate system [108, 103] or by modifying traffic management at relays [43]. Due to the undoubtedly significant development effort required to implement these changes, these
solutions are yet to be deployed.

In addition, Tor’s anonymity guarantees breakdown in some cases due to its path selection being oblivious to Internet routing. For example, on some paths, an Autonomous System (AS) may be present on the Internet routes both between the client and the entry relay and between the exit relay and the destination. Such an AS can statistically correlate traffic on the entry and exit segments of the path and potentially infer the destination with which the client communicated. Though this problem has been recognized previously \cite{63, 60} and the default Tor client attempts to preempt such cases by ensuring that no two relays in a path are in the same /16 IP prefix, we find that this heuristic is insufficient for detecting most instances of potential snooping by ASes.

In this chapter, we seek to address both of the above shortcomings with Tor today by making only client-side modifications. This approach ensures that a user can obtain the resultant benefits in latency and anonymity simply by updating her Tor client, without having to wait for changes to the rest of the Tor network. Therefore, we seek to answer the following question: what latency improvements can a Tor client obtain today, without any modifications to the rest of Tor, while also avoiding paths on which an AS could break the client’s anonymity by correlating traffic? Towards this end, we design and implement \textit{LASTor}, a new Tor client that differs from the default Tor client only in its path selection algorithm.

\section{Background and motivation}

In this section, we provide some background on Tor and discuss results that motivate our work.
2.1.1 Tor overview

Tor [34], a low-latency open source application that allows users to use the Internet anonymously, was developed in September of 2002. In Tor, clients download a list of relays and some information about these relays from directory servers. To establish a connection to a destination, a client selects three relays—entry, middle, and exit nodes—and builds a circuit[1] through these three relays. The client appropriately encrypts the data it sends to the entry relay so that each of these three relays only knows the nodes before and after it on the path, i.e., the entry relay knows the source and the middle relay, the middle relay knows only the entry and exit relays, and the exit relay knows only the middle relay and the destination. This form of onion routing [106] preserves the client’s anonymity by ensuring that no one other than the client knows that it communicated with the destination.

To avoid statistical profiling attacks, the default Tor client restricts its choice of entry nodes to a persistent list of three randomly chosen nodes named “entry guards” [118]. For the middle node, the Tor client sorts Tor relays based on their access link bandwidth and randomly selects a relay, with the probability of selection being higher for relays with higher bandwidth. For the selection of the exit node, clients are constrained by the fact that a large fraction of relays choose to not serve as exit nodes. This is because destination servers see the exit node as the computer that communicates with them; if any malicious activity is detected by the destination, it will assume that the exit relay is responsible. Therefore, when selecting an exit node, a client chooses at random (again with bias for higher bandwidth relays) among those relays willing to serve as an exit node for the particular destination that the client is attempting to contact and the particular service with which this communication is associated.

[1] We use the terms path, circuit, and tunnel interchangeably in this thesis.
2.1.2 Motivation

The motivation for our work stems from two sources of inefficiency in path selection as above in Tor today—high latency due to circuitous routing and degradation of anonymity because of path selection being oblivious to Internet routing.

**Poor latency.** First, as discussed above, a client selects entry, middle, and exit nodes in a circuit more or less at random. As a result, the circuit between a client and a destination can often be circuitous, causing significant latency overhead compared to latency on the default Internet path between the client and the destination. Since Tor is predominantly used for interactive communication [91], e.g., to visit websites, this increased latency degrades user experience. Fig. 2.1 presents such an example. A client in the US communicates with a server in Canada. The client incurs significant latency overhead due to relay selection inefficiencies because all packets from the client travel around the world two times before they reach their destination.

To quantify the extent of this latency overhead, we measured the latency of visiting
the top 200 websites [27] from 50 PlanetLab nodes [26] spread across the globe. We measured the latency between every PlanetLab node and every website as the median latency of 5 HTTP HEAD requests. We first measured latencies by having the PlanetLab nodes contact the websites directly. Next, we repeated the same with the communication happening over the default Tor setup. We finally measured latencies via Tor when choosing entry, middle, and exit nodes that result in the shortest end-to-end path based on the geographical locations (inferred using MaxMind’s IP geolocation database [23]) of the client, the destination, and the relays on the path. Fig. 2.2 shows the distribution across (PlanetLab node, website) pairs of the latencies measured in the three cases. First, we see that latencies measured using default Tor are more than 5x greater than via the direct Internet path (no Tor) in the median case. Second, latencies over the shortest path on Tor (SP Tor) result in a 2x reduction in median latency compared to default Tor.

Circuit establishment in Tor however cannot simply be modified to select the shortest path between the client and the destination; this makes path selection deterministic and enables adversaries to strategically setup relays that can subvert the client’s anonymity. Instead, moti-
vated by the latency improvements possible by choosing geographically shorter paths, our goal is to enable probabilistic path selection that can deliver some of these latency benefits without significantly compromising client anonymity.

**Lack of AS-awareness.** Though Tor’s use of onion routing tries to ensure that no one other than the client has knowledge of the destinations with which it communicates, there are a variety of attacks possible (e.g., [61] [69]) from which this information can be inferred. One such attack arises because of Tor’s path selection being oblivious to Internet routing. In the case where the routes through the Internet from the client to the entry node and from the exit node to the destination both traverse a common Autonomous System (AS), such an AS can correlate the traffic it observes to infer the (client, destination) pair [90, 99]. Fig. 2.3 shows an example in which AS2, which appears on both the routes from the source S to the entry relay R1 and from the exit relay R2 to the destination D, can potentially infer that S is communicating with D. We hereafter refer to such ASes that have the potential of correlating traffic by snooping as **snooping ASes**. Note that even though traffic between the client and the entry node is encrypted, ASes can observe the client’s IP address in the headers of the packets that the client sends to the entry node.

Feamster and Dingledine [63] showed that the probability of existence of snooping ASes is 10–30%. This observation was re-evaluated 5 years later by Edman and Syverson [60]. They observed that while there are many more Tor relays than before, this growth has only a
Figure 2.4: False negatives in detecting snooping ASes with default Tor client.

slight effect on mitigating attacks by snooping ASes. This is because Tor relays are not scattered uniformly among ASes, and so the growth of the network does not guarantee path location diversity. Further, the presence of ASes that can snoop is especially likely in cases where the client and destination are in the same location, because the entry and exit segments of the circuit may go through the same ASes with presence in that region.

Therefore, to protect its anonymity, a Tor client needs to ensure that its algorithm for path selection prevents, or at least minimizes, the existence of common ASes across both ends of a circuit. To preempt AS-level attacks and preserve anonymity, Tor’s default path selection algorithm ensures that the entry and exit nodes on any particular circuit do not share the same /16 IP address prefix [39].

We however find that this heuristic performs poorly in practice in avoiding snooping ASes. First, in the deployment of Tor as of June 2011, we observe that 60% of ASes that have Tor relays resident in them have at least two relays that are in different /16 subnets. In addition, we evaluated the /16 prefix heuristic on a dataset of measured AS paths (the PL-BGP-Rand dataset described later in Section 4.2). For every (client, destination) pair in our dataset, we
<table>
<thead>
<tr>
<th>Goal</th>
<th>Technique</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reduce latency of communication on Tor</td>
<td>Weighted Shortest Path (WSP) algorithm for probabilistic selection of paths with preference for low-latency paths</td>
<td>2.3.1</td>
</tr>
<tr>
<td>Defend against strategic establishment of relays to increase probability of compromised relays on chosen path</td>
<td>Clustering of relays in nearby locations</td>
<td>2.3.2</td>
</tr>
<tr>
<td>Enable user to choose trade-off between latency and anonymity</td>
<td>Augment WSP with parameter $\alpha$ that can be varied between 0 (lowest latency) and 1 (highest anonymity)</td>
<td>2.3.4</td>
</tr>
<tr>
<td>Account for distributed destinations</td>
<td>DNS lookup service on PlanetLab nodes</td>
<td>2.3.3</td>
</tr>
<tr>
<td>Preempt traffic correlation attacks by ASes</td>
<td>Lightweight algorithm to determine set of ASes through which Internet may route traffic between a pair of IP addresses</td>
<td>2.4</td>
</tr>
</tbody>
</table>

Table 2.1: Overview of techniques developed to build LASTor.

computed the false negative rate of the /16 heuristic, i.e., of all entry and exit node combinations in which there was a common AS across the entry and exit segments, the fraction that the /16 heuristic deemed as safe from snooping ASes. Fig. 2.4 plots this false negative rate for this heuristic across (client, destination) pairs. The /16 heuristic for avoiding snooping ASes miss over 40% of instances of snooping ASes for more than 80% of (client, destination) pairs. Furthermore, we find that simply accounting for the ASes in which the relays reside (the “Same AS” line in Fig. 2.4) is also insufficient.

To address the shortcomings of these heuristics, Tor clients need to determine the ASes through which the Internet routes traffic between them and entry nodes and between exit nodes and destinations. Since we seek only client-side solutions, modifying relays to measure routes is not an option. Querying a route prediction service (e.g., iPlane [87]) for this information is not an option either since the client and destination will be revealed to the service. On the other hand, having clients download pre-computed AS paths between themselves and all entry guards and between all exit relays and all end-hosts will require clients to download a prohibitively
large dataset. For example, even if we aggregate Tor relays and all end-hosts on the Internet into BGP atoms \(^4\) based on the average AS path length of 4 on the Internet, we estimate that clients will have to download on the order of 500 MB of data. Further, this data will have to be continually updated to account for flux in the Internet’s routing.

Instead, it is imperative that clients download a snapshot of Internet topology and routing information and make route predictions locally. However, enabling such local route predictions with current techniques poses two problems. First, it is impractical to expect clients to download several gigabytes of data, e.g., iPlane’s Internet atlas, to make such predictions. Second, AS path inference techniques that operate on a compact Internet atlas \(^8\), have high computational overhead and take on the order of a second to estimate the AS path between a pair of IP addresses. Since a Tor client has to choose from around 1000 exit relays in setting up a circuit, the use of such computationally-heavy techniques to estimate AS paths can impose high overhead on path selection, rendering the latency benefits of avoiding circuitous routes moot.

2.2 Overview

Next, we define the precise problem statement that we target and provide a brief overview of our work. We also discuss the datasets that we use throughout our work to evaluate the techniques that we develop.

2.2.1 Problem statement

Our goal in this chapter is to address the shortcomings in Tor discussed above with respect to latency and anonymity without requiring a revamp of Tor’s design. Leveraging the fact that intelligence in Tor resides at the client, we seek to only modify the client-side path
selection algorithm so that clients can benefit today without waiting on updates to relays to be developed and deployed. In doing so, we respect conventional wisdom on how to preserve client anonymity in Tor, e.g., the use of three entry guards to protect against statistical profiling attacks and the need for sufficient randomness in relay selection to protect against colluding relays.

Table 2.1 summarizes the techniques that we present in the rest of this chapter to address this problem by developing LASTor.

### 2.2.2 Measurement datasets

To evaluate LASTor’s components, we make use of three datasets (summarized in Table 2.2), with PlanetLab nodes serving as clients in all three cases; we pick 50 PlanetLab nodes to use as clients, in keeping with the distribution across countries of Tor clients [36]. In
our first dataset, *PL-Tor-Web*, we use 200 websites [27] as destinations and the relays in the actual Tor network serve as relays. In this dataset, while we can measure both latencies and AS-level routes from PlanetLab nodes to Tor relays, we do not have access to either information on paths from relays to destinations. Second, we use the *PL-BGP-Rand* dataset, in which BGP routers seen in various BGP feeds [94, 29] serve as relays and the .1 IP address in 500 randomly chosen /24 prefixes serve as destinations. Here again, we can directly measure latencies and AS paths from PlanetLab nodes to BGP routers. In addition, we obtain the AS paths from the BGP routers to the destinations from various BGP feeds, but we do not have latencies along these paths. This dataset enables to evaluate our techniques for AS-awareness in path selection using measured AS-level Internet routes, unlike prior work in this area [63, 60] that has relied on inferred AS-level routes. Though we have to use BGP routers as proxy for Tor relays for this purpose, Fig. 2.5 shows that the distribution of relays across ASes in the *PL-BGP-Rand* dataset is similar to that in the case of real Tor relays.

Finally, in the *PL-PL-Web* dataset, we use PlanetLab nodes as both clients and relays and the top 200 websites as destinations. In this case, we can measure latencies and AS paths both from all clients to all relays and from all relays to all destinations. To emulate typical Tor clients, we ensure throughout our evaluation that we do not provide as input to iPlane [87] any Internet topology measurements from the 50 PlanetLab nodes used as clients; as we describe later in Section 2.4, we use AS path length estimates from iPlane for our AS set prediction.

### 2.3 Path Selection

Path latency on the Internet is a sum of three factors—propagation delay (time spent by packets on the wire), queueing delay (time spent by packets enqueued at end-hosts or intermediate routers, waiting to be put onto the wire), and transmission delay (time to put a packet
onto the wire). Since access link bandwidths of the client and Tor relays is beyond our control, we cannot reduce transmission delay. On the other hand, as we show later in Section 2.6, a modification of Tor relays would be necessary to reduce queueing delays. Therefore, we focus here on reducing propagation delay.

### 2.3.1 Preferential selection of low-latency paths

To reduce propagation delays, we need to reduce the probability of selection of circuitous paths. We cannot however simply pick the shortest possible path through three relays between a client and a destination. This would make path selection deterministic and hence, susceptible to strategically placed adversarial Tor relays. Therefore, we implement a Weighted Shortest Path (WSP) algorithm. WSP orders all possible paths between a client and a destination based on the expected latency on each path. The latency along a path is the sum of latencies on each of the four segments of the path—(client, entry relay), (entry relay, middle relay), (middle relay, exit relay), and (exit relay, destination). The probability of a particular path being selected is then inversely proportional to the expected latency on it.
However, in order to estimate the latency along every possible path of three relays between the client and the destination, we would need latencies between the client and all candidate entry relays, between all candidate exit relays and the destination, and between all pairs of relays. As proposed in previous approaches to improve latency in Tor\cite{108,103}, gathering this latency information would require a modification of Tor relays. Instrumenting measurements of the Internet at such a scale is a non-trivial undertaking. As a result, these prior proposals are yet to translate into practice.

Our focus here instead is on a practical implementation of WSP, with changes only at the Tor client. Therefore, we use the end-to-end geographical distance along a path as a proxy for the latency along it. This ensures that we do not need to modify relays to track latencies between them, but we can rely instead on the estimated geographic locations of clients, relays, and destinations. We compute the end-to-end geographical distance along a path by summing up the distance along each segment, which we in turn compute based on the (latitude, longitude) coordinates of the hosts at either end of a segment. We can estimate the geographic locations of end-hosts and relays using an IP geolocation database, such as MaxMind\cite{23}. Out of all candidate paths, WSP then selects one path with the probability of a path’s selection being proportional to the weight associated with it; the weight associated with a path is the difference between the maximum end-to-end distance across all paths and the distance along this particular path.

Though the use of geographical distance ignores the effect of routing on latency (the Internet may forward packets along a circuitous route\cite{77}), we confirm empirically that our use of geographical distance as the weight for every edge in the graph when running WSP is a reasonable substitute for the latency of every edge. Since we can compute end-to-end latencies of paths only on the PL-PL-Web dataset, we perform this analysis on that data. We perform this analysis first using latency as the edge weight metric for running WSP and then repeat the same
using geographic distance for edge weights. Fig. 2.6 shows that the end-to-end latencies of chosen paths are similar irrespective of whether WSP uses latencies or geographic distances as edge weights. Therefore, we believe that our use of geographic distances delivers most of the benefits of reducing propagation delays without warranting the need for a distributed infrastructure that measures latencies between all pairs of relays, an unarguably arduous undertaking.

### 2.3.2 Clustering of relays

A straightforward implementation of WSP however causes two problems. First, WSP’s preference for paths with lower end-to-end geographical distance results in a greater preference for paths through relays that are close to the direct line between the client and the destination. For example, in Fig. 2.7 WSP will select the path through relay $R_1$ with a higher probability than the path through $R_2$. As a result, if an adversary wishes to ensure that a relay under his control is on the chosen path between $S$ and $D$, then the adversary can choose a location that is close to the direct line between $S$ and $D$ and setup a large number of relays at that location. It is relatively easy for an adversary to setup several relays in the same location, for example,
by renting several virtual machines in a cloud service. The high probability of at least one of the adversary’s relays being on the selected path increases the chances for the adversary to use recent traffic analysis attacks on Tor [96] and infer that $S$ is communicating with $D$.

The second problem with a strawman implementation of WSP is its runtime. Today, Tor has over 2500 relays with roughly 1000 of these relays willing to serve as exit nodes. The number of candidate paths between a client and a destination is therefore in the order of billions. So, a naive computation of the end-to-end geographical distance on every candidate path is computationally expensive and takes roughly 6.5 seconds to run even on a 2.5 GHz processor. This large runtime—in comparison to Internet path latencies that are of the order of tens or hundreds of milliseconds—to even select a path can render the selection of a low latency path redundant.

To address both of these problems, we cluster Tor relays that are located in geographically nearby locations. We employ a simple clustering algorithm in which we divide the globe into a grid of square cells and cluster all relays within a cell; the edge length of the cells is a configurable parameter. We then execute WSP on the clustered Tor network where every node is a cluster of relays, and each candidate path is through three clusters. WSP computes the end-to-end distance on every cluster-level path and then selects one path with preference to shorter paths as before. We translate the chosen cluster-level path to a path through three Tor relays by picking one relay at random from each of the clusters on the selected path.

This modification of WSP reduces its runtime to select a path between a client and a destination through today’s Tor network to 245 milliseconds, in comparison to the runtime of 6.5 seconds with the naive implementation. More importantly, the modified WSP ensures that the establishment of a large number of relays in the same location does not bias the selection of paths through them since WSP considers paths at the granularity of the cluster to which all of them belong; paths through different relays in a cluster are not considered independently. Thus
the modified WSP increases the onus on an adversary to establish relays in multiple locations in order to have one of those relays be on the chosen path with a very high probability.

We conduct the following experiment 1) to choose the cell size to be used in clustering of relays, and 2) to demonstrate the improved resilience of WSP to an adversary as discussed above. In the PL-Tor-Web dataset, for every (client, destination) pair, we emulate an adversary who controls the 5% of relays that are closest to the direct line between the client and the destination. We then model the adversary increasing the number of relays that he controls by replicating these 5% of closest relays by a factor of 25. We run WSP on this modified Tor network with and without clustering of relays. In either case, given a (source, destination) pair, we compute the probability of the path between them selected by WSP traversing at least one compromised relay. This value represents an upper bound on the fraction of cases in which the chosen path will traverse a relay controlled by the adversary, if the adversary controls at most 5% of relays.

![Figure 2.8: Clustering with higher cell sizes provides better resilience.](image-url)

Fig. 2.8 compares the distribution across (source, destination) pairs of this upper bound when clustering relays with different cell sizes. We vary the edge length of every cell.
Figure 2.9: Clustering of relays reduces the probability of an adversary compromising a large fraction of paths.

from 0.25 to 4—measured in terms of the difference in latitude or longitude—and, in each case, we compute the fraction of paths that traverse a relay controlled by the adversary. We see that using a edge length of 2 for each cell significantly decreases the influence of the adversary compared to the effect when using lower edge lengths, and increasing the edge length further has minimal impact.

Next, we evaluate the resilience offered by running WSP after the clustering of relays.

