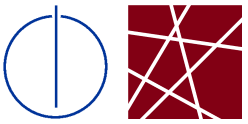
Technische Universität München
Fakultät für Informatik
Lehrstuhl für Netzarchitekturen und Netzdienste



Privacy-Implications of Performance-Based Peer Selection by Onion-Routers: A Real-World Case Study using I2P

Masterarbeit in Informatik

durchgeführt am
Lehrstuhl für Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München

von
Michael Herrmann

März 2011

Technische Universität München
Fakultät für Informatik
Lehrstuhl für Netzarchitekturen und Netzdienste

Auswirkung auf die Anonymität von performanzbasierter Peer-Auswahl bei Onion-Routern: Eine Fallstudie mit I2P

—

Privacy-Implications of Performance-Based Peer Selection by Onion-Routers: A Real-World Case Study using I2P

Masterarbeit in Informatik

durchgeführt am
Lehrstuhl für Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München

von
Michael Herrmann

Aufgabensteller und Betreuer:	Christian Grothoff, PhD (UCLA)
Tag der Abgabe:	29. März 2011

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I assure the single handed composition of this thesis only supported by declared resources.

Garching, den 29 März 2011

Acknowledgment:

This thesis is based on a paper with the same title from Michael Herrmann and Christian Grothoff, published at *Privacy Enhancing Technologies Symposium* (PETS) 2011.

I thank my advisor Christian Grothoff for the possibility to work with him on this thesis and his great support. Not only was working with him very inspiring, he also pushed me to do the best I possibly can. Also I thank Katie Haus and Nathan Evans. Katie for her great work on the pictures in this thesis and her amazing patience to get them exactly as I wanted them to be. Nathan always took the time when I was unsure about something and gladly discussed with me every issue I had with this work.

Thank goes also to my father, Sarah and all my friends. They all greatly supported me through the sometimes tough times of this thesis.

My special thanks go to my grandparents. Without their support and understanding, studying would not have been as easy as it was. Especially I thank my grandfather, who could not live to see the end of this thesis. He always believed in me.

Abstract:

The *Invisible Internet Project* (I2P) is one of the most widely used anonymizing Peer-to-Peer networks on the Internet today. Like Tor, it uses onion routing to build tunnels between peers as the basis for providing anonymous communication channels. Unlike Tor, I2P integrates a range of anonymously hosted services directly with the platform. This thesis presents a new attack on the I2P Peer-to-Peer network, with the goal of determining the identity of peers that are anonymously hosting HTTP (Eepsite) services in the network.

Key design choices made by I2P developers, in particular performance-based peer selection, enable a sophisticated adversary with modest resources to break key security assumptions. Our attack first obtains an estimate of the victim's view of the network. Then, the adversary selectively targets a small number of peers used by the victim with a denial-of-service attack while giving the victim the opportunity to replace those peers with other peers that are controlled by the adversary. Finally, the adversary performs some simple measurements to determine the identity of the peer hosting the service.

This thesis provides the necessary background on I2P, gives details on the attack — including experimental data from measurements against the actual I2P network — and discusses possible solutions.

Kurzfassung:

Das *Invisible Internet Project* (I2P) ist eines der am meisten verbreiteten Peer-to-Peer Anonymisierungsnetzwerke im Internet. Grundlage für anonyme Kommunikationskanäle sind Tunnel, die — ähnlich zu Tor — Onion Routing einsetzen. Im Gegensatz zu Tor bildet I2P ein eigenständiges Netzwerk in dem eine Vielzahl integrierter anonymener Dienste angeboten werden. Diese Arbeit präsentiert einen neuen Angriff auf das I2P Netzwerk mit dem Ziel die Identität eines anonym angebotenen HTTP Dienstes (einer sogenannten Eepsite) herauszufinden.

Nach dem I2P Protokoll wählen sich Peers gegenseitig nicht zufällig, sondern aufgrund beobachteter Performanz aus. Diese Design-Entscheidung erlaubt es einem Angreifer mit moderaten Ressourcen wichtige Sicherheitsannahmen zu brechen. Der Angreifer schätzt zuerst den Blick des Opfers auf das Netzwerk ab. Danach wird er andere I2P Knoten angreifen, von denen er glaubt, dass sie vom Opfer mit überdurchschnittlicher Performanz bewertet wurden. Die Performanz dieser Knoten wird mit einem denial-of-service Angriff signifikant reduziert. Gleichzeitig bietet der Angreifer dem Opfer die Möglichkeit feindliche I2P Knoten mit guter Performanz zu verwenden. Schließlich werden einfache Messungen durchgeführt um die Identität des Opfers zu bestimmen.

Diese Arbeit stellt das nötige Hintergundwissen von I2P vor und beschreibt detailliert die Einzelheiten des vorgeschlagenen Angriffs. Messergebnisse, die vom I2P Netzwerk stammen, werden ebenfalls vorgestellt und erläutert. Zusätzlich diskutiert diese Arbeit mögliche Lösungen um den Angriff zu vermeiden bzw. zu erschweren.

Contents

1. Introduction

An anonymous communication system aims to provide anonymity, which according to [39] is the state of being not identifiable within a set of peers. We also differentiate between *sender-anonymity* and *receiver-anonymity*. In the first case, the sender of a message is not identifiable and in the latter case the receiver. Furthermore the type of communication to anonymize has a big impact on the difficulty to design such a system. *High-latency* anonymity networks are built for applications that do not require a fast response. For instance e-mail communication is a typical representative of a high latency application. If e-mail communication is delayed by a couple of minutes the quality of service is not significantly reduced. On the other hand, if an user is surfing the web or uses instant communication, delaying the communication more than a couple of seconds is not acceptable. Therefore *low-latency* anonymity networks are needed.

One of the established techniques to provide sender- or receiver-anonymity is Onion routing [40]. It is especially applicable for building low-latency anonymous communication systems. Two of the most widely used anonymizing networks, Tor [20] and I2P [57] are using onion routing to achieve anonymity for low-latency applications. Furthermore, both are open, peer-to-peer (P2P) networks. However, there are significant differences in the details of how these networks implement the basic technique. For many of the differences, the existing related work does not provide a clear answer as to which approach is better.

In this work, we report on our exploitations of some of the *design choices* in I2P to deanonymize I2P services, specifically I2P Eepsites.[1] An Eepsite is a website hosted anonymously within the I2P network and accessed via HTTP tunneled through the I2P network, which also acts as an anonymizing SOCKS proxy. Our attack requires a modest amount of resources; the only special requirement, to run I2P peers in several different /16 peers, could be met by any Internet user for example, by using cloud based services. While this requirement may put us outside of the I2P attacker model, our other requirements — participation in the I2P network and a modest amount of bandwidth — are easily within common attacker models for anonymizing P2P networks, including I2P and Tor.

Our attack is primarily based on exploiting I2P's performance-based selection of peers for tunnel construction, I2P's usage of unidirectional tunnels and the fact that Eepsites are located at a static location in the network. Using a combination of peers that participate as monitors in the network and other peers that selectively reduce the performance of certain other peers, our attack deduces with a high degree of certainty the identity of the

[1] Our basic technique could be applied to other kinds of I2P services as well.

peer hosting the targeted Eepsite. In contrast to previous deanonymization attacks (such as [22, 32]), our attack does not primarily rely on congestion-induced changes to latency and is hence able to provide stronger evidence against the victim.

We have evaluated our technique not merely in simulation or a testbed but against the real I2P network. This work presents experimental results obtained in early 2011 using I2P version 0.8.3, modified for our attack.[2]

The main contributions of this thesis are as follows:

- An independent characterization of the I2P protocol
- A novel attack on anonymity based on the heuristic performance-based peer selection for uni-directional tunnels
- Experimental evaluation of the attack
- Recommendations for improving the I2P design to thwart the attack

The rest of this work is structured as follows. In Chapter 2 we give an detailed overview about related work. Chapter 3 provides a detailed overview of the I2P network and gives an brief overview about the differences between Tor and I2P. Our attack is described in Chapter 4 and Chapter 5 presents the experimental results. Finally, we discuss in Chapter 6 possible solutions and relates our attack to previous work on deanonymization for similar systems.

[2]The modified I2P version and additional data can be downloaded from http://i2p.net.in.tum.de/

2. Related Work

Research on anonymous communication systems started in 1981 with Chaum's work [9] on anonymous email transfer. Due the increased importance of digital communication, also an increased interest on anonymous communication was created, generating several techniques to build anonymous communication systems.

The goal of this chapter is to give an overview about this research. The focus is set on literature closely related to this work, namely low latency networks and attacks on them.

In Section 2.1 we give a brief introduction on two basic techniques for building anonymous communication systems. Section 2.2 deals with specific proposals for anonymizing networks. Attacks on the widely used low-latency network Tor are introduced in Section 2.3.

2.1 Basics on Anonymous Communication Systems

In the literature, the two main techniques on how to built anonymizing networks are *mix networks* and *onion routing*. Since mix networks have a minor impact on this work, we only give a brief overview of them in Section 2.1.1. Onion routing is a key concept to achieve anonymous communication. We introduce it in Section 2.1.2.

2.1.1 MIX

A *mix* [9] is a node in an network serving as a proxy. Its goal is to hide the relationship between ingoing and outgoing messages and thereby providing anonymity to its users. Multiple mixes are combined into a *mix network*, in order to increase anonymity. Originally mixes were designed for high-latency communication, such as email communication or online voting.

A mix network can use multiple strategies to hide the relationship between messages going through the network. A message might be transformed cryptographically or might be delayed. Also a mix network might add dummy traffic. Since mix networks are outside the scope of this work, for theoretical works on mix networks the reader is referred to [6, 12, 13, 18, 21, 24, 25, 27, 33, 35, 37, 42, 51] and for anonymous communication networks based on mixes the reader is referred to [5, 8, 16, 17, 43, 45, 47]. Information about attacks on mix networks can be found in [1, 14, 15, 34, 48, 56].

2.1.2 Onion Routing

Onion Routing [40] is a technique to hide a peer seeking for anonymity behind a set of peers. In Figure 2.1 Onion Routing with three other peers is shown. After a buildup phase, node S sends a message with layers of encryption — thus the name *onion* — down the path. Every layer is encrypted with a key only the desired receiver knows. After the receiver decrypted the message he finds information on where to send the message next. As in Figure 2.1 shown, the sender S sends a message to node n_1, which has been encrypted with a secret only S and n_1 share. Additionally, after n_1 removed one layer of encryption it finds the information to forward the message to node n_2. When such a message receives its destination, i.e. no layers of encryption are left, receiver R evaluates the message.

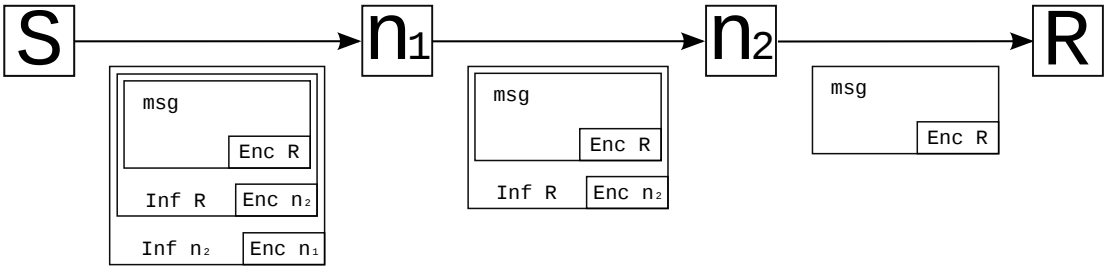


Figure 2.1: Onion routing.

Advantages of this approach are that no intermediary node is able to learn the information of the message, and through the embedded routing information, it is assured that only nodes who are supposed to participate know how to forward the package. Finally, with the exception of the sender, every node on the path only knows its predecessor and successor. Consequently, the originator of a message remains anonymous.

2.1.3 Garlic Routing

Garlic routing [19], shown in (Figure 2.2), is a variation of onion routing. The key difference is that the creator of a garlic can combine multiple messages, which can themselves be garlic encrypted, when creating a "clove". After the corresponding decryption step the decrypting peer, finding multiple messages, can either find another layer of encryption and forward the encapsulated messages to the next targets or find fully decrypted messages and process those as specified by the protocol.

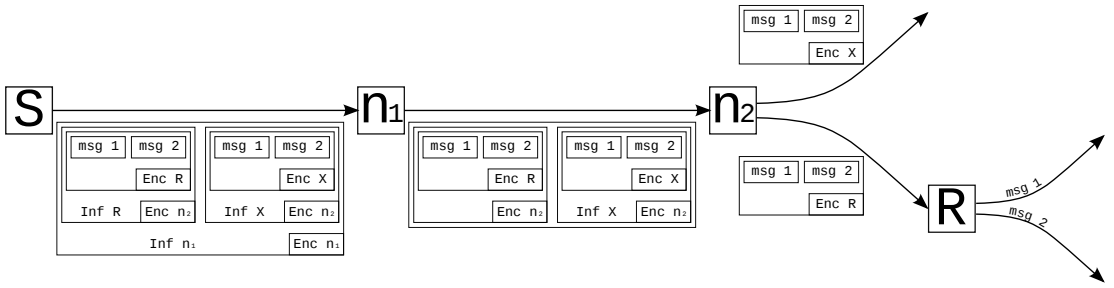


Figure 2.2: Garlic routing

2.2 Anonymizing Networks

In this section, we introduce systems proposed to achieve anonymity. These systems can be separated in two categories. One category, we introduce in Section 2.2.1, is proposed

to offer anonymous and censorship-resistant data exchange. The other category, which is introduced in Section 2.2.2, is low latency networks. These networks aim to offer fast low latency on anonymous communication and thus they are aplicable for applications like web-browsing or instant messaging.

2.2.1 Anonymous Storage

Systems providing anonymous storage, aim to enable people to store and exchange information censorship-resistant on the Internet. This means, that if a person stores or reads information in the network, a third party should not be able to identify the person. All systems have the two basic operations *storing* and *reading* in common.

The first work on this topic was published by Anderson in his *Eternity Service* [2]. Its goal was merely to store information on different computers accessible via the Internet, thereby ensuring that it is impossible to completely delete a particular piece of information. The system itself is not designed to ensure anonymity or to use cryptography. In [4], Benes proposed *The Strong Eternity Service* as a advancement of the original Eternity Service. Advancements of this work are the establishment of persistent pseudonyms for the users and to provide a fully distributed name space. Also, for storing and requesting data, mix networks are used, in order to provide anonymity.

In [44] another censorship resistant system is proposed. The key idea of the system is to add a level of indirection between the nodes storing data and the nodes visible in the network.

