BAMBERGER BEITRÄGE

ZUR WIRTSCHAFTSINFORMATIK UND ANGEWANDTEN INFORMATIK

ISSN 0937-3349

Nr. 77

Improving the Tor Hidden Service Protocol Aiming at Better Performances

Christian Wilms

November 2008

FAKULTÄT WIRTSCHAFTSINFORMATIK UND ANGEWANDTE INFORMATIK

OTTO-FRIEDRICH-UNIVERSITÄT BAMBERG

Distributed and Mobile Systems Group

Otto-Friedrich Universität Bamberg

Feldkirchenstr. 21, 96052 Bamberg, GERMANY

Prof. Dr. rer. nat. Guido Wirtz

`http://www.uni-bamberg.de/pi/`

Due to hardware developments, strong application needs and the overwhelming influence of the net in almost all areas, distributed and mobile systems, especially software systems, have become one of the most important topics for nowadays software industry. Unfortunately, distribution adds its share to the problems of developing complex software systems. Heterogeneity in both, hardware and software, concurrency, distribution of components and the need for interoperability between different systems complicate matters. Moreover, new technical aspects like resource management, load balancing and deadlock handling put an additional burden onto the developer. Although subject to permanent changes, distributed systems have high requirements w.r.t. dependability, robustness and performance.

The long-term common goal of our research efforts is the development, implementation and evaluation of methods helpful for the development of robust and easy-to-use software for complex systems in general while putting a focus on the problems and issues regarding the software development for distributed as well as mobile systems on all levels. Our current research activities are focussed on different aspects centered around that theme:

- *Robust and adaptive Service-oriented Architectures*: Development of design methods, languages and middleware to ease the development of SOAs with an emphasis on provable correct systems that allow for early design-evaluation due to rigorous development methods and tools. Additionally, we work on approaches to autonomic components and container-support for such components in order to ensure robustness also at runtime.
- *Agent and Multi-Agent (MAS) Technology:* Development of new approaches to use Multi-Agent-Systems and negotiation techniques, for designing, organizing and optimizing complex distributed systems, esp. service-based architectures.
- *Context-Models and Context-Support for small mobile devices*: Investigation of techniques for providing, representing and exchanging context information in networks of small mobile devices like, e.g. PDAs or smart phones. The focus is on the development of a truly distributed context model taking care of information reliability as well as privacy issues.
- *Peer-to-Peer Systems*: Development of algorithms, techniques and middleware suitable for building applications based on unstructured as well as structured P2P systems. A specific focus is put on privacy as well as anonymity issues.
- *Visual Programming- and Design-Languages*: The goal of this long-term effort is the utilization of visual metaphores and languages as well as visualization techniques to make design- and programming languages more understandable and, hence, easy-to-use.

More information about our work, i.e., projects, papers and software, is available at our homepage. If you have any questions or suggestions regarding this report or our work in general, don't hesitate to contact me at `guido.wirtz@uni-bamberg.de`

Guido Wirtz

Bamberg, April 2008

Improving the Tor Hidden Service Protocol Aiming at Better Performances

Christian Wilms

Lehrstuhl für Praktische Informatik, Fakultät WIAI

Abstract Offering services anonymously on the Internet using so-called *location-hidden services* requires complex protocols with many different nodes involved. These properties result in performance problems, e.g. a simple website request taking tens of seconds. This work describes a setup to measure the performance of hidden services using the worldwide *Tor* network. It analyzes the results and proposes changes to the protocol to improve the performance without losing anonymity.

Keywords Tor, anonymity networks, performance, measurements, location-hidden services

Contact: christian.wilms@uni-bamberg.de

Contents

1 Introduction

In the past two decades the Internet has gathered a wide public acceptance. More and more activities of daily life enter the virtual world and completely new ideas are implemented, that would not be possible without the worldwide communication network.

The most common protocol on the Internet and its fundamental technical base is the *Transport Control Protocol* (TCP) in combination with the *Internet Protocol* (IP), ensuring that data packets are routed over many computers to the right destination, potentially on the other end of the world. The TCP/IP protocol stack was developed with respect to numerous performance issues, but ignoring certain other issues, like anonymity or personal privacy.

According to the Internet Protocol all packets sent through networks have a header in addition to the user content. This header specifies the sender and receiver of the packet, more precisely the IP address of both parties. In the current version of the Internet Protocol, IPv4, these 32-bit addresses are assigned to all clients and servers connected to the Internet, the latter with a static long-term IP address, the former usually with a dynamic short-term IP address, assigned by their Internet service provider.

Using unencrypted communication, every router on the path between sender and receiver of a packet can read the data and relate the content to a specific sender IP as well as a specific receiver IP. The problem of revealing data to unauthorized parties can be addressed by using end-to-end encryption, offered by cryptographic protocols like *Transport Layer Security* (TLS), which is widely used for applications like online banking. When browsing the web the security protocol must be supported by the web server and the client. While the user can choose a browser supporting the protocol, he has no influence on the server's decision to support the protocol or not.

But there are many websites and services that do not support encryption, although the transfered data should be protected. If someone wanted to learn something about the symptoms of a specific decease a few decades ago, he probably just took an encyclopedia from his bookshelf and read in it. Today an alternative is to use an online encyclopedia instead. If the person looks the decease up during lunch break in the office using his employers network, the system administrator could easily notify this to the employer. The employer might become curious, if the employee has the decease and if this could be a problem for the company in the future or if he better fires the employee.

It is not only the data sent and received, which can reveal details of a person's privacy, but also the fact which servers one connects to. An example might be an employee who visits websites of self-help discussion groups in the break, because he has some kind of problem, e.g. alcoholism. The employer could learn about it, even though this is completely private and does not affect the employees work.

While the first example could be solved with an end-to-end encrypted connection, encryption does not help in the second case. Although the content of the data packets is encrypted and not readable by anyone else but the specific recipient, the header information about sender and receiver is still plain text.

Both examples have an adversary in the local network in common, but an adversary can also monitor any other node that routes a packet from the client to the server and vice versa or even the server. If a company has a web form on its website to enable employees to suggest improvements, this form can be used for *whistle-blowing*, i.e. uncovering for example malicious behavior of a supervisor, which could cost the company a lot of money. Even if the employee uses his Internet access at home to avoid the adversary in the local network, his computer's IP appears in the logs of the server. With the help of the employee's Internet provider it would be easy to find out the customer, who used the specific IP address at the time when the web form was used. So the supervisor could try to cover his malicious behavior and pressure the employee or even fire him.

In the recent example a simple proxy could help to hide the employee's IP address from the server. The packets are sent from the client to the proxy server, that changes the sender address to its own address and relays them to the server. For the server it looks like the proxy has sent the request and responds to it. After receiving the response the proxy forwards it to the client. So the client cannot be identified by the server, but the proxy is a single point of trust. The client must trust the proxy not to reveal his identity to the server.

That this can be a problem shows the case of Julf Helsingius in 1995. He was the operator of an anonymous remailer, a special proxy for email communication. A user made an anonymous posting about the Church of Scientology and the organization convinced the Finish police to force Helsingius to reveal the true name of the poster. After two more incidents involving Scientology, Helsingius decided to shut the service down.[1]

More recent attempts to provide anonymity try to distribute trust, instead of relying on a single entity. Random paths of network participants are created, who act as multiple proxies in a row. If each node in a path between sender and receiver has only a fraction of the information necessary to route the packet to the receiver, no involved entity except for the sender has the "big picture", i.e. knowing who is talking to whom. A node at the beginning of a route knows the sender, but it does not know where the message goes to except for the very next hop. A node at the end of the route needs to know where to send the message, but it does not know who was the original sender. Therefore the networks provide a feature called *unlinkability*, because sender and receiver cannot be linked by any other node but the sender.

These anonymity networks hide a single user in an anonymity set, consisting of all users of the network. This means the sender of a message could still be found by enumerating all users of the network. Therefore the level of anonymity depends on the size of the network and a network with 10 users provides less anonymity than a network with tens of thousands of users.

All concepts shown above try to provide anonymity for the sender of a message, who is contacting a public server. So the initiator is the client who request information and the server responds usually on the same path in the other direction. This is possible by keeping up a path of nodes through the network for some time, instead of discarding it right after the request. But one can also think of situations when the responder needs to stay anonymous. This means a service, e.g. a website, should be provided without revealing the server's IP address.

Such a so-called *location-hidden service* enables dissidents to write in a weblog about their situation or the situation in their country without the fear of being persecuted by the country's government. In addition to the identity of the author the identity of the server is hidden, so the government cannot simply shut the server down to suppress a disagreeable opinion. The same is possible for journalists in countries, where the freedom of press is less respected than in others. In those countries the government controls the media and might want to suppress independent opinions.

A location-hidden service provides blocking resistance, i.e. information published using the service cannot be blocked easily by others. For the same reason, i.e. not revealing the IP address of a service, it can resist distributed denial of service attacks.

A disadvantage of location-hidden services is that their performance compared to regular non-anonymous protocols can be fairly bad. This is caused by complex protocols, involving a large number of nodes with a lot of encryption and decryption. While requesting a small website from a regular web server usually takes a fraction of a second, requesting the same website over a location-hidden service can take tens of seconds. Therefore anonymity is not simply added as another quality of service, but it is traded off for performance, especially against a low latency during connection setup.

While a user, whose life might depend on their anonymity, accepts a fairly high latency, other users just protecting their privacy when using the Internet have a much lower threshold regarding performance. It depends on how much anonymity is worth for users. For the benefit of all users it is important to increase the usability, of which performance is a part of, of anonymity networks in order to increase the anonymity set mentioned above.[2]

This thesis intends to find ways to improve the location-hidden service protocol of an anonymity network, aiming at its performance.

1.1 Overview of this Work

This thesis is organized in the following way: Section 2 reviews available anonymity systems and protocols that feature location-hidden services or comparable approaches. Further it mentions research in the area of anonymity systems performance. Section 3 describes the Tor anonymity network in detail as well as the protocol for establishing and accessing location-hidden services. Section 4 introduces a measurement environment to measure the performance of all sub steps during the process of accessing location-hidden services using the global Tor network. Section 5 presents a number of performance enhancing changes to the current Tor hidden service protocol and their implementation. Section 6 shows the results of measurements of the changes using the measurement environment and performs statistical tests to prove the significant impact of the implemented changes. Section 7 ends with some concluding remarks and suggests future work based upon this thesis.

2 Related Work

This section describes the protocols of other anonymity systems, comparable to the one that is improved in this paper. Further other performance related work considering the performance of anonymity systems is presented.

2.1 Comparable Systems

Other low-latency anonymity systems and protocols, that provide location-hidden services, are analyzed below. The selection is further limited to systems that use a message-based approach, while document-based approaches are not considered.

This is because document based system merely focus on publishing content anonymously and censorship-resistantly, which is written once and then passively accessed by others. Message based systems are more capable to protect other ways of communication and low-latency interaction between users, e.g. web surfing, instant messaging, or chatting on Internet Relay Chat (IRC).

2.1.1 Onion Routing

Goldschlag et al. introduced in 1996 an approach called *Onion Routing*, hiding routing information by routing a data stream over a number of nodes to the destination. Each node on the path between client and server knows only the previous and next hop in the communication chain.[3] This approach was later used in several ways to design anonymity protocols.

The network consists of a number of routing nodes, which have link encrypted connections to other routing nodes. Some of the routing nodes additionally act as proxy to enter and leave the network.

To start communication with a server an initiator, i.e. the client, connects to a routing node that also provides proxy functionality. This proxy identifies a number of routing nodes to form a route through the network. It builds the *onion*, a data packet surrounded by several layers of encryption together with routing information in each layer. This design, with layers similar to the layers of an onion, gives the approach its name. At the end of the route is again a routing node with proxy functionality, that forwards the data to the responder, i.e. a server.

If the client wants to establish a connection to the server it contacts the proxy node. The proxy node encrypts two pairs of a symmetric key and a function specifying a cryptographic operation, one pair for every direction of communication, with an out-of-band retrieved public key for the last proxy in the route. This packet is encrypted together with the contact information of the last node and two pairs of key and function for the preceding node, and so on back to the first node it will send the onion.

After sending the onion to the first node, the latter decrypts it with its private key and finds the two symmetric keys and functions in there as well as the contact information for the next node and an encrypted packet. So he sends the packet to the next node and keeps the keys for further communication on this route. Since one layer is stripped off the onion for each hop, a random bit string is added to keep the size of the onion equal.

Using the onion a circuit over a number of routing nodes is created. Each node has two symmetric keys as well as two functions chosen by the client's proxy. The proxy can now apply the inverse of the cryptographic operation using the symmetric keys to prepare the actual messages it receives from the client, again in several layers, starting with the last node in the circuit. The message is sent through the circuit and each node applies the cryptographic operation using the symmetric key for the forward direction. At the end the last node finds the plaintext message and sends it to the server. After receiving a response from the server, each node applies the cryptographic operation and symmetric key for the backward direction. The client's proxy again performs the inverse operations and sends the response to the client. Figure 1 shows an overview of all roles involved.

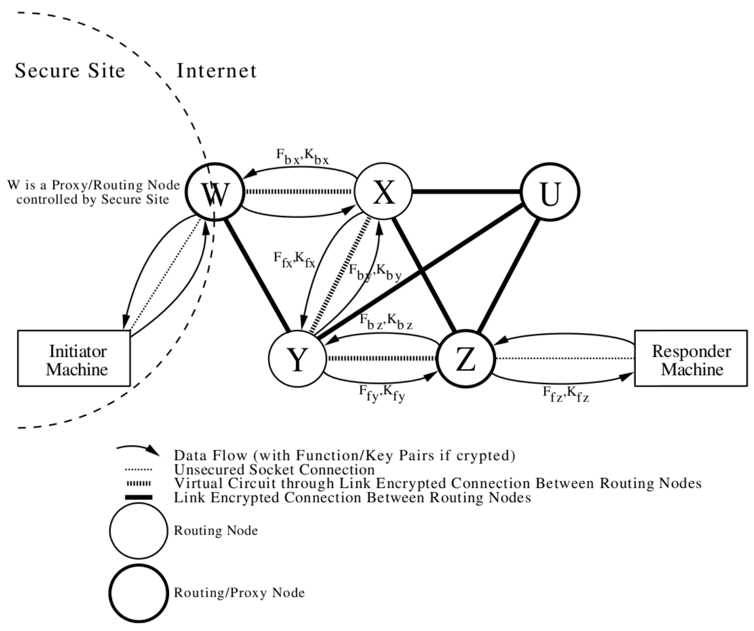
Secure Site
Internet
W is a Proxy/Routing Node controlled by Secure Site
W
X
U
Y
Z
Initiator Machine
Responder Machine
F_{bx}, K_{bx}
F_{fx}, K_{fx}
F_{by}, K_{by}
F_{bz}, K_{bz}
F_{fy}, K_{fy}
F_{fz}, K_{fz}
Data Flow (with Function/Key Pairs if crypted)
Unsecured Socket Connection
Virtual Circuit through Link Encrypted Connection Between Routing Nodes
Link Encrypted Connection Between Routing Nodes
Routing Node
Routing/Proxy Node

Figure 1: Onion routing overview (taken from [3])

The authors mention a possible extension for a completely anonymous communication between two parties. Both parties need to connect to some anonymity server, that mates circuits sharing some token. This is not yet usable for location-hidden services, but the Onion routing approach was a foundation for later systems.

2.1.2 Anonymous IP Infrastructure

In 2000 Goldberg presented the *Anonymous IP Infrastructure* (AIPI).[4] The AIPI enables clients to access public services without revealing their IP address. He also presented an extension to offer a service anonymously.