Fig. 2.9 compares the distribution across (source, destination) pairs of the fraction of paths that traverse a compromised relay in the following three cases: 1) when running WSP on the PL-Tor-Web dataset without clustering of relays (No clusters, default), and when an adversary replicates relays in this dataset as above and WSP is executed 2) after clustering relays (using a cell size of 2x2) (With clusters, 25x), or (3) without clustering (No clusters, 25x).

By comparing the “No clusters, default” and “No clusters, 25x” lines, we see that, in the absence of clustering, the adversary can increase the fraction of paths that traverse a compromised relay from around 35% to over 65% on average by replicating the relays that he controls by 25x.

In contrast, when relays are clustered into cells of size 2x2, the adversary gains nothing by
replicating relays.

Clustering of relays however has a negative impact on the latencies along paths chosen by WSP. This is because, in cases where there are several relays in a location close to the direct line between the source and the destination, the basic version of WSP can choose from the several candidate paths through these relays. In contrast, after these relays have been clustered, WSP has only path of choice through these relays. Hence, as shown in Figure 2.10 the geographic distance along the path chosen by WSP increases by roughly 15% in the median case when relays are clustered. This inflation in path length due to relay clustering is a compromise that we have to bear, in exchange for increasing the onus on adversaries to setup relays in several locations to attract traffic through compromised relays with high probability.

Finally, we evaluate the latency improvement obtained with WSP in practice. We modify the default Tor client to implement the WSP path selection algorithm and use the modified client to measure latencies over the Tor network to the top 200 websites from 50 PlanetLab nodes. For each (client, destination) pair, we run WSP 5 times and on each attempt, we measure
the median latency of 5 HTTP HEAD requests. We then compute the median latency across the 5 attempts. We repeat the same process using the default Tor client and compute the median latency across 5 paths chosen by it, considering the median latency across 5 HTTP HEAD requests on each path. Fig. 2.11 presents the latency distribution measured across (client, destination) pairs when using WSP as compared to that when using the default Tor client. We see that WSP results in a 25% reduction in latency in the median case.

2.3.3 Accounting for distributed destinations

Thus far, our exposition of WSP has assumed that the destination has a single location associated with it. In practice, the destinations associated with interactive communication (e.g., webservers) are often replicated across several geographic locations. In such cases, users specify the destination by its hostname, and upon DNS resolution of the hostname, the webservice provider returns the IP address of the server located closest to the end-host that performs the DNS lookup. This implies that when a client uses a Tor circuit to contact a destination, the particular server with which the client ends up communicating depends on DNS resolution of
the destination’s hostname at the exit node on that circuit. Therefore, when WSP estimates the end-to-end distance on any candidate path, it must take into account the location of the particular IP address to which the exit node on that path will be redirected.

However, at the time of path selection, it is impractical to perform DNS lookups for the destination on all candidate exit relays. Doing so would require the client to setup a circuit for every candidate exit relay; the client cannot simply ask a relay to resolve the destination hostname since that would leak the client’s anonymity. Establishing one circuit for every candidate exit relay every time a path needs to be selected would not only impose significant overhead on Tor but also take several tens of seconds, thus nullifying the benefits of selecting a low-latency path.

Instead, we setup a DNS lookup service across a set of 15 geographically distributed PlanetLab nodes. When a client needs to run WSP for a destination, it submits a request to resolve the destination’s hostname to each of the PlanetLab nodes running the DNS lookup service. The client submits these requests via any one of the circuits that it had previously established, e.g., the default Tor client establishes three circuits when it starts up. The client uses

Figure 2.12: Lower latencies obtained with WSP when accounting for distributed destinations.
HTTPS to submit these DNS resolution requests to the PlanetLab nodes so that the exit node on
the circuit used for communicating with the PlanetLab nodes cannot infer the destination. Once
the client receives the set of IP addresses obtained for the destination, we assume any candidate
exit relay would be redirected to the IP address that is geographically closest to it amongst this
set. Thus, when we subsequently run WSP to pick a path to the destination, we compute the
end-to-end distance on each candidate path by using the distance along the exit segment as the
distance between the exit node on that path and the destination’s IP address to which we believe
the exit node will be redirected.

To evaluate the utility of this modification to WSP, we consider the top 1000 websites
from Quantcast and focus on those that return IP addresses in multiple locations when resolved
from all PlanetLab nodes. We then measure latencies over the Tor network to these websites
with 50 PlanetLab nodes as clients. We measure latencies in two cases. In the first case, we
run WSP as described above where it uses IP addresses obtained by resolving the destination on
15 geographically distributed PlanetLab nodes. In the second case, we run WSP assuming the
destination to have a single IP address obtained by DNS resolution at a randomly chosen exit
relay. Fig. 2.12 compares the latencies measured in these two cases. We see that accounting
for the fact that destinations could be potentially distributed reduces path latency in the median
case by 15%.

2.3.4 Latency versus anonymity tradeoff

Though clustering of relays reduces the chances of compromised relays being present
on a large fraction of chosen paths, WSP’s preference for shorter paths naturally reduces the
entropy of path selection. All users may not wish to trade-off this reduction in entropy for lower
latencies. Therefore, we make path selection with WSP tunable with a parameter $\alpha$. A user can
Figure 2.13: Increasing the value of $\alpha$ when using WSP results in (a) higher latencies and (b) greater entropy of path selection.

We incorporate this parameter $\alpha$ into WSP as follows. As previously mentioned, after computing the end-to-end distance on every candidate path, WSP associates a weight with every path that is equal to the difference between the maximum end-to-end distance across all paths and the distance on that path. The probability of WSP choosing a particular path is then proportional to its weight. We now modify this weight $w$ for a path to instead be $w(1-\alpha)$. In the case when $\alpha$ is equal to 0, WSP defaults to the original version we presented above, which
picks paths with a preference for shorter ones. On the other hand, when $\alpha$ is equal to 1, all paths have a weight of 1 and thus, any particular path is chosen at random. For any other value of $\alpha$ between 0 and 1, path selection is appropriately biased towards low latency or higher entropy.

Fig. 2.13 shows the effect that varying $\alpha$ has on both latencies and entropy. Figure 2.13(a) shows latencies measured with $\alpha$ equal to 0, 0.25, 0.5, 0.75, and 1 in the same setting as that used in Section 2.3.2—median latency from 5 HEAD requests each to the top 200 websites from 50 PlanetLab nodes as clients. Lower values of $\alpha$ result in lower latencies.

To capture the corresponding variance in entropy, we use the Gini coefficient metric [67], which has previously been used to measure anonymity of path selection in Tor, e.g., in [110]. Gini coefficient is a measure of skew in a set of values. A value of 0 for the Gini coefficient indicates perfect equality—that all values in the set are equal, whereas a value of 1 indicates perfect inequality. We use this metric to measure, for each (client, destination) pair in the PL-Tor-Web dataset, the skew across candidates paths of the probability of them selected by WSP. Fig. 2.13(b) shows that higher values of $\alpha$ result in lower values for the Gini coefficient, which corresponds to a lower skew across paths in the probability of their selection.

Finally, we use the parameter $\alpha$ to also guide the selection of entry guards. To avoid
statistical profiling attacks, the default Tor client restricts its choice of entry nodes to a persistent list of three randomly chosen nodes selected when the client starts up [118]. All circuits setup by the client thereafter choose entry relays from one of these three entry guards. As one would expect, this constraint on the selection of entry relays, though good for anonymity, hurts the selection of low-latency paths by WSP; the path between a client and a destination may be unavoidably circuitous if all three entry guards chosen happen to be distant from both the client and the destination.

Therefore, in keeping with our goal of making path selection tunable between a preference for low latency or anonymity, we modify the selection of entry guards as follows. After we cluster relays as above, we order all clusters that contain candidate entry relays based on their distance from the client. We then choose three clusters at random from the closest \((g + \alpha \cdot (100 - g))\)% clusters in this ordering, and pick one relay at random from each of these clusters as the three entry guards, where \(g\) is a configurable parameter; in our implementation we use a value of 20 for \(g\). Thus, when \(\alpha\) equals 0—a preference for the lowest latencies—we choose the entry guards at random from the closest 20% of relays to the client. This minimizes the probability of circuitous routes when \(\alpha = 0\), while still providing good anonymity by selecting entry guards from a fairly large subset (20%) of the candidate entry relays. On the other hand, when a user chooses a value of 1 for \(\alpha\) to get the best level of anonymity, selection of entry guards defaults to the current best practice of choosing from all candidate entry relays at random. Fig. 2.14 shows the effect that this modified entry guard selection algorithm has on the end-to-end distance of the chosen path in the PL-Tor-Web dataset. With increasing \(\alpha\), the randomness of entry guard selection increases and results in longer path lengths.

3The default Tor client considers a subset of all Tor relays for selection as entry guards based on their stability.
2.4 AS Awareness

Next, we address the second limitation of interest in the default Tor client—avoiding paths in which an Autonomous System (AS) can correlate traffic across the routes between the client and entry relay and between the exit relay and the destination. Since our goal is to not require any modifications to Tor relays, we cannot avoid such paths by simply having all relays measure routes from them to the client and to the destination. Therefore, we next discuss how a client can locally make estimations of routing in the Internet in order to identify and ignore paths that present the possibility of snooping ASes.

2.4.1 AS set estimation

Precise inference of AS-level routes between arbitrary IP addresses is hard, as seen in the fact that no existing technique for doing so [83, 89, 105, 87, 88] is close to perfect. Therefore, when evaluating whether a particular combination of entry and exit relays offers the possibility of a snooping AS, we preclude the approach of estimating the AS-level route on the entry and exit segments of the circuit. Instead, we take the approach of predicting for either segment, a set of candidate ASes through which the Internet is highly likely to route traffic on the segment. We can then determine the potential existence of snooping ASes by checking if the intersection between the AS sets for the paths between the client and the entry relay and between the exit relay and the destination is non-empty.

To enable such inference of AS sets by Tor clients, we require clients to download three inputs. First, we use the Internet’s AS-level topology represented as a set of inter-AS links. Second, we need an estimate of the AS path length between every Tor relay and every end-host on the Internet. We need this information as input because the AS path selected by BGP is often longer than the shortest path in the AS topology [88]. As we show later, AS path
lengths can be stored much more compactly and are significantly more stable compared to AS paths. Third, we store AS three-tuples as described below to represent routing policies being employed by ASes.

Given this AS-level topology and an estimate \( L \) for the AS path length between a source \( S \) and destination \( D \), we put together the set of ASes through which traffic may be routed from \( S \) to \( D \) as comprising any AS that is on any policy-compliant route of \( L \) AS hops between \( S \) and \( D \) in the topology. Here, we stress on policy-compliance because every path in the AS-level topology does not conform to routing policies of ASes. Therefore, to ensure that we only consider the ASes on policy-compliant paths, we borrow the technique of using AS three-tuples from iPlane Nano [88]. From a collection of AS path measurements—obtained from BGP feeds [94, 29] and by mapping traceroute measurements [4, 87] to AS paths—we identify every sequence of three consecutive ASes seen on any AS path and add them to a set of AS three-tuples. For example, if we observe an AS path \( AS1 \rightarrow AS2 \rightarrow AS3 \rightarrow AS4 \rightarrow AS5 \), then we add \((AS1, AS2, AS3), (AS2, AS3, AS4), \) and \((AS3, AS4, AS5)\) to our set of AS three-tuples. Any such AS three-tuple \((A, B, C)\) represents routing policy by showing that \( B \) is willing to transit traffic from \( A \) on to \( C \) (in other words, \( B \) passes along route announcements received from \( C \) on to \( A \)). We generated such a set of AS three-tuples by aggregating various BGP feeds, and we are able to represent this data in about 1 MB. Note that though Internet routing can be asymmetric in practice, i.e., the route from \( S \) to \( D \) can differ from the route from \( D \) to \( S \), we assume routing asymmetry here and add the three-tuple \((C, B, A)\) to our set of three-tuples for every tuple \((A, B, C)\) discovered from the AS path measurements.

Given an estimate \( L \) for the AS path length between a pair of IP addresses \( S \) and \( D \), we estimate the set of ASes that are likely to occur on the the route between them using the following two phase algorithm. In the first phase, we run Dijkstra’s shortest path algorithm...
to compute the length of the shortest path from every AS to \( D \)'s AS. We modify the standard Dijkstra’s algorithm to ensure that shortest path lengths are computed only across those paths that satisfy the criterion that any three consecutive ASes on a path are in the set of AS three-tuples. Next, we determine for every AS in the topology, the set of path lengths to \( D \) available via any of the AS’s neighbors.

In the second phase, we determine the output set of ASes by performing a modified breadth-first search (BFS) from \( S \). While performing BFS, we traverse a neighbor \( B \) of an AS \( A \) that is \( k \) hops away from \( S \) only if \( B \) has a path of length \((L - k - 1)\) available to \( D \) via one of its neighbors. In addition, we enforce the valley-free nature of Internet routes [66] by ensuring that once the BFS goes from a node \( A \) to a neighbor \( B \) that has a shorter shortest path to \( D \) than from \( A \), thereafter, we never traverse a node’s neighbor that has a longer shortest path to \( D \) than from that node. Furthermore, we again ensure that the input AS three-tuples are respected; we traverse a neighbor \( B \) of \( A \), whose parent in the BFS is \( C \), only if \((C, B, A)\) is in the input set of AS three-tuples. Algorithm 1—which takes as input the AS graph \( G \), the set of AS three-tuples \( T \), the source \( S \), the destination \( D \), and the estimated AS path length between them—summarizes the pseudocode of this algorithm.

2.4.2 Avoiding snooping ASes

When selecting a path from itself to a destination, a client needs to use the above procedure to determine AS sets for paths between itself and its 3 entry guards and between all exit relays and the destination. For the latter set of paths, we do not compute the AS sets independently. Instead, we run the first phase of our AS set estimation algorithm once, and thereafter run the BFS in phase two of the algorithm from each exit relay independently. We can then ignore from consideration all paths that potentially have snooping ASes on them by ignoring those combinations of entry and exit relays for which the intersection between the AS
sets for the (client, entry relay) and (exit relay, destination) paths is non-empty. This algorithm can prune out paths with snooping ASes in around 3 seconds, even when choosing from 1000 exit relays.

Other than being efficient in terms of computation, our approach also minimizes the data to be downloaded by a client to make local inference of AS sets. First, the set of inter-AS links and the set of AS three-tuples are each roughly about 1 MB in size and changes to these datasets are rare. Second, all Tor relays and all end-hosts on the Internet can be grouped into roughly 600 and 50K BGP atoms [48, 87], respectively. Therefore, we need every client to download AS path lengths for 30M paths—between every (relay, end host) pair.

We evaluate the expected size to store these AS path lengths and the stability of this data using traceroutes gathered daily by iPlane [19] from all PlanetLab nodes to all IP address prefixes at the edge of the Internet. We analyze this data for the period of three weeks in July 2011. On each day, we map all traceroutes to their corresponding AS-level routes and compute the AS path length, i.e., the number of ASes seen on the route. First, we find that less than 0.05% of paths traverse more than 8 AS hops. So, every AS path length can be stored in 3 bits,
making the size of the AS path length data to be downloaded initially by a client to be around 11 MB.

For each week in the considered period, we then compare AS path lengths on every day with those measured on the first day in that week. We perform the comparison by computing the fraction of paths that have a different AS path length on day $i$ compared to that on day 0. As shown in Figure 2.15, AS path lengths changed on a little over 5% of paths even after a week. Therefore, in summary, our design requires clients to initially download 13 MB of data across inter-AS links, AS three-tuples, and AS path lengths—a close to 40x reduction in size compared to pre-computed AS paths between all Tor relays and all end-hosts—and a client need only fetch less than 1.5 MB weekly thereafter to keep the data up-to-date.

2.4.3 Evaluation of AS-awareness

Next, we evaluate our technique for AS set estimation in two parts. First, we examine if the estimated AS sets accurately cover actual AS paths. For this, we estimate AS sets for the paths from PlanetLab nodes to Tor relays in the *PL-Tor-Web* dataset. Fig. 2.17(a) and 2.17(b)
show that the estimated AS sets are typically compact—90\textsuperscript{th} percentile size less than 10 ASes—and at most one AS on the actual AS path is not in the estimated set for over 75\% of paths.

Second, we use the PL-BGP-Rand dataset to study the accuracy with which AS sets enable prediction of potential snooping ASes; we do not have AS paths from exit nodes to destinations in the PL-Tor-Web dataset, and the PL-PL-Web dataset is biased for this analysis.  

For every (client, destination) pair in the PL-BGP-Rand dataset, we partition all entry and exit relay combinations into those that have a common AS across the entry and exit segments and those that do not. We compute the false negative rate in predicting the presence of snooping ASes as the fraction of entries in the former partition not caught by our approach of computing intersections between estimated AS sets. Fig. 2.18(c) shows that our median false negative rate is 11\%. This compares to median false negative rates of 28–57\% with alternate approaches—using iPlane’s predicted AS paths, using the approach proposed in [60] (the “E&S” line), or when only accounting for ASes of end-hosts and relays (the “Same AS” line). On the flip side, in Fig. 2.18(d), we see that AS sets produce a much greater false positive rate—fraction of paths

\footnote{Paths between PlanetLab nodes typically traverse a different set of ASes, e.g., research and educational ASes, compared to paths from PlanetLab nodes to random destinations on the Internet}
that do not have a snooping AS but are declared as having one by our technique—compared to other approaches. However, as we see in Fig. 2.16, the fraction of paths with potential snooping ASes is low for most (src, dst) pairs. So, pruning out about 45% of candidate paths in the median case still leaves a sizeable set of paths from which WSP can choose.

### 2.4.4 Impact of AS-awareness on path latency

Finally, we evaluate the impact that the incorporation of AS-awareness has on path latencies obtained with WSP. WSP has to now select from a subset of all possible candidate paths, because it has to ignore those detected by our AS set estimation algorithm as potentially traversing an AS capable of inferring the (client, destination) pair by traffic correlation. Though the subset of candidate paths with snooping ASes is typically small in practice, the high false positive rate of our detection procedure significantly reduces the subset of paths considered. Therefore, we again use WSP (with $\alpha$ set to 0) to measure latencies over the Tor network from 50 PlanetLab nodes to the top 200 websites. Fig. 2.19(a) compares these latencies with those obtained when using WSP without AS-awareness and when using the default Tor client. We
Figure 2.19: Comparison of (a) latencies and (b) normalized geographical distance along paths chosen with WSP ($\alpha = 0$) with and without AS-awareness.

see that the pruning of paths to avoid snooping ASes results in a slight increase in latency. Fig. 2.19(b) shows that this increase in latency is due to an increase in the length of the chosen path when using WSP informed by AS sets. In future work, we plan to pursue a reduction in false positives to further improve latencies when using WSP with AS-awareness.
2.5 Implementation

We implement all of the algorithms developed thus far—to improve path latency, to make path selection tunable, and to incorporate AS-awareness into path selection—in the LAS-Tor Tor client. In this section, we summarize LASTor’s path selection algorithm and provide an overview of our implementation.

2.5.1 Client in action

In the default Tor client, the client sets up a few circuits on startup and thereafter, when the user chooses to communicate with a particular destination via Tor, the client routes the user’s traffic over one of the established circuits. LAS-Tor mimics the default Tor client in this respect. In addition, once LASTor learns the destination that the user wishes to communicate with, it quickly selects a path using AS-aware WSP, sets up a new circuit along the chosen path, and then transitions the user’s traffic to the destination to this new circuit. Thus, the latency obtained with LASTor matches that of the default Tor client in the case when the user’s communication with the destination is short. In the case when the user’s interaction with the destination is prolonged, e.g., when the user visits several web pages on a website, LASTor significantly improves latencies for most of the user’s interaction, i.e., once LASTor switches the user’s traffic to the circuit chosen with WSP.