The Free Haven Project [19] is a design for a system of anonymous storage. Several anonymity definitions, a censorship-resistant system should meet, are provided:

Publisher Anonymity It is impossible for an adversary to link a publisher to a document.

Reader Anonymity An Adversary cannot determine which other users reads a certain document.

Server Anonymity It is impossible for an adversary to determine the servers storing a certain document.

Document Anonymity A server does not know which documents it is storing.

To request previously stored data, a node performs a broadcast. Furthermore a reputation system is sketched. Every server stores about every other server two values: reputation and credibility. Reputation represents the belief on how the other server will obey the protocol. Credibility represents a belief on how valuable the information of the other server are. Some of the suggestions of the paper have been implemented, in order to research their impact and effectiveness. However, to the best of our knowledge, this project is currently under no active development.

In [52] another attempt of a censorship-resistant network named *Publius* is presented. There also exists a software version from 2001[1]. Unlike the Free Haven Project, a user is able to update or revoke previously published documents. Naturally, this leads to security risks, because an attacker might exploit these mechanisms and delete or tamper with a document.

[1] http://www.cs.nyu.edu/~waldman/publius/software_download.html

All already mentioned suggestions on censorship-resistant storage systems have either not been implemented or only a prototype was developed. There are also systems, that have been implemented and that are in use today.

StealthNet[2] aims to provide anonymous communication with acceptable download rates. In MUTE[3] and ANts P2p[4], file exchange partners do not have any direct contact, but the communication happens encrypted via intermediary peers.

Freenet is a peer-to-peer information storage and retrieval system that provides sender and receiver anonymity. The key concept of Freenet is *plausible deniability* achieved through the routing algorithm. For both, storing and requesting data, requests are transmitted hop by hop, thus the next hop never knows if the predecessor was the initiator or an intermediary node. Furthermore, Freenet uses a closeness measure to achieve that similar content gets stored on close nodes.

In general GNUnet is a framework for secure peer-to-peer communication. We list it in this section, because at its current state, it provides censorship-resistant file-sharing. Similar to Freenet, requests in GNUnet are transferred hop by hop through the network. If the file matching the request is available, the reply is routed back on the same path. Peers choose next hops based on their closeness to the target and their observed performance (hot paths). In GNUnet requests are prioritized and peers monitor each other in order to reward better performing peers also with a better service. Another key design choice is a mechanism called *shortcut feature*. According to this feature, peers react different on requests based on their current load, in order to achieve better network performance. If a peer has resources left it follows the protocol. If the resources are exhausted, it simply forwards the request or may even drop it. This means that if a peer A sends a request to the peer B and B has resources left, B replaces contact information of A with its own and contacts the next peer C. If B does not have enough resources, it just forwards the traffic to C, whereby C learns that the predecessor of B was A.

Achieving anonymity in information exchange comes with a lot overhead and thereby quality of service is significantly decreased. *OneSwarm* was proposed to mitigate this problem. It aims to provide anonymous information sharing with a low performance overhead. A user can define several anonymity levels for its published files. A piece of information can be publicly available, thereby OneSwarm behaves like BitTorrent [10]. A piece of information can also be configured with permissions, which means that this piece of information is only accessible for a well defined number of peers. These peers built the swarm to exchange the information under each other. The last possible configuration is to define the information *without attribution*. In that case, data is relayed through a unknown number of peers, thus achieving sender and receiver anonymity.

2.2.2 Anonymous Communication Systems

Unlike systems for anonymous storage, there have been several proposed networks to anonymize real-time Internet communication, like web-browsing or instant messaging. Most proposals were analyzed and simulated, but, to the best of our knowledge, most were never widely deployed. In the following we give a brief overview of these networks.

The first attempt to achieve anonymity for HTTP communication was done by *Anonymizer*[5]. Clients do not request a web page directly, moreover they send their requests to the single anonymizer proxy. This proxy requests the web page on behalf of the original requester

[2] http://www.stealthnet.de/
[3] http://mute-net.sourceforge.net/
[4] http://antsp2p.sourceforge.net/
[5] http://www.anonymizer.com

and sends the reply back. Thereby the IP address of the requester is hidden from the web server. Obviously, the user has to trust the proxy provider.

In order to remove this single instance of trust Crowds [49] tries to hide the requester behind a crowd of other peers, that are also participating in the network. An original requester sends the request to an arbitrary other peer. This peer flips a coin and based on the result either forwards the request to another peer or does the actual HTTP request. However, a drawback of this attempt is that a passive adversary might still be able to determine the originator of the request.

P5 [46] uses cover traffic to make statistical analysis by a passive adversary infeasible.

A system using multicast routing is proposed with *Hordes* [28]. The idea is to construct the forward path as with Crowds, but for the return path to send the message to a multicast group, instead to a single computer. With this strategy the authors develop an anonymous communication system with better performance but with less anonymity than Onion Routing. A drawback of this approach is that every peer has to know every other peer in the network.

SAS [54] is an approach to let the user decide about the tradeoff between anonymity and quality of service he wants to use by defining an *anonymity level*. This level determines the type and the number of nodes on a message's path. Nodes with lower levels will use fewer hops to reach the target than nodes with higher levels. Based on its anonymity level, a node is also assigned a *load level*. The higher the load level, the more likely it is that this node is picked as hop by another node sending a message.

In *Tor* [20], each peer retrieves a list of candidate mixes from a set of hard-coded *directory servers*. Every peer can introduce itself to the directory server, provide state information and become a Onion Router (OR). Based on this information, the directory servers achieve an overview about the current state of the network and spread OR information around. The path through the mix network in Tor is called a circuit, which by default consists of three ORs. The first node in the circuit is particularly important for privacy and is called a guard node. A peer in the Tor network tries to stick with a small set of guard nodes. The last node in the circuit is typically an exit node which allows the user to establish a TCP connection to the Internet. Exit policies are used to describe which TCP connections a particular exit node is willing to support and Tor has to respect these policies when constructing a circuit for a particular application. The middle node is selected at random from the list of available peers, but biased towards peers with high bandwidth.

Since all ORs can be learned from the Tor network, a network provider could prevent computers from connecting to any OR. Consequently, people could be stopped from using Tor. To overcome this problem, Tor introduced in December 2007 the *bridge concept*. A *bridge* is a Tor peer, that voluntarily participates in circuits and unlike ORs is not listed in the Director Server. Bridges can be learned from the *bridge authority*. To avoid that all bridges can be learned by a single attacker, peers only can learn bridges from the same /24 subnet.

Tor also allows users to anonymously host so-called *hidden services* within the Tor network. Here, the peer providing the hidden service creates a circuit to an *introduction point*, a peer in the Tor network that serves as a point of contact to the hidden service. Next, information about the introduction point is published in the Tor DHT. Another peer can then contact the introduction points with information about a *rendezvous point*, a peer acting as the crossover-node between two circuits. Finally, both peers communicate anonymously via two circuits to the rendezvous point.

Tarzan [23] is a completely peer-to-peer based anonymous communication network, that — in contrast to Tor — does without special nodes. Tarzan differs in two ways to all

other proposals. At first a peer connects — and thereby communicates — to only a few particular peers, such called *mimics*. Secondly, to address the threat of an global passice adversary, cover traffic is used on the connections to these mimics. Another characteristic of Tarzan is that it assumes every peer knows virtually every other peer. This is achieved by the *gossip protocol*. In this protocol a new peer learns from existing peers about all other peers in the network. To confirm that every learned peer actually exists, every peer is contacted once.

In comparison to Tor, *SHALON* [36] is another network using onion routing. By abdication of link layer encryption, SHALON slightly reduces the level of anonymity in order to increase performance. Furthermore the network is completely based on standardized protocols, like HTTP and SSL.

The *Invisible Internet Project* (I2P) [57] is in the focus of this work. In Section 3 we will give a detailed explanation of this network.

2.3 Attacks on Tor

Tor is a widely used anonymous communication system and is thus the dominant target of attacks. The goal of Tor is to be secure against a locally active attacker, who controls some peers in the network that might not follow the Tor protocol. This is appropriate, because to control most of the network, an attacker has to be extremely strong. Also it is easy to see, that such a strong attacker — or an attacker able to observe most of the traffic — can deanonymize Tor. In this section we give an overview about published and implemented attacks on Tor. All of these attacks use a locally active attacker and some of them have also an impact on other anonymous communication systems.

In [32] a *low cost traffic analysis attack* on Tor is presented. The authors exploit the fact, that Tor ORs serve a number of Tor peers in a round-robin fashion. Since ORs have only limited resources, they delay traffic if they are confronted with high load. This delay can be measured by a malicious Tor. Furthermore this delay can be increased by causing traffic on a particular OR. In the attack, the authors assume that a Tor peer, seeking anonymity, requests data from a malicious server. The server responds in data bursts, instead of steady data flow, thereby increasing the load on ORs participating in the victims circuit. Consequently, this results in an increase of the delay time and the three Onion Routers being used can be identified, which reduces the anonymity of Tor. Note that this attack has been deployed in a lightly loaded Tor network with only 13 ORs. As shown in [22], the attack is no longer practical in today's Tor network.

The authors of [32] state, that the attack found on Tor probably works for any low latency anonymous communication network. In [53] it is shown, that the descibed attack is not applicable for Tarzan and *MorphMix* [41]. For the latter, this attack is not applicable, because the malicious peer, measuring the latency of the other nodes, cannot know all MorphMix peers. For Tarzan it is not clear if this attack might work. According to the gossip protocol, a corrupt Tarzan peer can be assumed to know all other peers in the network. The mimics concept in Tarzan influences the attack in two ways. At first cover traffic between mimics hides communication pattern and thereby the delay of nodes might not change. Consequently, the delay of a node might not change even if it is involved in the response path. The authors state, that the only way to ensure this point is to measure in the actual Tarzan network. Secondly, since a node only connects to its mimics, it is questionable if a corrupt Tarzan peer can connect to every other peer, in order to measure it.

The attack in [3] deanonymizes Tor peers, while they are building circuits. Naturally, if a peer builds a circuit with three collaborating peers, the initiator can be identified.

The attack aims to reduce the probability for that scenario in two steps. At first, it is not necessary that all three peers in the curcuit are attacker peers; the guard and exit node are sufficient. The middle node is not necessary because the timing information of the circuit builtup can be observed and the attacker can recognize ORs. Secondly, the attacker exploits, that the performance claims of peers, sent to the directory servers, are not being verified. Thus attacker peers are able to claim having lots of resources, which causes other peers to use them more frequently for circuit building.

In [31] the authors attack Hidden Services of Tor. The attack assumes an attacker who knows several computers, that might host a hidden service in Tor. To verify which computer is the actual host, the authors stress a hidden service by continually requesting a 10 MB file from the hidden service. After that, for another two hours no requests are performed. By this experiment the attackers first — due to stressing the host — heats the CPU up after which the CPU cools down. While continually requesting TCP timestamps [26] of the suspects outside the Tor network, a clock skew, caused by the temperature change of the CPU, is identifiable and thereby the host of the hidden service.

The attacks in [55] also uses clock skews to deanonymize Hidden Services. Unlike the attack in [31], the new attacks do not need the attacker to stress the possible hosts. Basic idea of all new proposed attacks is to also measure the clock skew on timestamps requested in the Tor network. Note that TCP timestamps are not available in the Tor network, since TCP headers are modified on the circuit. This only leaves the usage of HTTP timestamps, which have a lower resolution and thus being less accurate. In the first new attack, the change of the clock skew — every computer has — is measured inside network from the Hidden Service and outside of the Tor network of all possible candidates over some time. The clock skew pattern obtained from outside the Tor network that is most similar to the clock skew pattern of the Hidden Service reveals the identity of the host. The second attack is a faster version of the first one. The attack exploits that the difference of clock skews on different machines is significant.

As already mentioned, the low cost traffic analysis attack in [32] is in today's Tor network is no longer practical. Another congestion attack, that works on today's Tor network, is proposed in [22]. For the attack three Tor features are crucial. At first ORs do not delay data packages, thus latency of OR is observable. Secondly, an attacker is able to learn about all ORs in the network, because they can be requested from the Directory Servers. Finally, genuine Tor peers build circuits of length three, but the Tor protocol does not constrain the length of a circuit, enabling an attacker to build circuits of arbitrary length. One drawback of the low cost traffic analysis attack in [32] is that an attacker has to measure all ORs simultaneously, which is impractical for a large number ORs. The proposed attack in [22] injects a piece of JavaScript on the web browser of the victim that periodically requests a malicious web site outside of the Tor network. In every request, the script adds a timestamp and thereby the attacker can detect possible congestion on one of the ORs in the circuit. To actually induce congestion to a possible circuit participant the attacker builds a long circuit whose peers are iterations of the same three ORs over and over again. This has an excessive impact on the number of circuits an OR has to deal with, since every time an OR receives a circuit request, it naturally assumes that it does not participate already in this circuit. If the attacker now induces traffic in a long circuit, every OR in the circuit has to transfer it multiple times. This increases the load on the OR and thereby the attacker induces delay. If an attacked OR is used as a circuit of the victim, this delay is traceable in the periodic HTTP requests.

An attack using Tor bridges is introduced in [30]. The attack deanonymizes a user running a bridge and accesses a pseudoanonymous web site (for example a forum) through the Tor network. Crucial for the attack is the fact that a user, whenever online in the Tor network,

also serves as a bridge. At first the proposed attack circumvents the restriction that one user only is allowed to learn a small set of bridges. Actually all ORs are supposed to be treated with one virtual IP address, because otherwise an attacker can build circuits with different ORs and thereby send requests from different /24 networks. Unfortunately this was not the case and so the authors were able to learn many more bridges. In the next step of the attack, the online times of the bridges are compared with the pseudoanonymous activities on the web. Consequently, a patient attacker is able to track down the bridge that is always online during the pseudoanonymous activities on the web.

Low latency networks like Tor are vulnerable to two latency attacks proposed in [50]. In the first attack called *passive linkability attack* a malicious server measures the latency of the circuit, by causing the victim's browser to establish many additional connections to the malicious server. For every connection, the exit node has first to establish a TCP session, after which the victim is being notified that the session is established. On a connection established signal, the victim sends a HTTP GET request. The server measures the time difference between the TCP ACK signal and the time the HTTP GET request is received, as the latency of the circuit. The more similar these round trip times are, the more likely it is, that two Tor users used the same circuit. For the second attack, the authors exploit that subnetworks have distinct latencies. If the attacker possesses distinct latency measures from all possible routable IP address prefixes, the measured latency of the victim can be compared with them. Thus the uncertainty of the victim's position can be reduced.

3. Background: I2P

I2P is a multi-application framework for anonymous P2P networking written in Java. Figure 3.1 gives an overview of the I2P architecture.