Clients who want to use the AIPI need an application-level proxy, which routes data from any client application that can use proxies, into the network. The basic component of the AIPI is the so-called *IP Wormhole*, an Internet service, the client can send messages to. One or more dedicated hosts are set up around the Internet, called *Anonymous Internet Proxy* (AIP). Each AIP has at least two IP addresses, a regular Internet address and a Wormhole address.

Messages send by clients to the regular Internet address of the AIP are IP-in-IP messages, i.e. an IP packet contained in another IP packet. The inner packet has a blank source address and the destination is the desired server. The source address of the outer packet is the client IP address and the destination address the Wormhole IP address. The AIP assigns a certain port and the Wormhole IP address to the client IP address and keeps this combination in a table, if more packets from the same client arrive. Then it sets the assigned Wormhole IP address as the source address of the inner IP packet and sends the packet to the desired server.

When the server replies, the destination address is the Wormhole address. The AIP receives the packet, encapsulates it in another IP packet and sends this packet to the client. The client address can be looked up in the table mentioned before.

This scenario hides the client from the desired server, but an attacker reading the IP-in-IP packages can still easily link client end server, because he knows the sender of the outer packet and the destination of the inner package. To prevent this, the inner IP packet is encrypted with the AIP's public key, when the packet is sent to the AIP and with the client's public key, when the AIP forwards a reply of a server to the client.

To prevent an attacker from correlating messages coming in and out of the AIP, packet padding is used to make all packets between client and AIP the same size by adding random data to the inner IP packet before encrypting it. While packets carrying a simple GET command of the Hypertext Transfer Protocol (HTTP) are rather small, the response can be much bigger. To make padding more efficient, multiple packet sizes are used and each packet is packet to the smallest size, in which it fits. In addition link padding is used for packets between client and AIP, which means that also the times when packets are sent are changed. If less packets are sent, dummy traffic can be added and if more packets are sent than fit in the slots, smaller packets can be combined to a packet of larger size. This is to protect the packets against packet timing correlations.

The AIP must still be trustworthy and a malicious AIP or an AIP under control of an adversary would reveal all connections between clients and servers. Therefore not a single AIP is used, but a chain of AIPs. Adjacent pairs of AIPs are connected by secure links, similar to the one between client and first AIP in the chain. But the links between AIPs are permanent, so all AIPs constitute a network, with each AIP having a few links to others. To learn which AIPs are connected clients can retrieve the current *AIP graph* from the *Network Information Database* (NIDB). The NIDB is distributed among all AIPs. Every AIP has a list of its neighbors, the nodes it has secured links to, and exchanges its list with them.

A client can query a subset of AIPs for their lists and construct the AIP graph based on the lists and pick a number of linked AIPs to form a chain. Then it builds a secure link to each of the AIPs in the chain, tunneled over the secure links between the nodes.

To add the functionality of anonymous servers to the AIPI *rendezvous servers* are introduced, which can be part of the AIPI but need not to. To locate a rendezvous server a decentralized database is used, e.g. the Gnutella network. The server can connect to the rendezvous server via the AIPI and hand over a service tag, which is the identifier for the service. The rendezvous point publishes the tag and its own address in the decentralized database. Clients who want to connect to the service query the database and learn the tag and the rendezvous server from it. Then they can contact the rendezvous server and using the tag the server can forward requests to the anonymous server. If clients want to stay anonymous they can contact the rendezvous server via the AIPI also.

2.1.3 Tarzan

Freedman et al. introduced a peer-to-peer anonymous network layer that provides generic IP forwarding called Tarzan in 2002.[5] Packets are routed through tunnels, consisting of a sequence of Tarzan nodes, and they are encrypted in layers hop-by-hop. On one end of a tunnel is the client requesting a service and on the other end is a Tarzan node running a network address translator (NAT), which forwards the unencrypted request to the original destination, e.g. a web server.

The client who seeks anonymity creates the tunnels. First it needs to know which other peers are available. Tarzan uses a distributed hash table (DHT) and the Chord algorithm. So the directory is distributed over all Tarzan nodes, forming a logical Chord ring based on a cryptographic hash of their IP address. This hash is used as key in the DHT. The hash is created using a one-way function, so it is not possible to rebuild the IP address from the hash. Each node learns about a few neighbors after initially contacting a single node.

To randomly pick nodes for tunnel creation, the client can simply generate random lookup keys and send a request to the matching node or the node, whose hashed IP is the closest successor of the lookup key. The node that is found answers with its IP address and public key.

After collecting the contact information of a number of random peers the client can start building the tunnels. It does so by sending a User Datagram Protocol (UDP) packet to the first node, encrypted with the receiver's public key. The packet includes a symmetric session key for further communication. The receiver responds with an acknowledgment encrypted with the symmetric key.

To add a second hop to the tunnel, the client sends another symmetric key through the tunnel, now encrypted for the second node and encapsulated with the first symmetric key. The first hop in the tunnel cannot distinguish if the packets, he is relaying, are already data packets carrying user data or control packets for tunnel creation. It can further not identify if the sender is the originator of the tunnel or just another relay. The first hop just strips one layer of encryption off the packet and relays the still encrypted content to the IP it also finds in the packet.

The following hops are built accordingly. In the Tarzan client an IP forwarder intercepts the packets coming from user applications e.g. a web browser, sanitizes the IP header and routes them through the tunnel. The last hop in a tunnel uses the built in pseudonymous network address translator (PNAT) and sends the packet to the destination. For the destination it looks like the packet originates at the last hop of the tunnel. Figure 2 shows an overview of the architecture.

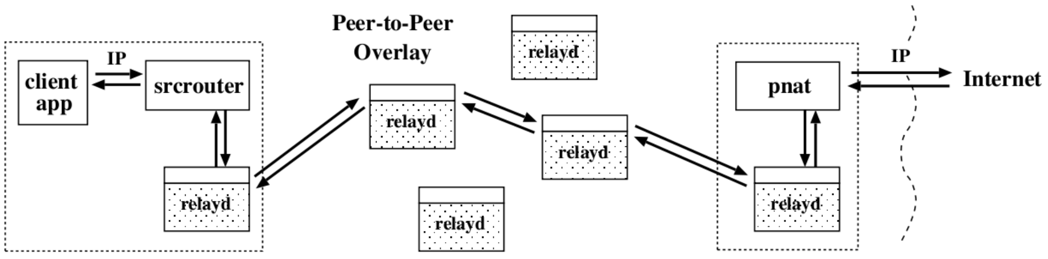

Figure 2: Tarzan architecture overview (taken from [5])

To offer a service anonymously instead of accessing one, the PNAT at the end of a tunnel can be configured to forward ports. So even regular clients not using Tarzan can access the service, as it appears to be hosted on the node at the end of the tunnel. Tarzan clients can also access an anonymous server anonymously by using two tunnels ending at two different PNATs, one representing the server and the other one representing the client. But it is important to mention that such a connection would not be end-to-end encrypted, because the messages between the two PNATs at the end of both tunnels are not secured by Tarzan.

The authors have measured the performance of Tarzan and found a latency of 68.37 milliseconds on average for building a tunnel consisting of three hops. Varying the length of tunnels showed, that each hop adds another 20 milliseconds to the build time. In these measurements looking up other nodes in the Chord table is not included, but the node information is fetched in advance. When the node information is fetched on demand, the latency for a 3-hop tunnel is on average 106.70 milliseconds. Measurements concerning the latency a client experiences using Tarzan to access a public server anonymously or accessing a location-hidden service are not presented.

2.1.4 I2P

The *Invisible Internet Protocol* (I2P) is an open source peer-to-peer network started in 2003.[6] It provides anonymity for typical Internet activities by routing traffic through other network nodes, using separated tunnels for inbound and outbound messages. I2P is capable of anonymizing activities like web browsing, chatting on IRC, file transfers, file sharing and email, as well as web hosting. The network does not hide the usage of I2P, but what the user is doing and who he is communicating with.

After starting a client it retrieves the information about other routers from a distributed hash table, the *network database*, including public keys for encrypting and signing messages, contact information in form of IP address and port, a time stamp of publication, and signature against all data to ensure its authenticity. To reach the hash table, which is distributed over all routers, the client needs to know only a single peer, that can be asked for other routers. To find this single peer a seeding Uniform Resource Locator (URL) is available.

With the information, which routers are available, the client can start to build tunnels. Usually a client has five to ten outbound tunnels and two inbound tunnels. Those are the tunnels for client communication. In addition exploratory tunnels are opened for tunnel maintenance and network database maintenance. While exploratory tunnels are selected randomly from the set of known peers, client tunnels need to fulfill certain requirements like being fast and having a high capacity. The tunnels have a life time of 10 minutes, when they are discarded and new tunnels are built. The default length of tunnels is two hops.

Both inbound and outbound tunnels have a tunnel gateway, where messages enter the tunnel, and a tunnel endpoint, where messages leave the tunnel. That means the client serves as a gateway for outbound tunnels and as an endpoint for inbound tunnels. Figure 3 illustrates the usage of tunnels in I2P.

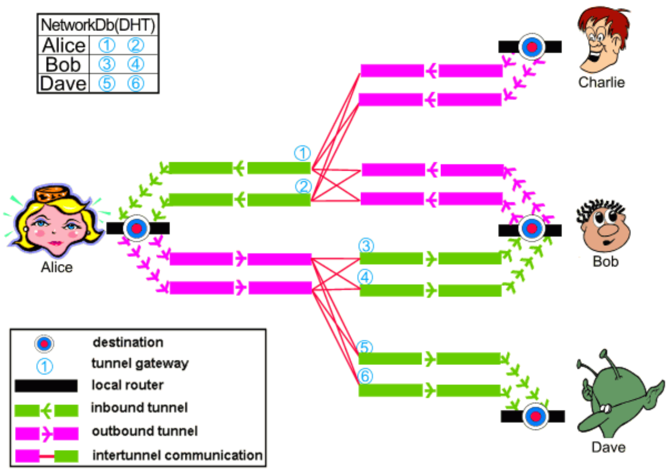

Figure 3: I2P overview (taken from [6])

When the tunnel gateway receives messages, it encrypts them in several layers of encryption, one layer for each hop in the tunnel. It is also possible, that the gateway accumulates several messages and sends them together through the tunnel. This is one feature of so-called *garlic routing*. Another feature is to include gateway information of an inbound tunnel to a message that is sent to another client. If the other client wants to answer that message it does not need to query the network database, where the gateway information of all peers is available. Instead it can use the gateway information included in the message it has received.

Due to the peer-to-peer character of I2P every client automatically takes part in the network when connecting. That means it also receives and sends traffic of others. The peers communicate with each other using the UDP protocol.

To deploy a location-hidden service within I2P the client creates a destination key pair. The private key is used for authentication of the service, while the public key serves as identifier. This identifier is also uploaded to the network database together with gateway information and a human readable domain name, ending with the top level domain *.i2p*. In addition the physical service must be deployed on the machine, e.g. a web server to host an anonymous website or an IRC server.

When a client wants to access an .i2p domain while being connected to the I2P network, it retrieves the contact information from the network database and can therefore connect to the gateway of the service, that relays requests to the client providing the service.

2.1.5 Peer-to-Peer Personal Privacy Protocol

In 2002 Sherwood et al. presented a protocol for scalable anonymous communication, called P^5 for Peer-to-Peer Personal Privacy Protocol. The protocol provides anonymity for senders and receivers of messages.[7]

The basic idea underlying the protocol is a global broadcast channel. That means that a message sent to the broadcast group is received by every member of the group. The messages are encrypted with the public key of the receiver, so that only the desired receiver can decrypt and read the message. All participants send fixed-size packets at a fixed rate. Even if they do not communicate with anyone they send noise packages. Otherwise the package contains useful information and is called a signal package. As last feature the broadcast is organized as a peer-to-peer ring with spoofing the sender addresses and with hop-by-hop encryption, so that outgoing messages cannot be mapped to a particular incoming message of a node.

An adversary cannot distinguish between noise and signal packages and therefore cannot find out who is actually communicating. Because everyone in the group receives a message, the adversary cannot determine, who is the actual recipient of the message and because of the sender spoofing and hop-by-hop encryption the sender of a message is not revealed. In addition this idea provides unlinkability, that means nobody can find out, if and which two users are communicating.

But the idea has one major problem. It does not scale. When the broadcast group increases, the available bandwidth for useful messages decreases linearly. Dividing the group into smaller separated groups would help, but if two users end up in different groups, they would not be able to communicate.

The authors solve that problem by creating a hierarchy of broadcast channels. Users can locally choose different levels in the hierarchy and thereby trade-off between anonymity and communication efficiency. Foundation of the hierarchy is the bitstring of the user's hashed public key. Groups in the hierarchy are determined by the prefix of the bitstring, whose length is set by a bitmask. Figure 4 shows the upper levels of the hierarchy in the format *(bitstring/bitmask)*, with the null-bitstring as root.

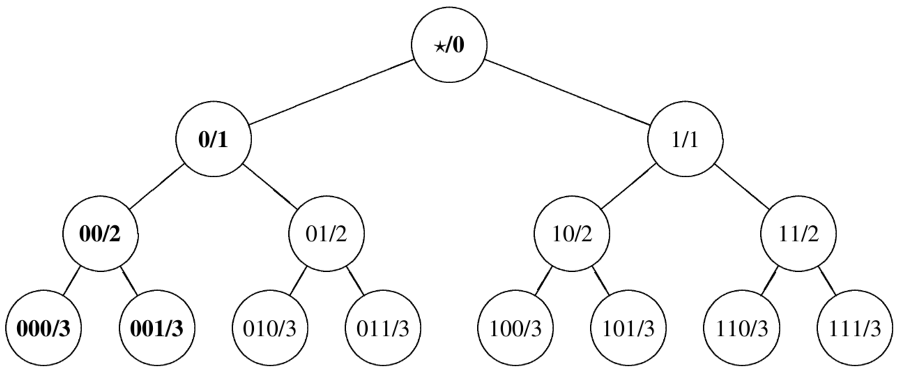

Figure 4: P^5 Logical broadcast tree (taken from [7])

A message sent to a certain group reaches all members of that group, all members of the ancestor groups, i.e. the groups that have a lower bitmask and their bitstring is a prefix of the targeted group, and all members of the descendant group, i.e. the groups with a higher bitmask and the bitstring of the targeted group is a prefix of their bitstring. A message to the root is sent to all users in all groups, while a message to the group (00/2) is received by members of the bold groups in Figure 4 only.

If two users want to communicate, they need to obtain each others public key out-of-band. They can create a bitstring from it, and choose a common group, i.e. not necessarily the group both users are in, but a group both receive messages from. But with only one key per user there is a 50 % chance that the lowest common ancestor is the root. Messages sent through that group have a maximum of anonymity, but also have higher chance to get lost, because peers are able to drop messages, if they receive more messages than they can handle.

Therefore clients create two additional locally generated *routing keys*, to join more groups in different parts of the tree. When joining a group, they periodically send messages to the group, listing other groups they are in.

The P^5 protocol has not yet been implemented in the real world, but the authors provide a packet-level simulator to analyze the behavior of the protocol for certain parameters. They show that the number of public key decryptions on every node can easily handled by current processors. The bandwidth necessary to communicate in a network with 8192 nodes at hundreds of kilobits per second is approximately 2 megabyte per second. This is because of up to 40 % packet loss. This is low compared to a pure broadcast system with the same parameters. The group size in this scenario is at least 100 members. Latency is not considered in the simulation.

2.1.6 Anonymous Peer-To-Peer File Sharing Protocol

When developing the *Anonymous Peer-to-peer File Sharing* (APFS) protocol [8] in 2001, Scarlata et al. adapted existing protocols for initiator anonymity to provide responder anonymity focusing on file-sharing.