To select a path to the specified destination, LASTor executes the tunable AS-aware WSP algorithm with the following sequence of steps.

- Upon initialization, the LASTor client clusters all available relays, and using the value for $\alpha$ specified in its input configuration, it chooses three entry guards at random from the $(20 + \alpha \cdot 80)$% closest relay clusters to the client.
- When required to select a path to a destination, LASTor resolves the destination’s hostname
on a distributed set of nodes that service requests to perform DNS lookups. These requests are submitted via one of the circuits established upon initialization of the client.

- LASTor estimates the AS sets for the paths from the client to the entry guards and from all exit relays to the destination, mapping every candidate exit relay to the closest among the IP addresses obtained for the destination.

- LASTor then computes the end-to-end distance on every candidate path through three clusters that satisfy the check of the AS sets for the entry and exit segments being disjoint. One cluster-level path is then selected with the probability of a path being chosen dependent on the end-to-end distance on it and the input value of $\alpha$.

- The circuit to the destination is then established via one relay selected at random from each of the clusters on the chosen cluster-level path.

### 2.5.2 Modification of default Tor client

We implement LASTor by building upon the default Tor client. We have implemented a Java application which connects to the default Tor client on its control port. This control port is a port on the Tor client which can be used to manage and monitor the Tor client based on a standard protocol [38]. By issuing commands to the control port, our Java application can either obtain information such as the description of all available relays, or manage the Tor client by establishing or closing a circuit, attaching streams to a circuit, and clearing Tor’s DNS cache. To setup a circuit, our program first fetches relevant information through the Tor control port and provides this as input to our tunable path selection algorithm. It then issues commands to the Tor client, again via the control port, to build desired circuits. We implement LASTor to take as part of its input configuration 1) a value of $\alpha$ to guide path selection, and 2) a file with a list of nodes that provide the DNS lookup service.
To run the tunable AS-aware WSP path selection algorithm, our Java program needs several datasets as input. First, it fetches a IP geolocation database that maps IP addresses to locations from MaxMind [23]. Second, the first time it is executed, the program downloads 1) an AS-level representation of the Internet topology, 2) the set of AS three-tuples used to determine policy-compliant paths, and 3) a snapshot of AS path lengths for paths in either direction between all Tor relays and all end-hosts, grouped at the granularity of BGP atoms. We put together the first two datasets by aggregating AS paths from various sources [94, 29, 87, 4]. To estimate AS path lengths, we issue queries to iPlane [20]. We find that iPlane can process roughly 1000 queries per second, and so, we can re-query iPlane every day for all 60 million IP pairs (600 BGP atoms with Tor relays \( \times \) 50K BGP atoms comprising all end-hosts, in either direction) for which we need AS path length information. As mentioned before, all three datasets can be stored in less than 13 MB in size. Since these datasets are the same across all clients and the information of a client having downloaded this data does not hamper its anonymity, clients can download this data from each other via a peer-to-peer file distribution system such as BitTorrent, so as to not overwhelm the bandwidth requirements of any central server. Bandwidth-constrained clients can however download relevant subsets of this data from the central server, e.g., only AS path length information necessary for communication with popular websites. Lastly, every week, the client downloads a roughly 1.5 MB update for AS path length information, and more infrequently, fetches updates for the set of inter-AS links and AS three-tuples. These updates are fetched from a central server since the update depends on the version of the data already on the client. For all datasets required by LASTor, we can enable clients to verify integrity of the data they download using an approach similar to that used to guarantee integrity of the default Tor client—by posting a cryptographic hash of the dataset on the Tor website.
2.6 Discussion

In this section, we discuss the extensions to Tor necessary to further reduce latencies and the impact on load balancing if LASTor is widely adopted.

2.6.1 Accounting for dynamic load

Though we showed that WSP can significantly reduce latencies for communication on Tor, there remains a significant overhead compared to communication over the default Internet path. Therefore, to reduce latencies further, other than reducing propagation delays with the use of the WSP path selection algorithm, it is necessary to minimize queueing delays by taking into account the load at each relay at the time of path selection. Here, we present some preliminary results from our efforts to do so.

First, we observe that access link bandwidths of Tor relays are spread over a wide range, as shown in Figure 2.20(a). Therefore, we investigate the potential for reducing queueing delays by restricting the choice of relays among those with high bandwidth. To study this, we measure path latencies on the Tor network when visiting the top 200 websites from 50 PlanetLab
Figure 2.21: Variation across time of latencies on paths spanning similar end-to-end geographical distances.

nodes in two cases. We measure latencies first when choosing relays at random from those which have bandwidth greater than 100 KBps, and then repeat the same choosing from all Tor relays. To keep propagation delay similar in both settings, for every path that we pick from relays with bandwidth greater than 100 KBps, we pick a corresponding path with the entry, middle, and exit nodes in the same locations, but with no restriction on relay bandwidth. For either path selection strategy, we measure latencies between every (client, destination) pair on five different paths. The lines “All Relays” and “BW $\geq 100$ KBps” in Fig. 2.20(b) show that the distribution of median latency (across the 5 chosen paths) is identical whether we account for relay bandwidth or not. In this case, we use the “Estimated” bandwidth estimate for each relay—the value used by the default Tor client to perform path selection—but we found the results to be similar when using other estimates of relay access link bandwidth provided by the Tor directory.

Next, we studied the variation in latencies over time on a given path. We selected 20 (client, destination) pairs at random, and for each of them, we considered two different disjoint paths with the same end-to-end geographical distance; either path traversed three Tor relays. For each (client, destination) pair, we measured latencies once every half hour on either path
selected for it and noted the relative difference between latencies measured on the first and second path; we randomly order the two paths chosen for every (client, destination) pair and fix that ordering across all measurement rounds. Fig. 2.21 shows the variation of this difference in measured latencies across the period of a day. We see that, though the pair of paths selected for every (client, destination) pair span identical geographical distances, the path that provides better latencies significantly varies over time.

Therefore, these results seem to indicate that we can reduce queueing delays only by modifying relays—either by having them track and report load at finer granularities of time or by introducing a new queue management algorithm at relays—which is outside the scope of our goal of enabling immediate latency improvements for Tor clients. Given the current implementation of Tor relays, biasing relay selection based on their bandwidth may help improve throughput, but this will not improve latencies for interactive transfers.

2.6.2 Load balancing

When choosing a path, the default Tor client currently selects relays with a probability proportional to their access link bandwidth. As a result, the fraction of all of Tor’s traffic that traverses any particular relay is roughly proportional to that relay’s access link bandwidth, thus balancing the load across relays.

In contrast, load across Tor relays could be significantly skewed if LASTor were widely used. If most users choose to use LASTor with a value close to 0 for $\alpha$, paths chosen by each client will be biased towards traversing relays that result in lower end-to-end distances to the destinations with which the client communicates. On the other hand, even if all users use LASTor with a value of 1 for $\alpha$, the consequent selection of relays at random will result in an equal distribution of load across relays, which is undesirable given the significant skew in access link bandwidths across relays (seen earlier in Fig. 2.20a).
Though addressing this issue requires further investigation outside the scope of this work, we present two recommendations that we speculate would enable widespread use of LAS-Tor without harming the balance of load across Tor relays. First, we recommend that Tor users who use the network for bulk transfers, such as BitTorrent, should continue to use the default Tor client. Since bulk transfers account for a majority of the traffic on Tor [91], the use of the default Tor client for such traffic will ensure a distribution of load across relays that is reasonably close to the distribution of their access link bandwidths. The loss of anonymity due to protocol-specific path selection requires further investigation. Second, LASTor’s path selection algorithm itself will need to be modified to take the access link bandwidths of relays into account. However, to do so, we will need to discover the distribution of the value of $\alpha$ used by Tor users who use the LASTor client. Discovering this distribution should be possible by means of an anonymous survey across users. LASTor’s path selection algorithm can then be tweaked to not simply have a preference for paths with a lower end-to-end distance but to also account for the access link bandwidths of relays and the distribution of $\alpha$ across users.

2.7 Related Work

We build upon three lines of prior work—1) improving performance in Tor, 2) improving anonymity with Tor, and 3) AS path inference. We discuss related efforts in these areas.

**Improving performance in Tor.** To improve performance on Tor, Sherr et al. [108, 109] proposed a path selection algorithm based on the concept of link-based relay selection. In this approach, a client computes a cost for each path by aggregating values for the chosen metric (e.g., latency, bandwidth) across segments on the path, and then picks a path with probability based on this cost. With the aid of simulations, they showed that their approach offers better performance on each of the objective functions mentioned above. In order to obtain these
performance benefits, they discuss relays disseminating information among themselves using, for example, a network coordinate system. However, modifying relays to build such a distributed system for performing measurements and then disseminating this information is not a trivial task. Therefore, we focus on latency benefits possible without any modification to relays. Furthermore, to evaluate the anonymity of their approach, Sherr et al. count the number of traversed ASes on the path and consider the traversal of a lower number of ASes to provide better anonymity. Instead, we explicitly detect common ASes on the entry and exit segments of a path and avoid such paths.

Panchenko et al. [103] propose two algorithms to improve the performance on Tor. First, to reduce latency, they measure the latency between every pair of relays and choose a path with a probability related to the end-to-end latency on that path. Second, to help throughput-oriented applications, they perform passive measurements to infer the available bandwidth on each relay and pick a path based on the expected end-to-end throughput. However, again, modifications to all Tor relays are necessary to implement these approaches. Also, since most connections on Tor correspond to interactive traffic [91], we focus only on reducing latency and show how to do so with only client-side modifications.

The authors of [102] studied the influence of geographical diversity on the performance of Tor and found a tradeoff between improved performance and anonymity. They found that though low diversity of relays may lower the latencies in setting up circuits, greater geographical diversity of nodes is an important factor to provide strong anonymity guarantees. We similarly illustrate the loss in anonymity when preferring low latency paths, but make path selection tunable to enable latency benefits to be overridden for better anonymity, when desired.

Snader and Borisov [110] showed how a client can trade off between performance and anonymity when selecting paths. However, Snader and Borisov focused on improving throughput on Tor (their evaluation revolved around the download of a 1 MB file), while we focus
on latency. We showed that the selection of lower latency paths warrants the need for several techniques not necessary when optimizing throughput, such as the careful selection of entry guards and accounting for destinations that are geographically distributed. DefenestraTor \cite{43} improves latencies in Tor by modifying traffic management in Tor relays to reduce congestion-related queueing delays. We pursue a complementary approach that reduces propagation delays without any modifications necessary to Tor relays.

**AS-awareness in path selection.** In 2004, Feamster and Dingledine \cite{63} studied the Tor network to investigate the problem of an AS eavesdropping both ends of a circuit. First, they showed that there are Tor relays with different IP addresses that are in the same AS, and that Tor clients should avoid selecting two relays from the same AS. Second, they discovered that the probability of an AS observing both ends of a circuit varies between 10% and 30% across (client, destination) pairs. To reduce this probability, they proposed the passive monitoring of BGP feeds to determine AS paths. However, they did not elaborate on how clients should fetch and maintain up-to-date information from BGP routing tables. Instead, motivated by their observation, we make AS-aware path selection practical by reducing both time and space complexity.

Later, in 2009, Edman and Syverson \cite{60} showed that although the number of Tor relays increased significantly since Feamster and Dingledine’s analysis, the probability of an AS being able to observe both ends of a connection did not decrease much. To protect against occurrences of snooping ASes, the authors suggest that all Tor server authorities agree upon a snapshot of ASes based on Routing Information Bases (RIB). Client can then use AS topology snapshots to select a path in which AS-level routes from the client to the entry node and from the exit node to the destination span a disjoint set of ASes. As we showed in our evaluation, our approach of using AS sets significantly reduces the rate of missing snooping ASes compared to that proposed by Edman and Syverson.

**AS path inference.** Several systems and algorithms have been developed for infer-
ence of AS paths between arbitrary IP addresses on the Internet. Approaches for this can be broadly classified into two classes. One set of approaches [87, 83, 105] enable computationally efficient estimation of AS paths but use a large corpus of path measurements as input. Such approaches are ideal for hosting services that can be queried for AS path inferences, but this is not an option in the case of Tor since the queries for AS paths can leak client anonymity. The second set of approaches [89, 88] for AS path inference require much lesser data as input, e.g., only the Internet’s PoP-level or AS-level topology, but are computationally prohibitive in processing queries. The use of such techniques to select paths that avoid snooping ASes will render the selection of low latency paths moot. Given these shortcomings of prior approaches for AS path inference, we develop a new technique that both has low runtimes and requires compact inputs.

Other related work. Several measurement studies [82, 86, 91] of the Tor network have been performed to determine the location diversity of Tor users and the popularity of different kinds of traffic such as HTTP, BitTorrent, and E-mail. These studies have shown that though HTTP transfers account for a small fraction of the traffic on Tor, they constitute a large majority of connections. Hence, for most Tor users, latency is more important than throughput. To the best of our knowledge, we are the first to show how to improve latencies on Tor in a practical manner with only client-side modifications.

Hopper et al. [69] studied the loss in a client’s anonymity by knowing the latency on the circuit in use by the client. While complementary to our effort, this study needs to be revisited in the light of our tunable AS-aware WSP path selection algorithm. We speculate that the knowledge that a client is using WSP to choose paths probably leaks more information about the client when path latency is known.
Algorithm 1 Pseudocode of AS set estimation algorithm.

1: Inputs: AS graph $G$, AS three-tuples set $T$, source $S$, destination $D$, AS path length $L$

2: Shortest_Path($G, T, D$)

3: Queue $Q$

4: List Node $PossibleSet$

5: List Node $AS_set$

6: $S.hops = 0$

7: Add $S$ to $Q$

8: while $Q$ is not empty do

9: \hspace{1em} $cur \leftarrow Q.pop$

10: \hspace{1em} $cur.added \leftarrow 0$

11: \hspace{1em} Add $cur$ to $PossibleSet$ if $cur \notin PossibleSet$

12: \hspace{2em} for $n \in cur.neighbors$ do

13: \hspace{3em} Skip $n$ if $(cur.parent, cur, n) \notin T$

14: \hspace{3em} Skip $n$ if $\exists m \in n.neighbors$ such that $m.pathLength + cur.hops + 2 = L$

15: \hspace{3em} if $n$ has ancestor $p$ with $p.pathLength < p.parent.pathLength$ then

16: \hspace{4em} Skip $n$ if $n.pathLength > cur.pathLength$

17: \hspace{3em} end if

18: \hspace{3em} $n.hops = cur.hops + 1$

19: \hspace{3em} Add $n$ to $Q$

20: \hspace{3em} $cur.added += 1$

21: \hspace{2em} end for

22: \hspace{1em} if $cur.added = 0$ then

23: \hspace{2em} Decrement $n.added$ for every ancestor $n$ of $cur$

24: \hspace{1em} end if

25: \hspace{1em} end while

26: for $n \in PossibleSet$ do

27: \hspace{1em} Add $n$ to $AS_set$ if $n.added > 0$

28: \hspace{1em} end for

29: return $AS_set$
Chapter 3

Innominate: Strongly Anonymous, Yet Scalably Performant, Online Communications

3.1 Introduction

The ability to anonymously conduct interactive communications on the Internet is important in a variety of scenarios. Dissidents may seek to anonymously exchange messages with each other, despite government surveillance of their network traffic. Users who want to provide sensitive information as input to journalists may also wish to do so anonymously. Furthermore, in some services (e.g., cloud storage), anonymity can be the only way to ensure user privacy since encrypting data may degrade functions desirable either by the service provider (e.g., deduplication) or by users (e.g., collaborative recommendations).

Though several solutions have been developed to cater to these various needs for anonymity, all existing solutions suffer from one of two drawbacks. On the one hand, systems
(e.g., Tor [35], Crowds [107]) that are optimized to provide low latency often use the approach of forwarding traffic from a source to a destination via several intermediate relays. Several studies have shown that the anonymity offered by all such relay-based forwarding approaches are susceptible to a variety of traffic analysis attacks [119, 65, 98, 45, 95, 101]. For example, a recent study [49] showed that more than 81% of Tor clients can be de-anonymized via traffic analysis. On the other hand, systems (e.g., Dissent [57, 117]) designed to offer strong anonymity suffer from poor performance at scale. The communication patterns in these systems (e.g., the use of DC-nets in Dissent), that are designed to ensure resilience to any traffic analysis, impose overhead that grows linearly with the size of the anonymity set.

In this chapter, we fill the gap in existing anonymity solutions by developing Innominate, an anonymity framework that guarantees traffic analysis resilient strong anonymity while offering good performance even at scale. Innominate relies on relay-based forwarding for low latency communication, but differs from prior systems that use this approach in two significant ways. First, rather than individual machines serving as relays, in Innominate, every relay comprises a small group of users, who shuffle received data among themselves via a DC-net before forwarding. Second, the membership of any group of users who together serve as a relay is not visible to those outside the group. Together, these features of Innominate ensure that, despite its use of relay-based forwarding, it is resilient to traffic analysis attacks, unlike prior systems that use this approach.

However, designing Innominate based on the approach described above is hard for several reasons. First, in order to keep group membership hidden from those outside of a group, the formation of groups must be decentralized, yet be resistant to Sybil attacks (i.e., a set of adversarial nodes colluding to form a group with a benign node). Second, the membership of any group should not be revealed during the process of the group forwarding traffic for others or for one of its members. Lastly, despite users routinely going offline, the process of forming
Table 3.1: Comparison of different anonymity networks where A, S and P stand for Anytrust, centralized Servers and Peer respectively.

and maintaining groups should ensure high availability and impose low overhead.

We address these challenges in our design of Innominate by judiciously combining the use of several techniques. First, we apply a distributed version of the dominating set algorithm \cite{71,79,80} on the social graph (that connects users to their friends and friends-of-friends) to scalably enable users to self-partition themselves into anonymity sets, each comprising tens of thousands of users. Second, for use within each anonymity set, we develop an efficient two-round protocol that enables users within a set to partition themselves into small groups of users, such that the membership of any group is not visible to those outside the group. Third, we empirically determine how many users should be each in group and how many groups each user should be part of, so as to maximize anonymity and availability at minimum overhead. In conjunction, these techniques ensure that when a user in Innominate communicates with an external entity (another user or a service), the user is indistinguishable from others within its anonymity set, yet enjoys good performance.
3.2 Background and Related Work

In design of *Innominate*, we target users of large scale online services. None of the current anonymity networks can provide strong anonymity for millions of users in scalable manner with good performance. In this section, we first discuss about two examples of these online services and then we describe the existing solutions and their limitations for those services.

Web browsing. A lot of users like blogger, whistleblower and journalist use anonymity network for web browsing e.g., 90% of Tor traffic is interactive \[91\]. For a blogger who is posting a message online from a danger zone, it is very important to have strong anonymity although the operation might take place a few minutes.

Online storage. Users who store their files in plain format on online storage providers such as Dropbox have no anonymity. To protect users’ anonymity, one way is to encrypt data before uploading it. Encrypting files by different keys generates random data and prevents the server to deduplicate the data. Data deduplication means eliminating duplicate copies of data across all users. In this workload, on one hand, users needs to anonymously upload/download their files of different sizes and on the other hand, storage provider wants to perform deduplication.

Next, we describe current anonymity solutions which generally falls into two categories: 1) relay-based approach and 2) strong anonymity approach. Table \[\text{3.1}\] compares these solutions.