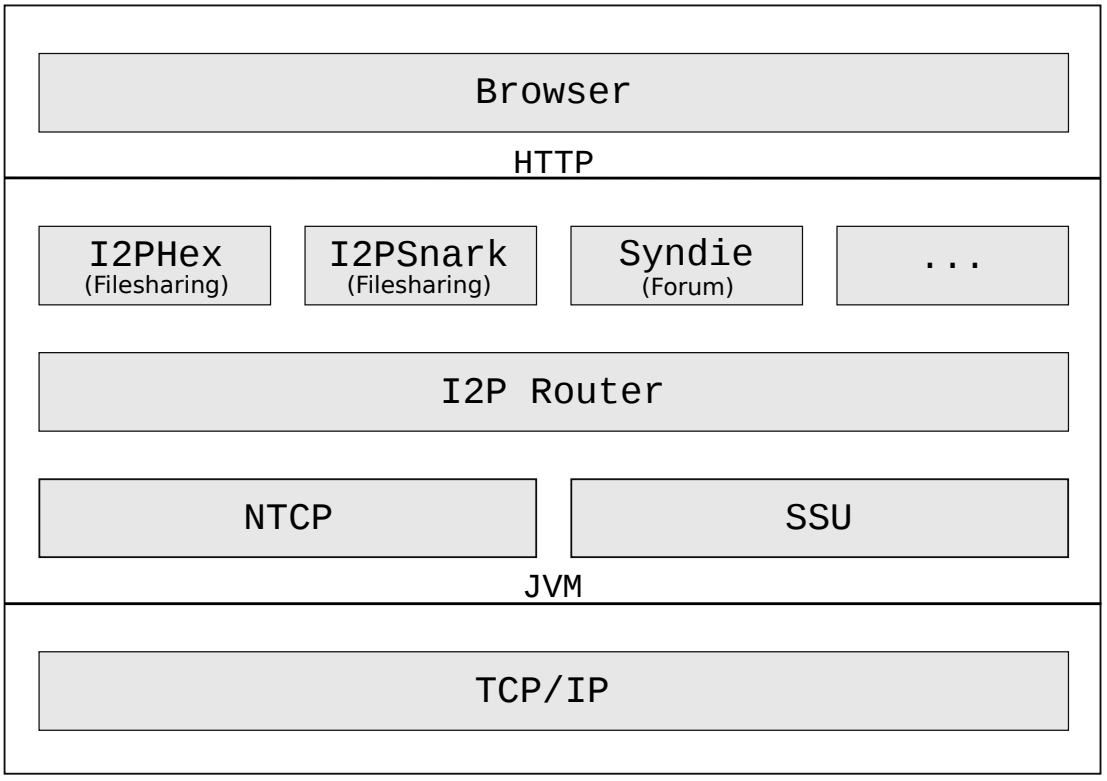


Figure 3.1: Scheme of the I2P architecture. On top of TCP/IP, I2P provides two connection oriented protocols NTCP and UDP. The I2P Router is the core block of the architecture. On top of the router there are several applications, which are usually configured by a web browser interface.

On top of the native Internet protocol, I2P specifies the use of two different peer-to-peer transport protocols. The first is called *NIO-based TCP* (NTCP), where NIO refers to the Java New I/O library. It extends basic TCP for key exchange and encrypted com-

munication. The second is called *Secure Semireliable UDP* (SSU), providing UDP-based message transfer. SSU extends the basic UDP protocol and provides areliable, encrypted, connection-oriented, point-to-point data transport. Generally I2P prefers NTCP to SSU, but if one protocol does not work, the other one is used.[1]

The core of the I2P framework is the I2P router, which implements key components of the I2P protocol. Tasks of the I2P router include: maintaining peer statistics, performing encryption/decryption, building tunnels and sending/receiving provided by NTCP or SSU. Tunnels are the key abstraction of I2P to achieve anonymity. They are composed of several peers and implement garlic routing as introduced in Section 2.1.2. I2P differentiates between four types of tunnels, based on their direction and their required performance. *Inbound tunnels* are for communication from the I2P network towards the tunnel owner and *outbound tunnels* are for the opposite direction. An inbound and outbound tunnel can be either an *exploratory tunnel* or a *client tunnel*, hence the four different tunnel types. Exploratory tunnels are used by the I2P router itself and client tunnels are used by I2P applications. A detailed explanation on tunnels is given in Section 3.2. I2P applications rely on the anonymizing tunnels provided by the I2P router for privacy protection; consequently, the I2P router is central to the security of all I2P applications and the analysis presented in this thesis.

Many Internet applications can be implemented on top of the I2P router. An application provided by a particular I2P peer is referred as a service. For example, I2P includes services to host HTTP servers, to provide IRC-based communication and to perform POP/SMTP-based email transfer. An extensive list of services currently available for I2P is given in Table 3.1. Note that all services are provided anonymously inside the I2P network. Most I2P services are controlled and used via a web browser interface.

3.1 Peer and Service Discovery

Like most other P2P networks, I2P has to deal with the problem of finding peers and subsequently the services offered by those peers. Every peer in the I2P network is uniquely identified by a data structure called *routerInfo*. This data structure holds all the key information about the peer, including public keys of the peer, a 256 bit hash-identifier and information about how the peer can be contacted.

I2P addresses the bootstrapping problem, the problem of initially discovering some other peers in the network, by using a non-anonymous HTTP download of a list of routerInfos for available I2P peers from a fixed location [11].

3.1.1 I2P's DHT: the netDB

After bootstrapping, I2P uses a super-peer DHT to build a network database, called the *netDB*, with information about all the peers and services available in the network. The super-peers that maintain this database are called *floodfill* peers; each floodfill peer is responsible for the information closest to its ID. Proximity is determined using Kademlia's XOR distance metric [29]. If a peer has sufficient bandwidth and its configuration allows it, a peer can promote itself to floodfill status and will do so as soon as the number of active floodfill peers in the network drops below a certain threshold.

3.1.2 Storing data in the netDB

Information about how to contact a service provided by an I2P peer is stored in the netDB in so-called *leaseSet*s. A leaseSet primarily specifies a set of entry points (called *leases*) to

Table 3.1: Services build into I2P

Anonymous service	Description
Eepsites	HTTP servers provided by I2P peers based on Jetty[2]
susidns	Address book for mappings from Eepsites to identifiers
BOB	API to connect arbitrary application to the I2P network
I2PSnark	As web application integrated bittorrent client
Robert	I2P bittorrent client using BOB
i2p-bt	Command-line based bittorrent client
Transmission for I2P	Port of the bittorrent client Transmission to I2P
I2Phex	Client for creating and joining Gnutella based peer-to-peer networks
iMule	Filesharing program based on aMule [3]
Susimail	E-mail web application to access an e-mail server voluntarily provided in the I2P network
I2P-Bote	Decentral e-mail communication system
I2P-Messenger	Instant messaging system for I2P
Syndiemedia (Syndie)	Blogging tool

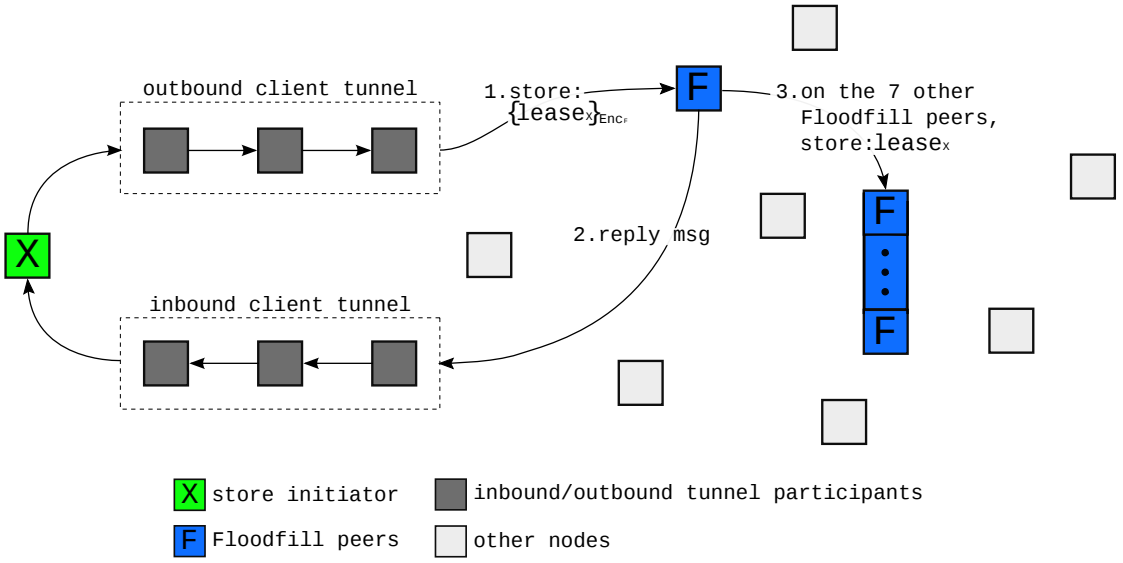


Figure 3.2: I2P uses tunnels to store a lease in the floodfill database to hide the identity of the (HTTP) server.

the service. An entry point is the identification of an inbound tunnel at a peer currently serving as an inbound gateway to the service.

The lookup and storage of leaseSets and routerInfos is achieved by sending the respective requests to a floodfill server. Figure 3.2 illustrates the storage process for a leaseSet. Figure 3.3 shows the storage process of a routerInfo.

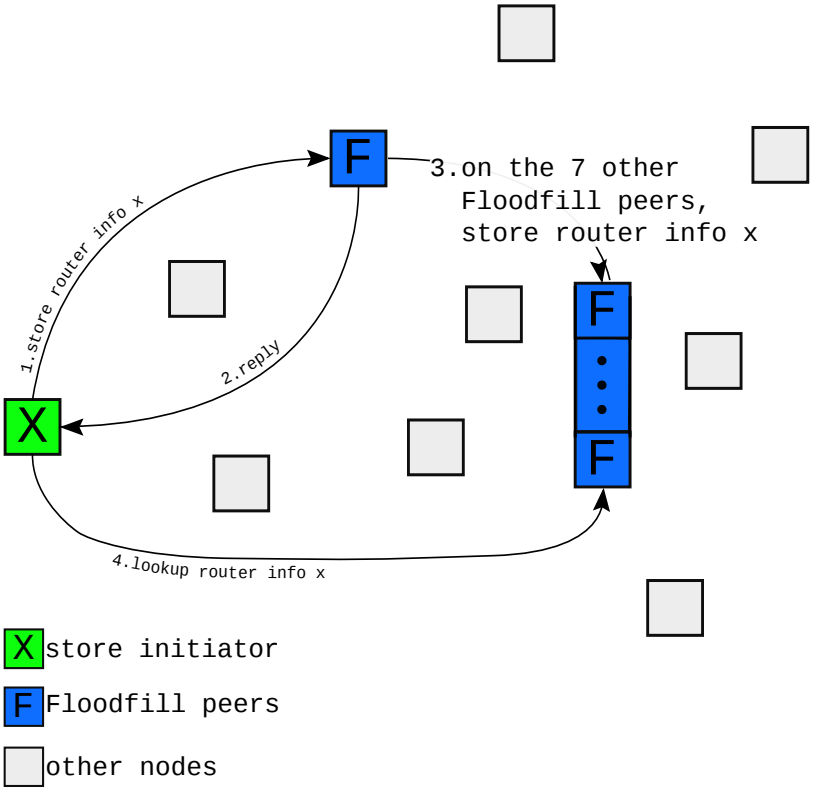


Figure 3.3: I2P stores routerInfos directly in the floodfill database.

The only difference between storing a routerInfo and a leaseSet is that in the latter case communication happens via a (client) tunnel in order to anonymize the peer offering the service. The information of which routerInfo belongs to which machine is not a goal of anonymization and is thus not sent through a tunnel. In both cases, after a floodfill peer receives a request, it replicates the received information at seven additional closest floodfill peers and sends a confirmation back to the initiator. In case of confirming a leaseSet, the floodfill peer sends the confirmation back via the inbound tunnel.

Finally, after 10 seconds, the initiator performs a test lookup for the previously stored data structure at another floodfill peer. The store operation is repeated if the test fails.

3.1.3 Retrieving Data from the netDB

Retrieving routerInfos and leaseSets are performed via exploratory outbound tunnels. The request is transmitted to the two — with respect to the destination address — closest floodfill peers known to the requester. If a floodfill peer does not have the requested information, a list of other close floodfill peers is sent back. The replies are transmitted to the initiator using an exploratory inbound tunnel. If both floodfill peers do not have the requested information, the requesting peer queries two other floodfill peers until all known floodfill peers are contacted. An example for a lookup process in the I2P network is given in Figure 3.4.

[1]For a detailed discussion on the preference of NTCP and SSU, please see http://www.i2p2.de/ntcp_discussion.html

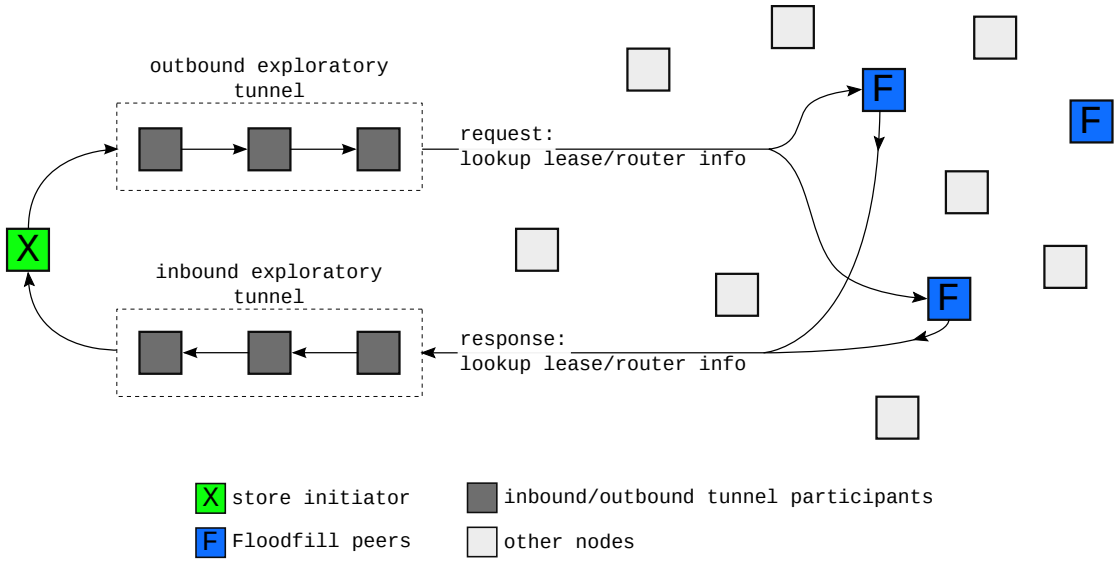


Figure 3.4: Looking up a Router Info or Lease using exploratory tunnels.