Using a proxy node it is possible to achieve responder and initiator anonymity and therefore mutual anonymity with existing initiator anonymity protocols. The responder, i.e. provider of a service, creates a connection to the proxy using the existing protocol and sets up a public alias. So the proxy only knows the alias but not the identity of the responder. An initiator also builds a connection to the proxy anonymously, handing over the alias. The proxy forwards the request over the anonymous channel to the responder, who sends the reply the same way back. Both roles connected anonymously to the proxy, so the latter does not know either one of them. But of course the proxy is a bottleneck and it needs to be trusted not to drop all request packets in a denial-of-service attack.

The authors divide the APFS protocol in two stages. In the first stage peers create an network providing initiator anonymity. That means they build connections over other peers using layered encryption as described in Section 2.1.3. In this stage anyone is allowed to see that the clients participate in the network. In the second stage peers anonymously query anonymous servers, which answer by providing contact information of other peers, that have the queried files available.

The initialization of the APFS protocol starts with a coordinator, whose IP and public key are known to all clients and which is the bootstrapping node. Clients joining the network notify the coordinator and receive a list of other nodes available for forming anonymous routes. After the initialization willing peers contact the coordinator anonymously and offer to act as server. The offer includes a unique ID, the address of a tail node, i.e. the node at the end of an anonymous route. Acting as a server means that other clients can send lists of available files to them and that clients can query for specific files. If the server knows a client providing the specific file, it responds with the client's tail node, so that the requesting client can connect to to the client providing the file.

Clients can learn about the servers from the coordinator, and send lists of shared files through anonymous tunnels to the servers, including unique IDs and tail node addresses. When querying servers, the queries can be sent to different servers to avoid heavily loaded or poorly connected servers. Before leaving the network clients notify the servers that they are not sharing files anymore.

In a variation of the protocol the authors propose a multicast solution for bootstrapping. This adds some complexity to the protocol, but does not need a central coordinator. An initiating node publishes a multicast address on a website or mailing list together with a join time. Other nodes interested in joining subscribe to the multicast group and start sending overt messages to the multicast address periodically. By receiving the messages sent to the address each node learns about the others. A peer willing to act as server just sends a message to the multicast address instead of the coordinator in the other variation, now using an anonymous tunnel. The search for files works like in the variation with a coordinator.

The performance of the APFS protocol was measured using a simulation. The transfer latency for a 2 megabyte file from one peer to another increased by 65 % comparing a path length of 0, without any anonymity, and a path length of 2. Increasing the path length to 4 results in another 13 % increase of transfer latency. The absolute average latency for the measurements with a path length of 2 is 11 seconds at 1 % loss in all links.

2.2 Anonymity Systems Performance Research

Research analyzing the performance of anonymity network as part of usability has become more important in the last years amending mandatory security and anonymity research. Most papers still focus on initiator anonymity only and do not consider location-hidden services. Nevertheless they are important to mention here, because the location-hidden services analyzed in this thesis base on the same foundations examined in the papers.

Dingledine and Matthewson established a basis for usability research of anonymity systems in 2006. In a number of case studies they showed the importance of usability. Because the best network design does not provide anonymity if it cannot attract enough users to create a sufficient anonymity set.[2]

Also in 2006 Köpsell examined the relation between the delay of message transfer resulting from using an anonymity system and the number of users. He found that the number of users decreases linearly as the delay increases. This decrease affects the anonymity set and therefore the anonymity the system provides. As a result designers of anonymity systems should not disregard the performance issue.[9]

Wendolsky et al. compared the latency of the Tor network, which is also subject of this thesis, to another low-latency anonymity network, called AN.ON, in 2007. AN.ON does not offer location-hidden services and was therefore not mentioned in this thesis. The authors found that Tor is subject to unpredictable performance with a high variability of latency. The average time to anonymously retrieve a website was measured at 4.231 seconds in the afternoon and little lower 3.790 seconds in the morning.[10]

In 2008 Panchenko et al. proposed new methods of path selection in Tor. The methods are based on measurable performance metrics like latency or throughput, instead of self-advertised values. Using the proposed path selection method the performance experienced by a user could be increased. The time to access a public website anonymously via Tor decreased from 4.04 seconds on average using the default Tor path selection to 1.35 seconds with the improved methods.[11]

3 Tor and Tor Hidden Services

In this diploma thesis enhancements of a location-hidden service protocol are developed. The anonymity network selected for those enhancements is the Tor network. This network is chosen for a number of reasons. Unlike other location-hidden service protocols the Tor protocol has been implemented and is widely in use. So measurements in the global network can be performed intending to measure the performance a regular user experiences, instead of simulating the usage of the protocol. Tor supports a wide range of applications, anonymizing data on the TCP protocol level. Further Tor is free software and its implementation is well documented. There is also a supporting community and it is analyzed in a number of scientific papers.

To comprehend the changes proposed in this thesis it is necessary to understand the way how Tor provides anonymity in general and how the Tor hidden service protocol works.

3.1 Tor

Tor is an overlay network built upon the Internet. That means, that it consists of a number of nodes, so called *relays* that are deployed on regular servers. These relays run as client-level processes and are used as proxies to provide different forms of anonymity.[12]

One form of anonymity is the anonymity of a client accessing a public service e.g. a web site. In this scenario neither the web server, nor any other router on the way between user and server should be able to link this particular client with the server access.

The Tor software was deployed in October 2003. The network had no down-time since then and currently consists of approximately 2,000 nodes worldwide. It is the largest distributed anonymity network currently in use with a weekly estimated 200,000+ users.

The Tor client builds so-called *circuits* consisting of three relays to contact an external server. The relays are all chosen before building the circuit. To find out which relays are available, the client retrieves a list of relays from a directory server, including their IP address, port number, and public key. The connection with the directory server is a regular TCP-connection. This only leaks the information that the user is using Tor, but not what services he is going to use.

Communication between Tor relays takes place in form of fixed-size cells, send over TLS secured connections. Using TLS prevents an adversary from impersonating a relay or modifying data. When the user requests a website, the client first establishes a TLS connection to the first relay, which is called entry node. To negotiate a symmetric session key with the entry node, the client sends the first part of the Diffie-Hellman handshake down the TLS connection. The entry node performs the second half of the handshake and replies it. Now both nodes share a common secret to generate a 128-bit session key for further communication, using the Advanced Encryption Standard (AES) algorithm. Although both parts of the handshake were send over the network, they would not help an adversary to learn the secret.[13]

To extend the circuit, the Tor client encrypts and sends a cell to the entry node to open a secured connection to the second relay, called middle node. The entry node decrypts that cell and sends a create cell to the desired node, followed by another Diffie-Hellman handshake between the client and the middle node, with the entry node relaying the messages between both.

The next step is, to create a message for the middle node to open a secured connection to the third relay, called exit node. This message is encrypted for the second relay and enhanced by contact information of the same relay. Both is encrypted for the first relay and sent to the first relay. The entry node decrypts the message and finds another encrypted message within along with the contact information of the middle node. Therefore it forwards the encrypted message to the middle node. The same form of layered encryption is applied when extending the circuit to the exit node.

After building the circuit it can be used to retrieve a website or send a message. The client prepares the message and wraps it in three encryption layers. The outermost layer is always stripped off by the three relays, first by the entry node, then by the middle node and at last by the exit node. After the exit node has encrypted the message it received, it finds the original message along with the contact information of the server to send it to.

Figure 5 shows a circuit over three relays between the client in the upper left and the public server in the lower right corner. The directory server is in the lower left. Solid lines represent encrypted connections, while dashed lines are unencrypted connections. The onion identifies nodes with Tor installed, either as client, relay or directory.

When the exit node sends the message to the server over an unencrypted TCP connection, for the server it looks like the message originates from the exit node. After receiving a reply from the server, it is sent down the circuit again, encrypted with the corresponding session keys by each node until the packet arrives at the client and he can decrypt it to read the reply.

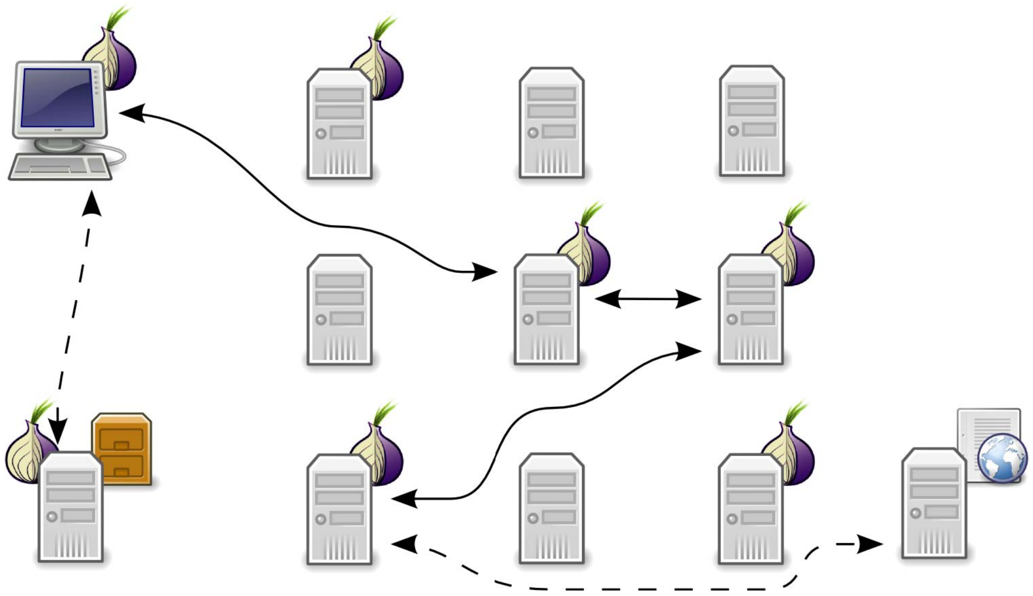

Figure 5: Tor overview

3.2 Hidden Service Protocol

This section describes the protocol underlying Tor hidden services. It explains all steps necessary to first establish a hidden service and later access that service. Further the different roles involved in the process and the messages, that are sent between nodes, are examined. This is essential to understand the measurement environment, presented in the next section as well as the changes proposed later.

The whole process to transfer content from a hidden server to a client can be divided in two parts: the establishment of a hidden service and the actual access by a client. The following description assumes that a fictional user Bob wants to offer a service without revealing his real location and therefore uses the hidden service feature of Tor. Another user called Alice wants to access this service.

3.2.1 Establishing a Hidden Service

The establishing part occurs once per hidden service at the very beginning. It makes the service available for clients. If it is successful, clients can access the service until it is shut down. The idea is to select other relays in the Tor network, that can be contacted by clients. These nodes are called *introduction points*. When a hidden server is started, it selects three nodes and builds a circuit to them. That means that two other relays are between hidden server and introduction point. Figure 6 shows the process of establishing a hidden service, which is explained in detail below.

All Bob needs to do to establish a hidden service is to set up the actual service, e.g. a web server or an IRC server and to start the Tor client, which is configured to provide a hidden service.

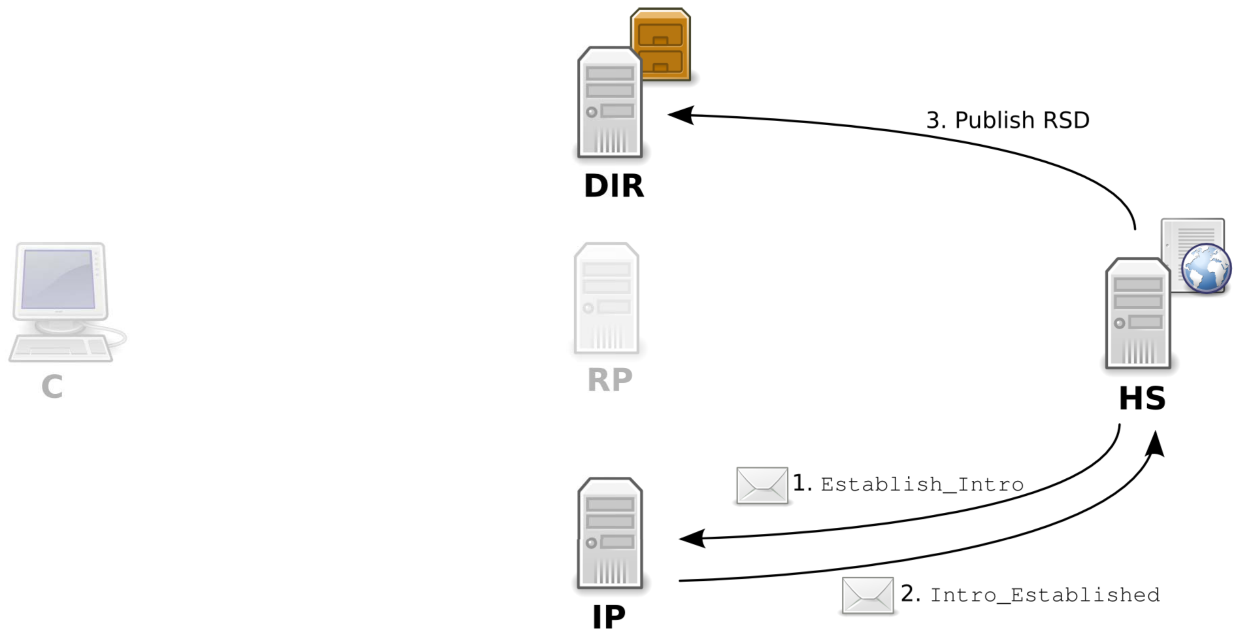

Figure 6: Establishing a hidden service

As soon as the circuit is built the hidden server sends an `Establish_Intro` cell over the circuit, containing the public key of the hidden server besides some signature information. The introduction point checks that signature and associates the circuit with the public key received in the cell.

During the hidden service access, clients will contact the introduction point and the hidden server's public key is used to find the right circuit, especially if the relay acts as rendezvous point for more than one hidden service.

After accepting its role, the introduction points acknowledges with an `Intro_Established` cell over the circuit.

After receiving the acknowledgments the hidden server builds the *rendezvous service descriptor*. This descriptor includes contact information about the three introduction points and the hidden server's public key. The hidden server uploads the descriptor and a unique identifier for the service, the onion address built from a random hash, to two of six hidden service directories. The directory consists of two sets with three directory servers each.

When the rendezvous service descriptor is available at the directories, clients can start to access the hidden service. To do so they need to know the onion address, which is distributed out-of-band, from a website listing hidden services or from a posting on a mailing list.

3.2.2 Accessing a Hidden Service

Accessing a hidden service is the part of the hidden service protocol, that matters to the user regarding performance. If the establishment of a service would take a lot of time, the user would not even notice, but the access protocol starts when the user clicks on a link on a web site and ends when the desired web site appears on his screen.

If a user wants to access a hidden service, he must have learned the onion address before. He might have found it on a web site or read it in an email. It's just like a regular domain name that one first needs to know before one can access it.

When the user types the onion address in the address bar of his browser or clicks on an HTTP or IRC link, the Tor client recognizes the request by the *.onion* notation and knows that it has to act different from the default Tor protocol, i.e. start the hidden service protocol.