**Relay-based approach: good performance, no strong anonymity.** In this approach, a client sends its traffic to the destination via several intermediate relays. Relay-based approach is scalable but it provides little protection against an adversary who eavesdrops network traffic i.e., it is vulnerable to traffic analysis attack \([65, 98, 45, 95]\). For example, in Tor \([35]\) which is one of the popular relay-based networks, traffic analysis attacks usually is performed by two entities:
First entity is an Autonomous System (AS) who can monitor traffic between entry path to Tor network and exit path from it. In worst case that both source and destination are in the same AS, there is no way for user to be protected against the AS. Second entity is a adversarial destination. The destination runs a lot of compromised relays until one of them is chosen as entry relay by the source \(i.e.,\) an adversarial destination can identify the source by correlating received traffic to what it is observed in compromised entry relay.

Aqua [81] is another relay-based network in which all data is padded or split in such a way that the traffic is indistinguishable for a group of clients. In the edge network, padding is used while the traffic is split in core network via multi-path routing techniques to reduce padding size. However, this approach is vulnerable to compromised relays running by adversarial destination. In addition to, groups are small, therefore the set of clients who might be the source of the communication is small. Herd [47] is a modified version of Aqua [81]. In Herd, all clients must be online all the time and they send data with constant rate. These assumptions limit the application of Herd to VoIP. In both Aqua [81] and Herd [47], a client has to trust one specific peer and the peer acts on the behalf of the client. This in not a very reasonable assumption as it is unclear why a client can trust a peer but not the destination.

**Strong anonymity approach: poor performance** In addition to the anonymity provided by relay-based solutions, strong anonymity methods aim to prevent traffic analysis attacks. In these attacks, an attacker collects and evaluates statistical information about the time series of packets and client activities to identify the source and the destination. Strong anonymity protocols usually uses Private Information Retrieval (PIR) or Dining Cryptographers network (DC-net) techniques. PIR protocols allow a client to retrieve an item from a destination in possession of a database without revealing which item is retrieved. For example, one trivial but inefficient way is to download the entire database by the client. PIR techniques require high bandwidth and/or computation and they usually provides data anonymity meaning that destination knows
the client but it does not realize about what records are read or written. Riposte [56] is a PIR based anonymity network but it is not a practical solutions for neither web browsing nor online storage services. The reason is that these services require source anonymity instead of data anonymity. Moreover, all destinations must be changed to support this protocol which is an impractical assumption. Finally, Riposte has poor performance in three cases: 1) large messages, 2) diversity in message sizes, and 3) there are a lot of writes.

Next technique is DC-net which is introduced by Chaum [53]. One way to implement DC-net protocol is random shuffling of the all messages by every client in such a way that nobody can correlate the shuffled messages to their owners. One instance of this protocol works as follows: first, all clients determine an order among themselves, then they send their messages which is encrypted by onion encryption technique [106] to the first client. It removes the external encryption layer, shuffles the messages randomly and finally forwards them to the next client. Other clients follow same procedure until all messages are decrypted. The last client broadcasts all messages to other clients. Performance of DC-net usually grows linearly with the number of clients.

One of DC-net based networks is Dissent. In Dissent V1 [57], clients broadcasts their messages to all other clients which has significant overhead in terms of bandwidth. In Dissent V2 [117], a few servers are introduced in order for clients to communicate with these servers and then the servers communicate with each other on behalf of the clients. However, to send a message, every client must send a message size equal to size of messages of all clients which makes it impractical for both web browsing and online storage service. Moreover, all servers forward decrypted messages to all clients. Figure 3.1 shows number of bytes received by every client if they use Dissent V2 to visit one of the top 200 websites in random. X-axis is total number of requests for all clients. For example, if total requests over time is about 20K requests, every client overall has to download about 21.6 GB of data. Dissent works for the
Figure 3.1: In Dissent, data is broadcasted to all clients. The more requests for visiting websites, the more data that every client has to download.

Figure 3.2: High level design of Innominate. (a) Data plane is responsible for transferring traffic through multiple groups to the destination. (b) Clients share their information about their buddies via DHT to build ASes which happens infrequently. Inside of each AS, groups are built frequently. A series of facilitators speed up group building process. (c) All clients are partitioned into ASes and ASes are split into groups.

cases that very few clients share very small messages.

Vuvuzela [112] is another anonymity network that allows clients to share small messages. In Vuvuzela, all messages must have fixed size and server pads them to the largest message size which is not efficient for online storage services. Vuvuzela is designed for private message sharing in which both sender and receiver pulls some information from Vuvuzela, therefore, if online services such as web servers want to communicate with their clients via Vuvuzela, they have to fetch clients’ messages i.e., server modification is required.

3.3 Overview

Innominate adopts relay-based technique for low latency communication, however
instead of a single client servers as a relay, group relay is used. In group relay, several clients first share received data anonymously in the group and then they forward the data. The main goal of group relay is to hide what everybody is sharing in the group by running a strong anonymity protocol such as DC-net. Besides, group relay prevents an attacker who can observe traffic of a specific client to know if the client sends any data. Figure 3.2-(a) shows a client which is communicating with a destination. All messages of the client gets to the destination after relaying by multiple groups. Inside every group, after running the strong anonymity protocol, all messages are assigned to a member to be forwarded by a deterministic function.

However, two issues are raised from this approach. First issue is that the number of intermediate group relays can not be fixed, otherwise the originator is always in the first group. A probability-based forwarding technique similar to Crowds [107] is used to establish variable path length. After running strong anonymity protocol inside the group, messages might be either forwarded to the destination or to the next group. This causes that the messages traverse variable number of groups and once a message is shared inside the group, it is unclear that the originator is in the group or the message is being forwarded by a member.

Second issue is that how groups are formed. Group building technique should consider these facts: 1) every client might join or leave Innominate at any time, 2) group building protocol must be light in terms of both control messages and time to build the group, 3) nobody can control over who be in the group and who be out, 4) a client has trust at some clients more than others to follow the protocol. The clients with higher trust value is called buddies. In this chapter, we consider friends and friends-of-friends from a social network as buddies. 5) group membership must be hidden to expand the set of clients who might be the originator to the set of all benign clients. The reason is that once an attacker observes a message in the group, the originator might be: a) a benign member in the current group, b) a benign member is forwarding

\footnote{An originator is a host which its ultimate goal is to anonymously exchange messages with an destination.}
the message from another group which is not compromised and it might be any of the benign clients who has been in another group with the benign member and it is unknown to attacker. We will discuss in more detail about this in Section 3.6.

In Innominat, control plane is separated from data plane. Control plane helps clients to build groups and no data is transferred through it to destinations. In our design, control messages are small and are fixed size, while data messages have different sizes. Innominat takes advantage of this property of control messages to design an efficient group building method. Figure 3.2 (b) represents physical design of control plane. Clients share some information such as their buddies through a Distributed Hash Table (DHT). If groups are constructed among buddies, then an attacker has no chance to run sybil attack assuming buddies of the client are not compromised. Based on data in DHT, clients run a distributed version of dominating set algorithm [71] to cluster themselves into ASes (Anonymity Sets). Every client in dominating set and its buddies form a AS. Next, clients inside every AS collaborate to construct the groups.

To build groups inside a AS, some servers called facilitators are used to securely share control messages among all clients. In analogy to Dissent V2 [117] and Riposte [56], we assume that all of facilitators do not collude together, although all of them might be malicious. The protocol has two rounds, in the first round, every client directly sends a message containing a temporary key and AS identity to one of these facilitators. These facilitators run a DC-net protocol on collected messages and publish it on a public bulletin which can be one of these facilitators or DHT. DC-net makes sure that nobody knows temporary keys’ owners. In the second round, based on temporary keys in the same AS, clients creates a random order among themselves and then every predetermined clients are in the same group. Clients in the same group encrypt their identity with shared temporary keys and publish it on the bulletin. Finally the clients download the identities and decrypts them to find other members in their group. Logical output of control plane is shown in Figure 3.2 (c) where all clients are clustered into
ASes and groups are formed inside every AS.

3.3.1 Challenges

**Sybil attack is likely to happen in distributed group formation.** An adversary might run a lot of compromised clients in hope to be in a group with only one benign client. An adversary also might compromise some clients inside a AS, in this case, if the clients is able to choose with whom they create a group, then compromised clients can build a group with the targeted client resulting in revealing the client’s communications.

**Frequent online/offline clients.** On one hand, DC-net protocol fails to run in the group when a member becomes offline, and on the other hand, a client should join a group for immediate communication after it becomes online. With a fast group building technique and keeping the group size small, we proactively form groups so as to tolerate any client going offline.

**Group memberships may leak resulting in a client achieving lower anonymity than the size of its AS.** Dividing clients into groups limits the set of clients who might be the originator to the group members. Thus, any protocol must make sure that group memberships are hidden to expand the potential set of originator to the benign clients inside th AS. Even though the group membership is hidden, they should not be used for independent communications *i.e.*, communication between different pair of originator and destination. As an illustration, in Figure 3.3 a message traverses three groups, G1, G2 and G3 before it gets to the destination. Let assume that one of the members in G1 and G3 are compromised. As they observe the message in their groups, they know who forwards the message in G1. Therefore, they know that the forwarder in G1 is in a group with one of the benign clients in G3. In long term communications, group memberships might leak. Moreover, after running strong anonymity protocol, the originator might be assigned to deliver its message to the destination. This is not desirable and our design must avoid any direct interaction between the originator and the destination.
Figure 3.3: A path with three groups, which first and last groups are compromised. The attacker can know the client who forwards the message to middle group and the one that forwards the message in third group are in same group.

**Adversarial facilitator might change group formation.** A facilitator might drop or modify a client’s message and it might also inject some messages in hope it can change group formation and gain some information about group memberships. We need an approach to enable a client to verify the information published in the bulletin to make sure that only clients in a AS can publish the data while their messages are intact.

### 3.3.2 Threat Model

In design of *Innominate*, every client trusts on its buddies to follow the protocol for forwarding the messages. However, while buddies run the protocol, they might collude with other attackers to identify the originator.

We assume an adversary controls all but one of the facilitators; clients do not need to know which facilitator. That is, as long as all facilitators do not share information of the control messages with each other, our clients are protected.

We consider an attacker to be capable of logging all of interactions with clients and analyzing these logs to identify the originator. The attacker might be the destination, other clients or anyone else who is able to monitor the traffic such as ISPs or government entities. However, we assume that monitoring is bound by local authorities, so clients in a group should make sure that at least one of the group members is in a location whose jurisdiction meets their privacy needs *i.e.*, they are in autonomous systems controled by different authorities, otherwise
the group is not safe enough and it should not used by the members. If an attacker can log all communications among all clients, the set of potential clients who might be the originator is limited to one group, otherwise it is included all benign clients in the AS.

We assume that a client exchanges its public key with its buddies either by relying on known centralized methods such as central authority [93] or UIA [64], or exchanging via traditional way such as email or social networks. Finally, we assume our cryptographic techniques are secure and if an originator’s communication with a destination is not end-to-end encrypted (e.g., using HTTPS), then the originator’s anonymity is trivially leaked.

3.4 Design

In this section, we explain design of Innominate in detail. We first discuss about how ASes are formed, then we describe our fast and efficient group building technique. Finally, Innominate takes advantages of onion routing to expand the set of potential originators who might sent the message, from group which is small to AS which is large. Table 3.2 summarizes main techniques and their goals in design of Innominate.

3.4.1 Distributed Anonymity Sets Creation

Straw man approach for group building is that every client asks other clients to form a group. For all clients who would like to join same group and have not joined any other group, a new group is formed. There are some problems with this approach. First problem is coordination among clients. If everybody starts building a new group, a client might join a lot of groups, at worst case number of its groups is $2^N$ where N is number of all clients (because a client can form groups with any subset of other clients); participating in this amount of groups requires a lot of resources. Second problem is that compromised clients can run selective attacks meaning
<table>
<thead>
<tr>
<th>Technique(s)</th>
<th>Goal</th>
<th>Applied to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group of clients acts as relay</td>
<td>Hide what is sharing by members inside the group</td>
<td>Data plane</td>
</tr>
<tr>
<td>Probability-based forwarding</td>
<td>Hide the group that the originator is in</td>
<td>Data plane</td>
</tr>
<tr>
<td>Hidden group membership</td>
<td>Extend the set of clients who might be the originator to AS</td>
<td>Data plane</td>
</tr>
<tr>
<td>Separating control plane from data plane</td>
<td>No data is transferred to facilitators, 2) control messages are small are same size, <em>Innominate</em> relies on this property to design a control plane that can handle millions of clients</td>
<td>Data and control plane</td>
</tr>
<tr>
<td>Limiting AS memberships to buddies and randomized group memberships</td>
<td>Prevent sybil attack</td>
<td>Control plane</td>
</tr>
</tbody>
</table>

Table 3.2: Summery of techniques and their goals in *Innominate*’s design.

that if compromised clients can identify the originator in the group, they obey the protocol for having a successful communication, otherwise they stop forwarding the traffic in hope that they identify the originator in her next attempt to build a new group. Third problem is that any client who initiates a group, has likely some data to send out.

To prevent an attacker runs a lot of compromised clients in hope to identify the originator, we introduce a community called Anonymity Set(AS). An AS is a set of clients who are buddies. AS is based on the fact that a client usually has more trust on some clients than others. The trust value can have a real number, but in this paper we consider it as binary number *i.e.*, either trust or distrust. As mentioned before, if two clients are friend or friends-of-friends on a social network, we assume that they are buddies.

We limit collaboration for group building to the clients inside the AS. AS building algorithm should cluster buddies into largest possible ASes so as to keep set of clients who might be the originator large. we modified the distributed dominating set algorithm [71, [79, 80] to find ASes. A dominating set is defined as one where all clients in the *Innominate* are either
in the set or have a buddy that is in that set. Every client in the dominating set and its buddies form an AS and the AS is identified by the client’s public key. This client is called AS head. AS head inserts the list of AS members into a DHT which is signed by its private key. Other clients look up DHT for entries related to the AS head to find out all AS members.

To have clients in largest size of ASes, we need to find dominating set of minimal size which is a NP-hard problem. There are centralized and distributed approximation algorithms \[71, 79, 80\] that compute a minimal dominating set which are optimal up to a certain factor. For ease of explanation and eliminating many unnecessary details, in this part we discuss about centralized algorithm and distributed dominating set algorithm will explain later.

In centralized approximation algorithms, a list of client degrees (number of buddies) sorted in descending order is obtained. First, the client with maximum degree is added to the dominating set; then, the client and its buddies are removed from the sorted list. This process repeats until no client is left in the list. If a client which is being to add to the dominating set has a buddy with higher degree (the buddy is already removed from the list because one of its buddy is in the dominating set), our algorithm add the buddy to the dominating set. This results in a larger AS.

In the distributed version of the algorithm, information about clients and buddies are shared through direct messages and DHT. Every client follows a protocol to decide about if it should create an AS or it should join another AS or it should wait until for its buddies to clear their status.

In the centralized version of the algorithm, repairing ASes can be done by repeating the algorithm to find new ASes or modify existing ASes. However, in the distributed version, the clients need to notify their neighbors and then in collaboration with their neighbors they fix the ASes.

**Obtaining authorization token to publish on bulletin.** After the AS is established, every client
obtains an authorization token from each facilitator which allows the client to publish on the bulletin. The purpose of the token is to prevent anyone from outside of the AS to publish any data related to the AS on the bulletin. The client uses blind signature technique \[52,41,40,55,92\] to get the facilitator’s signature on the token which is blinded. Every facilitator, firstly, verifies the client to be in the claimed AS, then it signs the token. For verification, the facilitator looks up DHT to make sure the client is in the claimed AS, as we mentioned before AS head inserts a list of AS members into DHT. This token is used later to allow only clients inside the AS to publish their data on bulletin.

3.4.2 Distributed Group Creation

In order to ensure anonymity, buddies are required to collaborate with each other to hide the originator from the attacker. To provide strong anonymity, the buddies have to run traffic analysis resilient protocol among themselves, however, such a protocol is expensive and it is scalable up to few tens of clients. Thus, in a system with millions of clients, the buddies has to be bundled into smaller communities which limit the scope of running the expensive protocol. This community is called group which is the set of collaborating clients that protects the originator’s anonymity by executing traffic analysis resilient. The groups are built every epoch which we will discuss about its duration in Section 3.7.

It is very important that group membership is hidden to anyone outside the group, otherwise an attacker might nail down the set of potential clients that the originator is among them to the group. If membership is hidden, even there is an attacker in the group, it can not know whether the message is originated from a benign client in the group or it is being forwarded by a benign client. The later case leads to expanding the set of clients who might be the originator to all benign clients in the AS because other groups memberships are unknown to the attacker, benign clients in its group may have a group with any of other benign clients.
To create groups with hidden membership, high level description of our method is as follows: 1) clients create a random order among themselves by sharing and shuffling temporary keys where every client owns exactly one key in the order. Nobody must not know the position of any other clients in this order. This key is used later to prove ownership of a position to other members. 2) every predetermined number of clients form a group, i.e., members of a group reveals their identity to each other. The method is explained in details next.

**Publishing temporary keys on the bulletin.** One way for anonymous sharing temporary keys is to run DC-net protocol among all clients. However, in a AS with millions of clients, this is impractical. Fortunately, all of these control messages (containing temporary keys) in control plane are same size which means message size does not leak any information about the owner. *Innominate* relies on this property to design group building method as it is shown in Figure 3.4

Every client shares a temporary key through facilitators on the bulletin. All facilitators have a predetermined order $F_1, F_2, ... F_n$ which is public. For ease of presentation, we assume that every one sends its message to $F_1$. We discuss about the conditions that this constraint can be safely removed in Section 2.6. Clients share temporary keys as follows:

1. a client generates an asymmetric key, $(pk, pr)$.
2. the client creates a message $M = (pk, AS(id), E_{F_1}(T_1), ..., E_{F_n}(T_n))$ where $AS(id)$ is the identity of anonymity set (public key of the AS head) and $E_{F_i}(T_i)$ is encrypting the
token that the client received after creating the AS by public key of facilitator $F_i$. Index $i$ represents facilitator $i$.

3. Onion encryption technique is applied on $M$ and it generates $M' = E_{F_1}\{E_{F_2}\{\ldots E_{F_n}(M)\}\}\}$ where $E_{F_i}\{msg\}$ is encrypting message $msg$ with public key of $F_i$.

4. the client sends $M'$ to $F_1$

5. the client receives a receipt from $F_1$. The receipt is hash value of $M'$ appended by a timestamp.

After collecting the messages by $F_1$ in one epoch, then $F_1$ removes first layer of encryption from all messages, shuffles them randomly and forwards them to the next facilitator. Next facilitator gives a receipt to the previous facilitator. Every facilitator also removes a layer of encryption, shuffles the messages and forwards them to next one. Finally, the last facilitator publish the messages on the bulletin. It also attaches epoch number to the message. Public bulletin can be a centralized database server or a distributed storage similar to Bitcoin [5]. Once data is published on bulletin, every facilitator validates its corresponding token of temporary keys. If a temporary key has a valid token, the facilitator signs hash value of the key and appends it to the message on bulletin. A message is valid if and only if all facilitator validate it. Finally, every clients directly downloads all shared keys of its AS from the bulletin.

Our method is different from Dissent [57] as follows: 1) size of control message that the client sends/receives to/from a facilitator is small while in Dissent, every client must send a message of size equal to size of all messages, 2) client does not transfer any actual data to the facilitators i.e., no actual data is passed in control plane where in Dissent control and data plane are combined and operated by central servers.

Random order of the clients. Every client downloads the list of all shared temporary keys in the AS. To prevent colluding clients leverage the ordering of keys in hope to be placed in a
group with a benign client, a new random ordering of these keys is generated as follows: As it is shown in Figure 3.5, every client runs cipher block chaining message authentication code (CBC-MAC) on the first shared key and other keys are given as the input key, one key per iteration. The output of CBC-MAC is a new key, the new key is used to map the current order to final order of shared keys. This method guarantees that keys are sorted completely randomly because the new key is generated after everybody commit to a temporary key.