3.2 I2P Tunnels

I2P uses *tunnels* to hide the IP address of a participant in an online interaction. I2P tunnels closely resemble onion routing as implemented in Tor with circuits [20]: the initiator selects the route through the network, no artificial delays are introduced when forwarding, and link- and layered-encryption are used to protect the data against observers.

3.2.1 I2P Tunnels are Unidirectional

Tunnels in I2P only transfer payload data in one direction. In order to achieve bi-directional communication, I2P uses *inbound* and *outbound* tunnels. Inbound tunnels are used to transmit data to the peer that constructed the tunnel and outbound tunnels are used to transfer data from the peer that constructed the tunnel. Note that only the peer that constructed the tunnel knows all of the peers in the tunnel. Figure 3.5 shows an example of inbound and outbound tunnel message flow.

For outbound tunnels, multiple layers of encryption are added by the creator of a message; each one is then removed by the corresponding peer as the message traverses the outbound tunnel.

For inbound tunnels, adding all layers of encryption at the first peer is not possible; this would require the first inbound node to know the secret tunnel keys for all of the participants of the tunnel. Instead, every node in an inbound tunnel *adds* an additional layer of encryption. Finally, the creator of the tunnel, who knows the tunnel keys used by each peer from the tunnel construction phase, removes all layers of encryption to obtain the original message.

3.2.2 Tunnel Diversity

Every I2P peer creates multiple tunnels; the specific number of tunnels and the tunnel length depend on the peer configuration. The length of the tunnel is considered to be a trade-off between speed and anonymity and I2P gives the end-user control over this setting. The user specifies two non-negative numbers, x and y. For each tunnel, I2P selects a random number $r \in [-y, y]$ and constructs a tunnel of length $\max(x + r, 0)$.

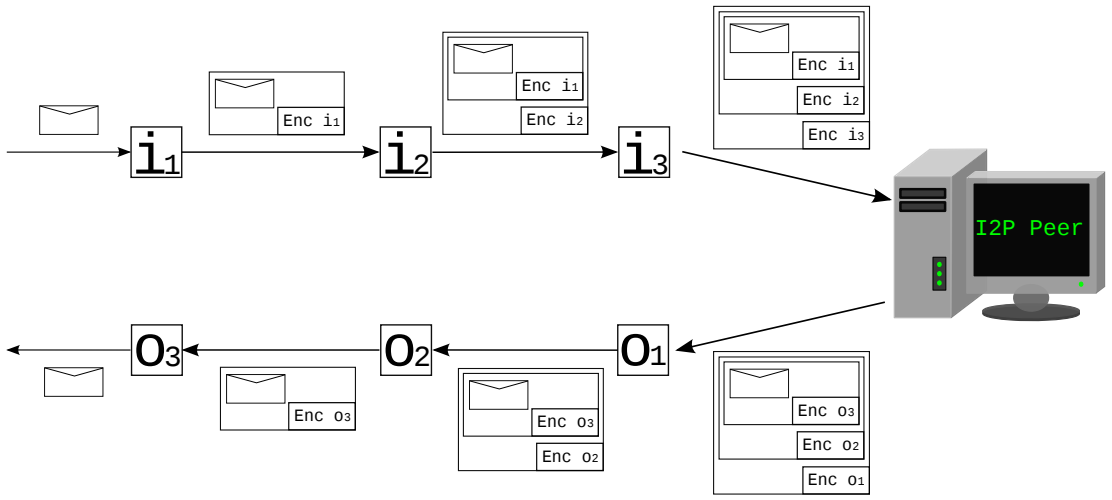


Figure 3.5: Message flow for an inbound and an outbound tunnel of an I2P peer. For the inbound direction, layers of encryption are added while the message moves towards the receiver. For the outbound direction, layers of encryption are removed.

In addition to the distinction between inbound and outbound tunnels based on the tunnel's transfer direction, I2P further distinguishes between exploratory and client tunnels. Exploratory tunnels are for requesting data from the netDB and for tunnel management. They are not used for application operations. Client tunnels are used for all typical application level network messages, like leases lookup, to anonymize requests from clients and to provide inbound/outbound tunnels for I2P services, such as Eepsites.

3.2.3 Tunnel Construction

In order to select peers for tunnel construction, I2P first categorizes all known peers into tiers. Depending on the type of tunnel that is being created, the peer selection algorithm then attempts to select peers exclusively from a particular tier. In addition to selecting peers from particular tiers, I2P also avoids the selection of multiple peers from the same /16 (IPv4) network for the same tunnel.

After selecting peers for the tunnel, the initiator sends tunnel construction requests (via that partially built tunnel) to the selected peers. A peer receiving a tunnel construction request is free to either accept to participate in the tunnel or to reject the request, indicating a reason for the refusal. Naturally, tunnels can still fail if peers that accepted a tunnel construction request are later unable to sustain the tunnel. The behavior of a peer faced with tunnel construction requests (including the reason given for rejection) as well as tunnel failures is important for the performance evaluation of peers, which is used for assigning peers to tiers.

3.2.4 Tier-based Peer Selection

An I2P peer chooses other peers randomly from a particular tier depending on the type of the tunnel. A tier consists of peers that share certain performance characteristics. I2P categorizes peers into four tiers:

Fast Peers with high throughput

High-capacity Peers that will accept a tunnel request with high probability.

Well-integrated Peers that claim to know many other peers

Not-failing All known peers

Every (known) peer is always considered to be in the "not-failing" tier; peers can additionally be in the high-capacity and well-integrated tiers. Peers must be in the high-capacity tier to be eligible for the fast tier.

The fast tier is considered the most valuable tier and is used for constructing client tunnels. In the theoretical case where the fast tier does not have a sufficient number of peers, I2P falls back to using peers from the high-capacity (or even well-integrated and not-failing) tiers for peer selection in the construction of client tunnels. In practice, we were unable to observe this behavior since the fast tier was always sufficiently populated during our evaluation.

The high-capacity tier is the default choice for exploratory tunnels, the well-integrated and non-failing tiers are only used as fallback options (also unlikely in practice).

Peers are placed into tiers based on certain performance metrics. A peer is put in a particular tier if its corresponding performance value exceeds a threshold calculated by I2P for that tier.[4] The size of the fast and high-capacity tiers is bounded. For the fast tier the number of peers is between 8 and 30 and for the high-capacity tier between 10 and 75. If the number of peers in those tiers drops below the threshold, the best-performing peers from lower tiers are promoted. If the number of peers in a tier exceeds the upper limit the lowest rated peers are demoted.

The I2P router keeps track of various performance statistics in order to sort peers into the correct tiers. A flow chart diagram of the tier placing algorithm is given in Figure 3.6. Performance metrics are gathered more often for peers in the fast and high-capacity tiers, since performance metrics are always gathered if a peer is used for a tunnel. Furthermore, performance scores are cumulative; this generally results in higher performance values for peers in the fast and high-capacity tiers and reduces fluctuation.

3.2.5 Metrics for Tier Assignment

I2P is careful about only including performance metrics that are hard to manipulate, relying only on measurements entirely controlled by the peer for throughput and tunnel maintenance properties. In particular, information about tunnels created by other peers is not taken into consideration.

The *capacity value* of a peer is based on the number of times the peer accepts a tunnel request, the number of tunnel rejections and the number of tunnel failures that happen after a peer accepted to participate in a tunnel. The actual capacity calculation of the current I2P version differs from the only I2P design paper authored by members of the I2P community [57].

The goal of the capacity calculation is to estimate how a peer is likely to behave in the future in terms of its participation in tunnels. The calculation is primarily based on the accept, reject and failure actions of that peer. Furthermore, if the peer rejected events in the last 5 minutes, the reason given for the rejection is also considered. A detailed description of the capacity calculation algorithm can be found in Appendix A.1; the main point for this thesis is that peers accepting tunnel requests score high, peers rejecting tunnel requests score low and peers failing tunnel requests score very low in terms of their capacity value.

[4] The complex threshold calculation is described in detail in Appendix A.3 since these details are not necessary for the understanding of this work.

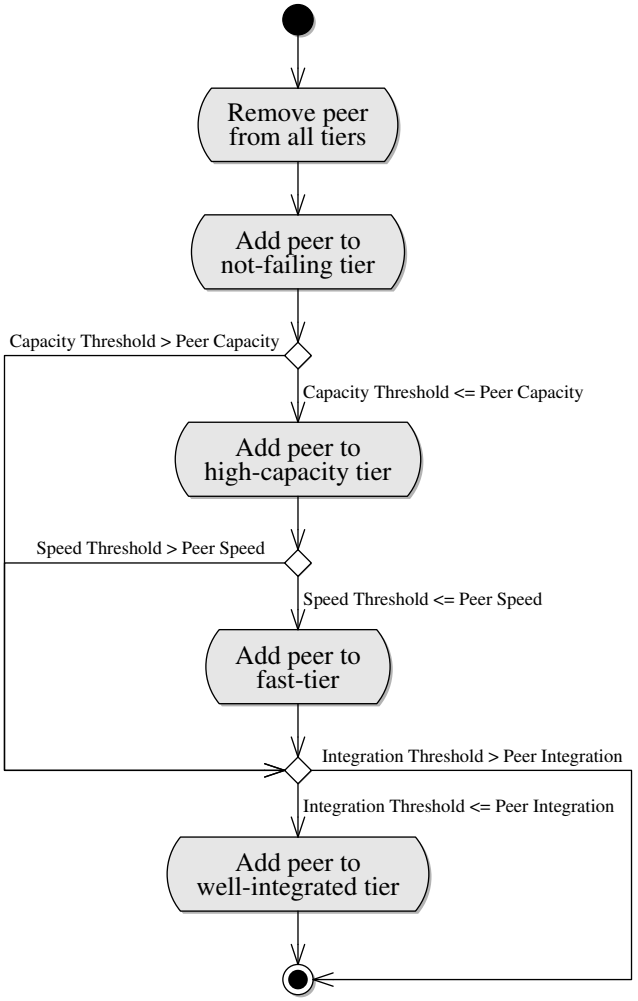


Figure 3.6: Flow chart diagram of the peer placement algorithm. A peer is always in the not-failing tier. To be in the high-capacity or well-integrated tier, its corresponding performance value must exceed the threshold. For the fast-tier, a peers speed value must also exceed the speed threshold and the peer has also to be in the high capacity tier.

A peer's *speed value* is the mean of its three highest, one second throughput measurements in any tunnel established by the measuring peer over the course of the last day. Throughput is measured whenever data is sent through a peer via a tunnel created by the measuring peer. Naturally, throughput is bounded by the throughput capacity of the measuring peer as well as, for each individual measurement, the slowest peer in the tunnel. While it would be nice to be able to influence speed values of other peers, the fact that I2P uses the observed maximum over an entire day makes this unattractive: attacking a peer to reduce its speed for a whole day is simply too expensive.

The *integration value* of a peer is a measure of how well a peer is integrated in the network. To accomplish this, the I2P router keeps track of how many new peers it has learned from the particular peer. The integration value is not relevant to our attack; details for how it is calculated are presented in Appendix A.2.

3.3 Eepsites

The I2P software comes with the *Jetty* web server. Using Jetty, every I2P user can offer HTTP web pages to the I2P network using a domain under the *.i2p* TLD. Given such a domain name, I2P creates inbound and outbound client tunnels for the service and (periodically) publishes a leaseSet in the netDB.

When a user decides to create a new domain, a 516 character Base64 encoded identifier based on a public key and a signing key, is created locally. The creator of the Eepsite can share this identifier with any other I2P user to provide access to the service. This mechanism requires, similar to an IPv6 Internet without DNS, the memorization and sharing of a cryptic set of numbers.

To overcome this issue, the local I2P application *addressbook* can be used to store a mapping of pet domain names and corresponding identifiers, similar to the `/etc/hosts` file on UNIX systems. With this, an I2P user can contact the web server by typing the pet domain name in its browser. A canonical `hosts` file is shipped with the I2P distribution.

Accessing an Eepsite involves several steps (illustrated in Figure 3.7):

1. The Eepsite host (server) creates inbound and outbound tunnels for sender-anonymity and publishes gateway information as a leaseSet in the netDB (as described in Section 3.1). Fresh tunnels and corresponding leaseSet updates are published at least every 10 minutes.
2. The peer running the HTTP client (client) uses a tunnel to access the netDB and retrieves the leaseSet information.
3. The client uses inbound and outbound tunnels (for receiver-anonymity) to contact the gateways from the leaseSet.
4. A handshake is performed via the tunnels for end-to-end encryption between server and client, using the public key in the leaseSet.
5. The HTTP request is transmitted through the outbound tunnel of the client and the inbound tunnel of the server.
6. The HTTP response is transmitted through the outbound tunnel of the server and the inbound tunnel of the client.

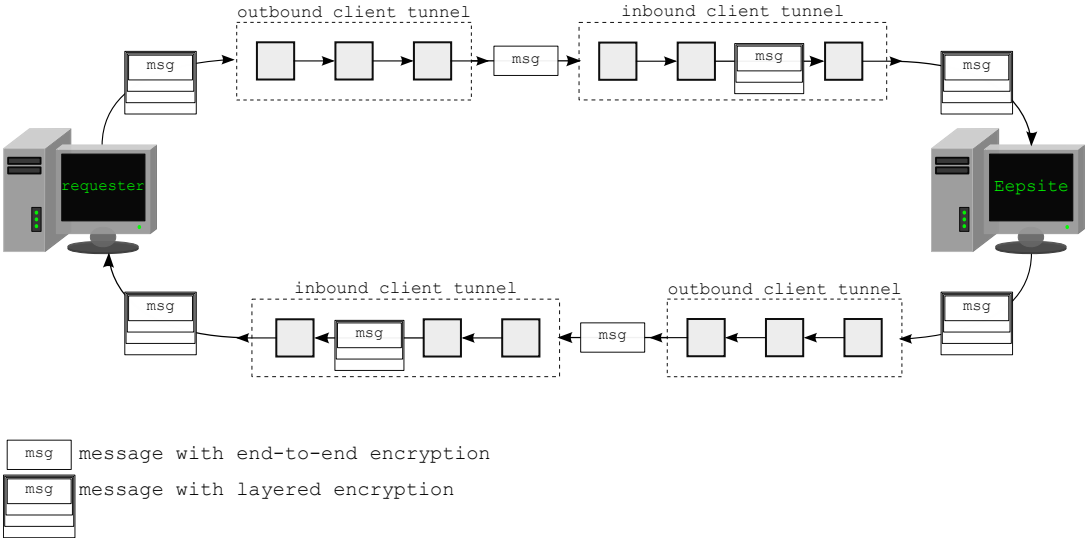


Figure 3.7: Accessing an I2P Eepsite.