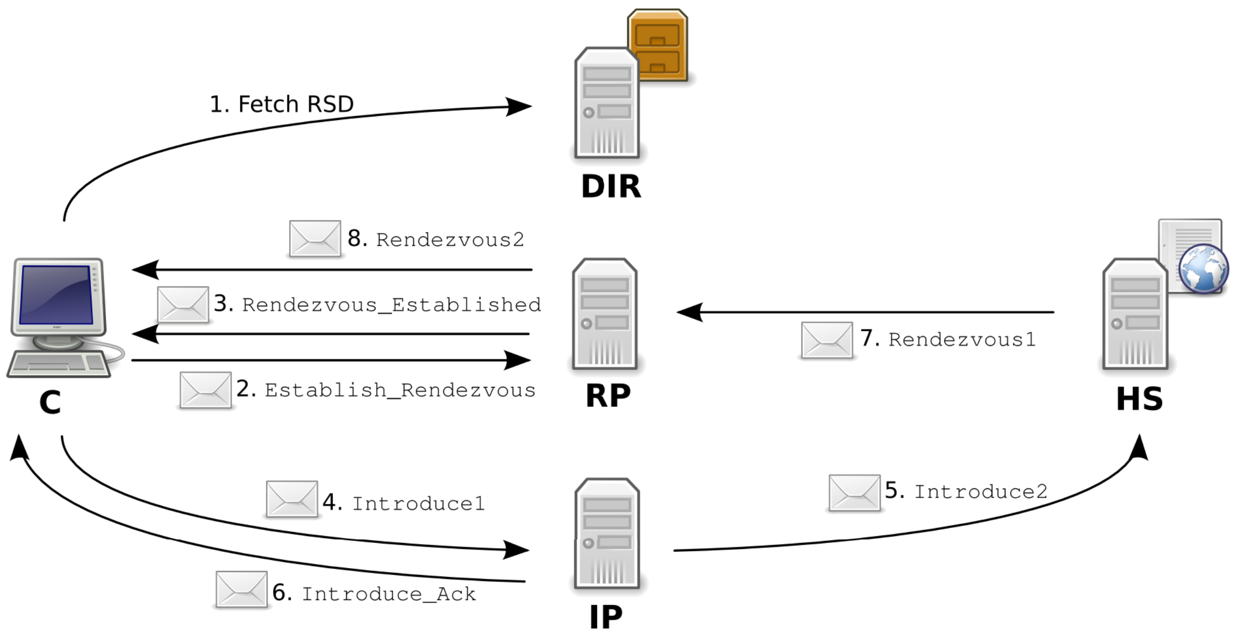

Figure 7: Accessing a hidden service

First the Tor client needs to know how to contact the hidden server at all. More precisely it needs the contact information of the introduction points of the hidden service. Those are listed in the rendezvous service descriptor mentioned in the last section. To retrieve the descriptor the client sends a request to one of the hidden service directories, using a 3-hop circuit. So the request is processed like a regular Tor connection described in Section 3.1.

After receiving the rendezvous service descriptor the Tor client randomly picks one of the introduction points it finds in the descriptor and builds a circuit to the introduction point. To accelerate the circuit building, so-called *cannibalization* can be used. That means, that an existing 3-hop circuit is extended to a desired relay, in this case the introduction point. Obviously this only works, if there is a usable circuit available to be cannibalized. Otherwise this circuit first needs to be built hop by hop.

While contacting the introduction point, another very important process takes place. Alice's Tor client selects another relay to act as *rendezvous point.* This rendezvous point plays a central role for the later data transfer between Bob and Alice. To establish a circuit to the rendezvous point, again cannibalization is used. But this time no 3-hop circuit is extended to the desired relay, but the third hop of an existing 3-hop circuit is chosen as rendezvous point. The advantage is that the rendezvous circuit is immediately built and Alice's Tor client can send an `Establish_Rendezvous` cell over the circuit. This cell contains a random rendezvous cookie for identification purpose.

The rendezvous point saves the rendezvous cookie and acknowledges its new functionality by replying with a `Rendezvous_Established` cell on the same circuit.

To proceed Alice's client needs to check two conditions. One is that the introduction circuit is successfully opened. The second condition is that the rendezvous point has replied with the acknowledgment cell. If both conditions are true, Alice's client sends an `Introduce1` cell over the introduction circuit. This cell contains contact information of the rendezvous point and the rendezvous cookie.

The introduction point receives the cell and forwards the content in an `Introduce2` cell to the hidden service. Afterwards it sends an acknowledge back to the client, the `Introduce_Ack` cell. When the hidden service receives the `Introduce2` cell it can decide if it wants to let the client access the service. If the decision is positive, the hidden service uses the contact information found in the `Introduce2` cell to contact the rendezvous point. First it builds a circuit to the relay acting as rendezvous point using cannibalization again. In this case an existing 3-hop circuit is extended to the rendezvous point. When the circuit is open, a `Rendezvous1` cell is sent down the circuit, including the rendezvous cookie.

When receiving the `Rendezvous1` cell the rendezvous point uses the rendezvous cookie to find the matching client circuit and connects it with the one it just received the cell over. At last the rendezvous point sends a `Rendezvous2` cell back to the client to notify it that a connection to the hidden service has been successfully established and is now ready to transfer TCP user data, e.g. an HTTP GET to retrieve a website.

In Section 2.2 the average latency of Tor, when anonymously retrieving a public website, is approximately four seconds. While for this use case only a single circuit is used, hidden services in Tor require more circuits and more nodes involved. Therefore the time from requesting a hidden service until receiving the reply is 24 seconds on average as shown in earlier measurements.[14] The variability is fairly high, with values over 2 minutes occurring.

This thesis intends to lower the latency of Tor hidden services and does not consider the other main performance attribute of computer networks: throughput. While latency is defined as the transfer time of a packet from one end of a network to another, throughput describes the amount of data transfered through the network in a certain time period.[15] Throughput is not examined, because most applications on the Internet rather require more connections handling small amounts of data, e.g. web surfing or instant messaging, than fewer connections handling big amounts of data, like file transfers.

Latency in this thesis is considered from a user's perspective. That means it is the time between sending a request to a hidden service and receiving the response, even though not a single message constitutes this value but a lot of messages between many different network nodes.

4 Measurement Environment

To measure the performance of most of the sub steps necessary to access a hidden service in detail a measurement environment is set up. This environment is based on the default Tor implementation, version *0.2.0.7-alpha*, retrieved from the official Tor subversion repository [1]. This version is chosen because in the following version *0.2.0.8-alpha* a new directory protocol is introduced. While having no direct impact on hidden service access, the consensus between directory nodes takes longer, up to 30 minutes. This makes measurements in private Tor networks with own directory nodes more difficult.

The basic idea of the environment is to enable repeatable measurements of hidden service access. Because all messages are encrypted, messages are not observed on TCP packet level, but the logging events generated by Tor during run time are analyzed.

After choosing the logging statements for observation purpose, it is necessary to own as many roles in the process as possible, because access to the log files is necessary to analyze them. The idea to set up a private Tor network is rejected, because it is expected that having all nodes in a private network or even on the same machine would affect message transfer times between nodes. The times measured in such an environment would be much lower than those experienced by a regular user of hidden services on the Internet.

So measurements have to take place "in the wild", actually using the same servers for hidden service access that everyone else in the Tor network uses. The number of nodes controlled by the author is minimized to those relays hosting the most important roles in the hidden service access process. First of all there is the Tor client, representing the user, accessing the hidden service and controlling most of the process like path selection or selection of routers to communicate with. On the other end of the process a Tor client configured as hidden service serves the request. In between two further Tor clients are configured as relays to act as rendezvous point and introduction point for the service.

Two main types of changes are made to the roles: regular configuration via the *torrc* configuration file of each node and changes in the implementation to change the behavior in a way that the configuration file is not capable of.

4.1 Client

Besides the default out-of-the box configuration of a Tor client in the *torrc* file the following changes are made. Logging is enabled and the output on info level is pointed to a certain file. A data directory is also specified to keep keys and router descriptors. Those two changes are necessary because three of the four roles are hosted on the same machine. To keep them completely separated within the machine all temporary and long-term data goes into separate directories. The configuration offers an option to select preferred nodes as rendezvous point. This is enabled and the identity of the rendezvous point is specified. To identify all logging statements, SafeLogging is disabled. It is enabled by default and removes potentially sensitive

[1]available at https://tor-svn.freehaven.net/svn/

strings like addresses from the logs. All options can be found in Listing 1.

```
1 SocksPort 9050
2 SocksListenAddress 127.0.0.1
3 Log info file .../log/infothree.log
4 RendNodes $094C0337F5A03A9D62B35358DDD9F3A8E48FF23B
5 SafeLogging 0
```

Listing 1: Client configuration

In addition to the configuration changes some changes to the source code of Tor are necessary to make the measurements work.

The RendNodes option to use a specific relay as rendezvous node is available, but does not work in case of cannibalization. The implementation expects an existing 3-hop circuit with the chosen relay at the end. But in the beginning the option is not considered and three internal circuits to other relays are opened.

To make it work, the following change is applied to a function in `circuituse.c`. In the `circuit_predict_and_launch_new()` function the algorithm to open new circuits before they are needed, checks if the number of existing internal circuits is zero and the RendNodes option is set. If this is the case an internal circuit to the rendezvous point of the measurement environment is opened, using its identity hash in line 3. That means that the first internal circuit ends at the rendezvous point, and is available for later cannibalization. If the first internal circuit is already open, the default path selection algorithm is applied. The change is shown in Listing 2.

```
1 if (options->RendNodes && num_internal < 1) {
2   circuit_launch_by_nickname(CIRCUIT_PURPOSE_C_GENERAL, 0,
3       ”$094C0337F5A03A9D62B35358DDD9F3A8E48FF23B”,
4       hidserv_needs_uptime, hidserv_needs_capacity, 1);
5 } else {
6   circuit_launch_by_router(CIRCUIT_PURPOSE_C_GENERAL, 0, NULL,
7       hidserv_needs_uptime, hidserv_needs_capacity, 1);
8 }
```

Listing 2: Client code to pick rendezvous node

But if the circuit is launched by nickname the regular length for general purpose circuit is four hops. To have the later rendezvous point at the end of a 3-hop circuit, the additional line 8 in Listing 3, affecting the `new_route_len()` function in `circuitbuild.c`, is necessary.

When the client receives the rendezvous service descriptor from the directory server it finds all three introduction points selected by the hidden service in there. Usually it randomly picks one of those three in function `rend_client_get_random_intro()` in `rendclient.c`, i.e. a random value between zero and two is assigned to the variable `i` in Listing 4.

To always use a specific introduction point this behavior is changed. The client now iterates through the list of introduction points in the rendezvous service descriptor and searches for the identifier of the introduction point used in the measurements. The identifier is shown in line 4.

```
routelen = 3;
if (exit &&
    purpose != CIRCUIT_PURPOSE_TESTING &&
    purpose != CIRCUIT_PURPOSE_S_ESTABLISH_INTRO &&
    /* CW If a particular router is chosen and the purpose is C_GENERAL, the
     * default path length is 4. I want it to be 3 for the rendezvous
     * circuit. */
    purpose != CIRCUIT_PURPOSE_C_GENERAL)
  routelen++;
}
```

Listing 3: Client code to decrease circuit length

```
int own_intro_pos;
own_intro_pos = −1;
char own_intro[] =
    "$68333D0761BCF397A587A0C0B963E4A9E99EC4D3";
int x;

for (x = 0; x < entry−>parsed−>n_intro_points; ++x) {
  if (!strcmp(own_intro, entry−>parsed−>intro_points[x])) {
    own_intro_pos = x;
  }
}

if (own_intro_pos > −1) {
  i = own_intro_pos;
} else {
  i = 0;
}
```

Listing 4: Client code to choose introduction point

If it finds that specific identifier the associated position is chosen. Otherwise something went wrong in terms of measurements and the first introduction point is chosen.

4.2 Hidden Service

The Tor client configured to host a hidden service reads the following configuration options from its torrc file. Like for the client own directories for logging and key storage are specified and SafeLogging is disabled.

The port listening on local application connections is disabled, because the hidden service node shall only respond to hidden service request coming from the client node.

To enable the hidden service functionality a directory for long-term hidden service information is specified as well as a port to forward hidden service requests to. With the HiddenServiceNodes option the given node is preferred as introduction point.

The bandwidth for relayed traffic is limited to 20 kilobyte per second, with bursts up to 80 kilobyte per second and the relay is not allowed to act as exit relay. So the relay can be used by everybody as first or second hop in circuits, as introduction and rendezvous point, but not as exit relay to access public websites or services. This is to protect the relay, because if Tor is used to cover any illegal activity, the IP address of the exit node appears in the log statements of a server and the computer might be subject to prosecution by law enforcement. This can interrupt or more likely even abort the measurements and should therefore be avoided.

The ORPort is set to 9001 to listen on that port for Tor connections and a contact information is provided as well as a nickname for the relay. The combined configuration options are presented in Listing 5

```
SocksPort 0
Log info file .../log/infoone.log
SafeLogging 0
DataDirectory .../data/one
HiddenServiceDir ...data/one/hidden_service
HiddenServicePort 80 127.0.0.1:80
HiddenServiceNodes $68333D0761BCF397A587A0C0B963E4A9E99EC4D3
Nickname testorone
RelayBandwidthRate 20 KB
RelayBandwidthBurst 80 KB
ContactInfo Delvey <delvey AT gmx dot de>
ORPort 9001
ExitPolicy reject *:*
```

Listing 5: Hidden service node configuration

4.3 Rendezvous Point

The Tor client acting as rendezvous point is configured as a regular Tor relay without any client or hidden service specific configuration. The relay gets a different nickname, logs into an own file, gets its own directory for temporary data and listens on a different port for Tor connections. See Listing 6 for complete options.

```
SocksPort 0
Log info file .../log/infotwo.log
SafeLogging 0
DataDirectory .../data/two
Nickname testortwo
RelayBandwidthRate 20 KB
RelayBandwidthBurst 80 KB
ContactInfo Delvey <delvey AT gmx dot de>
ORPort 9002
ExitPolicy reject *:*
```

Listing 6: Rendezvous point configuration

4.4 Introduction Point

The Tor client configured as introduction point is set up on a different machine than the other three roles. It could use a similar configuration like the rendezvous point. But the server is not set up primarily for the measurements, so some options like those for bandwidth differ from the rendezvous server, as it is shown in Listing 7. This relay is in addition configured as a Tor directory server. But this functionality is not used in the measurements and the according configurations are not shown in the listing.

```
SocksPort 0
log notice file /home/tor/gabelmoo/notices.log
log info file /home/tor/gabelmoo/info.log
DataDirectory /home/tor/gabelmoo/
Nickname gabelmoo
BandwidthRate 2 MB
BandwidthBurst 2 MB
ContactInfo 1024D/F7C11265 Karsten Loesing <karsten dot loesing AT gmx
ORPort 443
ORListenAddress 0.0.0.0:9001
ExitPolicy reject *:*
```

Listing 7: Introduction point configuration

4.5 Physical Setup

Three of the four roles are set up on the same machine. This is possible, because they never get in direct contact with each other, but always with two or three other Tor relays in between. Figure 8 visualizes the circuits used for hidden service access, following the regular protocol.

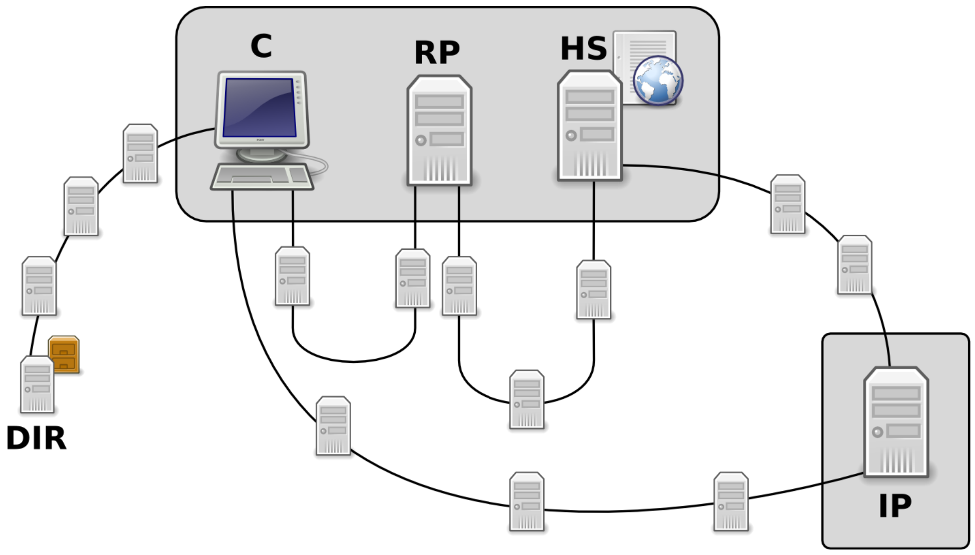

Figure 8: Measurement environment

The main measurement server is a virtual root server in Frankfurt on the Main in Germany, hosted by the German hosting company *1blu*.[2] The machine has an AMD Opteron^{TM}processor 246, 400 megabyte Random Access Memory (RAM) guaranteed, and can use a maximum of 2 gigabyte when available. There are no traffic limitations for this machine.