**Bundle the clients into the groups.** In the sorted list of keys, every predetermined number of subsequent keys form a group. In next step, clients whose temporary keys are in the same group share their public keys. Every member first, encrypts its identity with other member’s shared key and then it published the encrypted message on bulletin by the same protocol for publishing shared keys. Finally, every client decrypts all encrypted identities with its temporary keys and finds out other members’ identity. It might query DHT for further information about the group members.

Group creation using this method makes sure that the group members are unknown to external entities. This protocol can repeat multiple times or clients share multiple keys in order to create several groups for every client. Being in multiple groups has several advantages:

1) increasing reliability of a successful communication, *i.e.*, in case a client in a group goes
offline, the group is inactive and communication can not happen through it. 2) it improves the anonymity of the originator which will explain about this in [3,6].

### 3.4.3 Group-based Traffic Routing

After the groups are formed, members share their messages by a strong anonymity protocol inside the group in data plane. Usually in strong anonymity protocols, the last member which removes the last layer of encryption sends all messages to all members. This is necessarily in order to prevent members to learn whom is targeted for every message. However, in our modification, instead of sending all messages to all member, every client receives one random message to handle. A deterministic function of hash values of messages is used to assign messages to different members in the group. A member delivers the assigned message to the destination with the probability of $P_d$ or forwards it to another group of itself with the probability of $1 - P_d$ (this is similar to Crowds [107]). Probability based forwarding makes path length variable and it prevents an adversary to know that the originator is in same group or another group.

Probabilistic approach can degrade the performance and it takes long time before the message gets to the destination. To avoid long delay, the originator retransmit the message after few epochs. For example, if $P_d = 0.5$, then after four epoch the probability of delivering the message to the destination is about 0.937. Thus the originator can retransmit the message after four epoch.

A member might be assigned to handle her message with the probability of $1/G_{Sz}$ where $G_{Sz}$ is the group size. In this case, to prevent direct interaction between the originator and the destination, the originator always forwards the message to another group of itself.
3.5 Implementation

In this section, we first, describe how a client starts using Innominate and what it needs to share with other clients. Then, we describe how the clients communicate with a destination. Note that groups and ASes can be built ahead of time so as long as a client wants to send a message, it immediately forwards the message to one of its groups.

3.5.1 Registration

The clients use DHT to share information among themselves. The information contains: (i) \((IP, port)\): this is IP address and port number of the client which is used by other clients to establish the connection. (ii) OnlineStatus: it is used for discovering online clients. Clients update their status once per epoch until they will consider online in next epoch. (iii) BuddyList: every client keeps a list of public keys of its buddies. (iv) OnlineBuddies: every client provides a list of online buddies periodically. Main purpose is to reduce number of exchanged messages; i.e., instead of checking status of all buddies who are friends of friends which are about 27 thousand, a client queries only its direct OnlineBuddies which are about 99 clients \([11, 16, 37]\). The client queries for status of friends of friends if its friend is offline.

In DHT, public key of the client is the key and the rest of information is the value.

3.5.2 Communication with destination.

Send a message. When an originator wants to send a request \(R\) to the destination, it randomly selects one of its groups where all members are online, otherwise the communication will fail and another group should be used. Then, the client creates a package as follows: \((E_{OSP}[R], rand, circuitID, hostName)\), where \(E\) encrypts \(R\) with the destination key, \(circuitID\) is a number which indicates packages with same circuitID should goes through same
path) and *hostName* is the hostname of destination\(^2\) The client sends the package to the group.

**Receive a message.** Once the destination responds the message, it attaches the *circuitID* to the response. So the response traverses the same path to get to the originator. In the reverse path, all messages are broadcasted to all members.

### 3.6 Anonymity Analysis

In this section, we analysis the anonymity provided by *Innominate*. In *Innominate*, ASes are public meaning that everyone knows list of ASes and their members, however, our group building technique as described in Section 3.4.2 guaranties that group members are hidden and only the members know each others. Considering this fact, if there is no attacker in the group, then the originator can be any of the benign clients in the AS. But if there is an attacker in the group, then forwarding policy between groups prevents the attacker to scale down the set of possible clients who might be the originator to the benign clients in the group. In this section, we discuss about the security properties of both control and data planes. Although no actual data is transferred through control plane, an attacker must not be able to selectively change group building procedure.

#### 3.6.1 Control Plane

To ensure integrity and trustworthiness of data on control plane, we discuss about potential attacks and the way that *Innominate* protects itself.

**Attack 1: identifying owner of published messages.** If an attacker can find out owner of temporary keys, the group membership is revealed (because the attacker knows position of every client in the order). Our method for publishing control messages on bulletin is resilient to traffic

\(^2\)If IP address is used instead of hostName, resolving IP address should happen in the client that hands in the message to the destination, otherwise, the destination might be a multi locations destination, so resolving IP address by the originator leaks its location.
analysis attack, because 1) all clients uploaded encrypted message of exactly same size, 2) all clients in same AS download all temporary shared keys for the AS which is again same size, and 3) as we assume in threat model, at least one of the facilitator does not collude with other facilitators, therefore similar to Dissent V2 [117] and Riposte [56], DC-net protects associating output messages to inputs. In conclusion, these properties prevents not only nobody can correlated the published data on bulletin to the original messages, but also size of messages does not reveal anything about the owner.

**Attack 2: someone from outside of a AS can share a temporary key with AS members.**

Sharing a key with other AS member allows an attacker to participate in groups while she is not a buddy with AS head. To prevent this attack, Innominate should guarantees that: 1) every client inside the AS must publish at most one key per epoch. 2) any client outside the AS must not be able to publish any key.

Message validation that happens after publishing data on bulletin allows only AS members to have a valid message on bulletin. Assuming that at least one of the facilitators is honest, nobody outside the AS can obtain a token from the honest facilitator. This blocks the external attacker to participate in group building. Note that, since token is obtained by blind-signature technique, facilitator can not detect owner of the token.

**Attack 3: a facilitator drops, modifies or maliciously invalidates a message.** In this attack, a facilitator can exclude or add some members in process of group building. Innominate prevents this attack by identifying the malicious entity. In design of Innominate, after delivering any message, the sender receives a receipt from the receiver. The receipt is hash value of the message. Once a client does not observe its message on the bulletin or it is invalidated incorrectly, it sends an appeal to all facilitator where the appeal is its own delivery receipt (hash value of the message delivered to $F_1$). If it is a valid receipt, then $F_1$ broadcasts its receipt about forwarding the message to $F_2$ to the client and to other facilitators. This procedure is repeated until the
violated facilitator, the facilitator without valid receipt, is found. Note that the client has not
sent out her actual data yet, therefore revealing the receipt and the temporary key does not leak
any information about the actual data.

### 3.6.2 Data Plane

To analysis client anonymity in data plane, we, first, define a compromised group as
the group with at least one attacker. In the compromised group, the attacker knows about all
members and exchanged messages in the group. However, it can not still map messages to
benign clients in the group unless one benign client exists. In this part, we first, compute $P_c(G)$
which is the probability that a benign client is in a compromised group. Then, we calculate $P_e$,
the probability which exposes the originator to the destination. We use the parameters defined
in table 3.3 in our analysis. Our goal for tuning these parameters is addressing trade-off between
performance and anonymity.

<table>
<thead>
<tr>
<th>Param</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$AS_{sz}$</td>
<td># of clients in the AS</td>
</tr>
<tr>
<td>$NG$</td>
<td># of groups of the client</td>
</tr>
<tr>
<td>$NE$</td>
<td>Maximum # of epoch before the client resend the message</td>
</tr>
<tr>
<td>$GS_{sz}$</td>
<td># of clients in the group</td>
</tr>
<tr>
<td>$Att(AS)$</td>
<td># of attacker’s clients in the AS</td>
</tr>
<tr>
<td>$Att(G)$</td>
<td># of attacker’s clients in the group</td>
</tr>
<tr>
<td>$Bn(AS)$</td>
<td># of benign nodes in the AS</td>
</tr>
<tr>
<td>$Bn(G)$</td>
<td># of benign nodes in the group</td>
</tr>
<tr>
<td>$P_e$</td>
<td>Prob. of exposing the originator</td>
</tr>
<tr>
<td>$P_c(G)$</td>
<td>Prob. of existing at least one attacker in the group G</td>
</tr>
<tr>
<td>$P_d$</td>
<td>Prob. of forwarding the message to the destination</td>
</tr>
<tr>
<td>$1 − P_d$</td>
<td>Prob. of forwarding the message to next group</td>
</tr>
<tr>
<td>$R$</td>
<td>Average rate of sending message</td>
</tr>
<tr>
<td>$P_1$</td>
<td>Prob. no attacker is in the communication</td>
</tr>
<tr>
<td>$P_2$</td>
<td>Prob. sending a message of itself in the group</td>
</tr>
<tr>
<td>$P_3$</td>
<td>Prob. of exposing originator, given it is in the group</td>
</tr>
<tr>
<td>$P_4$</td>
<td>Prob. the message originated from the group where attacker is</td>
</tr>
</tbody>
</table>

Table 3.3: Parameter definition.
3.6.2.1 Group Anonymity

The more honest friends that a client has, the better anonymity it gets. For example, if all clients in the AS are compromised, then the client can not have a safe communication. $P_c(G)$ is defined as the probability of a client being in a group with at least one attacker. Given a client in a group $G$:

$$
P_c(G) = \sum_{i=1}^{\text{AS}_{G} - 1} \binom{\text{Att(AS)}}{i} \times \binom{\text{Bn(AS)}}{G_{G} - 1} - i ^{\binom{\text{AS}_{G}}{G_{G} - 1}} \quad (3.1)$$

The nominator is all combinations of benign clients and attackers in the group. Note that in a compromised group at least one benign client and one attacker must exist. The dominator is all possible cases that a group can be built. A client participates in $N_G$ groups, thus the probability that all groups of the client are compromised is

$$
P_c(G)^{N_G} \quad (3.2)$$

All groups of the client should be compromised to limit the anonymity set to the benign clients in the group to which it sent the message, otherwise the attacker can not recognize whether the client is forwarding the message or it is the originator. Next, we discuss about this in more detail.

3.6.2.2 The originator Anonymity

In this section, we find $P_e$, the probability which the originator is exposed to the attacker, in other words, the attacker can say with the probability of $P_e$, the originator is the one that sent the message. We compute $P_e$ in two different cases, in the first case no attacker involves in the communication and in the second case at least one attacker participate in the
communication.

**No attacker.** There is no attacker in the communication if none of the groups through which the message is going is compromised. The message delivered to the destination is passed through at most $N_E$ groups, thus the probability that there is no attacker in the communication is

$$P_1 = P_d \sum_{i=0}^{N_E-1} (1 - P_c(G))^i \times (1 - P_d)^{(i-1)}$$  \hspace{1cm} (3.3)

each iteration of the summation is the probability that the message has not being in a compromised groups and all of them forwarded the message to next group. Finally $P_e$ when there is no attacker in the communication is:

$$P_e(\text{No attacker}) = P_1 \times \frac{1}{(Bn(AS) - 1)}$$  \hspace{1cm} (3.4)

In this case, the anonymity set is equal to all benign clients in the AS. Equation 3.4 states that the originator can be any of the benign clients in the AS from the destination’ point of view. Since the originator does not directly send its message to the destination, the client who handed in the message to destination should be excluded from the list of all benign clients who might be the originator.

**At least an attacker.** In the second case, at least one attacker observers the message before being forwarded to the destination. Once a message is seen by an attacker in the first compromised group, traversing the message through other groups does not change $P_e$, because the originator does not exist in later groups. Therefore, we compute $P_e$ in the first compromised group *i.e.*, the probability that a benign client in the group is the originator. The probability that at least an attacker exists in one of the groups is $1 - P_1$. To compute $P_e$ in this case, three items should take into the account:
First, $P_e$ depends on how often the originator sends a message in the compromised group. This is related to two factors: $(i)$ how often a client has data to send denoted by $R$. For example, if a client has one message per year, then the probability of sending its data into the group is negligible if each epoch is order of seconds or minute. In Section 3.4.3, it is discussed that the originator resends its own message to another group if it has been assigned to forward its own message to the destination, so $R$ also includes retransmitting the message in this case. In other words, $R = R \times (1 + 1/G_{R_e})$. $(ii)$ how many groups a client has denoted by $N_G$. For example, if client is in only one compromised group and it has one message per epoch, then the probability of the client send a message in the group is one. Anticipating these two factors, the probability of using the compromised group by the originator in the an epoch is:

$$P_2 = \frac{1}{R \times N_G} \quad (3.5)$$

Second, running of DC-net protocol inside the compromised group provides strong anonymity, then the originator might be any of the benign clients in the compromised group which is:

$$P_3 = \frac{1}{B(G)} \quad (3.6)$$

Third, We should also consider that the message has not been forwarded by the benign client. That is, since the client attends in multiple groups, it might be forwarding the message of a client in another group. The probability that the message is not coming from another group is that all messages in other non-compromised groups of the client are delivered to the destination in previous epoch.

$$P_4 = \left( \frac{P_d}{R} \right) (N_G \times (1 - P_c(G))) \quad (3.7)$$
In this case, $P_e$ is:

$$P_e(\geq 1 \text{ attacker}) = (1 - P_1) * P_2 * P_3 * P_4$$  \hspace{1cm} (3.8)

which states that one of the begin clients in the group is sending a message of itself.

Finally $P_e$ is calculated as follows:

$$P_e = P_e(\text{No attacker}) + P_e(\geq 1 \text{ attacker})$$  \hspace{1cm} (3.9)

Figure 3.6 shows $P_e$ for different parameters of Equation 3.9. In Figure 3.6 in case of $P_d = 1$, considering the facts that the message of the current client might be send to any of its groups and any of benign client in the group might also sent a message in the group, then $P_e$ is still low.

### 3.7 Performance Analysis

Performance evaluation of the *Innominate* falls into three categories. First, we analyze instantiation of *Innominate* using several social networks. Then, we study performance of control plane and finally we measure throughput of *Innominate* for two different applications web browsing and storage back up system.

#### 3.7.1 *Innominate* Instantiation

In this part, we first analyze AS size based on multiple social networks. An AS contains all clients who might be the originator, thus it is important that the size of this set be large.
enough. Then we measure performance of Innominate as size of group increases. Our goal of this experiment is to find a reasonable group size.

### 3.7.1.1 Analysis of Anonymity Sets in Social Networks

To evaluate our algorithm for building ASes, we ran it on three social networks described in Table 3.4. All of these data set have several million users and relations. Orkut and Pokec are online social networks collected in [85]. Pokec is the most popular online social network in Slovakia. To the best of our knowledge, no data set of Facebook users is available for research purpose, so we use graph generator described in [42] to generate a Facebook graph with properties discussed in [111]. We used nearest neighbor algorithm with parameters U=0.82 and k=60.

Figure 3.7(a) shows sizes of the ASes in different dataset. In our evaluation, friends and friends of friends are allowed to be in a AS. However, in the case that the AS size is not
<table>
<thead>
<tr>
<th>Data set</th>
<th>Nodes</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orkut</td>
<td>3,072,441</td>
<td>117,185,083</td>
</tr>
<tr>
<td>Pokec</td>
<td>1,632,803</td>
<td>30,622,564</td>
</tr>
<tr>
<td>Facebook</td>
<td>3,000,000</td>
<td>29,653,046</td>
</tr>
</tbody>
</table>

Table 3.4: Social networks used in our analysis for ASes.

Figure 3.7: (a) Cumulative Distribution Function (CDF) of ASes vs their sizes (b) CDF of clients vs number of their ASes.

satisfactory in some setting, Innominate can include friends within three hop distance and more.

In this figure, median of AS size is 10,058, 23,580 and 67,334 for Facebook, Pokec and Orkut respectively.

Figure 3.7(b) shows the probability of overlapping for a client in our datasets. Overlapping is the number of ASes that a client is attending in. The more ASes that a client has, the more resources are required. This graph shows that in median a client participates in 10-22 ASes. Note that, in our experiment we remove all clients with degree less than 10, because a lot of accounts in social networks are created and they never used, we consider them as outlier and exclude them from our analysis.
3.7.1.2 Group

After building the ASes, groups are created. Since members are friends or friends of friends, they are usually inside the same country [111][44]. It is shown in [111] using 721 million Facebook users that 84.2% of friendships are within countries. The clients in very different time zone are less likely to be online at same time, so they usually do not collaborate. Therefore in the rest of our analysis, we set the delay between two clients to 34.9 millisecond by using Dummynet [12]. This is the average latency in USA during the peak periods [28]. We chose USA because it is one of the largest countries in the world and the propagation delay is high. We also set the average connection speed to 3.9 Mbits per second [1] which is the average in the world.

To examine affect of group size on performance, we ran our prototype on Emulab [15]. We ran one thread per client, and to emulate real clients over the Internet, threads on different machines could be friends but not on same machine. That is, if two threads represents two friends, they are placed on different machines.

We measured throughput of sharing messages of size 1KB (very small), 100KB (small), 1MB (medium) and 100MB (large) in the group of different sizes. We assume that everyone has same message to share in every epoch. Figure 3.8 shows the results of the experiment. In these figures, $P_{exposing}$ is added as a reference to compare affect of group size on both anonymity and performance. The left and right Y-axe represent sharing time and $P_{exposing}$ respectively. This figure shows that for small messages, the throughput is less than 100 seconds, however for large messages for more than 4 clients in the group, it takes more than an hour to run DC-net. This shows that a system like DC-net or a modified version of it, Dissent [57], are not scalable when the message size are large. As it is shown in the Figure, anonymity values does not change when the group size increase by few more clients. For example, anonymity value increases from
2.17 \times 10^{-4} to 1.04 \times 10^{-4} for group size of 4 and 8 respectively. We use group size of 4, for the rest of our evaluation. In this experiment, to plot anonymity line, we set \( P_d = 0.5 \).

### 3.7.2 Facilitators

We need to study performance of facilitators in order to set up epoch duration \textit{i.e.}, epoch duration depends on the time that a clients starts sending a request to participate in group building until the time that it finds out the identity of other members in the group. The clients might have different access link and computation power, so the epoch time should be large enough in such a way that all clients with different resources can participate in the group building.
Figure 3.9: (a) time to deliver the keys to the facilitators, (b) time to verify a message.

We evaluate the epoch time in two parts a) communication with the facilitator and b) decryption time in facilitators. We measure the time to send a request for participating in the group building from 200 Planet Lab nodes to a facilitator running on Amazon cloud in North California. Figure 3.9(a) shows that all clients are able to finish their request in 6 seconds. Since each client has to download and upload data for sharing temporary keys and also exchanging identities, we recommend that the total cut-off time for communication to be set to 12 seconds. Figure 3.9(b) also shows the result for decrypting the messages in facilitators. We used RSA2048 with full-domain-hash using SHA256 in the standard way [46] and AES-256-CBC. As it is shown this time is a function of number of clients and facilitators.

3.7.3 Throughput of Innominate

In this part, we evaluate effectiveness of Innominate to deal with two different types of traffic: (i) small messages: visiting the top 200 popular websites. (ii) mix sizes: uploading and downloading data of size 1-100MB to Dropbox.

In our evaluation we compare these cases: (i) “Direct” which means without using
any anonymous application, the result is demonstrated. (ii) “OR” shows the result when Onion routing is used as anonymous provider. This is not “apple-to-apple” comparison, we added this line to have a comparison between solutions with strong and week anonymity. (iii) “Innominate” is the case that DC-net is used as underline protocol to provide strong anonymity inside the groups. (iv) “Dissent” is the case that we run Dissent V2 protocol.