Steps 5 and 6 can then be repeated; I2P reuses the resulting channel for subsequent HTTP requests to improve performance. This is somewhat relevant to the attack presented in this thesis since it allows an attacker to repeatedly query the HTTP server without the need to perform the costly tunnel setup operations each time.

3.4 Threat Model

The I2P project does not specify a formal threat model, it instead provides a list of possible well-known attack vectors (such as intersection / partitioning, tagging, DoS, harvesting, sybil and analysis attacks) and the authors discuss how the design relates to these attack vectors.[5]

Based on the scenarios described, I2P's attacker model closely resembles that of Tor: malicious peers are allowed to participate in the network, collect data and actively perform requests. However, the attacker is assumed to be unable to monitor the entire network traffic, should not control a vast number of peers (80% is used as an example) and should not be able to break cryptographic primitives.

3.5 Summary: I2P vs. Tor

On the technical side I2P resembles Tor in many ways, but its developers often use slightly different terminology for almost identical features. Table 3.2 provides a mapping between the different terms used by the two projects.

Table 3.2: Terminology: Terms used by Tor vs. I2P. This list includes the terms relevant to this thesis where the technical differences are also sufficiently minimal. A more extensive (but not always technically close) mapping can be found at `http://www.i2p2.de/how_networkcomparisons`.

Tor	I2P
cell	message
circuit	tunnel
directory	NetDb
directory server	floodfill router
exit node	outproxy
hidden service	Eepsite or destination
hidden service descriptor	lease set
introduction point	inbound gateway
onion routing	garlic routing

The key philosophical difference between the well-known Tor network and I2P is that I2P tries to move existing Internet services into the I2P network and provide service implementations within the framework whereas Tor enables anonymous access to external Internet services implemented and operated separately. While Tor has hidden services and I2P has exit nodes, the canonical usage of Tor is accessing external services and I2Ps canonical usage is accessing internal services.

I2P and Tor also differ in a number of technical details, some of which are key to the attack presented in the following section. Table 3.3 summarizes the main technical differences between the two projects.

[5] `http://www.i2p2.de/how_threatmodel.html`

Table 3.3: Key technical differences between Tor and I2P.

Tor	I2P
3-hop tunnels	user-configurable, randomized number of hops
bi-directional tunnels	uni-directional tunnels
guards, bandwidth-based peer selection	performance-based peer selection
7 directory servers with complete data	super-peer DHT (floodfill peers)
link- and layered-encryption, but not (necessarily) end-to-end-encryption	end-to-end-, link- and layered-encryption
many exit nodes, few hidden services	one exit node, many services
hidden services are external TCP servers	build-in servers for many services
implemented in C	implemented in Java
transport over TCP only (for now)	transport over TCP or UDP

4. Our Attack

Our attack assumes an adversary that actively participates in the network with less than 70 nodes, ideally located in different subnets. The adversary should be distributed in order to work around I2P's restriction of one node per subnet per tunnel and to provide reasonably well-performing malicious peers as neighbors regardless of the location of the victim on the Internet. Each of the participating peers is expected to have resources comparable to typical normal peers in the I2P network. The peers participate in the I2P network according to the network protocol. Our adversary does not have the capability to monitor the traffic of any other node. Our attack influences the performance of I2P peers likely to be chosen by the host of an Eepsite — the victim — for creating its client tunnels. Therefore 40 adversary peers using all their resources to attack other peers to make it more likely that the rest of our peers are chosen as tunnel participants, in order to deanonymize the victim.

The goal of the attacker is to identify the peer "anonymously" hosting a given Eepsite with high probability. Furthermore, it is assumed that the Eepsite is available to the entire I2P network for the duration of the attack and hence resists intersection and partitioning attacks. We have implemented and tested the attack on the extant I2P network in early 2011, making this type of attacker a credible real-world adversary.

For our attack, the adversary uses three types of I2P peers (illustrated in Figure 4.1). The first type, a monitor peer, simply participates in the I2P network as "normal" peer, but reports certain statistics about tunnel operations back to the adversary. The most expensive operation (in terms of time and/or bandwidth) is getting the victim to select these monitor peers as its direct neighbors during tunnel construction. While there is always a (small) chance that the victim will select the adversary's monitor peers, the attack uses a second type of peer, an attack peer, which performs a limited type of DoS attack to influence the victim's tiers to the adversary's benefit. Finally, the adversary also uses one peer to act as a "normal" visitor to the Eepsite, querying the I2P NetDB for leaseSets and issuing HTTP requests to the Eepsite. The leaseSets are used to determine which peers should be attacked (by the attack peers), and the HTTP requests are used to create a pattern which is detected by the monitor peers.

4.1 Taking over the Fast Tier

The main challenge for the adversary is to control the nodes closest to the victim in the inbound and outbound tunnels of the Eepsite. The adversary's goal is to force the victim to

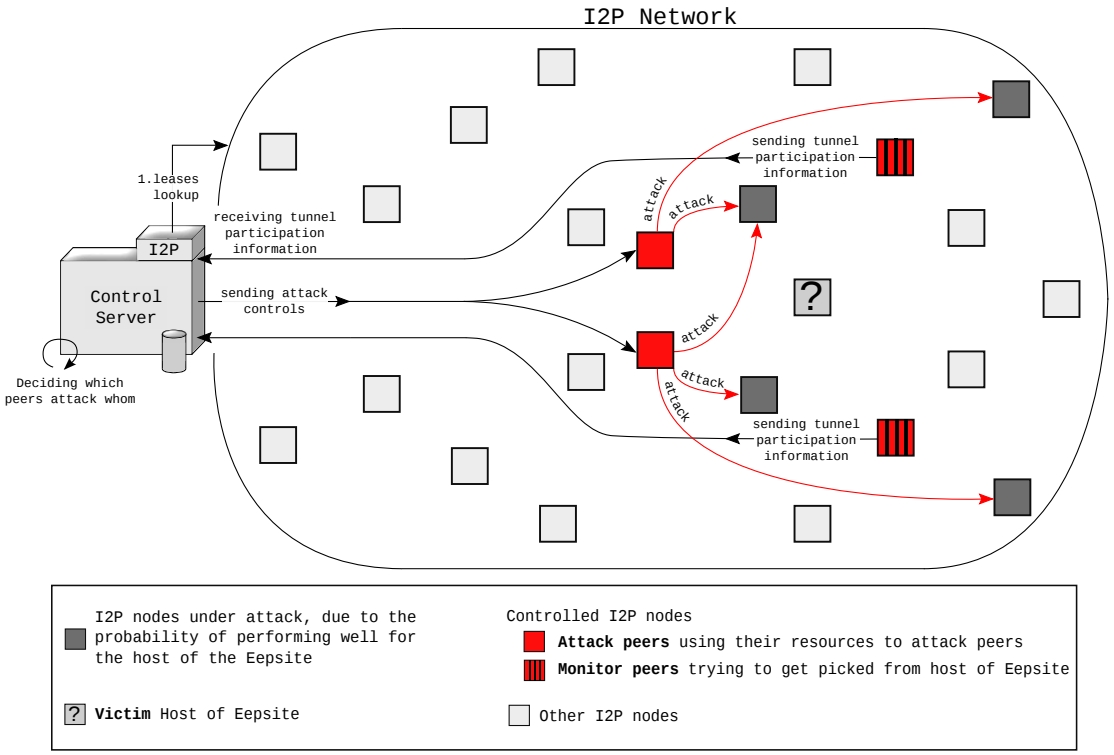


Figure 4.1: Our attack on I2P uses several participating peers in different roles. Monitor peers gather statistical evidence, attack peers accelerate getting the monitor peers into the right position and the control server orchestrates activities.

use the adversary's monitoring peers in its fast tier. Naturally, this requires the adversary to run several well-behaved and fast (monitor) peers. Since I2P never picks multiple peers from the same /16 IPv4 network, the adversary must operate monitor peers in at least a few different /16 IPv4 networks.

Clearly, depending on the size of the I2P network, just having a few monitor peers participate in the network would make it unlikely that the victim chooses these peers. Our attack takes advantage of the peer selection algorithm of I2P, which tries to select only well-performing peers for the tunnels. Thus, the adversary can increase its chances of entering the fast tier by actively hampering the performance of the peers that are currently in the fast tier. While our goal is to enter the victim's fast tier, I2P's use of the highest observed speed over the last 24h makes it impractical to remove peers from the fast tier directly. Furthermore, the adversary may not be able to simply perform faster than the fastest 30 peers in the network — not to mention the victim may normally take a long time to even evaluate nodes controlled by the adversary. Thus, our attack makes use of the fact that I2P only allows high-capacity peers to remain in the fast tier; hence our attack influences the peer selection algorithm by causing peers to reject tunnels, which in turn makes it likely that they will be removed from the high-capacity tier (and thereby also the fast tier). This increases the chance that the victim will then select the adversary's monitoring peers as replacements.

Before the adversary can get peers from the victim's fast tier to reject tunnel requests, the current nodes in the victim's fast tier must be identified. Our attack uses nodes that were recently specified in the leaseSet of the Eepsite as good targets. After all, nodes that are in the leaseSet must be in the fast tier of the victim at that time, and are thus likely to

remain in the fast tier for a while. We found that this method worked better than trying to predict the fast tier from performance measurements done by adversarial nodes.

Given a (small) set of peers that are likely in the fast tier, the adversary performs a denial-of-service (DoS) attack against these peers. Possible venues we considered were attacks against the CPU (by forcing the victims to perform many public key operations) and bandwidth exhaustion. In the end, overloading the peers with a large number of idle tunnels turned out to be the most cost-effective strategy for the current I2P release. It should be noted that the specifics of the DoS attack are not terribly relevant to the big picture of the attack, and alternative strategies such as long paths [22] would likely work as well.

4.2 Confirmation via Traffic Analysis

Eventually the adversary should succeed with placing a few of his nodes into the fast tier of the victim and the victim will then likely choose these nodes for its tunnels. While this will happen eventually, the adversary cannot directly observe that this has actually happened. Furthermore, the adversary's nodes will not necessarily remain in the fast tier indefinitely. As a consequence, the adversary must run the statistical pattern detection described in this section in parallel with the attack algorithm that facilitates its entry into the fast tier.

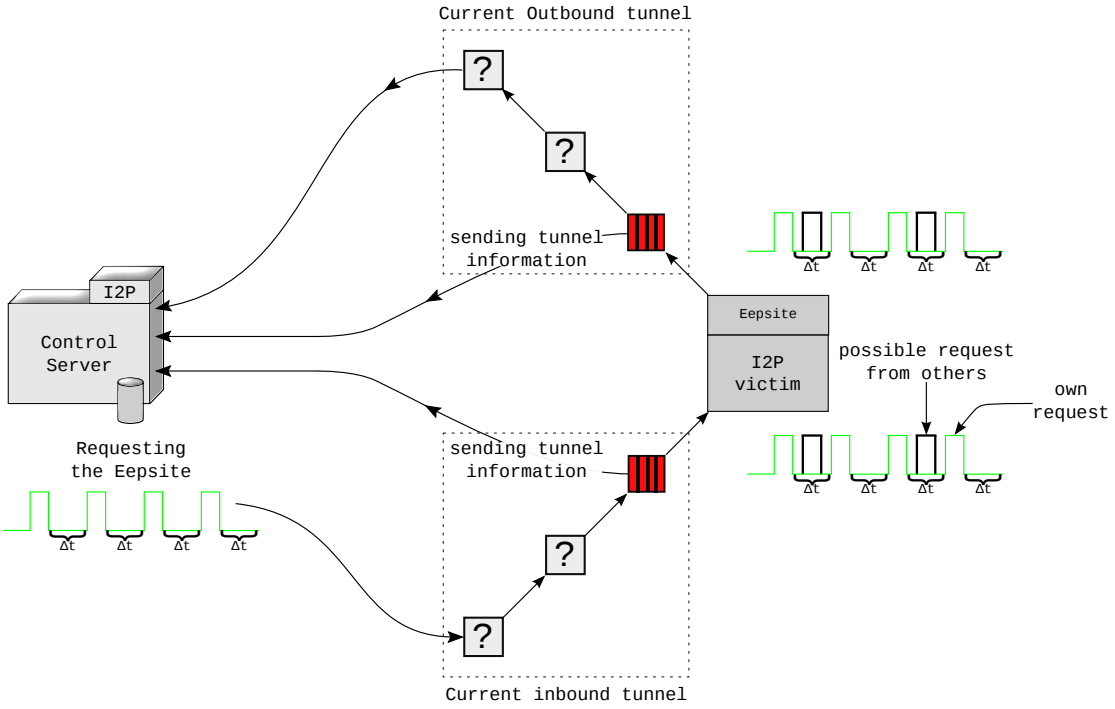


Figure 4.2: A periodic signal is induced by the control server and detected by the monitor nodes. They report likely Eepsite hosts to the control server which aggregates the information.

The statistical pattern we use to identify the correct tunnels at the monitor peers is, similar to [32], a periodic HTTP request issued by the control server at a fixed frequency t (Figure 4.2). For our experiments we use $t = 15s$. For each tunnel, each monitoring peer counts the number of packets received in buckets representing time intervals of packet

arrival times modulo t. If the total number of packets is smaller than those transmitted by the adversary to the Eepsite, the circuit is ignored. If the number of packets is close to the expected number, the monitoring peers compute how many standard deviations the largest bucket size is from the average bucket size. If the resulting factor is large, the packets were not equally distributed. Then, to exclude false-positives from short, non-periodic bursts, the monitoring peers perform the same calculation, this time for a time interval modulo q where $gcd(t, q) = 1$ and $|t - q|$ is small (we use $q = 16s$). If the signal had a frequency of t, the resulting factor should be very small; however, if a burst caused a false-positive, the resulting factor should be about as big as for the calculation modulo t. If the distribution is normalized modulo q, the tunnel is reported to the adversary as detected. If two monitoring peers report a peer between them at the same time, that peer is flagged likely to be the Eepsite host. The sensitivity used for the standard deviation factor threshold determines how often the same peer needs to be flagged before the adversary can be certain.

5. Experimental Results

For our experiments, we deployed a number of I2P nodes on PlanetLab [38]. Each node was configured to use at most 64 kb/s upstream and downstream bandwidth. We set up the control peer on a machine controlled by us to minimize jitter. Furthermore, one of our peers was set up to host an Eepsite to serve as a victim for testing. For our tests, we used up to 40 attack peers and 30 monitor peers. All tests were performed by having all of our peers join the live I2P network and participate normally (except, of course, for attack-specific behavior).