This server hosts the rendezvous point and the hidden service non-stop during the measurement period. That means both are started prior to the actual measurements and kept running until all measurements are finished. The hidden service is started a few minutes after the rendezvous point, because the latter must be already available when starting the former to be able to be picked as rendezvous point.

The introduction point is hosted on a different server, which is not necessary for the regular hidden service protocol, but for one of the implemented changes when rendezvous point and introduction point come in direct contact with each other. So the fourth role is hosted on a dedicated root server provided by the German hosting company *Hetzner*.[3] This server is located in Nuremberg, Germany and has an AMD Athlon^{TM}64 X2 Dual Core processor 3800+ and 1 gigabyte RAM.

If messages are sent from one server to the other, the corresponding log events for sending and receiving the message occur on different servers. Therefore it is necessary to make sure that the system clocks of the two servers are synchronized.

[2]https://www.1blu.de/
[3]http://www.hetzner.de

4.6 Creating Clients with PuppeTor

To enable repeatable measurements of the performance of accessing a hidden service the Java framework PuppeTor is used on client side. The framework "facilitates the configuration of a set of local Tor processes and the execution of automatic tests based on these processes".[4]

In this measurement the environment PuppeTor is used only to create Tor clients and make them access the hidden service. This is, because the measurement shall take place in the public Tor network, but not in a local environment. The PuppeTor class for the measurements also takes care about cycling through the five scenarios, four changes and the default protocol, presented in the next section.

Cycling through the scenarios is necessary to eliminate the impact of possible performance variability of the Tor network, caused by the day of the week or the time of the day. If a new change is measured after the other one with several hundred attempts is done, comparability cannot be assured, if the performance of the global Tor network changes for example during weekends due to changed amounts of traffic. By cycling through all protocols they are affected in the same way and therefore still comparable.

Cycling means that only one attempt for the original protocol and every change is performed one after another before the cycle starts over again with the original protocol. The time between the start of two attempts is set to five minutes. Therefore the time between two attempts of the same protocol is 25 minutes, due to five different protocols.

A PuppeTor class is executed by cron every five minutes. The class first determines which protocol to use and sets the according configuration options for the client. Then the client is started and the class waits for 75 seconds to let the client build circuits. Afterwards it asks the client to build a connection to the hidden service. After accessing the hidden service successfully, i.e. receiving a response to an empty HTTP GET request, or after receiving an error message from the Tor client, PuppeTor shuts down the client and exits.

PuppeTor also measures the round-trip time for accessing the hidden server, i.e. from requesting the service until receiving the answer. This value is also measured within Tor like all sub steps, but the additional measurement is performed to make sure that the value within Tor is exactly the time any client application using Tor experiences.

4.7 Analyzing the Log Events

As mentioned before log events are used to measure the performance of hidden service sub steps. This section explains, which particular events are used for which steps. The begin of the complete round-trip time is measured by the `Got a hidden service request for ID` statement, indicating that the Puppetor class has just requested the hidden service. The associated statement, that occurs when the Puppetor class has received the answer of the hidden service and immediately closes the Tor client is either `Catching signal TERM, exiting`

[4]Manual available at https://tor-svn.freehaven.net/svn/puppetor/trunk/doc/howto.pdf

cleanly. or Interrupt: exiting cleanly.

The time period to fetch the rendezvous service descriptor is the difference between the occurrence of Fetching rendezvous descriptor for service and Received rendezvous descriptor.

After receiving the rendezvous descriptor, introduction and rendezvous circuits are immediately requested and built. Therefore measurements start at the reception of the rendezvous descriptor and end for the rendezvous circuit with the rendcirc is open event. It is important to mention that the time measured is not necessarily the building time of the circuit, but the time until the first circuit is open. That means if a circuit attempt is discarded and another attempt is started, both are included in the time measured.

The same applies for the introduction circuit, whose beginning is the same as the one of the rendezvous circuit and its end is indicated by the introcirc is open event.

While all log events mentioned above occurred on the same machine and in the same role, the client, the transfer time of the Establish_Rendezvous cell starts at the Sending an ESTABLISH_RENDEZVOUS cell event in the client logs and ends at the Received an ESTABLISH_RENDEZVOUS request on circuit event in the logs of the rendezvous point. The latter event is also the begin of the transfer time of acknowledgment cell RENDEZVOUS_ESTABLISHED, which ends at the Got rendezvous ack. This circuit is now ready for rendezvous. event.

The INTRODUCE1 cell transfer time begins at the occurrence of the event Sending an INTRODUCE1 cell and ends at the Received an INTRODUCE1 request on circuit event in the introduction point log. For the acknowledgment the latter statement is again the beginning, ending at the Received ack. Telling rend circ... event in the client log.

The transfer time of the following INTRODUCE2 cell also starts at the reception of the INTRODUCE1 cell at the introduction point and ends at the occurrence of the Received INTRODUCE2 cell for service statement in the hidden service log. The third time to start at the same reception event is the rendezvous circuit building time at the hidden service. It ends at the Done building circuit * to rendezvous with cookie * for service * statement, when the circuit is open.

After building the circuit the RENDEZVOUS1 cell is immediately sent and a little later received by the rendezvous point, indicated by the log event Got request for rendezvous from circuit * to cookie *.. This is also the initial event for the RENDEZVOUS2 cell transfer time, that ends with the Got RENDEZVOUS2 cell from hidden service. statement in the client log.

The last time measured is the round-trip time of the user data, starting at the reception of the RENDEZVOUS2 cell and ending, when the Puppetor class shuts down the Tor client, because it has received the response of the hidden service. The event is the same that also indicates the end of the complete round-trip time.

5 Changes to be Evaluated

This section lists the changes implemented in the Tor code and describes the implementation in detail. There are four different changes of the hidden service protocol that are later compared to the original protocol regarding performance.

All changes are implemented in the same project and are easy to enable and disable via the torrc configuration file. The changes are not intended to reflect the final state that could be implemented in the actual Tor code. Their purpose is to enable measurements of the suggested protocol behavior.

5.1 Open More Pre-Built Internal Circuits

To use cannibalization it is necessary to open circuits before they are actually needed and used. This is implemented by a pool of pre-built circuits. The default size of the internal circuit pool on client nodes is two. This is the exact number needed for a single hidden service access, because one of the circuits is cannibalized as rendezvous circuit, i.e. the last of the three hops is used as rendezvous point, and the other one is extended to the introduction point, so it is used as introduction circuit.

The `circuit_predict_and_launch_new()` function in `circuituse.c` is called every second and checks if there are enough circuits available, that are not already used by a hidden service request and therefore free to use. But it does not check the building state of the circuits. That means if two circuits are still being built, but not open yet, the function sees no reason to open more circuits.

A problem resulting in a delay occurs if one or both internal circuit cannot be completely built successfully. Each hop in the circuit can be responsible for that. The timeout for circuit creation is 60 seconds. So after one minute the failed circuit is abandoned with a *waiting for keys* info in the log and the next time the function is called, it discovers a difference between the number of circuits needed and the number of available circuits and starts building a new one.

Usually the user does not notice the circuit building problems in the background. But it becomes a remarkable issue, if no open circuits are available at the time of cannibalization. At this time the user has already requested a hidden service and is now waiting for the connection to be established. If a circuit is already being built and waiting for completion the user also waits every second until the timeout. After the timeout he waits for a complete new circuit to be opened, because all three hops are now created.

The intention of this change is to simply increase the number of circuits in the pool and therefore reduce the probability that no open circuit is found at the time of cannibalization. Having more circuits available also increases the average circuit quality, when looking for the best circuits for a particular purpose.

In order to do so, the number of circuits needed is set to the value 5, if the configuration option

`OpenMoreCircuits` is set in the configuration file. The second configuration option mentioned in Listing 8 can be ignored for this change, but is important for another change described in Section 5.2.

```
int num_of_circuits_needed;
or_options_t *options = get_options();

if (options->UseTwoIntroCircuits) {
  num_of_circuits_needed = 3;
} else if (options->OpenMoreCircuits) {
  num_of_circuits_needed = 5;
} else {
  num_of_circuits_needed = 2;
}
```

Listing 8: Client code to open more internal circuits

A disadvantage of this change is that more Tor relays are involved than necessary. If every client in the network opens more circuits, each relay is part of more circuits and the traffic caused by circuit building increases as well as the relays have to handle more encryption and decryption operations. The additional three circuits are not wasted, because they can be used for further hidden service request. But as soon as they are bound to a certain request, new circuits are built to add them to the pool of circuits.

Also a guarantee for short circuit building times cannot be given. It is just the probability of short times that is increased, which can be proven empirically. But higher values may still occur occasionally.

The change does not decrease anonymity provided by Tor, although more relays are involved. The additional relays are distributed among different circuits. Therefore the probability that an adversary controls both entry and exit node does not increase if the absolute number of malicious nodes in the network does not change.

5.2 Start Building Two Introduction Circuits

The second change also focuses on circuit building, or more precisely on establishing the fourth hop of the introduction circuit. After receiving the rendezvous service descriptor from a directory server the Tor client wants to contact an introduction point found in the descriptor. Because of the circuit pool mentioned in the last section there should be two 3-hop circuits available, one to be chosen as rendezvous circuit and one to be extended to the introduction point. But being available, i.e. not being used for any other purpose, does not necessarily mean that the circuit is open and ready to use.

The circuit is marked as used now, but if the first three hops are still in the making, the waiting time until the circuit is abandoned is 60 seconds again. And also in case that the first three hops are already finished when the circuit building request arrives, it may take several seconds for the fourth hop to be built.

Again the intention of this change is similar to the intention presented in section 5.1. This time the idea is to cannibalize two circuits simultaneously. This takes care not only of finding an open 3-hop circuit like the first change, but also of the time to build the fourth hop to the introduction point. The fourth hop can be done in less than a second, but often took several seconds in earlier measurements.[14] Cannibalizing two circuits increases the probability that at least on of them is built within a short time. Figure 9 shows the original constellation on the left and the new way to contact the introduction point on the right. The two circuits are completely separated, but the extension target for both is the same introduction point.

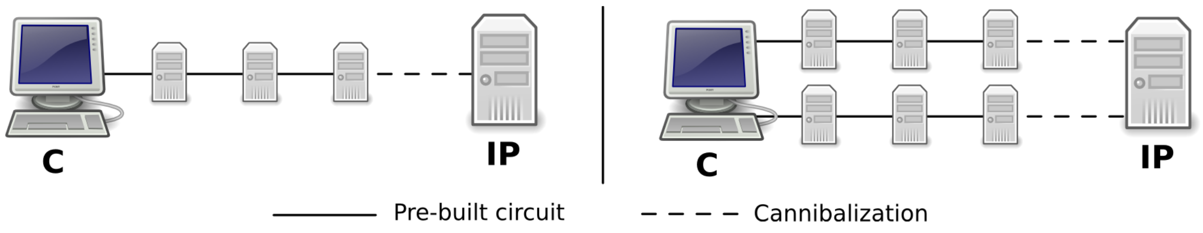

Figure 9: Building one and two introduction circuits

It would also be possible to build two circuits to two different introduction points. But for the measurements it is easier to keep the number of nodes in control, i.e. nodes the log events are observed on, low.

With now needing three internal circuits for a single hidden service access, two for the two attempts to contact the introduction point and one for the rendezvous circuit, it would not be consequent to start with two pre-built 3-hop internal circuit as in the default implementation. Because in that case an additional circuit has to be built completely. Therefore the number of pre-built internal circuits in the `circuit_predict_and_launch_new()` function in `circuituse.c` is set to three. This was shown in Listing 8 earlier. The configuration option to enable this change is `UseTwoIntroCircuits`.

The `circuit_get_open_circ_or_launch()` function in `circuituse.c` handles the circuit selection in all cases, one of those being a new introduction circuit. It either picks an appropriate circuit from the pool of pre-built circuits or launches a completely new circuit, if the pool does not contain a matching circuit. Listing 9 shows in the first three lines the default command. In case of the configuration option in line 6 being enabled and the purpose of the circuit to be created is to be an introduction circuit, a second circuit is launched with the exact same parameters as the first circuit. In line 12 the new circuit is also bound to the specific hidden service query it is built for. This is also done for the regular circuit, of course. The command is found just a few lines later.

The `circuit_has_opened()` function in `circuituse.c` is called when a new circuit is opened. Listing 10 describes what happens if the purpose of the new circuit is to be an introduction circuit. In the regular protocol just another function is called to notify the rendezvous component.

Additionally the change requires that as soon as one of the two introduction circuit attempts is successful, the other one is discarded. Therefore in line 5 the algorithm loops through the list of all circuits until it finds one with the following two properties. One is that the circuit's purpose is to act as introduction circuit, which is true for two circuits, the one recently opened and

```
circ = circuit_launch_by_extend_info(
    new_circ_purpose, want_onehop, extend_info,
    need_uptime, 1, need_internal);

or_options_t *options = get_options();
if (options->UseTwoIntroCircuits
    && new_circ_purpose == CIRCUIT_PURPOSE_C_INTRODUCING) {
  origin_circuit_t *secondCirc;
  secondCirc = circuit_launch_by_extend_info(
      new_circ_purpose, want_onehop, extend_info,
      need_uptime, 1, need_internal);
  strlcpy(secondCirc->rend_query, conn->rend_query, sizeof(circ->rend_query));
}
```

Listing 9: Client code to open a second introduction circuit

another one. So the second property is that the circuit is still in building state, i.e. not open yet. The circuit having both properties is the *loser circuit*, the one that needs to be closed.

In the measurement implementation the actual command to close the circuit in line 13 is commented out to gather more data about the time difference between opening the two circuits.

```
 1 case CIRCUIT_PURPOSE_C_INTRODUCING:
 2   if (options->UseTwoIntroCircuits) {
 3     origin_circuit_t *loserCirc = NULL;
 4     circuit_t *myCirc;
 5     for (myCirc = global_circuitlist; myCirc; myCirc = myCirc->next) {
 6       if (myCirc->purpose == CIRCUIT_PURPOSE_C_INTRODUCING
 7           && myCirc->state == CIRCUIT_STATE_BUILDING) {
 8         loserCirc = TO_ORIGIN_CIRCUIT(myCirc);
 9         break;
10       }
11     }
12     if (loserCirc) {
13       //circuit_mark_for_close(loserCirc, END_CIRC_REASON_NONE);
14     } else {
15       log_info(LD_REND, "CW: Couldn't find loser circuit.");
16     }
17   }
18   rend_client_introcirc_has_opened(circ);
19   break;
```

Listing 10: Client code to close the loser introduction circuit

This change again increases traffic on relays by opening and extending more circuits as well as cryptographic operations on relays. The anonymity for clients is not affected for the same reasons as in Section 5.1.