**Visiting top 200 websites.** In this experiment, we measure throughput of downloading the homepage of top 200 websites from Alexa [2]. As it is shown in Figure 3.10 all websites can be visited in less than 45 seconds.

**Upload files to Dropbox.** We also measure upload and download time for files of size 1MB, 2MB, ..., to 100MB to Dropbox. Figure 3.11 (a) shows the result of uploading through Innominate is equal or more than onion routing technique, because a message might forwarded through multiple groups. That is, multiple rounds of onion routing is executed. Figure 3.11 (b) also shows the result of downloading from Dropbox, which is less than 2 hours for all files.
Figure 3.11: Performance of Innominate for upload and download files to Dropbox.
Chapter 4

DeltaTrack: Differential Dependency Tracking for Attack Forensics

In this section, we first establish notation and review the classic method for intrusion backtracking, then we discuss about the problem.

**Dependency Graph (DepGraph)** The *DepGraph* is a directed graph for tracking dependency relationships between OS resources across hosts in an enterprise. In the *DepGraph*, a node represents a socket, pipe, file, or a process and an edge represents a system call event. Direction of an edge depends on direction of data or control flow, e.g., \((\text{process}) \rightarrow (\text{file})\), \((\text{file}) \rightarrow (\text{process})\) and \((\text{parent}) \rightarrow (\text{child})\) represent a file write, a read, and a fork system call, respectively.

**Detection Point** Given a detection point (system event) it is common for an administrator to want to find the initial cause. For example, the administrator may want to track the origin of a suspicious file or the execution of a malware process. For ease of presentation, we only consider the case of a single detection point, but our method can be easily extended to handle multiple
detection points. Given a detection point, the goal of intrusion backtracking is to find all the resources the detection point is casually dependent on. Intuitively, the collection of all such dependencies gives rise to a succinct description of how the detection point was created, i.e., it traces back to the attack origin.

**Backtracking Graph** A typical algorithm for intrusion backtracking filters the DepGraph to only include events temporally dependent on the detection point. Timestamps are used to eliminate any event occurring after the detection point since it has no causal effect on past events. We will illustrate the process through an example. Figure 4.1(a) shows a DepGraph in which the label of an edge shows the timestamp of the event (the time when the event occurred), a square node represents a process, and an oval node represents a file.

The detection point, denoted by a bold red edge, has a timestamp of 6. Backtracking to `useradd` causes the edges to both `ld.so.preload` and `malware` to be added since they both occurred before $t = 6$. However, it does not add the edge $(ldconfig) \rightarrow (ld.so.preload)$ since its timestamp (7) is greater than the previously backtracked edge (timestamp of 5). The analysis proceeds to the `malware` node and uses the timestamp of 5, the highest of the two outgoing

---

Figure 4.1: DepGraph (a) and its backtracking graph (b), where rectangles are OS processes and ovals are files.
edges, to proceed. The edges to the malware file and wget are added since they have a lower timestamp than 5. Finally, the outgoing edge from the upper bash node is included.

Figure 4.1(b) shows the resulting backtracking graph, which explains the actions performed on the system. A bash process forks a wget process to download a file named malware, and then forks a malware process which reads the downloaded file, modifies ld.so.preload, and executes useradd. Subsequently, useradd adds a user by modifying the passwd file.

**Problem Definition** The goal of intrusion backtracking is to create a concise summary of the actions of an attacker for the administrator. However, the DepGraph generated by existing methods tends to have many nodes and edges even for trivial applications. For example, our experiments show that a simple echo application written in Python can trigger 819 system calls, and because of this, the DepGraph can be very large. Consider the backtracking graph generated for compiling and installing gcc: it has 15,301 nodes and 197,670 edges. The large size of the backtracking graph often prohibits human administrators from conducting attack forensics. The objective of our work is to automatically remove irrelevant parts of the backtracking graph to make it easier to understand.

We present a new differential dependency tracking method to drastically improve the performance of intrusion backtracking by automatically pruning irrelevant resource dependencies while maintaining the precision of the backtracking graph. Our work is motivated by the fact that existing domain-specific heuristics are insufficient for mitigating many sources of dependency explosions that we have observed in practice. Toward this end, we reformulate the intrusion backtracking problem as a problem of tracking the delta dependency. The intuition is that the backtracking graph is meant to be used by a system administrator to study how an attack transpired and therefore should highlight the difference between the attack behavior and the normal system behavior, as opposed to the entire system behavior.
Indeed, most of the backtracking graph’s components include activities completely unrelated to the attack and therefore can be safely excluded. However, state-of-the-art techniques [75] are not effective. For example, one pruning heuristic is to remove files creating dependencies across many processes using a white list of files. However, our experiments found 44,592 files used by many processes in one week of data logged from ten hosts. Maintaining a list of all such files manually is far too expensive and also error prone: if the white-list is too aggressive it may filter a file necessary to understand the root cause of an attack.

In contrast, we apply a differential dependency analysis. To discover the subset of attack related dependencies that differ from the normal system behavior, we make the following observations: (1) in most attacks, the majority of the hosts inside the enterprise are not compromised; (2) since enterprise systems have a large number of hosts similar to each other, it is possible to establish a baseline of common activities. Therefore, our approach relies on mining common system behaviors on all hosts within an enterprise to establish the baseline. The greater the number of hosts and the longer the period of monitoring, the better the baseline reference model is.

Figure 4.2(a) shows the physical design of our system, DeltaTrack, which builds upon
a ubiquitous framework to monitor and log the resource usage of all hosts in an enterprise. On each host, a light-weight agent continuously records events of selected system calls and reports in real time to a Backend Server. The Backend Server translates reported data from each host into a dependency graph (DepGraph) and sends it to the Analysis Server. Figure 4.2(b) shows the logical design of DeltaTrack. The DepGraph is fed to a frequent pattern miner to extract information of observed events and produce a reference model. During intrusion backtracking, the reference model is used to derive a relevancy score for each pattern. Finally, given a detection point and the reference model, DeltaTrack generates the backtracking graph consisting of only the difference between the attack and the normal system behaviors.

The main advantage of DeltaTrack is that it does not rely on any domain knowledge and ad hoc heuristics and thus is more generally applicable. Since all hosts in the enterprise are continuously monitored, as software evolves, the reference model can also be adapted automatically to these changes. This allows DeltaTrack to more accurately identify attack related events, based on the intuition that behaviors occurring frequently on many hosts in the enterprise are unlikely to be relevant to attack since an attack would almost always trigger some rarely seen behaviors. The intuition has been confirmed by our experiments using real attacks on popular applications.

4.1 Motivation

To experimentally identify the primary causes of dependency explosion in the DepGraph we used two datasets: one collected from 50 hosts over the course of a month, and another collected from 10 hosts over a week. Table 4.1.1 shows the most significant sources that we identified from these datasets together with their frequency. We also observed that it was possible for a single backtracking graph to have more than one source of dependency explosion.
In this section, we discuss, in detail, the main sources of dependency explosion, and limitations of existing techniques to handle these sources.

### 4.1.1 Sources of Dependency Explosion

<table>
<thead>
<tr>
<th>Source</th>
<th>Frequency</th>
<th>Existing Method [75]</th>
<th>Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single file, multiple writers*</td>
<td>44,592</td>
<td>Remove hub nodes</td>
<td>Binary decision, requires deep domain knowledge</td>
</tr>
<tr>
<td>Infinity dependency chain</td>
<td>141,503</td>
<td>None</td>
<td>n/a</td>
</tr>
<tr>
<td>Unix domain socket, multiple senders</td>
<td>3,351</td>
<td>None</td>
<td>n/a</td>
</tr>
<tr>
<td>Single writer, multiple readers</td>
<td>226,037,910</td>
<td>Remove read-only files</td>
<td>Hard to identify read-only files, requires domain knowledge</td>
</tr>
<tr>
<td>Pipes</td>
<td>10,029,688</td>
<td>Remove hub nodes</td>
<td>Insecure pipes, require deep domain knowledge</td>
</tr>
</tbody>
</table>

Table 4.1: Sources of dependency explosion that we identified together with frequency, whether they can be handled by existing techniques, and why, based on one month of data from 50 hosts (* was based on one week of data from 10 hosts).

Figure 4.3: Examples of sources of dependency explosion in the backtracking graph. (a) `bash_n` depends on all previous bashes. (b) installation of an package depends on all previous package installations. (c) `cupsd` depends on all applications that use the printer. (d) all processes sharing a pipe depend on each other.
Figure 4.4: Inflation in number of (a) nodes and (b) edges of the backtracking graph.

**Single File, Multiple Writers** A file can act as a hub in the backtracking graph and “broadcast” to multiple processes. For example, a terminal reads from `bash_history` at startup and writes to it on exit, making `bash_history` a hub by which any instance of a terminal is dependent on all previous instances. This is shown in Figure 4.3(a). In our one week trace from 10 hosts, there are 44,592 files with at least one reader and two writer processes sharing this type of dependency through 8,051 unique files; an example we identified in this group is the `xsession-errors` file, to which graphical applications write error messages.

We quantify the impact of this source of dependency explosion as follows. For each occurrence in our dataset where a single file had multiple writers, we ran the backtracking algorithm on one of the read events as the Detection Point and computed the number of nodes and edges in the resulting backtracking graph. Figure 4.4 shows that, for about 35% of the occurrences of this case, more than 5K nodes and 20K edges are added to backtracking graph, indicating it as a main source of dependency explosion.
Infinite Dependency Chain  We define a chain as a sub-graph of DepGraph consisting of some nodes and their relationships. In this source of explosion, the output of a chain in the DepGraph is the input to a similar chain. For example, Figure 4.3(b) shows a sub-graph for installing a package with apt-get. First, apt-get checks the status of installed packages by reading the status file and then forks an http process to download the package. Next, dpkg reads the package and updates the status file after successful installation. These interactions create an “infinite dependency chain” in which the input of a new chain, (dpkg)→(apt-get), depends on the output of a previous chain, (dpkg)→(status). The result of this dependency is that installing a new package depends on all previous installations. If only one package is malicious, a backtracking algorithm would incorrectly include all packages installed before it in the backtracking graph.

Unix Domain Socket, Multiple Senders  A Unix domain socket (UDS) causes a dependency explosion if multiple processes send a message through it. Figure 4.3(c) shows an example, which has a dependency between cupsd (Common Unix Printing System Daemon) and all applications that have ever printed a document such as lp (printing tool). If an attacker performs a privilege escalation on cupsd and the attack is backtracked upon, the backtracking graph would contain the actions of all applications that have printed a file. In our month long dataset, we observed 3,351 cases of shared UDSs, such as syslog, mail, dbus, cron, PHP, and Tor. Figure 4.4 shows that, for 20% of the occurrences of this case, more than 2K nodes and 5K edges are added to the backtracking graph.

Single Writer, Multiple Readers  This is a special case of “single file, multiple writers” where only one writer process exists. Although each occurrence of this type only slightly inflates the backtracking graph, the number of occurrences can be high, e.g., the echo python program
has 175 files of this case in the backtracking graph. Most library files fall into this category: they are written to by an installer process and then used by multiple reader processes. Another example is `/etc/hosts`, a static lookup table for host names accessed by every process that needs to resolve an IP address. If an administrator modifies this file by an editor such as `Vim`, all subsequent processes resolving an IP would be dependent on `Vim`. Across 50 hosts over the course of a month, we observed 226 million unique files with at most one writer process and at least two reader processes. While every occurrence of such files does not drastically inflate the backtracking graph, dependency explosion occurs due to the large number of such cases occurring in the backtracking graph.

**Pipes**  A pipe is accessible by the process that created it and any descendants of the process. When a process ends, a pipe it created can still be used by a descendant process that has an open file descriptor for the pipe. As a result, all descendant processes share a parent’s pipe through which they all become related. Thus, as shown in Figure 4.3(d), backtracking to a child process causes us to backtrack to all of its ancestor and sibling processes. In our month long dataset, we observed the creation of 10 million pipes. Figure 4.4 shows that backtracking on pipes results in graphs with over 1K nodes and 2K edges for half of these cases.

**Long Running Process**  A long running process receives many inputs and generates many outputs. These outputs causally depend on all inputs which leads to an explosion in the backtracking graph. For example, an Apache bsserver process may respond to a lot of requests. However, it is not usually the case that different requests are dependent on each other. We observed 74,703 processes in our dataset collected during the course of a day.
4.1.2 Limitations of Existing Heuristics

The original algorithm for backtracking [75] determined whether an event is related to the attack using reachability. That is, in the DepGraph, any event that could reach the detection point was included in the backtracking graph. However, the size of the resulting graph was often far too large. Although the authors proposed a set of heuristics to prune the graph, they are often not accurate and effective enough, due to limitations outlined in the last column of Table 4.1.1. In the remainder of this section, we discuss these heuristics in details.

Removing Hub Files One pruning heuristic in [75] relies on the user manually creating a white-list including files such as /root/.bash_history, /var/log/lastlog, /var/run/utmp and /etc/mtab. Any dependencies related to these files is removed from the backtracking graph. Creating such a white-list is both time consuming and error prone. It requires the user to have deep domain knowledge about every edge of every hub file to make a decision on whether to retain it. Due to the large number of hub files, as shown in Column 2 of Table 4.1.1, asking the user to manually keep track of all these hub files is infeasible. Additionally, the white-list needs to be updated frequently and requires the administrator to know the intimate details of each process in the enterprise.

Removing Pipes Another pruning heuristic [75] is to remove all pipe nodes regardless of their position in the backtracking graph. The rationale is that removing pipe nodes does not disconnect the detection point from the attack origin because pipes are inherited from a process’s ancestor which would be traced back anyway through process creation events. However, this is not always true since removing pipe nodes sometimes does eliminate attack edges. We refer to these pipe nodes as insecure pipes. Figure 4.5(a) shows the DepGraph of executing cat passwd | grep USER, where the detection point is the red bold edge. If the pipe node
Removing pipe nodes may lead to wrong results. (a) the original DepGraph for `cat passwd | grep USER`. (b) the DepGraph after removing the pipe node.

is removed, the `cat` process would not be excluded from the backtracking graph as shown in Figure 4.5(b). In general, if a node is connected to the backtracking graph only through a pipe node, removing that pipe node may incorrectly filter out the node.

Removing Read-only Files Another pruning heuristic proposed in [75] is pruning all the read-only files based on the assumption that if a file has been read from but never written to during the period of analysis (which is one day in their experiments) it can be removed from the backtracking graph. This heuristic is useful, for example, in filtering out shared library files loaded by a process. Unfortunately, we have found many potential issues with this heuristic. For example, if the analysis period includes a point when a system update is performed, then all the files affected by the update are no longer read-only.

Additionally, we found the accuracy of the read-only heuristic to, in some cases, be low. We conducted experiments to assess the accuracy when the analysis period is shorter than the life time of the OS. Our goal was to see how the time scope of the analysis affects the read-only heuristic. In other words, how long a time period is required for many files on the system to no longer be read-only.

We analyzed `write` system calls on 10 hosts for 7 days in an enterprise system using...
Figure 4.6 shows the total number of files written for an increasing analysis period. The result demonstrates that, if we limit the analysis period, some files will be incorrectly considered as read-only. The sharp increases in the number of files written usually corresponds to new software being installed or data files being copied. Therefore, if the time frame of monitoring is too short, many files would be incorrectly classified as read-only. But if the time frame is too long, the read-only file removal heuristic would fail to provide a reduction in the backtracking graph.

4.2 Challenges and Assumptions

In this section, we outline the challenges and assumptions in our design of DeltaTrack, as well as the threat model.

4.2.1 Assumptions

We assume that events occurring frequently on all hosts in the enterprise are unlikely to be relevant to an attack, whereas events occurring infrequently across several hosts have a higher chance of being involved in the attack. We assume that hosts are identical in the enterprise and clocks on the hosts are synchronized, e.g., via NTP, which allows us to correlate network events across hosts. We will further justify these assumptions in Section 4.6.

We assume that an attack generates at least a few rare events, especially during the exploit process to gain additional privileges. For example, if an Apache btserver is compromised to spawn a shell – such operation would be considered as a rare event since it is not what the btserver would normally perform. However, after the exploit, we do not assume events involved in the following steps are rare. For instance, if a shell is used to add a new user, it would be considered as normal. Since the Detection Point is typically at some time after the exploit, many
of the backtracking graph edges involved in the following steps might look completely normal.

**Non-Goals** Although the framework provided by *DeltaTrack* (shown in Figure 4.2) can in principle be used for a wider range of applications, including intrusion detection or fault detection, our focus in this work is only on attack forensics.

### 4.2.2 Challenges

In the design of *DeltaTrack*, we face three challenges: accuracy, robustness, and scalability. The first one is related to the backtracking algorithm whereas the last two are related to the reference model.

**Accuracy** We define the false negative rate as the fraction of truly attack-related events mistakenly removed from the backtracking graph, and define the false positive rate as the fraction of unrelated events mistakenly included in the backtracking graph. False negatives can hinder the understanding of the attack, whereas false positives can overwhelm the human administrator with unrelated information. Therefore, we need to minimize both the false negative rate and the false positive rate.
**Robustness**  The reference model building method should be resilient against poisoning attacks. That is, an attacker might frequently trigger some system calls to build a pattern in the reference model and later leverage it to perform an attack that would go undetected in the backtracking graph.

**Scalability**  Our goal is to scale up our method to enterprise systems of realistic size. Toward this end, the main bottleneck is the size of the *DepGraph*: after just a few days, it could contain millions of nodes and billions of edges. Storing all possible patterns (subgraphs of *DepGraph*) in the reference model would require a prohibitively large amount of storage and computation resources. Moreover, as the pattern (subgraph) size gets larger, it is less likely to repeat, thereby decreasing its chance of being removed from the backtracking graph.

### 4.2.3 Threat Model

**The Reference Model**  We assume that the reference model is constructed based on logs of hosts when the majority of them are not under attack. The reference model can also be improved by ignoring logs of the known-to-be-compromised hosts in the reference model at the time of backtracking. If an attack happened before installing the monitoring agent, however, *DeltaTrack* would not be able to prune the backtracking graph. For example, *DeltaTrack* would not be able to detect an intrusion resulting for installing a compromised OS image.

**The Monitoring Agent**  We assume that the agent requires enterprise administrator authorization to report the log to the *Backend Server*. This requirement prevents an attacker from running the agents on virtual machines to pollute the log.
4.3 Design

In this section, we first provide a formal definition of the backtracking problem. Then, we describe, in detail, our approach to build the reference model and our K-hop Backtracking algorithm. Finally, we discuss the security of K-hop Backtracking.

4.3.1 Definitions

Event We define an event as a triple \( \langle s, e, d \rangle \), where \( s \) and \( d \) are nodes and \( e \) is an edge in the DepGraph, denoted \( s \xrightarrow{e} d \). We identify edges by their type (e.g., Read/Write, or ProcessCreate). We identify nodes by their path, i.e., the location of the executable in the case of process nodes, or the location of the resource in the case of all other nodes. We aggregate events of the same type between a pair of source and destination in the DepGraph (e.g., a process reading a specific file multiple times).

The Backtracking Problem Backtracking requires two inputs: the DepGraph and the Detection Point. The DepGraph, denoted \( G = (N, E, \tau) \), is a graph constructed from the system audit log, where \( N \) is a set of OS objects, \( E \) is a set of events between objects, and \( \tau \) is a function mapping each event \( e \) to a timestamp \( \tau(e) \). The Detection Point, \( d \in E \), is an observed anomaly resulting from the attack.