5.1 Tier Evolution

First, we wanted to see how well we would be able to predict the victim's fast tier. This determines how big the impact of our attack actually is. Our first attempt was to combine the knowledge of several attacker peers, which turned out to be infeasible. The number of correct guesses for candidates of an random victim peer grows indeed, unfortunately the number of possible candidates grows even more. Table 5.1 shows a summary of this experiment. All details can be found in B.1.

Table 5.1: Accuracy of the prediction for the fast tier of an arbitrary peer using the combined fast tiers of several attackers.

Number of attacker	Number of fast tier hits	Number of candidates	Resulting fraction
1	5	30	16.7%
2	6	53	11.32%
3	8	74	10.81%
4	11	93	11.82%
5	12	110	10.91%

Our next approach was to use the information an Eepsite leaks about its fast tier. Leases of an Eepsite are public information in the I2P network. Table 5.2 shows what fraction of the last n peers observed in the leaseSet were actually in the fast tier of the victim at the time. We configured the victim to use only one inbound and one outbound tunnel. Obviously, this data describes a worst case, since with more tunnels also more leases can be learned.

Table 5.2: Accuracy of the prediction for peers in the fast and high-capacity tiers using the n most recently observed peers from the lease set. The given percentage refers to the fraction of the peers from the n most recent leases that are actually in the respective tier. The fast tier typically consists of 30 peers, the high-capacity tier typically has 75 peers. At the time of the measurement, the I2P network contained at least 1921 peers in total.

# leases (most recent)	% nodes from lease set	
	in fast tier	in high-capacity tier
5	60%	60%
10	40%	50%
15	40%	47%
20	45%	55%
25	36%	52%
30	30%	50%

5.2 Attack Effectiveness

Next, we determined the impact of the DoS attack, first on the attacked peers (to confirm that the attack works as expected), and then on peer fluctuation in the fast and high capacity tier. Table 5.3 shows the impact of our attack on a single peer. It compares the tunnel request acceptance rate of an ordinary peer with the acceptance rate when that peer is attacked by several attackers. Table 5.4 shows the typical churn rate for peers in the high-capacity and fast tier of the victim ordinarily and while under attack. The data corresponds to the adversary attacking the last 30 peers observed as leases (with the expected inaccuracies as listed in Table 5.2). The data shows that the DoS attack is effective at obstructing tunnel operations and that the victim reacts to these obstructions by replacing peers in its high-capacity and fast tiers more often.

Table 5.3: Direct impact of the tunnel acceptance rate of a peer under attack from various number of attackers with a configured bandwith limit of 64 kb/s. Note that an increasing number of attackers not only causes the peer under attack to reject tunnels, but additionally causes requests for tunnels to be lost and hence not be answered at all.

		under attack, number attacker				
	normal	2	3	5	7	10
Tunnels accepted	82%	63%	52%	16%	9%	1%
Tunnels rejected	18%	36%	41%	40%	36%	28%
Tunnels lost	0%	1%	7%	44%	55%	71%

5.3 Deanonymization

Finally, we measured how effective our statistical analysis is at determining the victim once the measurement nodes are in place.

We provide some examples for what the statistical patterns observed by the monitor peers (Section 4.2) look like. Figure 5.1, Figure 5.2 and Figure 5.3 are showing representative patterns for the case where the adversary deanonymizes the victim with the signal and performs the statistical analysis using the correct modulus (here $t = 15$). Internals of the I2P implementation (which we can no fully explain) seem often to create two distinct peaks at about a close distance. Additionally those figures show the same data using a

Table 5.4: Impact of the DoS attack on the network using 40 peers with a configured bandwidth limit of 64 kb/s. This table shows the increase in the churn for the high-capacity and fast tiers of the victim that the attacker tries to deanonymize. Each value represents the churn of nodes per 45 seconds tier evaluation cycle of the victim. Note that the attack uses our (limited-precision) leaseSet-based prediction heuristic (Section 4.1) to determine which peers to attack. If the attacker could be certain about which peers are in the respective tiers, the increase in churn would be significantly higher. Monitor peers provided by the attacker are not subjected to the attack.

	normal	under attack
High-capacity tier churn	0.89 peers/cycle	3.41 peers/cycle
Fast tier churn	0.76 peers/cycle	1.71 peers/cycle

different modulus (here $q = 16$), resulting in the peaks being destroyed. This would not be the case if the signal was not due to requests at the adversaries frequency of t.

Finally, Figure 5.4 and Figure 5.5 are showing a typical pattern for a deanonymization that does not contain the signal. It should be noted that most circuits do not reach the required minimum number of messages and are hence filtered long before this analysis is even performed.

Table 5.5 lists the number of times the measurement nodes identified the correct host of the Eepsite (the target of the attack) vs. the second highest score assigned to any of the other peers in the I2P network (false positives). Since the statistical analysis uses a threshold, we report the data for different possible thresholds. The data was obtained over the course of four hours with the attacker controlling the entire fast tier of the victim.

Table 5.5: Final result of the statistical analysis for both the actual target for the deanonymization and the highest-ranked false-positive (FP) peer for different thresholds. The table shows the number of times the respective peer exceeded the given statistical threshold (for the correct signal frequency, minus the statistical significance computed for a non-signal frequency) for the duration of the measurement.

Detection	1-hop tunnel		2-hop tunnel	
Threshold	Target	top FP	Target	top FP
1 Std. Dev.	78	130	60	45
2 Std. Dev.	69	28	47	6
3 Std. Dev.	48	2	42	2
4 Std. Dev.	41	2	36	2
5 Std. Dev.	30	0	26	0
6 Std. Dev.	30	0	21	0
7 Std. Dev.	28	0	19	0
8 Std. Dev.	23	0	19	0
9 Std. Dev.	15	0	17	0
10 Std. Dev.	12	0	15	0

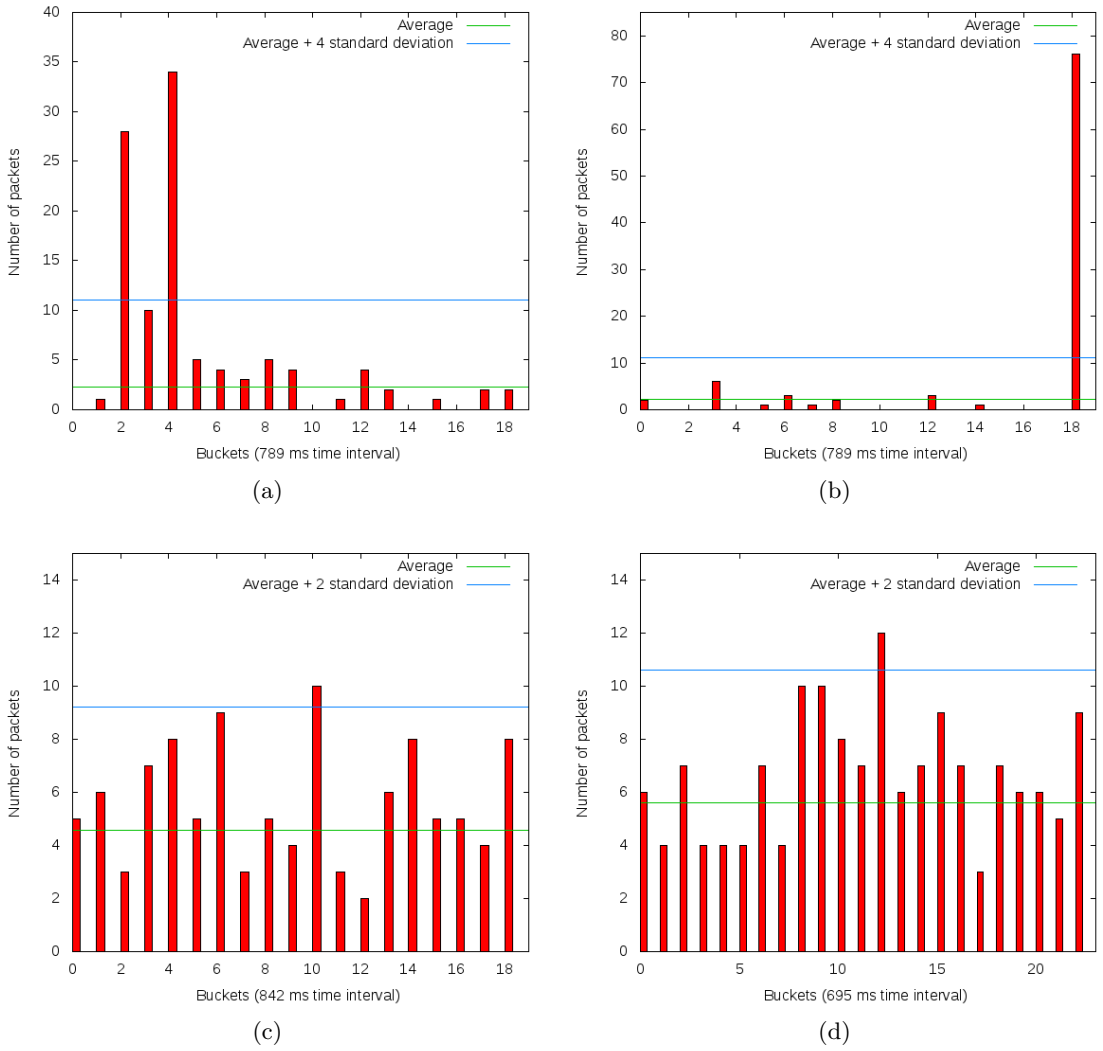


Figure 5.1: Graph (a) and graph (b) show the observed data of the predecessor and successor in the case the victim got deanonymized and the data was calculated with the correct modulus q. Graph (c) and graph (d) show the same data, but calculated with the wrong modulus t.

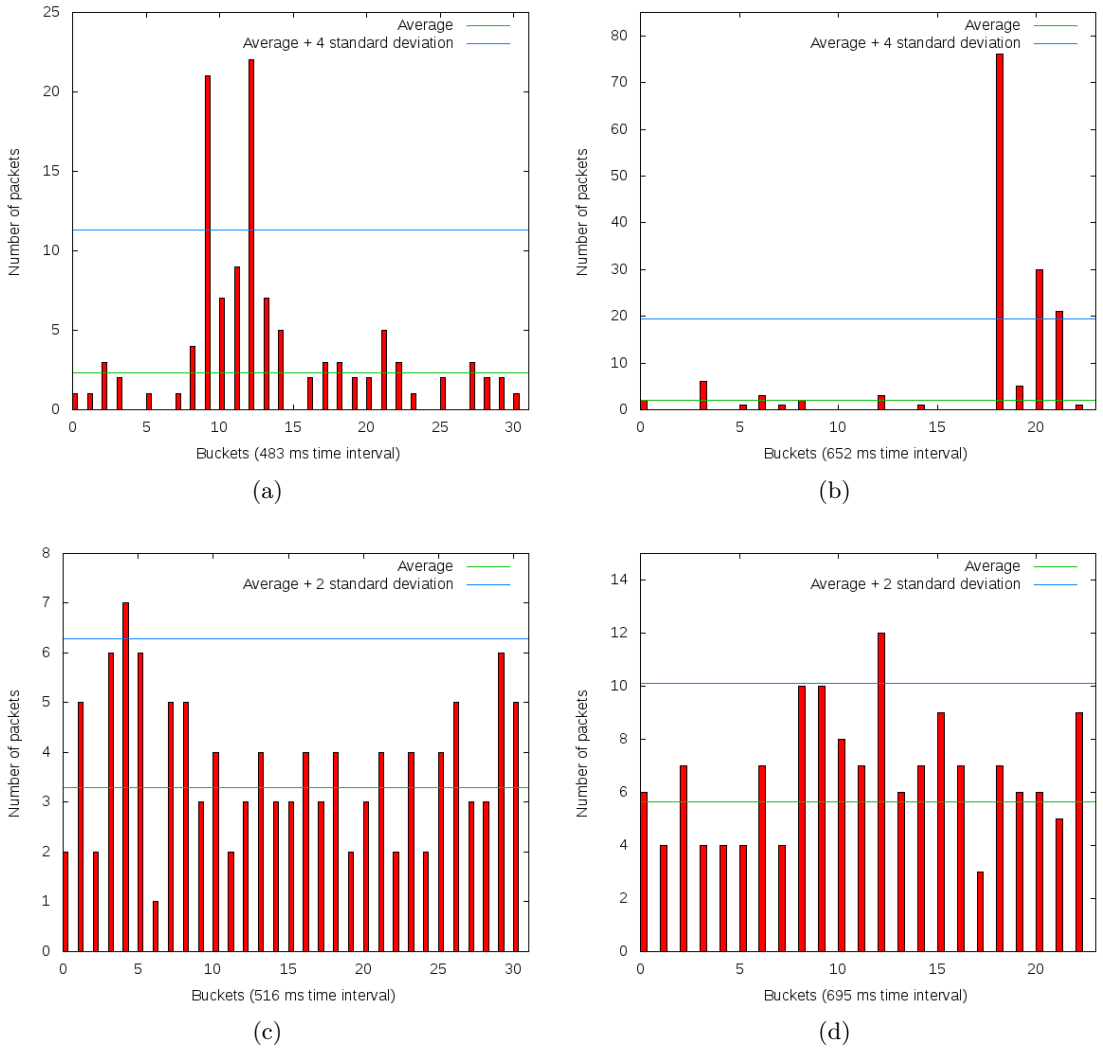


Figure 5.2: Graph (a) and graph (b) show the observed data of the predecessor and successor in the case the victim got deanonymized and the data was calculated with the correct modulus q. Graph (c) and graph (d) show the same data, but calculated with the wrong modulus t.