5.3 Simplifying Hidden Service Access

Øverlier and Syverson propose measures to simplify circuit establishment and the hidden service protocol. They write that the problem of the existing protocol is “that it has become too complex” and want to “drastically reduce both complexity and latency when connecting to a hidden service”.[16]

The changes of the hidden service protocol are only one part of the suggested improvements. The authors also propose improvements of the circuit establishment protocol for faster circuit building, further reduced load on relays, and improved client and network efficiency. These changes also affect the general Tor protocol for anonymous access of public servers.

To increase the performance of circuit establishment Øverlier and Syverson propose to reduce the number of Diffie-Hellman key exchanges per circuit. These key exchanges are more expensive in terms of computations compared to the ElGamal key agreement. Furthermore in the new circuit protocol less cells are necessary to establish a circuit. While in the current protocol each circuit extension by one hop is started at the client and after the successful establishment propagated back, only a single cell is sent by the client, including all the information for the circuit, still encrypted in layers. After the client has established a connection to the entry node, the latter connects to the middle node and so on. After creating all hops the last node sends a single cell down the circuit to acknowledge.

The authors’ hidden service protocol improvements require another concept added to hidden services: so-called *valet nodes.*[17] These valet nodes help to hide the introduction points from the clients. The hidden service writes the contact information of a random relay, the valet node, in the rendezvous service descriptor, instead of the introduction point contact information, as in the original protocol. It also adds a so-called valet token to the rendezvous service descriptor. This token is the contact information of an introduction point, encrypted with the valet node’s public key. If a client wants to access the hidden service it retrieves the rendezvous service descriptor from the hidden service directory, contacts the valet node and sends it the valet token. The valet node extends the circuit to the introduction point it finds in the token and the client can communicate with the hidden service to proceed with the original protocol.

With the introduction of valet nodes the authors propose to discard the rendezvous point and instead route the data traffic, that is sent over the rendezvous point in the original protocol, through the introduction point or over a new connection to the last node before the valet node in the circuit between client and introduction point. This requires less circuits to be built and less relays involved, without compromising on anonymity due to the new valet nodes protecting the introduction points.

This work only shows the impact of the hidden service protocol changes, still using the current circuit establishment protocol. This is because changing the internal circuit establishment protocol additionally goes beyond the scope of this thesis.

5.3.1 Combining Introduction Circuit and Rendezvous Circuit

Instead of using two different circuits as introduction circuit and rendezvous circuit, in the new protocol only a single circuit is opened, ending at the rendezvous point. This circuit is first used to send the `Establish_Rendezvous` cell and receive the `Rendezvous_Established` cell. Then the same circuit is cannibalized to contact the introduction point. After receiving the acknowledgment, the circuit is labeled back as rendezvous circuit. By doing so, the number of relays involved in the hidden service access is reduced by three, the three relays used in the earlier introduction circuit. Figure 10 illustrates the changed protocol.

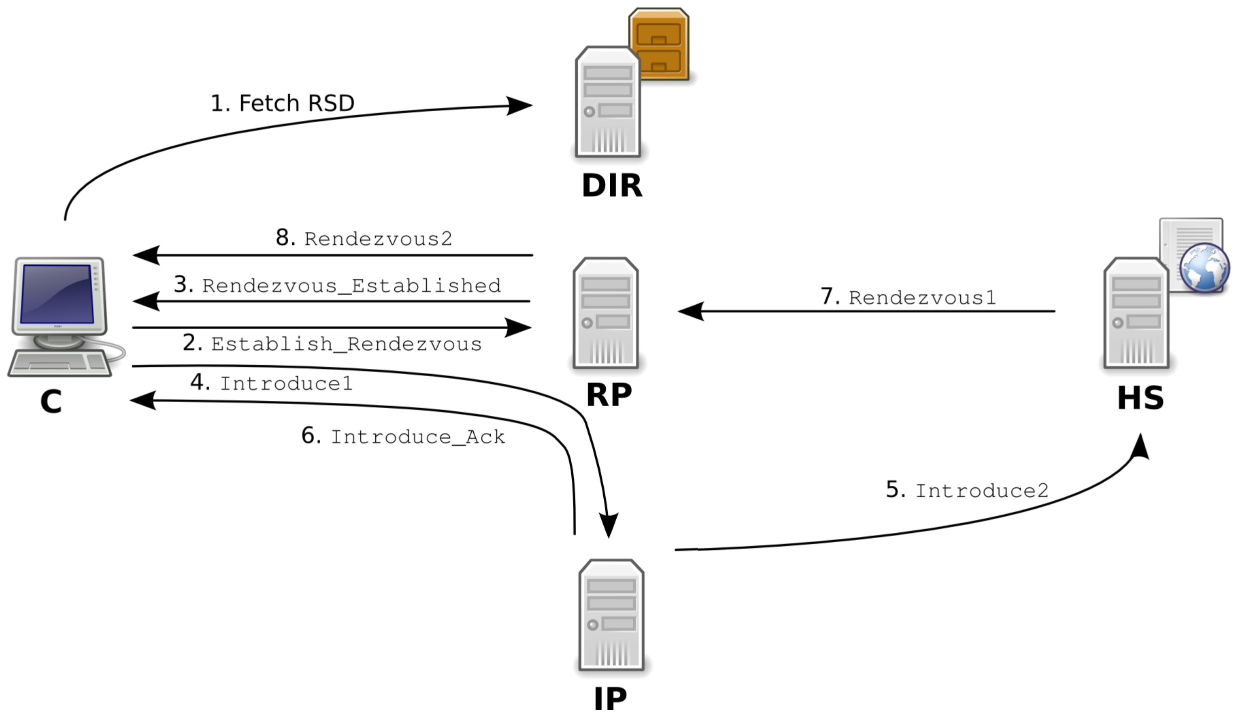

Figure 10: Combining introduction circuit and rendezvous circuit

For this change it is necessary, that the introduction circuit is built after the rendezvous circuit and even after the rendezvous point has acknowledged its role. This is different from the regular protocol, when rendezvous and introduction circuits are built simultaneously and handled independently.

The `connection_ap_handshake_attach_circuit()` function in `circuituse.c` is called every second after receiving a hidden service request. It checks if new circuits have to be built. The configuration option to activate this change is `UseOverlierChange`. In lines 2 to 7 of Listing 11 it is checked, if the rendezvous acknowledgment was already received. The `rendcirc` struct must not be null and its purpose must be one of the three purposes named in the listing.

If the rendezvous acknowledgment is available, a circuit to act as introduction circuit can be searched in line 10.

The next behavior to change is the selection of the introduction circuit. This takes place in the `circuit_launch_by_extend_info()` function in `circuituse.c` and is shown in Listing 12. If the configuration option to use this change is activated and the client is looking for a circuit

```
if (get_options()->UseOverlierChange) {
  int rend_ack_available = rendcirc && (rendcirc->_base.purpose ==
                                 CIRCUIT_PURPOSE_C_REND_READY ||
                                 rendcirc-> _base.purpose ==
                                 CIRCUIT_PURPOSE_C_INTRODUCING ||
                                 rendcirc->_base.purpose ==
                                 CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT);
  if (rend_ack_available) {
    /* it's on its way. find an intro circ. */
    retval = circuit_get_open_circ_or_launch(
               conn, CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT, &introcirc);
    if (retval < 0) return -1; /* failed */

    if (retval > 0) {
      /* one has already sent the intro. keep waiting. */
      tor_assert(introcirc);
      log_info(LD_REND, "Intro circ %d present and awaiting ack "
                 "(rend %d). Stalling. (stream %d sec old)",
                 introcirc->_base.n_circ_id,
                 rendcirc ? rendcirc->_base.n_circ_id : 0,
                 conn_age);
      return 0;
    }
  }
}
```

Listing 11: Client code to determine if introduction circuit can be built

to cannibalize and extend to the introduction point, not an unused circuit for general purpose is returned as in line 20. Instead the new code returns the circuit that was earlier used as rendezvous circuit in line 4. This circuit still has the rendezvous purpose and is bound to the hidden service identifier used in the measurements.

```
1  or_options_t *options = get_options();
2  if (options->UseOverlierChange &&
3      purpose == CIRCUIT_PURPOSE_C_INTRODUCING) {
4      circ = circuit_get_by_rend_query_and_purpose(”xpw5lcjsag7u6l6w”,
5              CIRCUIT_PURPOSE_C_REND_READY);
6      log_info(LD_CIRC,”CW Select rendcirc to cannibalize.”);
7      if (extend_info) {
8        /* need to make sure we don’t duplicate hops */
9        crypt_path_t *hop = circ->cpath;
10       do {
11         if (!memcmp(hop->extend_info->identity_digest,
12                     extend_info->identity_digest, DIGEST_LEN))
13           goto next;
14         hop=hop->next;
15       } while (hop!=circ->cpath);
16       next: ;
17     }
18 } else {
19   /* see if there are appropriate circs available to cannibalize. */
20   circ = circuit_find_to_cannibalize(purpose, extend_info,
21                                      need_uptime, need_capacity, internal);
22 }
```

Listing 12: Client code to select rendezvous circuit as introduction circuit

The client checks the purpose of the rendezvous and introduction circuits before sending the `Introduce1` cell to the introduction point in the `rend_client_send_introduction()` function in `rendclient.c`. Since the two circuits are now identical and the current purpose of the circuit is to be an introduction circuit, the assertion that the rendezvous circuit has another purpose has to be disabled, as shown in Listing 13.

```
or_options_t *options = get_options();
if (!options->UseOverlierChange) {
  tor_assert(rendcirc->_base.purpose == CIRCUIT_PURPOSE_C_REND_READY);
}
```

Listing 13: Client code to ignore purpose of rendezvous circuit during introduction

The hidden server needs to know the contact information of the rendezvous point to build a circuit to it. In the regular protocol this is the exit node of the rendezvous circuit, shown in line 7 of Listing 14. But in the changed protocol the fourth and last node in the combined rendezvous and introduction circuit is the introduction point. The rendezvous point is the third hop in this circuit. The `cpath` variable in line 3 is a pointer to a linked list of structs, representing the relays that belong to this circuit, starting at the entry node. The `prev` variable in each struct points to the next relay. This mistakable description arises from the fact that counting the nodes starts at the chosen exit node and the `next` pointer points towards the client. Nevertheless it is easy to find the third struct representing the rendezvous point and return its contact information in form of the `extend_info` struct.

```
extend_info_t *extend_info;
if (options->UseOverlierChange) {
  extend_info = rendcirc->cpath->prev->prev->extend_info;
  log_info(LD_REND, ”CW Routers: ’%s’.”,
           rendcirc->cpath->prev->prev->extend_info->nickname);
} else {
  extend_info = rendcirc->build_state->chosen_exit;
}
```

Listing 14: Client code to select rendezvous point

When receiving the acknowledgment that the introduction point has received the `Introduce1` cell and forwarded the content to the hidden server, the rendezvous circuit of the client is notified in the `rend_client_introduction_acked()` function in `rendclient.c`. Usually it is found in the global circuit list by the rendezvous query and the circuit's purpose. But after reusing the rendezvous circuit as introduction circuit, the purpose of the combined circuit is now to wait for the introduction acknowledgment. The changed code can be found in Listing 15.

```
or_options_t *options = get_options();
if (options->UseOverlierChange) {
  rendcirc = circuit_get_by_rend_query_and_purpose(
             circ->rend_query, CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT);
} else {
  rendcirc = circuit_get_by_rend_query_and_purpose(
             circ->rend_query, CIRCUIT_PURPOSE_C_REND_READY);
}
```

Listing 15: Client code to find combined rendezvous and introduction circuit

After receiving the acknowledgment usually the introduction circuit is not needed anymore. Since the introduction circuit in the change was created by just extending the rendezvous circuit, this 4-hop circuit needs to be truncated by the last hop, resulting in the originally created 3-hop circuit with the rendezvous point as third node. In the same function the code shown in Listing 16 was implemented.

The internal representation of a circuit is the linked list mentioned above. This list is used to create cells, encrypted for all relays on the circuit. In lines 7 and 8 the pointers are corrected to exclude the last relay, and in the following line the length of the circuit is reduced.

To truncate the real circuit, a cell with a `TRUNCATE` command is sent down the circuit. There exists an acknowledgment cell to notify the sender, that the circuit was successfully truncated. But in the regular protocol the `TRUNCATE` command is never used and the acknowledgment is used to propagate an error in the circuit, which is handled by the client by marking the circuit for close and therefore not using it anymore. In the change the acknowledgment is expected after sending the `TRUNCATE` cell and does not indicate an error. That is why a boolean flag is set in line 18 to remember this when the acknowledgment is received.

```
 1 if (options->UseOverlierChange) {
 2   log_info(LD_REND, "CW Checking cpath.");
 3   log_info(LD_REND, circuit_list_path(rendcirc, 1));
 4   log_info(LD_REND, "First node: %s, second %s.",
 5              rendcirc->cpath->extend_info->nickname,
 6              rendcirc->cpath->next->extend_info->nickname);
 7   rendcirc->cpath->prev->prev->next = rendcirc->cpath;
 8   rendcirc->cpath->prev = rendcirc->cpath->prev->prev;
 9   rendcirc->build_state->desired_path_len =
10               rendcirc->build_state->desired_path_len - 1;
11   log_info(LD_REND, circuit_list_path(rendcirc, 1));
12   if (relay_send_command_from_edge(0, TO_CIRCUIT(rendcirc),
13                                    RELAY_COMMAND_TRUNCATE, NULL, 0,
14                                    rendcirc->cpath->prev) < 0) {
15     log_info(LD_REND,"CW: Couldn't send TRUNCATE cell");
16   } else {
17     log_info(LD_REND,"CW: TRUNCATE cell successfully sent");
18     options->ExpectTruncatedCell = 1;
19   }
20 } else if (options->UseOverlier2Change) {
21   log_info(LD_REND,"CW Do nothing.");
22   //circ->_base.purpose = CIRCUIT_PURPOSE_C_REND_READY_INTRO_ACKED;
23 } else {
24   /* close the circuit: we won't need it anymore. */
25   circ->_base.purpose = CIRCUIT_PURPOSE_C_INTRODUCE_ACKED;
26   circuit_mark_for_close(TO_CIRCUIT(circ), END_CIRC_REASON_FINISHED);
27 }
```

Listing 16: Client code to truncate introduction circuit

In the `connection_edge_process_relay_cell()` function in `relay.c` all cells are processed. If an incoming cell is identified as acknowledgment of a `TRUNCATE` cell sent before, the circuit does not need to be closed like in line 5 of Listing 17, but the cell can simply be ignored.

```
1 if (get_options()->UseOverlierChange &&
2     get_options()->ExpectTruncatedCell) {
3   // we expect this cell. so do nothing.
4 } else {
5   circuit_truncated(TO_ORIGIN_CIRCUIT(circ), layer_hint);
6 }
```

Listing 17: Client code to receive truncated cell

5.3.2 Direct Hidden Service Usage

Another protocol proposed by Øverlier and Syverson is to send user data over the introduction point instead of opening a new circuit. Therefore only the circuit between client and introduction point needs to be cannibalized, while the hidden service already has a circuit to the introduction point resulting from the hidden service establishment. To measure the performance of this protocol, rendezvous circuit and introduction circuits are unified again. Figure 11 illustrates the changed protocol.