The output of backtracking is a connected sub-graph \( G' \subseteq G \) rooted from the Detection Point consisting of only events temporally reachable to the Detection Point. More specifically, \( G' = (N', E', \tau) \), where each \( n' \in N' \) must be able to reach \( d \) via a temporal path \( p \). Let every directional event \( e \) be from the source node tail(\( e \)) to the destination node head(\( e \)). We define the temporal path between \( e_0 \) and \( e \) as \( p = \text{tail}(e_0) \xrightarrow{e_0} n_0 \xrightarrow{e_1} \ldots \xrightarrow{e_j} n_j \xrightarrow{e} \text{head}(e) \) such that \( \forall i \in \{0, \ldots, j\}, \tau(e_i) \leq \tau(e) \). Intuitively, \( p \) contains only events occurred before or at the same time as \( e \).
The original backtracking method \[75\] relied on only the temporal reachability constraint and heuristics when generating the backtracking graph. As shown in the previous sections, this often leads to many false positives and false negatives.

In contrast, a concise backtracking graph should represent the essential steps of an attack. Therefore, we formalize backtracking as a constrained global optimization problem as follows. We associate each event in DepGraph with a relevancy score, \( r(e) \in \mathbb{R} \); the higher the real valued score, the more relevant the event is to the Detection Point. Thus, the goal of backtracking is to produce a backtracking graph \( G'' \subset G \) that, under the temporal reachability constraint, maximizes the sub-graph’s relevancy score: 
\[
R(G'') = \sum_{e \in G''} r(e).
\]
The relevancy function, \( r(e) \), is a critical piece in formulating the backtracking problem. A higher value promotes inclusion of a relevant event, while a lower value encourages exclusion of an irrelevant event.

### 4.3.2 The Reference Model of Relevancy Scores

We first describe the relevancy function and the way that the reference model is constructed.

**Relevancy Score** The relevancy function \( r(e) \) must have these properties: 
1. \( r(e) \) is the highest if \( e \) occurs rarely on a few hosts, e.g., the Apache btserver spawns a bash. 
2. \( r(e) \) is the lowest if \( e \) occurs frequently on all hosts, e.g., a bash fork and execute another bash. 
3. \( r(e) \) is lower when \( e \) occurs frequently in only a small subset of hosts. This property is needed to cope with the fact that different hosts have different roles in a network and thus may have differing usage profiles. For example, if there are only a few hosts used for Java development, one might observe many similar events generated by Eclipse on these hosts. 
4. \( r(e) \) is lower when \( e \) is observed rarely in many hosts. This property is needed to identify normal events that are less
common. For example, joining the host to the enterprise domain by a legitimate administrator may happen once in each host.

We compute the relevancy score of an event in the backtracking graph based on how frequent it appears across hosts. The method takes into the account the frequency of the event at each host and over all hosts. Specifically, we define the relevancy score function of event $e$ as follows:

$$r(e) = \log \left( \sum_{h \in H} \frac{F_h}{f(e,h)} \right) \times \frac{\|H\|}{\#\text{hosts observed } e}$$  \hspace{1cm} (4.1)

where $H$ is the set of all hosts, $f_{(e,h)}$ is the frequency of the event $e$ in the host $h$ and $F_h$ is the frequency of all events in the host $h$. Our relevancy function is similar to the Term Frequency-Inverse Document Frequency (TF-IDF [115]). The summation term measures the rareness of the event per host; if the event happens less frequently, the relevancy score is higher. The logarithmic function is used to keep the first and second term in same range. The second term focuses on importance of the event across hosts—it is the fraction of hosts that observed the event.

Let $r_{\min}$ and $r_{\max}$ be the minimum and maximum values of $r(e)$ among all events, respectively. We normalize the relevancy score of each individual event $e$ as follows:

$$r(e) = \begin{cases} 1 & \text{no host has } e \\ \frac{r(e) - r_{\min}}{r_{\max} - r_{\min}} & \text{otherwise} \end{cases}$$  \hspace{1cm} (4.2)

In this way, $r(e)$ can be easily be computed for any subgraph of $DepGraph$. The relevancy score for a connected subgraph is $score(E) = \sum_{e \in E} r(e)$, where $E$ is set of events in the subgraph.

In general, this function gives higher weight to less frequent events. Therefore, events that do not occur frequently can easily be flagged as anomalous. This is the reason why the
relevancy score is a good proxy for the probability that the event is anomalous.

**Building a Reference Model** The reference model can be viewed as a map from an event to its relevancy score and a list of its frequencies on different hosts. To create the reference model, we iterate over each edge in the *DepGraph*, obtain the source and destination nodes to build the event triple described in Section 4.3.1 and then increase the frequency of the event for the corresponding host.

After computing the frequency of each event, the relevancy score of every event across all hosts is calculated. We only consider subgraphs of single event in order to reduce complexity. Although considering larger subgraphs (and thus learning more specific patterns) might result in more accurate results, we found that such approach is difficult to scale up to graphs with millions of nodes. Our experimental results in Section 4.5 confirmed that subgraphs of single event are already accurate enough indicating that it is a good design choice.

**Robustness** The naive frequency counting algorithm might be vulnerable to data poisoning attacks. That is, the attacker can repeatedly execute malicious actions in order to increase their frequency in the *DepGraph*. To mitigate such attacks, we allow each host to increment the frequency of an event once per a user specified *time window*. This increases the difficulty of making a malicious event appear to be normal: the attacker would have to frequently perform the event over a longer period of time and/or on many hosts. Ideally, the *time window* should be as long as possible, e.g., in the order of weeks or months.

**Data Ambiguity** Pipes may introduce another challenge since they are anonymous communication channels that cannot be identified by paths. To deal with this problem, when building the reference model and similarly when checking the relevancy of an edge, we consider all outgoing edges to a pipe as being connected to all incoming edges of the pipe. As a result, the pipe itself
is ignored and we are left with direct read/write edges between processes. For example, consider the pipe node in Figure 4.5 where after we remove the pipe itself, it is represented using edges such as \((\text{cat})\rightarrow(\text{grep})\).

During backtracking, we use the following scheme to decide if an edge to a pipe node should be considered: if any edge to the pipe is relevant, then include the pipe node and all its incoming edges.

### 4.3.3 K-hop Backtracking Algorithm

Our new backtracking algorithm aims to maximize the relevancy scores of events included in the backtracking graph while reducing its size. It adds any reachable temporal path to the backtracking graph until no path with the temporal reachability constraint is left. The events on all the paths must have the relevancy score higher than a cutoff threshold \(\theta\).

However, given the relevancy information, naive methods for pruning irrelevant events do not produce good results, since they may exclude relevant events that are not directly reachable from another relevant event. For example, if we have a subgraph

\[
\begin{align*}
&n_0 \xrightarrow{e_0} n_1 \xrightarrow{e_1} n_2 \xrightarrow{e_2} n_3,
\end{align*}
\]

where \(e_0\) and \(e_2\) are relevant events and \(e_1\) is an irrelevant event, then \(e_0\) may be excluded if, after seeing the irrelevant event \(e_1\), our backtracking algorithm stops examining other events.

We extend the previously defined temporal reachability constraint to cope with the pruning of irrelevant events that are connected with relevant events. We define a hopping parameter \(k \in [1, \infty)\), which is controllable by the user, such that an event \(e\) is included in the backtracking graph if there exists a path

\[
\begin{align*}
&\text{tail}(e_0) \xrightarrow{e_0} n_0 \xrightarrow{e_1} \ldots \xrightarrow{e_j} n_j \xrightarrow{e} \text{head}(e)
\end{align*}
\]

that satisfies all of the following conditions:

1. \(\tau(e_i) \leq \tau(e)\) for all \(i \in \{0, \ldots, j\}\), meaning that all events in \(p\) occur before or at the same time as \(e\);

2. Either \(r(e) \geq \theta\) or there exists \(i \in \{0, \ldots, j\}\) such that \(r(e_i) \geq \theta\); and
3. The length of $p$ is less than or equal to $k$.

Intuitively, only events that can reach a relevant event within a path of length $k$ are included in the graph.

Our method, referred to k-hop backtracking, is described in Algorithm 1. The algorithm assumes that a call to $r()$ returns the relevancy score of an event as defined in Section 4.3.1. The input consists of the DepGraph, the Detection Point, a value for $k$, and a minimum relevancy score, $\theta$. The function RELEVANT conducts a depth-$k$ limited search from the event $ev$. It is an implementation of the non-local greedy constraints previously described. For each edge encountered during the search, if it is considered as relevant, the event is added to the backtracking graph. Here, an edge is considered relevant if its relevancy score is more than the specified threshold ($\theta$).

Limiting the depth of the search to $k$ hops has two advantages: (1) It will include relevant events even if there are $k$ irrelevant events in between; and (2) since the DepGraph is a cyclic graph, and the relevant event can be reached from multiple paths, the use of parameter $k$ prevents longer paths from being included. For example, if an Apache bts server spawns a bash, the shortest path is $(Apache) \xrightarrow{create} (bash)$; however, longer paths, e.g., via bash_history, are possible as explained in Section 4.1. K-hop allows only shortest paths to be included in the backtracking graph.

4.3.4 Security Analysis of the K-hop Algorithm

$DeltaTrack$ aims to help an administrator detect the attack origin by automatically pruning the backtracking graph. An adversary with knowledge of the k-hop algorithm may attempt to render the algorithm ineffective. In this section, we analyze the likelihood of success by such an adversary.
In general, we recommend that an administrator start with a small $k$ to prune the graph aggressively. If the resulting graph explains the cause of the attack, then she can stop. Otherwise, $k$ can be gradually increased until the root cause is revealed. In the theoretically worst case, $k$ needs to be increased to $\infty$, when the backtracking graph degenerates to the naive backtracking algorithm without pruning. In practice, however, such cases are rare. In all the real attacks we investigated during our experimental evaluation, a small $k$ (around 2) was able to create a backtracking graph accurate enough to explain the attack.

Another problem is whether an adversary can create a long chain of events to intentionally inflate the DepGraph, in turn requiring a large $k$, to prevent our method from including attack related events. We identify two scenarios how the adversary can achieve this:

First, an adversary can create a long chain of process creations, e.g., by continuously forking shells from a parent shell to create a process-to-process chain. However, a process-to-process chain can only inflate the backtracking graph linearly. Anticipating this, we have modified Algorithm ?? to always consider processes creation events as relevant. Furthermore, an adversary usually wants to remain stealthy; creating a long process-to-process chain defeats this purpose by increasing the chance to trigger the alarm of intrusion detection systems.

Second, an adversary can create a chain of data dependencies such as $(\text{file}) \rightarrow (\text{process})$ and $(\text{socket}) \leftarrow (\text{process})$. However, such chains are inherently difficult to create. Figure 4.7(a) shows possible scenarios under which an attacker process can create a data dependency chain. We assume that the very first attack action $e_1$ is considered relevant, which maliciously manipulates the data node $D_1$. The independent victim process $P_V$ has to have a vulnerability or special behavior that can be exploited when reading the data. Here, the independent victim process means that neither the process nor its descendants have any interaction directly with the attacker processes.

Depending on the nature of the vulnerability, the victim process $P_V$ may extend the
data dependency chain either by (1) having another data node $D_3$ maliciously manipulated or
(2) forking an attack process $P_A$ which then writes to the data node $D_3$. In both cases, the
adversary may leverage $D_3$ to compromise another victim process $P_{V_2}$ to further extend the data
dependency chain. However, finding two independent vulnerabilities in $P_V$ and $P_{V_2}$ is already
difficult. Furthermore, to successfully inflate the dependency chain without being detected,
both $P_V$ and $P_{V_2}$ must generate normal events otherwise the attack events will be included in
the backtracking graph. If processes are not independent, a path to the attack origin exits and
K-hop backtracking will not miss it. For example, Figure 4.7(b) shows $DepGraph$ of running
catcommand||echo||...||P_A, $P_A$ is connected directly (one-hop) to the bash through a process
creation edge, therefore it is guaranteed to backtrack from $P_A$ to the origin.

Figure 4.7(c) shows an example of an attack on the Apache btserver that generates
rare event $e_3$. Once k-hop backtracking reaches the rare event, it resets the $k$, i.e., by inspecting
all data dependency events within $k$ hops, and therefore includes the origin of the attack.

To summarize, the adversary needs to satisfy two requirements to successfully extend
an undetected long data dependency chain: (1) a series of vulnerabilities or special behaviors ex-
ist in separate independent victim processes, and (2) exploiting the vulnerabilities will generate
only normal events.

The only example we can come up with is shown in Figure 4.7(d), where an attacker
first modifies the sudoers file to allow everybody to become a sudoer, then uses unauthenticated
telnet to add a user to the machine, finally uses the newly added user to run another attack. Here,
the victim process is sshd, which takes the maliciously manipulated input passwd file, yet its
behavior of forking a shell is completely normal (sshd is designed to fork shells).

Even in this case, $k = 3$ is sufficient to identify the origin of the chain of attack actions
(though $k = 2$ is insufficient). Therefore, we believe that generally $k = 2$ or $k = 3$ would be
4.4 Implementation

In this section, we first provide an overview of our light-weight monitoring agent, which gives us ubiquitous visibility throughout the enterprise, and then describe our approach to convert collected data into a DepGraph. The implementation of all components of DeltaTrack consists of 15K lines of C++ code and 57K lines of Java code.

The Monitoring Agent We implemented a monitoring agent to achieve ubiquitous auditing, and deploy it to all participating systems in the enterprise. The agent monitors the host activities using both the system’s built-in auditing mechanism, i.e., the Linux Auditing subsystem[22], which collects system call information of every process, as well as auxiliary information sources such as the proc filesystem. The agent performs a light-weight transformation on the collected
events and reports the event data to a central *Backend Server* in an aggregated and compressed format.

To ensure minimal resource and performance impact to the system being monitored, we make two main design choices. First, the agent does not record snapshots of files or network messages, which greatly reduces the storage and network overhead. Second, the agent selectively monitors only a subset of system calls that have important security implications.

In addition, we make trade-offs to improve monitoring overhead while keeping the accuracy at an acceptable level. For example, from simple benchmarking, we discover that *read* and *write* are among the most frequently used system calls. However, no read/write operations could occur on a file without a process opening it first, and the usage of *open* and *close* system calls are over an order of magnitude less. We therefore chose to monitor only *open* and *close*, and infer the data operations with slightly lower granularity and accuracy. That is, we infer when a file is opened with read/write permissions that the process actually performs a read/write syscall on the file. However, it may be the case that a process opens a file and does not read or write it, thus reducing accuracy.

<table>
<thead>
<tr>
<th>Event Type</th>
<th>System Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>fork, vfork, clone, execve, exit, exit_group</td>
</tr>
<tr>
<td>Network</td>
<td>socket, bind, connect, accept</td>
</tr>
<tr>
<td>File</td>
<td>open, creat, link, unlink, mount, symlink, stat, access, dup, fcntl, close</td>
</tr>
<tr>
<td>IPC</td>
<td>pipe, socketpair, shmget, msgget, socket, bind, connect, accept</td>
</tr>
</tbody>
</table>

Table 4.2: The system calls being monitored (partial list).

Table 4.2 provides a list of system calls that our agent monitors, classified into four categories. Overall, 71 system calls are monitored, but for brevity we list a subset.

Our experiments show that on average, the resource consumption of the agent is neg-
ligible. The agent processes around 30 system calls per second and consumes less than 1.0% of a single processor core and less than 100 MB of memory on an idle system. During intense workloads such as compilation, the agent processes around 5800 system calls per second and consumes up to 14.8% of a single processor core and 200 MB of memory. Network traffic from the monitoring agent is under 13 Kbps. It requires 130 MB of storage per day to monitor 50 hosts.

**DeltaTrack** We implemented a graph generator to convert audited events from all hosts into a DepGraph. The graph generator contains security relevant states of every OS object (such as processes, files and network connections) on each monitored host and models their state transitions across reported events. The reported events and state transitions are then converted into their corresponding graph elements and finally serialized into storage. The resulting graph is expected to be extremely large, therefore, we chose Neo4J\textsuperscript{[24]}, an open source large-scale graph database, to store and analyze the DepGraph.

DeltaTrack takes a DepGraph as input and creates a reference model. We use RocksDB\textsuperscript{[30]} to count the frequency of events and store rules in the reference model. The average runtime of K-hop Backtracking is 26.96 seconds. Our reference model builder uses around 4 GB memory to process one hour of data of 50 hosts, and takes around 10 minutes. The btserver building the reference models has a Xeon E5-1650 processes with 64 GB memory. Although further improvements are possible, e.g., by parallelizing the reference model construction step, the task of creating the reference model only needs to be performed infrequently, making it less of a bottleneck in our system.
4.5 Evaluation

Our evaluation consists of three parts. First, we demonstrate the effectiveness of \textit{DeltaTrack} at reducing the size of the backtracking graph for real attacks. Second, we evaluate the capability of \textit{DeltaTrack} in dealing with the hard-to-prune sources of explosion mentioned in Section 4.1. Finally, we evaluate the accuracy of the reference model and discuss its impact on performance. On all the experiments, we set $\theta = 0.3$.

We deployed \textit{DeltaTrack} on fifty hosts in an enterprise including both desktops and servers. We used one week of data to build the reference model in all experiments with the time window of one day. The examples were tested using a parameter of $k \leq 3$.

4.5.1 Analysis of Real Attacks

Similar to prior work \cite{75,76}, we demonstrate the effectiveness of our method in pruning the backtracking graph generated from real attacks. The first attack is related to the one of the main sources of dependency explosion discussed in Section 4.1. The second attack spans across multiple hosts and is used to demonstrate the effectiveness of \textit{DeltaTrack} to detect inter-hosts attacks. Finally, we discuss real attacks on six other popular applications.

In our experiments, we relied on manual inspection to confirm the validity of our automated analysis results. The rationale is that only a human expert can make a final decision on whether an event is related to the attack. Since in all the attacks we inspected, there were no false negatives, meaning that \textit{DeltaTrack} never removes relevant portions of the backtracking graph, we conclude that our method is capable of pruning the backtracking graph while maintaining the relevant information. Therefore, in the remainder of this section, we only report the false positive rate as a metric for measuring the accuracy of our algorithm.
Intra-Host Attack: Privilege Escalation The intra-host attack we considered is CVE-2008-5377 [7]. The attacker escalates a user’s privilege by exploiting a vulnerability in CUPS. Specifically, a corrupted PostScript file is passed to CUPS, which causes CUPS to load a malicious library file. After elevating their privilege, the attacker creates a backdoor into the system by adding a new user. The detection point is the modified passwd file.

Figure 4.8 shows the result of our K-hop Backtracking algorithm, which starts backtracking on the bold red edge between useradd and passwd. The graph shows that a malicious file called exploiter is downloaded by wget. Then, exploiter creates a C file named /tmp/getuid.c, and compiles it to /tmp/getuid.so. The exploiter process also creates a corrupted PostScript file

Figure 4.8: The backtracking graph for an intra-host attack (CVE-2008-5377 [7]), where the attacker escalates privilege by exploiting a vulnerability in CUPS to add a user. The detection point is the unauthorized access to passwd.

Intra-Host Attack: Privilege Escalation The intra-host attack we considered is CVE-2008-5377 [7]. The attacker escalates a user’s privilege by exploiting a vulnerability in CUPS. Specifically, a corrupted PostScript file is passed to CUPS, which causes CUPS to load a malicious library file. After elevating their privilege, the attacker creates a backdoor into the system by adding a new user. The detection point is the modified passwd file.