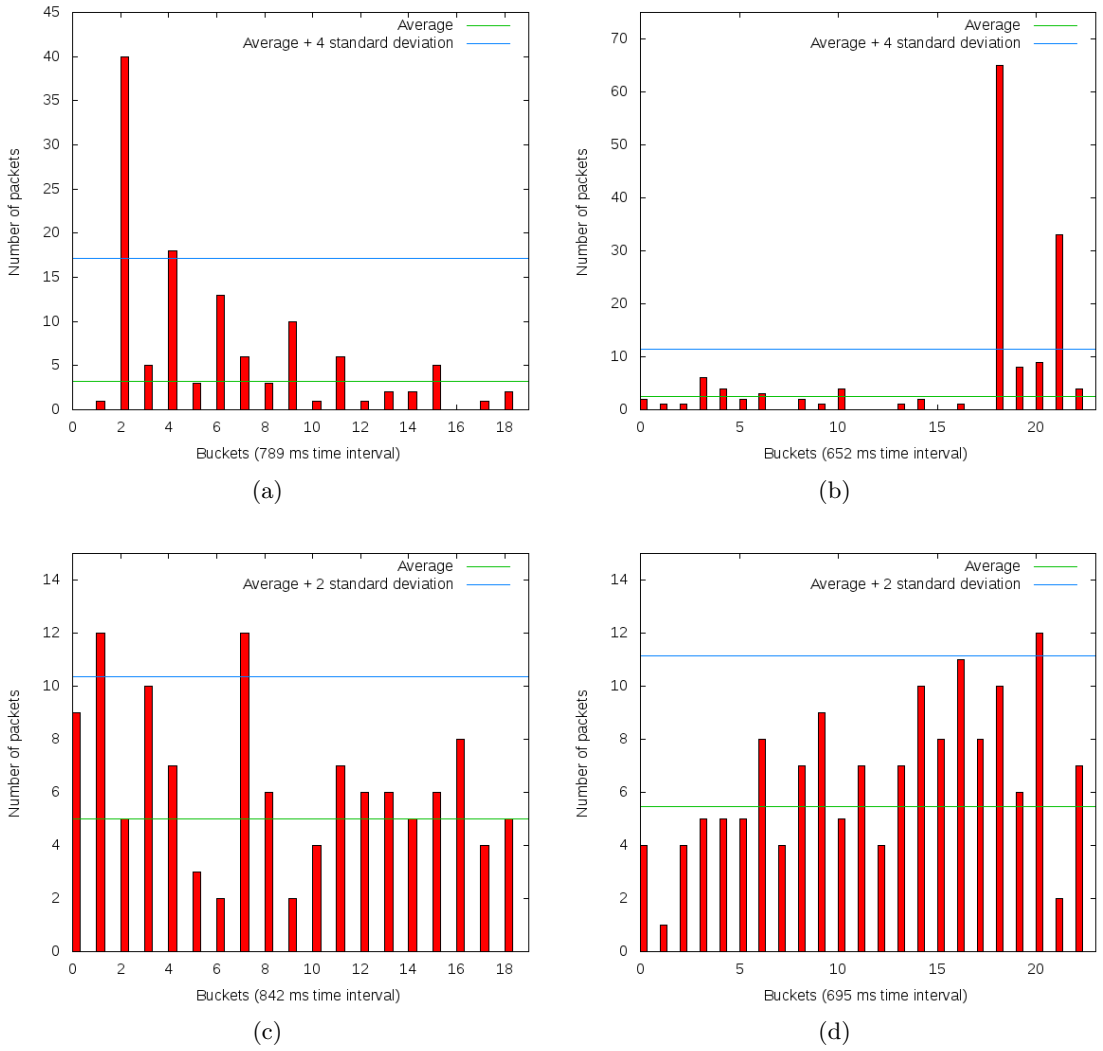
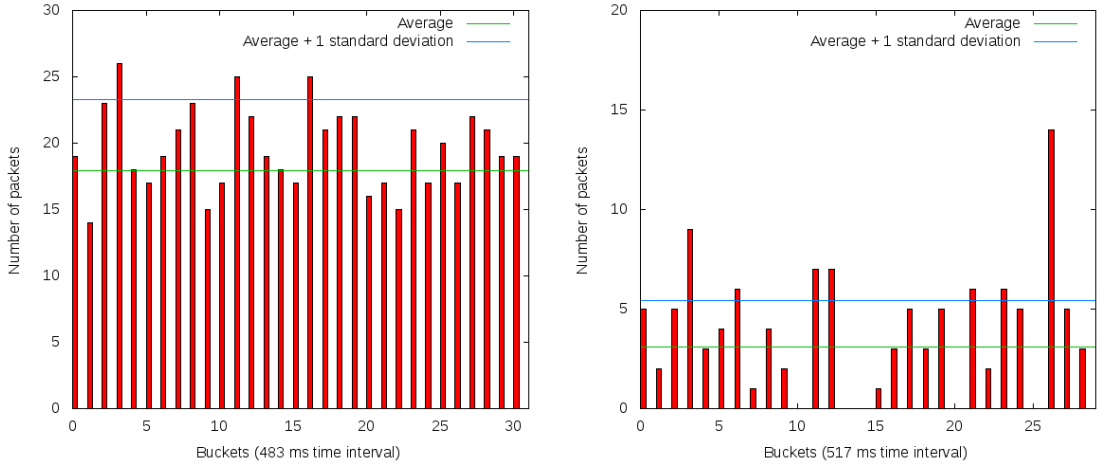
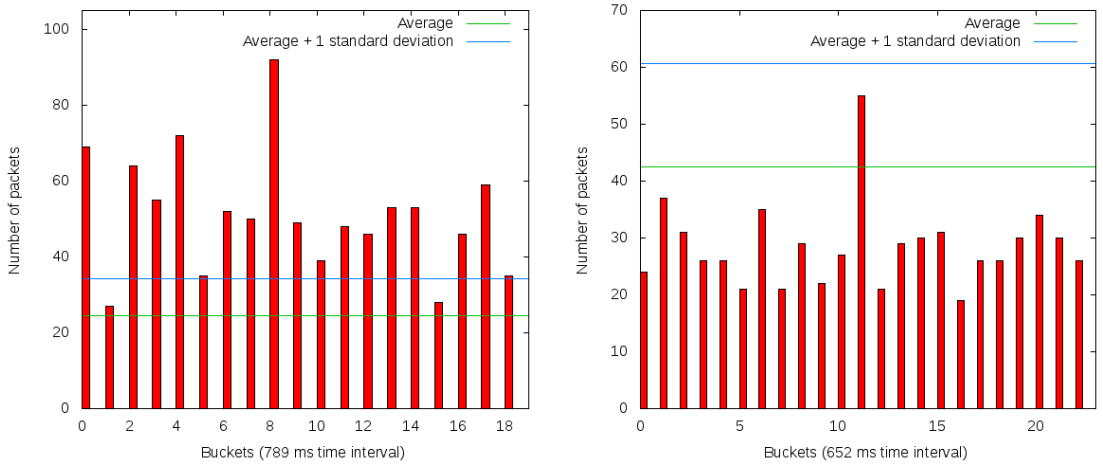


Figure 5.3: Graph (a) and graph (b) show the observed data of the predecessor and successor in the case the victim got deanonymized and the data was calculated with the correct modulus q. Graph (c) and graph (d) show the same data, but calculated with the wrong modulus t.



(a) (b)

Figure 5.4: Subfigure (a) and (b) show the observed data of the predecessor and successor in the case the victim got wrongly deanonymized.



(a) (b)

Figure 5.5: Subfigure (a) and (b) show the observed data of the predecessor and successor in the case the victim got wrongly deanonymized.

6. Discussion

This work confirms the well-known result [7] that attacks on availability or reliability of an anonymizing service can be used to compromise anonymity. What we have shown specifically is that anonymizing networks that have a strong bias towards well-performing peers for tunnel construction are particularly vulnerable to this type of attack. Once the tunnel is compromised, other researchers have shown that latency measurements could be used to determine the likely identity of the victim [50].

Because of the uni-directional nature of the I2P tunnels the attacker has to wait a longer time to possibly deanonymize a victim. Monitoring peers have to be in the correct position for both the inbound and the outbound tunnel. Thus, with a being the number of monitor peers in the fast tier of the victim, the probability for deanonymization in a fast tier of size 30 is:

$$\left(\frac{a}{30}\right)^2 \tag{6.1}$$

For bi-directional tunnels the attacker would only need one peer in the correct position, resulting in the probability:

$$\frac{a}{30} \tag{6.2}$$

However, the significance of a deanonymization is different for both cases, due to the probability of deanonymizing the wrong peer (false-positive). For the uni-directional, I2P case the probability to deanonymize the wrong peer requires the same ordinary tunnel participant in certain positions. At first the victim has to choose an ordinary, non-malicious peer for the inbound tunnel. This happens with probability $\frac{a-30}{30}$. The same peer also needs to be in the outbound tunnel, which happens with probability $\frac{1}{30}$. Additionally the victim has to choose a monitor peer for the second hop of one inbound and one outbound tunnel, which happens with probability $\left(\frac{a}{29}\right)^2$. Combining all these probabilities, the probability for false positives with uni-directional tunnels is[1]:

[1]Note that in principle this deanonymization setting of uni-directional path possibly leads to another case of false positives. Whenever in a tunnel an ordinary peer has an monitor peer as predecessor and as successor, this ordinary peer can falsely be identified to be the victim. However, since we induce our signals with HTTP requests/responses, it is possible to create a certain, distinguishable ratio between the data packets of the request and response. Obviously, since in this additional case the data ratio does not change, we do not need to take this case into account.

$$\frac{30-a}{30}\frac{1}{30}\left(\frac{a}{29}\right)^2 = \frac{30-a}{30^2}\left(\frac{a}{29}\right)^2 \approx a^2\frac{30-a}{30^4} \tag{6.3}$$

With bi-directional tunnels, the probability for a false positive is higher, because any other peer between a monitor peer and the victim can falsely be accused. The overall probability for this case is:

$$\frac{30-a}{30}\frac{a}{29} \approx a\frac{30-a}{30^2} \tag{6.4}$$

Correlating the probability of deanonymization and the probability for getting a false positive in the uni-directional and bi-directional case shows the difference in the significance. Dividing Equation 6.1 and Equation 6.2 results in $\frac{a}{30}$ and shows that the attacker has to wait 30 times longer in the uni-directional case than in the bi-directional case to be in a position to possibly deanonymize the victim. Doing the same with Equation 6.3 and Equation 6.4 results in $\frac{a}{30^2}$ and shows that the accuracy for the uni-directional case is 900 times bigger than in the bi-directional case.

This result indicates that uni-directional tunnels help an attacker due to the higher certainty an attacker can get from possible deanonymizations. Consequently, using uni-directional seems to be a bad design decision. Figure 6.1 clarifies the trade-off for an attacker. Especially when the attacker managed to control around the half of the victims fast tier, the false positive rate increases up to 25%.

Consequently, with choosing between uni-directional and bi-directional tunnels, the developers of anonymous communication systems can influence the certainty and the speed an attacker can deanonymize a peer. We want to stress that the false positive rate of bi-directional path is not tremendously high and might also be manageable for an attacker. But since Eepsites in I2P are accessible for a long time, time is not an issue and we think that uni-directional paths helps us to clearly deanonymize the victim.

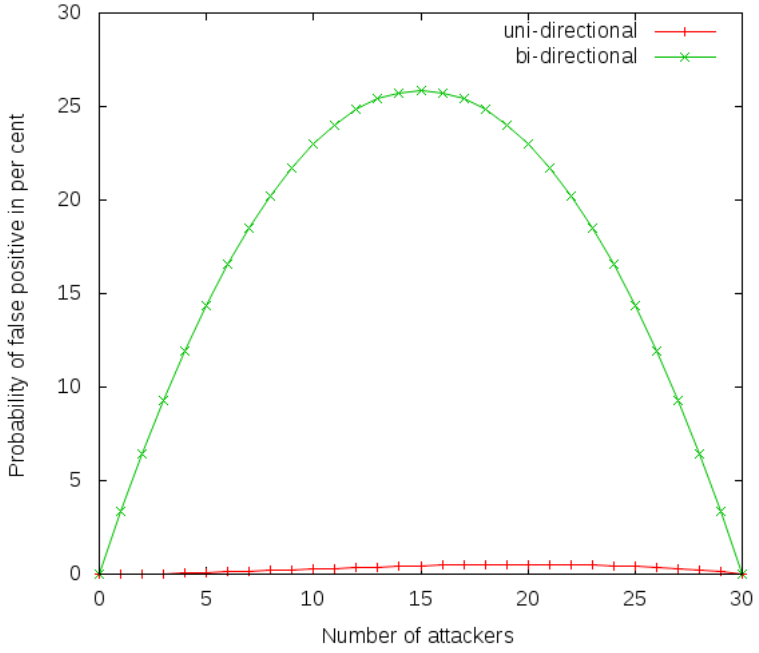


Figure 6.1: Showing the probability of false positive deanonymizations for both uni-directional and bi-directional case. The rate is much higher for bi-directional tunnels.

While making the I2P network more robust towards DoS attacks is always a good goal, we do not believe that this would address the main problem, the ability of the adversary to influence peer selection. While I2P's heuristics seem to make it hard for an adversary to directly influence the metrics used for peer selection, influencing performance itself is likely always possible. Hence, a better solution would be to limit churn in the fast and high-capacity tiers, similar to how Tor uses a limited set of guard nodes.

Another problem is the fact that Eepsites allow repeated measurements, giving the attacker the opportunity to possibly collect data for many months. This problem is not unique to I2P, but also applies in exactly the same way to Tor's hidden services. However, since philosophically I2P services are more integrated, I2P's design may offer a solution, at least for static content: instead of having a single peer act as the HTTP server, I2P should allow HTTP services to be offered in a distributed fashion. The secure distributed filesystem *Tahoe*[2] is currently be ported to I2P, which may address this concern.

Most importantly, I2P should avoid leaking information about its fast tier by using random peers, that are in the not-failing tier but not in the fast tier, for the leases. This would make it harder for an adversary to determine which peers should be attacked with the DoS attack while maintaining performance advantages for the rest of the tunnel.

[2] http://tahoe-lafs.org/~warner/pycon-tahoe.html

7. Conclusion

Biasing peer selection towards well-performing peers has previously been seen as a mostly theoretical issue. This work shows that combined with a limited, selective DoS attack on a few peers it enables an adversary to compromise the anonymity of long-running services. When our DoS-attack was detected by the I2P developers, they decided to make the peer selection algorithm switch peers even faster in response to performance anomalies for the next release. This work shows that peers reacting, and especially reacting quickly, to changes in observed network performance can be a bad idea for anonymizing networks.

A. I2P Heuristics in Detail

This appendix gives details on the various heuristics in the original I2P codebase that have been described in the paper. For our attack, the specifics given in this section are not significant; modest variations in how these heuristics are implemented would not change the attack.

A.1 Capacity calculation

Actions from the distant past are not considered to be as relevant as more recent events; hence the capacity calculation considers event counts for different time intervals, specifically 10 minutes, 30 minutes, 60 minutes and 24 hours.