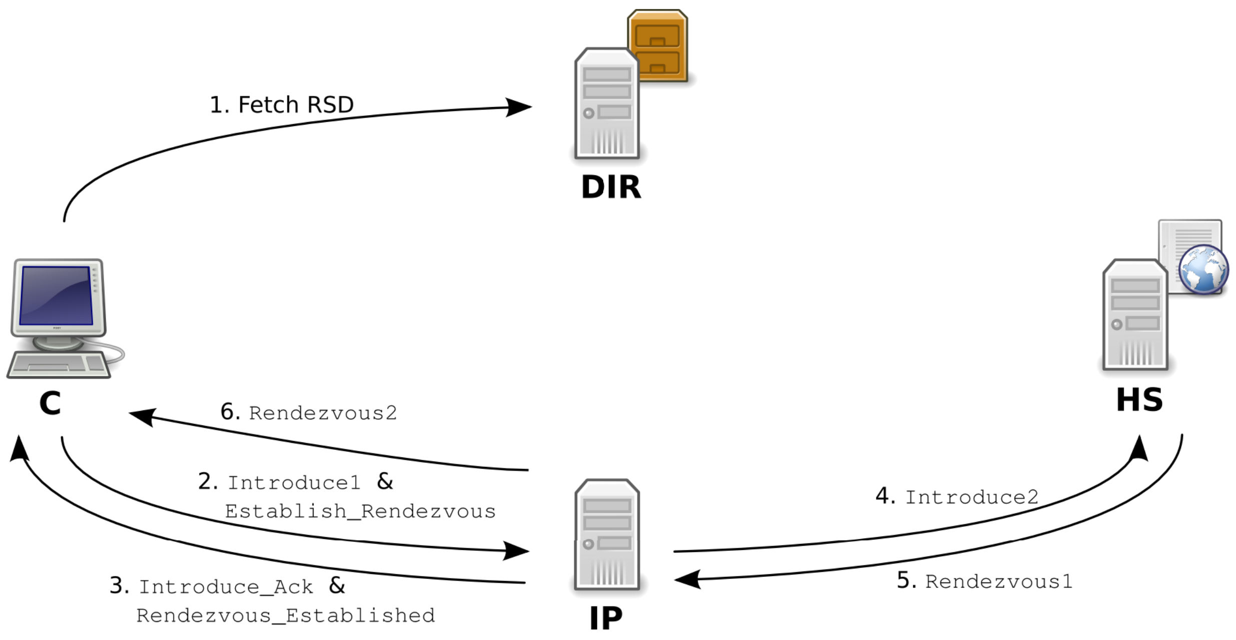
1. Fetch RSD
DIR
C
HS
6. Rendezvous2
2. Introduce1 & Establish_Rendezvous
4. Introduce2
3. Introduce_Ack & Rendezvous_Established
5. Rendezvous1
IP

Figure 11: Direct hidden service usage

The `connection_ap_handshake_attach_circuit()` function in `circuituse.c` is called every second, checking for all client connections. If a connection is associated with the measurement hidden service identity hash in line 5, the code in Listing 18 is executed. After determining that the second Øverlier protocol is used, a circuit with purpose `REND_JOINED` associated with the measurement hidden service is searched in lines 4 and 5. The configuration option to enable this change is `UseOverlier2Change`.

If a circuit meeting both conditions is found, the variable `rendcirc` is pointed to the same circuit struct in line 7 and the circuit is linked to the client connection in line 20. Otherwise a circuit is launched in lines 27 and 28. If the return value `retval` of that function is 0, the circuit is available and a message can be sent in line 47.

`rend_client_send_introduction_without_rendcirc()` in `rend_client.c` is the function that packs and sends the `Introduce1` cell. `rend_client_send_introduction()` is the original function, which has two parameters, pointers to the introduction circuit struct and the rendezvous circuit struct. This is because it takes the contact information and rendezvous cookie of the rendezvous circuit and sends it down the introduction circuit.

The new function is a copy of that function, using only the introduction circuit. This is because the circuit is used as rendezvous circuit before and therefore a rendezvous cookie is generated. The contact information is actually useless, because the hidden service uses the already existing circuit to the introduction point as rendezvous circuit, instead of launching a new circuit.

To notify the hidden service that the second Øverlier protocol is to be used, the port in the contact information is set to 9999 in line 3 of Listing 19. The contact information is not

```
origin_circuit_t *rendcirc=NULL, *introcirc=NULL;
tor_assert(!conn->cpath_layer);
if (get_options()->UseOverlier2Change) {
  introcirc = circuit_get_by_rend_query_and_purpose(
              "xpw5lcjsag7u6l6w", CIRCUIT_PURPOSE_C_REND_JOINED);
  if (introcirc) {
    rendcirc = introcirc;
    tor_assert(rendcirc);
    /* one is already established, attach */
    log_info(LD_REND,
             "rend joined circ %d already here. attaching. "
             "(stream %d sec old)",
             rendcirc->_base.n_circ_id, conn_age);
    /* Mark rendezvous circuits as 'newly dirty' every time you use
     * them, since the process of rebuilding a rendezvous circ is so
     * expensive. There is a tradeoffs between linkability and
     * feasibility, at this point.
     */
    rendcirc->_base.timestamp_dirty = time(NULL);
    link_apconn_to_circ(conn, rendcirc, NULL);
    if (connection_ap_handshake_send_begin(conn) < 0)
      return 0; /* already marked, let them fade away */
    return 1;

  } else {
    /* find an intro circ. */
    retval = circuit_get_open_circ_or_launch(
             conn, CIRCUIT_PURPOSE_C_INTRODUCE_ACK_WAIT, &introcirc);
    if (retval < 0) return -1; /* failed */

    if (retval > 0) {
      /* one has already sent the intro. keep waiting. */
      tor_assert(introcirc);
      log_info(LD_REND, "Intro circ %d present and awaiting ack "
               "(rend %d). Stalling. (stream %d sec old)",
               introcirc->_base.n_circ_id,
               rendcirc ? rendcirc->_base.n_circ_id : 0,
               conn_age);
      return 0;
    }

    tor_assert(introcirc->_base.purpose == CIRCUIT_PURPOSE_C_INTRODUCING);
    if (introcirc->_base.state == CIRCUIT_STATE_OPEN) {
        log_info(LD_REND,"found open intro circ %d; sending "
                 "introduction. (stream %d sec old)",
                  introcirc->_base.n_circ_id, conn_age);
        if (rend_client_send_introduction_without_rendcirc(introcirc)
            < 0) {
          return -1;
        }
        introcirc->_base.timestamp_dirty = time(NULL);
        assert_circuit_ok(TO_CIRCUIT(introcirc));
        return 0;
    }
  }
}
```

Listing 18: Client code to build combined circuit

necessary in this version of the protocol, as mentioned above, so the port can be used without any problems. Afterwards the cell is sent down the circuit in line 5.

```
1  extend_info = introcirc->build_state->chosen_exit;
2  [...]
3  extend_info->port = 9999;
4  [...]
5  if (relay_send_command_from_edge(0, TO_CIRCUIT(introcirc),
6                                   RELAY_COMMAND_INTRODUCE1,
7                                   payload, payload_len,
8                                   introcirc->cpath->prev)<0) {
9    /* introcirc is already marked for close. leave rendcirc alone. */
10   log_warn(LD_BUG, "Couldn't send INTRODUCE1 cell");
11   return -1;
12 }
```

Listing 19: Client code to send `Introduce1` cell

When receiving the rendezvous acknowledgment from the rendezvous point the purpose of the circuit is checked in the `rend_client_rendezvous_acked()` function in `rendclient.c`. If the circuit does not have the purpose that is expected in the regular protocol, the circuit is closed. This behavior is disabled via configuration option, when the usage of the second Øverlier protocol is detected, as shown in Listing 20.

```
or_options_t *options = get_options();
/* we just got an ack for our establish-rendezvous. switch purposes. */
if (!options->UseOverlier2Change &&
    circ->_base.purpose != CIRCUIT_PURPOSE_C_ESTABLISH_REND) {
  log_warn(LD_PROTOCOL,"Got a rendezvous ack when we weren't expecting one. "
           "Closing circ.");
  circuit_mark_for_close(TO_CIRCUIT(circ), END_CIRC_REASON_TORPROTOCOL);
  return -1;
}
...
if (!options->UseOverlier2Change) {
  circ->_base.purpose = CIRCUIT_PURPOSE_C_REND_READY;
}
```

Listing 20: Client code to ignore rendezvous acknowledgment

The client receives the `Rendezvous2` cell over the circuit, that is still the introduction circuit. As shown in in Listing 21, checking the purpose of the circuit is therefore disabled in the `rend_client_receive_rendezvous()` function in `rendclient.c`. Instead the purpose is set to the one indicating a successfully joined rendezvous circuit.

When the introduction point receives an `Introduce1` cell over the same circuit, it used for rendezvous establishment, this indicates the usage of the second Øverlier protocol. Instead of closing the circuit because of a protocol violation, which would happen in the regular protocol, a global option is set in line 5 of Listing 22. Otherwise the global option is disabled in line 7 to use the regular protocol.

```
or_options_t *options = get_options();
if (options->UseOverlier2Change) {
  circ->_base.purpose = CIRCUIT_PURPOSE_C_REND_JOINED;
} else {
  if ((circ->_base.purpose != CIRCUIT_PURPOSE_C_REND_READY &&
       circ->_base.purpose != CIRCUIT_PURPOSE_C_REND_READY_INTRO_ACKED)
       || !circ->build_state->pending_final_cpath) {
    log_warn(LD_PROTOCOL,"Got rendezvous2 cell from hidden service, but not "
                "expecting it. Closing.");
    circuit_mark_for_close(TO_CIRCUIT(circ), END_CIRC_REASON_TORPROTOCOL);
    return -1;
  }
}
```

Listing 21: Client code to receive `Rendezvous2` cell

```
or_options_t *options = get_options();
...
if (circ->_base.purpose == CIRCUIT_PURPOSE_REND_POINT_WAITING) {
  /* CW Recognize Overlier2 Change */
  options->UseOverlier2Change = 1;
} else {
  options->UseOverlier2Change = 0;
  if (circ->_base.purpose != CIRCUIT_PURPOSE_OR || circ->_base.n_conn) {
    log_warn(LD_PROTOCOL,
                "Rejecting INTRODUCE1 on non-OR or non-edge circuit %d.",
                circ->p_circ_id);
    goto err;
  }
}
```

Listing 22: Introduction point code to determine usage of second Øverlier protocol

The global option is used to know when to ignore the circuit purpose of the circuit the `Rendezvous1` cell is received over in `rend_mid_rendezvous()` in `rendmid.c`. The wrong purpose would again result in a protocol violation. This is shown in Listing 23.

```
or_options_t *options = get_options();
// CW Ignore purpose, if using Overlier2Change
if ((!options->UseOverlier2Change
    && circ->_base.purpose != CIRCUIT_PURPOSE_OR)
    || circ->_base.n_conn) {
  log_info(LD_REND,
           ”Tried to complete rendezvous on non-OR or non-edge circuit %d.”,
           circ->p_circ_id);
  reason = END_CIRC_REASON_TORPROTOCOL;
  goto err;
}
```

Listing 23: Introduction point code to complete rendezvous

The indicator for this protocol variation on hidden service side is the port number of the contact information of the rendezvous point as mentioned above. The `Introduce2` cell containing that information is processed in the `rend_service_introduce()` function in `rendservice.c`. So the hidden service can also change a global variable that is accessible in all functions in line 5 of Listing 24.

```
1  /* CW If the port of the would-be rendezvous node is 9999, the hidden
2   * service must act according to the second Overlier protocol.
3   */
4  or_options_t *options = get_options();
5  if (extend_info->port == 9999) {
6    log_info(LD_REND, ”CW Detected Overlier2Change.”);
7    options->UseOverlier2Change = 1;
8  } else {
9    options->UseOverlier2Change = 0;
10 }
```

Listing 24: Hidden service code to determine usage of second Øverlier protocol

In the same function usually a circuit to the rendezvous point is started and saved in the variable `launched` in line 8 of Listing 25. In the changed protocol this variable is set to the same struct the variable `circuit` in line 3 points to. The latter has a reference to the circuit, the `Introduce1` cell was received on. It is the connection to the introduction point.

Since the circuit in `launched` does not need to be cannibalized anymore, because it is already open, in the same function the function to signal an opened rendezvous circuit can be called in line 6 of Listing 26.

The way the circuit between hidden service and introduction point is relabeled as rendezvous circuit and therefore used for user data in this implementation for measurement purposes bears a problem that is solved with the following changes. Since the purpose of the former introduction

```
if (options->UseOverlier2Change) {
  /* CW Use the existing introduction circuit to send the RENDEZVOUS1 */
  launched = circuit;
} else {
  /* Launch a circuit to alice's chosen rendezvous point.
   */
  for (i=0;i<MAX_REND_FAILURES;i++) {
    launched = circuit_launch_by_extend_info(
                    CIRCUIT_PURPOSE_S_CONNECT_REND, 0, extend_info,
                    circ_needs_uptime, 1, 1);
    if (launched)
      break;
  }
}
```

Listing 25: Client code to send `Rendezvous1` cell

```
if (options->UseOverlier2Change) {
  /* CW The introduction circuit is already open. So change its purpose and
   * propagate the open circuit.
   */
  launched->_base.purpose = CIRCUIT_PURPOSE_S_CONNECT_REND;
  rend_service_rendezvous_has_opened(launched);
}
```

Listing 26: Hidden service code to report opened rendezvous circuit

circuit is now to be a rendezvous circuit, the hidden service thinks, that it has lost the node as introduction point and therefore chooses a new one.

The `rend_services_introduce()` function in `rendservice.c` is called every second and checks, if the introduction points are still available. After a client has accessed the hidden service using the changed protocol, the second condition in line 15 in Listing 27 fails and the router is not used as introduction point any longer. This is no problem for the current access attempt, but for the following attempts in the measurement environment, which are forced to use the same introduction point.

When the algorithm iterates through all three introduction points to check them, the lines 12 to 14 are added to see if one of the three introduction points is the one necessary for the measurements. The identity hash of each introduction point is simply compared to the one used in the measurements, whose identity hash is set in line 5. If the router is found a boolean variable is set.

```
 1 int have_own_router;
 2 char *own_intro;
 3 ...
 4 have_own_router = 0;
 5 own_intro = "$68333D0761BCF397A587A0C0B963E4A9E99EC4D3";
 6 ...
 7 /* Find out which introduction points we have in progress for this
 8    service. */
 9 for (j=0; j < smartlist_len(service->intro_nodes); ++j) {
10   intro = smartlist_get(service->intro_nodes, j);
11   router = router_get_by_nickname(intro, 0);
12   if (!strcmp(intro, own_intro)) {
13     have_own_router = 1;
14   }
15   if (!router || !find_intro_circuit(router,service->pk_digest)) {
16     log_info(LD_REND,"Giving up on %s as intro point for %s.",
17              intro, service->service_id);
18     tor_free(intro);
19     smartlist_del(service->intro_nodes,j--);
20     changed = 1;
21     service->desc_is_dirty = now;
22   }
23   smartlist_add(intro_routers, router);
24 }
```

Listing 27: Hidden service code to check if measurement introduction point is used

The information collected with the change above is now used in the same function when a new introduction point is chosen. According to the regular protocol this is a random router, but for the measurements the specific introduction point is chosen in lines 2 and 3 in Listing 28 using its identity hash.

```
if (!have_own_router) {
  router = router_get_by_nickname(
            "$68333D0761BCF397A587A0C0B963E4A9E99EC4D3", 1);
  have_own_router = 1;
} else {
  router = router_choose_random_node(service->intro_prefer_nodes,
            service->intro_exclude_nodes, exclude_routers, 1, 0, 0,
            get_options()->_AllowInvalid & ALLOW_INVALID_INTRODUCTION,
            0, 0);
}
```

Listing 28: Hidden service code to rebuild circuit to measurement introduction point

6 Results

The implementations presented in the previous section were measured with the measurement environment described in Section 4. The measurements started on Tuesday, 22 April at 3:45 p.m. and ended on Tuesday, 13 May at 12:00 noon, hence lasting for almost three weeks.