Figure 4.8 shows the result of our K-hop Backtracking algorithm, which starts backtracking on the bold red edge between useradd and passwd. The graph shows that a malicious file called exploiter is downloaded by wget. Then, exploiter creates a C file named /tmp/getuid.c, and compiles it to /tmp/getuid.so. The exploiter process also creates a corrupted PostScript file
Table 4.3: Comparing performance for the intra-host attack.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Proc</th>
<th>File</th>
<th>Sckt</th>
<th>Pipe</th>
<th>Edge</th>
<th>FPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>No pruning</td>
<td>233</td>
<td>25</td>
<td>9</td>
<td>164</td>
<td>1617</td>
<td>1485/3750</td>
</tr>
<tr>
<td>Heuristics in [75]</td>
<td>139</td>
<td>22</td>
<td>9</td>
<td>93</td>
<td>1071</td>
<td>839/2450</td>
</tr>
<tr>
<td>K-hop Backtracking</td>
<td>18</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>42</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.3: Comparing performance for the intra-host attack.

named /tmp/cve-2008-5377.ps, which is printed by the *lp* command. Printing this file causes *cupsd*, the CUPS daemon, to write to the system file /etc/ld.so.preload through a symbolic link /tmp/pstopdf.log. These two files are merged into a single node because one of them is a symbolic link to the other. The file ld.so.preload contains shared libraries to be loaded before a program starts. This causes *bash* to read the malicious library file /tmp/getuid.so, escalating the privilege of the attacker. Finally, the attacker runs *su* and successfully invokes *useradd* to create a new user account.

In Table 4.5.1, we compare the result of our fully automated backtracking method with the results of using no pruning heuristics and using the human-in-the-loop approach by King et al. Columns 2–6 show the number of processes, files, sockets, pipes, and edges of the backtracking graph, respectively. The last column (FPR) shows the false positive rate for nodes/edges, e.g., 1485/3750 indicates a false positive rate of 1485 and 3750 for nodes and edges, respectively. Overall, our new method leads to a significantly smaller backtracking graph, whereas the current state of the art has 839 and 2460 times the inflation in the number of nodes and edges, respectively.

As one can see, our method produces a concise description of the attack without distracting the user with any unrelated nodes or edges. Furthermore, it contains all the essential steps involved in the attack demonstrating that our new method can significantly reduce the size of the backtracking graph while preventing false negatives.

The increased number of nodes and edges in the heuristic approach is mainly due to
some hub nodes such as /dev/log and /dev/pts. Our method is able to automatically handle such cases whereas existing methods would require the system administrator to manually identify such files as being benign. Additionally, there are sources of dependency explosion caused by a pipe created by sshd, the ssh daemon, to all of its children processes.

**Inter-Host Attack: Remote to Local Access**  
Figure 4.9 shows an overview of an inter-host attack we examined. An external attacker first compromised a Nginx server to open a reverse shell on the attacker’s machine. Next, a git server was exploited (CVE-2014-4511 [10]) giving the attacker access to the server. The attacker left behind a file named `backdoor` which is the detection point.

Figure 4.10 shows the result of our K-hop Backtracking algorithm illustrating all three steps of the attack. First, a malicious file named `reverse-shell.gif` is uploaded to the Nginx server and then executed by `php5-cgi` giving the attacker a shell on the server. Next, the attacker downloads a suspicious file named `/tmp/git`, which when executed creates a file `x.php`. Finally, the attacker runs this file, creating a reverse shell to the git server and leaving behind the `backdoor` file.

Similar to our experimental analysis for the intra-host attack, here, we compare the result of our new method with the results of using no pruning heuristics and using the heuristics by King et al. [75]. Table 4.5.1 shows the experimental results. Compared to existing methods,
Figure 4.10: The backtracking graph for the inter-host attack in Figure 4.9. The external attacker gains a shell to an internal version control server through an enterprise web server.

our fully automated K-hop Backtracking algorithm can provide a more drastic reduction in the graph size. In addition, we are able to to keep the false negative rate at zero.

**Six Additional Attacks** Next, we present the results of applying DeltaTrack to six additional attacks. Due to space constraints, we cannot present the full graphical outputs of these experiments. Instead, we summarize the results in Table 4.5.1, where Column 1 shows the name of each test case, Columns 2–3 show the size of the original backtracking graph, Columns 4–5 show the size of the new backtracking graph, and Column 6 shows the reduction ratio in the number of nodes/edges.
The first two test cases are attacks on Distcc. The vulnerability (CVE-2004-2687 [6]) allows a remote attacker to execute arbitrary commands via compilation jobs. The third test case is an attack on PHPCGI. The vulnerability (CVE-2012-1823 [8]) allows remote attackers to execute arbitrary code by placing command-line options in a query string. The fourth test case is an attack on apt-get. A user downloads a malicious package that opens a reverse shell to the attackers computer allowing the attacker to add a backdoor to the user’s system. The fifth test case is an attack on MySQL. The vulnerability (CVE-2012-4255 [9]) allows a remote attacker to obtain the installation path via an error message. The last test case is an attack on SSH. The vulnerability (CVE-2014-7169 [11]) allows attackers to write to files on a remote system.

For all test cases, the backtrack graph generated by our method could accurately explain the attack’s steps without introducing false negatives. Furthermore, the reduction ratio ranges from 3x to 131x in the number of nodes, and from 4x to 512x in the number of edges. For the few cases where the reduction was less significant, we noticed that it was because our algorithm was unable to prune certain portions of the graph, e.g., operations for installing a package in apt-get create many intermediate files that are not frequently observed on the system.

In this case, although the resulting graphs are still small enough for a human to understand, it does point to a limitation of our method: since it was not designed to mine high-level concepts, such as the dashed boxes in Figure 4.10, we may leave behind many of these intermediate files.
Table 4.5: Experimental results on six additional attacks.

| Test case       | | Backtracking Graph | K-hop Backtracking | Reduction |
|-----------------|-----------------------|---------------------|-----------|
|                 | Node | Edge | Node | Edge | Ratio |
| Distcc+nmap     | 526  | 1421 | 32   | 47   | 16/30 |
| Distcc+passwd   | 72   | 165  | 22   | 37   | 3/4   |
| PHPCGI+nmap     | 106  | 280  | 19   | 24   | 5/12  |
| apt-get         | 285  | 1108 | 27   | 40   | 10/28 |
| MySQLDumper     | 983  | 5439 | 57   | 73   | 17/74 |
| SSH             | 1052 | 4102 | 8    | 8    | 131/512 |

Package Installation In Section 4.1, we discussed repeated paths as a frequently encountered source of dependency explosion. In order to quantify the extent to which it affects DeltaTrack’s performance, we conducted an experiment by installing make, running a system update, and then installing a Java runtime library. Backtracking, without our pruning method was performed on the installed make binary. The resulting backtracking graph contained 664 nodes and 2,224 edges. After its installation, we backtracked on the Java library file, which lead to a graph containing 10,571 nodes (16X) and 118,682 edges (53X). The backtracking graph contained the entire installation of make. In contrast, using K-hop Backtracking, we can prune away all irrelevant edges, leaving only the graphs of the installation of make and the Java runtime library. Therefore, we conclude that our new method is effective in mitigating this particular type of dependency explosion.

4.5.2 Reference Model Evaluation

To evaluate the effectiveness of building and using the reference model in DeltaTrack, we first analyze how the accuracy of the model is affected by the amount of data used to create it. Then, we analyze how effective it is at mitigating the dependency explosion.

Accuracy of the Reference Model Our experiments show that overall our algorithm performs better when more data is available to build the reference model. In the worst case, when the
reference model is empty and all events are considered as relevant, our new method degenerates into the method labeled as No Pruning in Tables 4.5.1.

During experiments, we used a DepGraph of 50 hosts, and varied the amount of data used to build 30 reference models ranging from data included from 0 to 55 days. The number of false positives were averaged across the 30 datasets for different thresholds. Here, we define false positive as the events in reference model that remain irrelevant after a certain amount of days, assuming the reference model is holding events irrelevant to the attack. The results summarized in Figure 4.11 shows that the number of false positives decreases as the amount of data increases.

The reason why the rate is decreasing as more days are evaluated is because (1) software keeps updating so some events which are originally rare become non-rare after a few days, and (2) the user’s usage pattern, such as how often a user performs the exact same task, is different. For example, adding a user to the system is a rare event that requires us to have a large amount of data to witness the event and classify it as relevant.

Figure 4.11 also shows that as the amount of data used to build the reference model
increases the effect of the threshold on the number of edges marked as relevant decreases. That is, with more data the value chosen as the threshold has less impact on the resulting backtracking graph. As a result, with more data the selected threshold causes less of an effect on the performance of the algorithm. This makes our algorithm more practical in that the system administrator will not have to spend a lot of effort fine tune the threshold in order to get acceptable results. Instead, the administrator can start with higher value of $\theta$ which results in smaller graph (the smaller the graph is, the easier it is to understand) and then gradually decrease $\theta$ to include more events and expand the graph.

**Effectiveness for pruning**  
Next, we evaluated the effectiveness of using the reference model inside our K-hop Backtracking algorithm to prune the backtracking graph by automatically identifying and managing the sources of dependency explosion shown in Figure 4.4. Specifically, we calculated the reduction ratio, or, the total number of nodes/edges associated with a given source of dependency explosion divided by the number of nodes/edges that remain after applying the k-hop algorithm.

Figure 4.12 shows the cumulative distribution function (CDF) of the reduction ratio. The data set used to create Figure 4.12 was collected during the course of a week. Each source of dependency explosion (Figure 4.4) was found in the dataset and the reduction ratio was calculated. For example, for 40% of pipe occurrences, we completely remove the pipe while for the rest of them, we partially remove the edges. For the nodes that we remove, reduction ratio is infinity, we use 10,000 as a proxy for infinity to plot the graph, this is the maximum value. Overall, the results show that our method is capable of significantly reducing the complexity of backtracking by coping with sources of dependency explosion.
Figure 4.12: Reduction ratio in the number of (a) nodes and (b) edges. A reference model of one week data is used.

4.6 Discussion

In this section, we discuss the validity of our assumptions.

Threats to Validity First, we assume that it is unlikely an attacker compromises all hosts in the enterprise, although it is possible if, for example, the attacker breaks into a single machine and then compromise the rest of the enterprise. But even in such case, DeltaTrack can still be effective with additional effort from the system administrator. Specifically, the administrator would have to estimate the time when the majority of hosts became compromised. Reference models built before then can still be used. Moreover, if the enterprise uses a stable set of software
(i.e., behavior of programs does not change often), then the older reference model can be used even after the entire enterprise is compromised, thereby enabling a reduction in the size of the backtracking graph.

Second, we assume that frequently occurred events are unlikely to be related to the attack. As a result, we could miss attacks if the attacker were able to closely mimic the normal behavior of the user. For example, an SQL injection attack may simply trigger the web btserver to send a malformed SQL request to the database. Without fine-grained monitoring, it will simply look like a regular socket connection (common event). However, to fully evade our scheme, an attacker may have to significantly constrain his behavior.

Another example is that $n_0 \xrightarrow{e_0} n_1$ and $n_1 \xrightarrow{e_1} n_2$ may be normal events but $n_0 \xrightarrow{e_0} n_1 \xrightarrow{e_1} n_2$ together is an attack, as discussed in [113]. However, during our experimental evaluation, we found that all attacks performed some kind of rare event. For example, an exploit to a mail btserver caused it to fork a shell; this exploitation behavior was uncommon even though the follow-up attack steps might look normal. We did not find any attack that consisted of only common events.

**Rarely Used but Benign Programs** Some programs are rarely used meaning that a long period of time is needed for their respective events to gain a proper relevancy score. As a result, when an intrusion occurs, these irrelevant events could be mistakenly reported to the system administrator. While this will at first add some noise to the backtracking graph, the longer DeltaTrack is used, the better it will be at pruning such events. One solution is for the administrator to run the clean program on a clean host and build a database of trusted events. Events in the database can be used to remove benign but rare events from the backtracking graph.
4.7 Related Work

**Attack Forensics through System Audit Logs**  King and Chen [75, 76] pioneered the work on backtracking the resource dependencies from the intrusion detection point to the attack origin and proposed a set of domain-specific filtering heuristics for highlighting the important dependencies. However, they did not employ the idea of differential dependency analysis, which is a main contribution of our work. Furthermore, in contrast to their domain-specific, human-in-the-loop approach, our new method for pruning the backtracking graph is fully automated and works with more general cases without specific domain knowledge. Although our focus is on backtracking analysis, the proposed techniques are complementary to and can be combined with forward dependency tracking [122] to estimate the impact of the attack.

**Application-specific Attack Forensics**  There are also techniques for leveraging application-specific semantics to improve the precision of the forensic analysis while cutting down its computational cost [76, 50, 73, 51]. In contrast, our new method do not rely on the availability of any application-specific information and therefore is more generally applicable.

**Inferring the Reference Model**  There is a large body of work leveraging system call data to build models characterizing normal execution behaviors [116, 114]. However, none of these existing techniques has ever been used in the context of enterprise attack forensics to prune the backtracking graphs.

In designing *DeltaTrack*, we have emphasized efficiency and scalability, which differentiates our method from many heavy-weight analysis techniques. In the remainder of the section, we discuss these other options of backtracking with additional overhead. They represent different trade-offs in the design space and some of the solutions are customized for specific use cases.
**Dynamic Taint Analysis**  Taint analysis [100, 72, 70] is a widely used technique in the security domain for various purposes including vulnerability analysis [100] and malware analysis [120]. However, dynamically tracking the taint information across a large system can be prohibitively expensive. In contrast, our new method tracks only dependencies among OS resources such as processes and files, and therefore is significantly more coarse-grained and lightweight.

**Dynamic Program Analysis**  Long-running processes have been handled using dynamic program analysis to gather additional logs of program behavior [84]. While our method is more general at dealing with the dependency explosion problem we believe dynamic analysis can also supplement our approach.

**Replay/re-execution**  Reply is another approach for recovering from intrusions [68, 74]. Compared to attack forensics, intrusion recovery goes one step further, with the goal of equipping a system administrator with a tool to automatically or semi-automatically clean up the system. Since their goal is intrusion recovery, the CPU and storage overheads are about 2x and 30–1000x larger respectively compared to our method.

**Decentralized information flow control**  There is a large body of work on runtime enforcement of information flow policies [62, 121, 78]. These techniques are orthogonal to our work on backtracking analysis based attack forensics.
Algorithm 2 Backtracking algorithm.

1: function BACKTRACK(depGraph, Detection Point, k, θ)
2:     Let $S$ be a stack of edges
3:     Let $E_f$ be the set of all edges in the Backtracking Graph
4:     $E_f ← ∅$
5:     $S$.PUSH(Detection Point)
6:     while $¬S$.EMPTY do
7:         $e ← S$.POP
8:         $n ← e$.SOURCE
9:         $E_f ← E_f ∪ e$
10:        for all $i ∈ n$.INCOMING EDGES do
11:            $ev ← (i$.SOURCE, $i$, $i$.DEST)
12:            if RELEVANT($ev$, $θ$, $k$) then
13:                $S$.PUSH($i$)
14:        end if
15:    end for
16:    end while
17:    return $E_f$
18: end function

19: function RELEVANT($ev$, $θ$, $k$)
20:     Let $P$ be the set of all paths from $ev$ of length $k$ such that $∀p ∈ P, τ(p) ≤ τ(ev)$
21:     for all $p ∈ P$ do
22:         if $∃e ∈ p$ | $r(e) ≥ θ$ then
23:             return true
24:         end if
25:     end for
26:    return false
27: end function
Chapter 5

Conclusions

In this chapter, I summarize the work presented in this dissertation and the contributions made.

5.1 Anonymous Communications

As of user privacy over the Internet, two systems have been developed.

LASTor Though Tor is the most widely used anonymity network today for low latency anonymous communication, poor latencies on it and the fear of traffic correlation attacks by underlying ASes are the biggest problems with Tor's usability today. Prior proposals have either focused on improving the performance on Tor in terms of throughput, which does not help interactive communication, or they mandate significant modifications to Tor relays, which places the onus on developers and thus are yet to be deployed.

In this thesis, we developed a new Tor client, called LASTor, to demonstrate that both significant latency gains and protection against snooping ASes can be obtained on Tor today without requiring any modifications to Tor relays. Based on measurements along paths between 10K (client, destination) pairs, we showed that LASTor can deliver a 25% reduction in median
path latency. To deliver these latency benefits, we showed that it is important to carefully select entry guards and account for replicated destinations. We also developed a space- and time-efficient technique for enabling LASTor to reliably detect the possible presence of snooping ASes on any path. Moreover, we have made path selection in LASTor tunable so that a user can easily choose an appropriate trade-off between latency and anonymity.

**Innominate** Existing solutions for anonymous online communications fall short in one of two ways: either the anonymity offered is susceptible to traffic analysis, or strong anonymity comes at the expense of poor performance at scale. In this thesis, we present Innominant, a new framework for traffic analysis resilient communications that both offers strong anonymity and scalable performance. Innominant uses relay-based forwarding (like Tor) to ensure good performance, but to ensure resilience to traffic analysis, Innominant differs from prior systems that use this approach in two significant ways. First, instead of individual nodes serving as relays, in Innominant, every relay is a small group of nodes that shuffle traffic among themselves via a DC-net before forwarding. Second, we design a lightweight protocol for users to self-organize themselves into these small groups in a manner that is resilient to Sybil attacks and hides the memberships of any group from those outside the group.

### 5.2 Dependency Tracking for Attack Forensics in Enterprises

Our new enterprise attack forensics system called DeltaTrack leverages the idea of differential dependency analysis to drastically reduce the size of the backtracking graph while maintaining its accuracy. It establishes a knowledge base of benign execution behavior by continually gathering execution traces from all hosts and then leverage the difference between attack and normal system behaviors to prune away irrelevant dependencies. We have implemented DeltaTrack and evaluated it on more than fifty hosts with real attacks on popular applications,
to demonstrate the effectiveness of the proposed techniques.

5.3 Contributions

This dissertation makes fundamental contributions in both of these two areas: (1) building secure and scalable anonymous network, (2) automatically removing irrelevant parts of graph of backtracking intrusion to conduct attack forensics in more efficient way.

First, not only LASTor makes default Tor client resilient to snooping AS, but also it improves performance of default Tor client. Second, in Innominate, a new framework that offers scalable anonymous communication is built. This framework addresses trade-off between performance and strong anonymity. Finally, in DeltaTrack, an efficient backtracking intrusion system which drastically improve the performance of intrusion backtracking has been deployed.
Bibliography


[24] Neo4j - the world’s leading graph database. [http://www.neo4j.org/]


[26] PlanetLab. [http://www.planet-lab.org]

[27] Quantcast. [http://www.quantcast.com/top-sites-1]


[29] RIPE Routing Information Service. [http://www.ripe.net/ris/]


[34] The Tor Project, Inc. [http://www.torproject.org]

[35] Tor. [https://www.torproject.org]

[36] Tor metrics portal: Users. [https://metrics.torproject.org/users.html]

[37] Your Facebook friends have more friends than you. [http://wapo.st/lI58Mj]

[38] The Tor directory protocol. [https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=dir-spec.txt] 2011.


[44] Lars Backstrom, Eric Sun, and Cameron Marlow. Find me if you can: Improving geographical prediction with social and spatial proximity. In WWW.


[65] Xinwen Fu and Zhen Ling. One cell is enough to break tor?s anonymity. In Black Hat, 2009.


[92] Sarah Meiklejohn, Hoyv Shacham, and David Mandell Freeman. Limitations on transformations from composite-order to prime-order groups: The case of round-optimal blind signatures. In ASIACRYPT, 2010.