For every time interval a capacity value is calculated, and the weighted corresponding results give the final value. If cap is the capacity calculation function, the capacity value for a peer p is:

$$\begin{aligned}\text{capacity}_p = 0.4\,\text{cap}(10\,\text{min}) + 0.3\,\text{cap}(30\,\text{min}) + \\ 0.2\,\text{cap}(60\,\text{min}) + 0.1\,\text{cap}(24\,\text{h})\end{aligned} \tag{A.1}$$

The calculation of the cap function itself is also separated into two steps. At first, the counted events of accepting and rejecting tunnel requests are considered. More specifically, the events in the particular time interval and the last time interval are summed up. For example in the cap calculation for the 10 minutes time interval, the data of the current 10 minutes and the previous 10 minutes are considered. Let a be the amount of accepted tunnel requests for the current and the last period and r be the corresponding rejection rate, then the temporary capacity value is calculated by $c_{\text{temp}} = \frac{a^2}{a+r}$. In the next step c_{temp} is adjusted by possible tunnel failures. If failures happened, let f be the amount of failures in the current and the previous period, the capacity amount subtracted is $c_{\text{sub}} = 0.5 + \frac{4f}{100}$.

Since a negative capacity is not considered in I2P, the result of the cap function is:

$$\text{cap} = \max\left(\frac{a^2}{a+r} - (0.5 + \frac{4f}{100}), 0\right) \tag{A.2}$$

In the second step of the overall capacity calculation the, the capacity value is adjusted if a rejection happened in the last five minutes, corresponding to the rejection reason. If

a rejection reason indicates a high request overload, the capacity value is set to 1. If a rejection reason was probabilistic nature, the capacity value is decremented by a random number r, with $r \in]0, 5]$.

A.2 Integration value calculation

If int_{cur} is a function giving the amount of successful peer lookups or received stores of a peer p for the current time interval and int_{prev} is the function for the corresponding previous time interval, then the integration value of p is:

$$\text{integration}_p = 96\,\text{int}_{\text{cur}}(1h) + 12\,\text{int}_{\text{cur}}(6h) + 8\,\text{int}_{\text{prev}}(6h) + \text{int}_{\text{cur}}(1d) \tag{A.3}$$

A.3 Threshold Calculation

In order to identify high performing peers, the I2P router needs to calculate additionally to the performance measures for every known peer, for every tier a threshold value. In the following, we introduce the threshold calculation for every tier.

The capacity threshold calculation depends on a list list_{all}, containing all known peers ordered by the capacity value, and the following values:

min_{hC} The minimum number of peers needed for the high capacity tier.

$\text{exceed}_{\text{hC}}$ The number of peers in list_{all}, that exceed the current capacity threshold.

mean_{hC} The mean value of list_{all}.

$\text{median}_{\text{hC}}$ The median value of list_{all}.

cap(min_{hC}) The capacity value of the element in list_{all} at position list_{all}.

cap($last$) The capacity value of the element in list_{all} at the last position.

With these values, the capacity threshold is calculated as follows:

- If $\text{min}_{\text{hC}} < \text{exceed}_{\text{hC}}$: The capacity threshold becomes mean_{hC}.
- If previous condition is false and $\text{mean}_{\text{hC}} > \text{median}_{\text{hC}}$ and list_{all} is at least two times more than min_{hC}: The capacity threshold becomes $\text{cap}(\text{min}_{\text{hC}})$.
- If the previous condition is false and list_{all} is at least two times more than min_{hC}: The capacity threshold becomes $\text{median}_{\text{hC}}$.
- In any other case, the capacity value becomes: cap($last$).

For the well integrated tier, the threshold is the mean value of the integration value of all known peers. For the fast tier the speed threshold is the mean of the speed value of all peers in the high capacity tier.

B. Tiers in Detail

B.1 Tier Prediction

Table B.1 shows the data of collaborating peers combining their tier-knowledge in order to predict the tiers of another peer.

Table B.1: Showing the benefit of several peers combining their tiers to guess the high-capacity (HC), fast (F), well-integrated (WI) and not-failing (NF) tier of a single peer, located in Aachen. The more peers collaborate, the more peers the team and Aachen have in common. However, for the team, the number of peers in a tier gets also larger.

Team	Tier	Tier Member	HC	F	WI	NF
Berlin	HC	75	27	12	20	67
	F	30	9	5	7	30
	WI	49	21	9	33	49
	NF	595	68	28	48	453
Berlin Kaiserslautern	HC	130	34	15	29	119
	F	53	13	6	13	53
	WI	60	26	11	42	60
	NF	777	74	30	48	541
Berlin Kaiserslautern Ilmenau	HC	173	37	16	38	156
	F	74	17	8	23	73
	WI	68	28	12	47	68
	NF	867	75	30	48	561
Berlin Kaiserslautern Ilmenau Hannover	HC	208	40	18	41	187
	F	93	22	11	27	92
	WI	69	28	12	47	69
	NF	932	75	30	48	574

Literatur

[1] Dakshi Agrawal, Dogan Kesdogan, and Stefan Penz. Probabilistic Treatment of MIXes to Hamper Traffic Analysis. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 16–27, May 2003.

[2] Ross Anderson. The eternity service. In *Proceedings of Pragocrypt '96*, 1996.

[3] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-resource routing attacks against tor. In *Proceedings of the 2007 ACM workshop on Privacy in electronic society*, WPES '07, pages 11–20, New York, NY, USA, 2007. ACM.

[4] Tonda Benes. The strong eternity service. In *Proceedings of the 4th International Workshop on Information Hiding*, IHW '01, pages 215–229, London, UK, UK, 2001. Springer-Verlag.

[5] Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 115–129. Springer-Verlag, LNCS 2009, July 2000.

[6] Oliver Berthold, Andreas Pfitzmann, and Ronny Standtke. The disadvantages of free MIX routes and how to overcome them. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 30–45. Springer-Verlag, LNCS 2009, July 2000.

[7] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of service or denial of security? In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 92–102, New York, NY, USA, 2007. ACM.

[8] Zach Brown. Cebolla: Pragmatic IP Anonymity. In *Proceedings of the 2002 Ottawa Linux Symposium*, June 2002.

[9] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, 1981.

[10] Bram Cohen. Incentives build robustness in bittorrent. http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.1911&rank=1, 2003.

[11] Curt Cramer, Kendy Kutzner, and Lin Tong. Bootstrapping locality-aware p2p networks. In *Proceedings of the IEEE International Conference on Networks (ICON 2004)*, volume 1, page 357–361, Singapore, 2004.

[12] George Danezis. Forward secure mixes. In Jonsson Fisher-Hubner, editor, *Proceedings of 7th Nordic Workshop on Secure IT Systems*, page 195–207, Karlstad, Sweden, November 2002.

[13] George Danezis. Mix-networks with restricted routes. In Roger Dingledine, editor, *Proceedings of Privacy Enhancing Technologies workshop (PET 2003)*, pages 1–17. Springer-Verlag, LNCS 2760, March 2003.

[14] George Danezis. The traffic analysis of continuous-time mixes. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)*, volume 3424 of *LNCS*, pages 35–50, May 2004.

[15] George Danezis. Breaking four mix-related schemes based on universal re-encryption. *International Journal of Information Security*, 6:393–402, 2007. 10.1007/s10207-007-0033-y.

[16] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 2–15, May 2003.

[17] George Danezis and Ben Laurie. Minx: a simple and efficient anonymous packet format. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, WPES '04, pages 59–65, New York, NY, USA, 2004. ACM.

[18] Claudia Diaz and Bart Preneel. Taxonomy of mixes and dummy traffic. In Yves Deswarte, Frederic Cuppens, Sushil Jajodia, and Lingyu Wang, editors, *Information Security Management, Education and Privacy*, volume 148 of *IFIP International Federation for Information Processing*, pages 217–232. Springer Boston, 2004.

[19] Roger Dingledine, Michael Freedman, and David Molnar. The free haven project: Distributed anonymous storage service. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies*, volume 2009 of *Lecture Notes in Computer Science*, pages 67–95. Springer Berlin / Heidelberg, 2001.

[20] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: the second-generation onion router. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[21] Roger Dingledine and Paul Syverson. Reliable mix cascade networks through reputation. In Matt Blaze, editor, *Financial Cryptography*, volume 2357 of *Lecture Notes in Computer Science*, pages 253–268. Springer Berlin / Heidelberg, 2003.

[22] Nathan S. Evans, Roger Dingledine, and Christian Grothoff. A practical congestion attack on tor using long paths. In *18th USENIX Security Symposium*, page 33–50. USENIX, USENIX, 2009.

[23] Michael J. Freedman and Robert Morris. Tarzan: a peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, pages 193–206, New York, NY, USA, 2002. ACM.

[24] Philippe Golle. Reputable mix networks. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)*, volume 3424 of *LNCS*, pages 51–63, May 2004.

[25] Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson. Universal re-encryption for mixnets. In Tatsuaki Okamoto, editor, *Topics in Cryptology - CT-RSA 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 1988–1988. Springer Berlin / Heidelberg, 2004.

[26] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. RFC 1323 (Proposed Standard), May 1992.

[27] Marek Klonowski and Miroslaw Kutylowski. Provable anonymity for networks of mixes. In *Proceedings of Information Hiding Workshop (IH 2005)*, pages 26–38, June 2005.

[28] Brian Neil Levine and Clay Shields. Hordes — A Multicast Based Protocol for Anonymity. *Journal of Computer Security*, 10(3):213–240, 2002.

[29] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. page 53–65, 2002.

[30] Jon McLachlan and Nicholas Hopper. On the risks of serving whenever you surf: vulnerabilities in tor's blocking resistance design. In *Proceedings of the 8th ACM workshop on Privacy in the electronic society*, WPES '09, pages 31–40, New York, NY, USA, 2009. ACM.

[31] Steven J. Murdoch. Hot or not: revealing hidden services by their clock skew. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 27–36, New York, NY, USA, 2006. ACM.

[32] Steven J. Murdoch and George Danezis. Low-cost traffic analysis of tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*. IEEE CS, IEEE CS, May 2005.

[33] Lan Nguyen and Rei Safavi-Naini. Breaking and mending resilient mix-nets. In Roger Dingledine, editor, *Proceedings of Privacy Enhancing Technologies workshop (PET 2003)*, pages 66–80. Springer-Verlag, LNCS 2760, March 2003.

[34] Luke O'Connor. On blending attacks for mixes with memory. In *Proceedings of Information Hiding Workshop (IH 2005)*, pages 39–52, June 2005.

[35] Miyako Ohkubo and Masayuki Abe. A length-invariant hybrid mix. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 178–191. Springer Berlin / Heidelberg, 2000.

[36] Andriy Panchenko, Benedikt Westermann, Lexi Pimenidis, and Christer Andersson. SHALON: Lightweight anonymization based on open standards. In *Proceedings of 18th International Conference on Computer Communications and Networks*, San Francisco, CA, USA, August 2009.

[37] Ginger Perng, Lin Tong, and Chenxi Wang. M2: Multicasting mixes for efficient and anonymous communication. In *Proceedings of the 26th IEEE Conference on Distributed Computing Systems*, July 2006.

[38] Larry Peterson. PlanetLab: Version 3.0. Technical Report PDN–04–023, PlanetLab Consortium, October 2004.

[39] Andreas Pfitzmann and Marit Hansen. Anonymity, unobservability, and pseudonymity: A consolidated proposal for terminology, July 2000.

[40] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16:482–494, 1998.

[41] Marc Rennhard and Bernhard Plattner. Introducing morphmix: peer-to-peer based anonymous internet usage with collusion detection. In *Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*, WPES '02, pages 91–102, New York, NY, USA, 2002. ACM.

[42] Marc Rennhard and Bernhard Plattner. Practical anonymity for the masses with morphmix. In Ari Juels, editor, *Proceedings of Financial Cryptography (FC '04)*, pages 233–250. Springer-Verlag, LNCS 3110, February 2004.

[43] Marc Rennhard and Bernhard Plattner. Practical anonymity for the masses with morphmix. In Ari Juels, editor, *Proceedings of Financial Cryptography (FC '04)*, page 233–250. Springer-Verlag, LNCS 3110, Springer-Verlag, LNCS 3110, February 2004.

[44] Andrei Serjantov. Anonymizing censorship resistant systems. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, pages 111–120. Springer Berlin / Heidelberg, 2002.

[45] Andrei Serjantov and Steven J. Murdoch. Message splitting against the partial adversary. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2005)*, page 26–39. Springer Berlin / Heidelberg, Springer Berlin / Heidelberg, May 2005.

[46] Rob Sherwood, Bobby Bhattacharjee, and Aravind Srinivasan. P5: A protocol for scalable anonymous communication. *Journal of Computer Security*, Volume 13:839 – 876, December 2005 2002.

[47] Eric Shimshock, Matt Staats, and Nick Hopper. Breaking and provably fixing minx. In Nikita Borisov and Ian Goldberg, editors, *Proceedings of the Eighth International Symposium on Privacy Enhancing Technologies (PETS 2008)*, pages 99–114, Leuven, Belgium, July 2008. Springer.

[48] Parisa Tabriz and Nikita Borisov. Breaking the collusion detection mechanism of morphmix. In George Danezis and Philippe Golle, editors, *Proceedings of the Sixth Workshop on Privacy Enhancing Technologies (PET 2006)*, pages 368–384, Cambridge, UK, June 2006. Springer.

[49] Lin Tong and Lin Tong. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1:66–92, 1998.

[50] Lin Tong, Eugene Y. Vasserman, and Eric Chan-Tin. How much anonymity does network latency leak? *ACM Transactions on Information and System Security*, pages 82 – 91, forthcoming 2010.

[51] Gergely Tóth and Zoltán Hornák. Measuring anonymity in a non-adaptive, real-time system. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)*, volume 3424 of *Springer-Verlag, LNCS*, pages 226–241, 2004.

[52] Marc Waldman, Aviel Rubin, and Lorrie Cranor. Publius: A robust, tamper-evident, censorship-resistant and source-anonymous web publishing system. In *Proceedings of the 9th USENIX Security Symposium*, pages 59–72, August 2000.

[53] Rungrat Wiangsripanawan, Willy Susilo, and Rei Safavi-Naini. Design principles for low latency anonymous network systems secure against timing attacks. In *Proceedings of the fifth Australasian symposium on ACSW frontiers (ACSW '07)*, pages 183–191, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[54] Hongyun Xu, Xinwen Fu, Ye Zhu, Riccardo Bettati, Jianer Chen, and Wei Zhao. Sas: A scalar anonymous communication system. In *Proceedings of ICCNMC*, pages 452–461, 2005.

[55] Sebastian Zander and Steven J. Murdoch. An improved clock-skew measurement technique for revealing hidden services. In *Proceedings of the 17th conference on Security symposium*, pages 211–225, Berkeley, CA, USA, 2008. USENIX Association.

[56] Ye Zhu and Riccardo Bettati. Unmixing mix traffic. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2005)*, pages 110–127, May 2005.

[57] zzz and L. Schimmer. Peer profiling and selection in the i2p anonymous network. In *PET-CON 2009.1.*, TU Dresden, Germany, 03/2009 2009.