In this time a total of 5999 access attempts were performed to access the hidden service, distributed equally among the four protocols, resulting in 1200 access attempts per protocol.

This section describes the measures necessary to clean the data set and the results for all four changes.

6.1 Timing Problems Between Measurement Servers

As stated in Section 4.5 logs were collected on two different servers. The main server hosts the client, rendezvous point, and hidden server roles, while another server hosts the introduction point. When looking at the results for the messages sent from the introduction point to the client or hidden service, a large number of negative values between 0 and approximately -1 occurred for the `Introduce_Ack` and `Introduce2` messages. That would mean that messages were received before they were sent, which is obviously impossible. Figure 12 shows all message transfer times for messages of the original protocol sent from the main server to the introduction point server over the 21 day measurement period. A daily pattern is clearly recognizable.

Looking at messages in the opposite direction, shown in Figure 13, a similar pattern can be identified. This indicates that there is a changing time deviation between the two servers. The difference seems to be reset on a daily basis and then increases until the next reset.

To compensate the time deviation between the two servers the deviation on the introduction point server is measured on an hourly basis for 24 hours between 15 May 11 a.m. and 16 May 11 a.m., resulting in the plot shown in Figure 14. The average deviation from regular time is -0.051802 seconds per hour, derived from the measurements. The system clock on the second server is synchronized with a set of official time servers every 24 hours at 6:25:17 a.m. With this information it is possible to correct the deviation for all events observed at the introduction

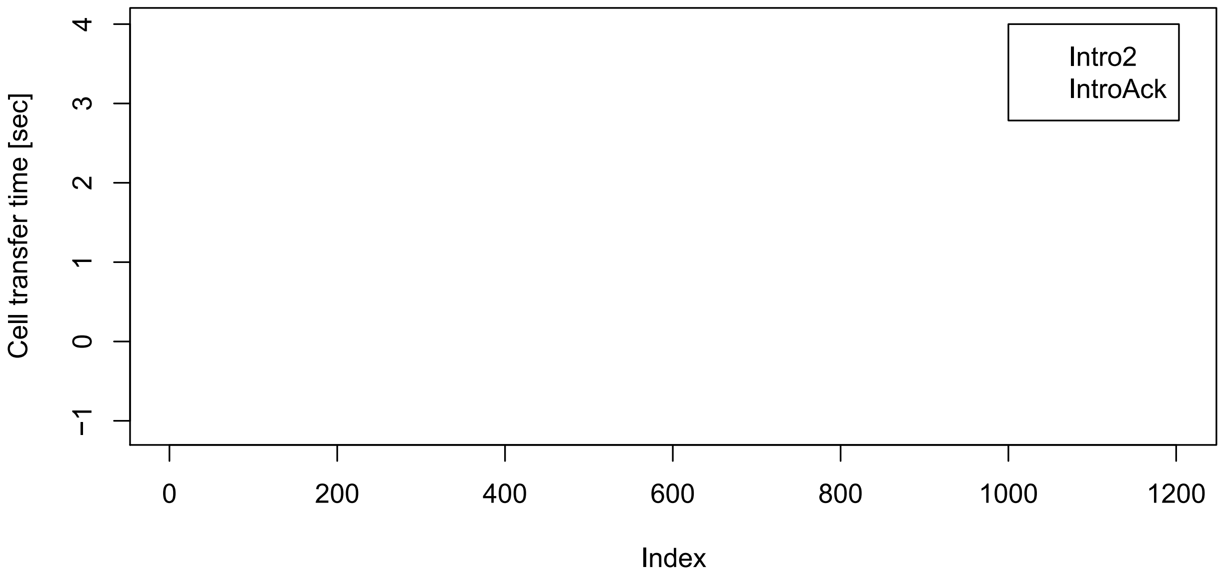

Figure 12: Cell transfer times from main server to introduction point server

point server.

To ensure that the main measurement server does not have a similar problem, corresponding measurements are performed on the main server. Figure 15 shows the results of these measurements between 21 May 11 a.m. and 22 May 11.am.

A pattern is not visible in the result and the amplitude of deviation is within the precision boundaries of the Network Time Protocol (NTP) used for the deviation measurements.

6.2 Deleted Records

In the results four records appeared that had overlapped with the following access attempt, clearly indicated by a PuppeTor warning when trying to start the following event.

In all four records all events until receiving the `Rendezvous2` occurred properly, but after performing the empty HTTP GET there was no reply from the hidden service and therefore PuppeTor kept waiting until a new access attempt was started. This resulted in values higher than 225 seconds for the round-trip time. In addition to the 75 seconds waiting time before requesting the hidden service, to let Tor build circuits in the beginning, this is the 300 second or five minute barrier before the next attempt is started.

Two of the four attempts occurred using the regular protocol on 24 April and 9 May, one during using the protocol with more pre-built circuits on 1 May and one when performing an access attempt according to the protocol with combined introduction and rendezvous circuits on 23 April.

So the distribution of this error among protocols and time seems random and the four records

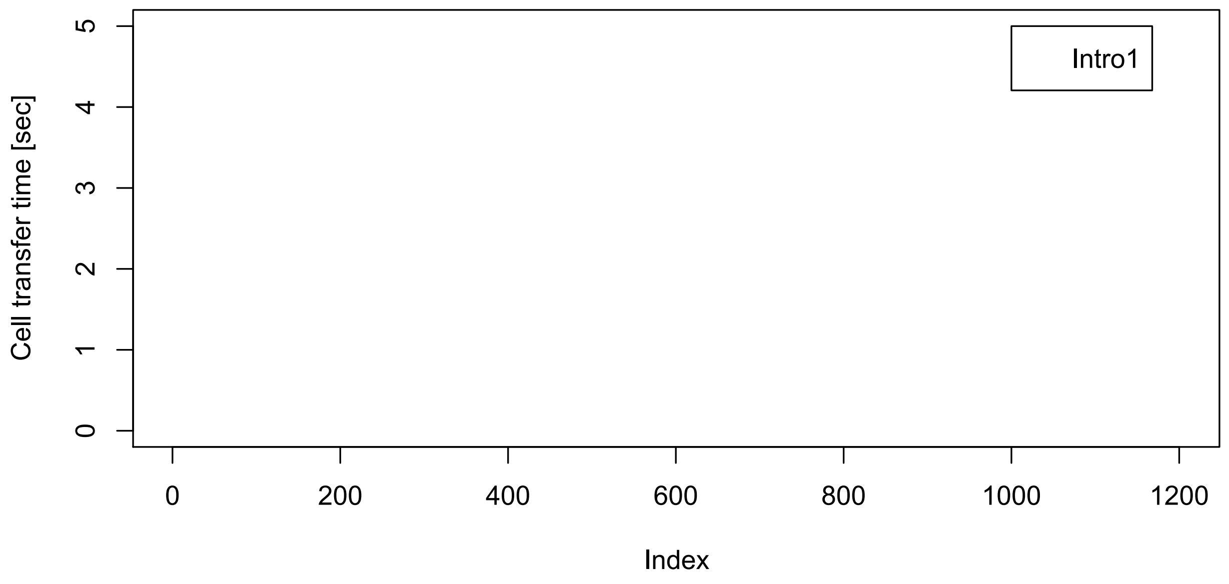

Figure 13: Cell transfer times from introduction point server to main server

were deleted, because they were aborted artificially by the measurement environment and did not finish naturally in terms of the appropriate protocol.

6.3 Use of Own Nodes

To collect data of all sub steps it was necessary to use the specific introduction point and the specific rendezvous point, which were set up in the measurement environment. Using other relays as one or both of the two roles resulted in log events that occurred on relays, which could not be accessed and therefore not used to calculate all sub step values.

But this is a problem only for some values, not for others, that are collected on the hidden service and client. Therefore it was possible for example to measure a round-trip time between requesting the hidden service and receiving the reply, although other nodes were chosen as introduction and/or rendezvous point than those specified in the measurement environment. This is possible because the round-trip time is based on two events that occurred on the client.

After deleting two records as mentioned in the previous section 1198 records remain for the regular protocol. 969 attempts used the specified rendezvous point for the measurements as well as the specified introduction point. In 229 attempts another node was chosen either for the introduction point or the rendezvous point of both of them. The mean of the round-trip time of the first group is 39.190 seconds and the median 27.610 seconds. The second group has a mean of 39.460 seconds and median of 30.490 seconds. Figure 16 shows a boxplot diagram of both samples.

To find out if this is just random or if the use of the specific measurement nodes had significant impact on the round-trip time, a statistical test needs to be applied.

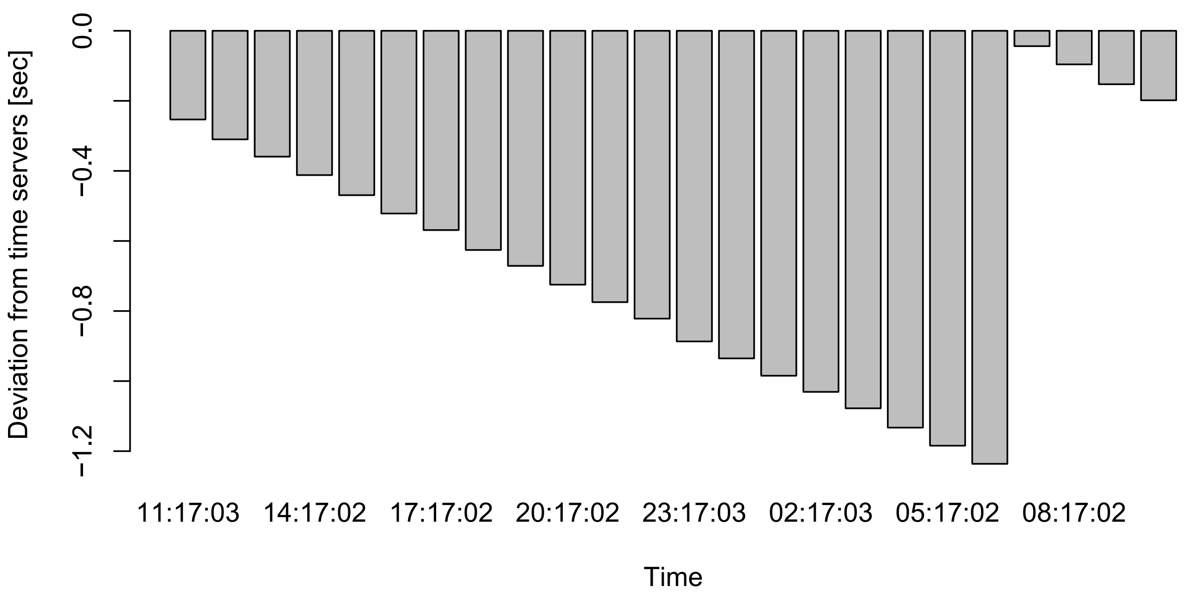

Figure 14: Time deviation of introduction point server

Student's t-test has three assumptions. The data has to be normally distributed, variances of the two samples have to be equal and the samples must be independent.[18] Figure 17 shows a histogram of both samples combined. The distribution is not symmetric and there are no values below zero, because round-trip times were measured, which cannot be negative.

The null hypothesis of the Shapiro-Wilk test for normality is that a distribution is normal.[19] Applying the test with a significance level of 99 % results in a p-value less than 2.2e-16. For a p-value less than 0.01 the null hypothesis must be rejected. Therefore in this case the null hypothesis can be rejected. The data is not normally distributed and so Student's t-test cannot be applied.

Because the data is not normally distributed, a test is chosen, that does not require a normal distribution: the Mann-Whitney U test.[20] The test is based on ranks, not on values. It only requires two independent samples and the null hypothesis is that both samples are from the same distribution.

The test is applied to the sample of 969 attempts using both measurement nodes on the one hand and the sample including the other 229 attempts on the other. The significance level is 99%. Applying the test results in a p-value of 0.8738, which is greater than .005 and therefore the null hypothesis cannot be rejected. This means, that there is no significant difference between the round-trip time resulting in using the measurement nodes and the times resulting in using other relays. Therefore both can be used for further analysis.

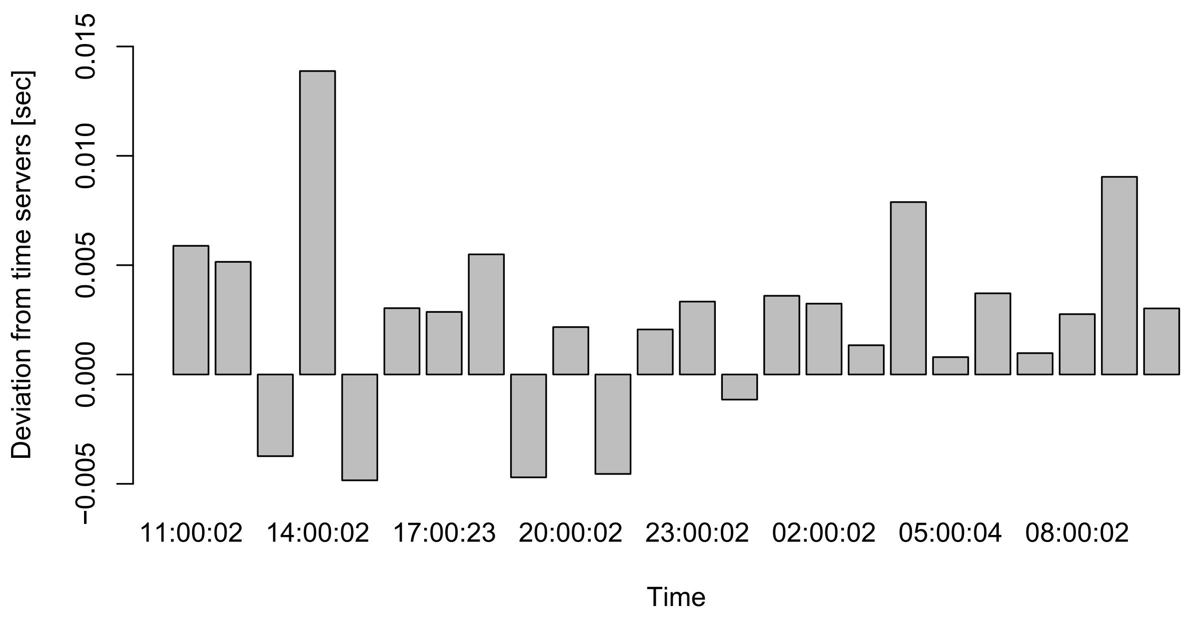

Figure 15: Time deviation of main server

Table 1: Changes due to using more internal circuits

	RTT		IC open		RC open	
	Median	Mean	Median	Mean	Median	Mean
Regular protocol	28.460	39.240	1.793	7.784	0.001	3.063
Changed protocol	25.300	36.190	1.862	7.397	0.001	2.938
Difference	-3.160	-3.05	+0.069	-0.387	±0	-0.125
P value	0.001478		0.6952		0.6245	

6.4 Evaluation of More Internal Circuits

The question is if opening more pre-built circuits really improves the performance of hidden services in terms of latency and therefore decreases the round-trip time. Table 1 lists the mean and median values for the round-trip time, the time until the introduction circuit is successfully opened after receiving the rendezvous service descriptor, and the same value for the rendezvous circuit. The values of the regular protocol base on 1154 successful access attempts, the values of the changed protocol base on 1156 successful attempts.

Due to the heavy-tailed distribution the median of the round-trip time is about 10 seconds higher than the mean. Both mean and average decrease by a little more than 3 seconds, when using the changed protocol. Figure 18 shows a boxplot diagram of the two round-trip times to compare the lower and upper quartiles indicated by the box.

As shown in the previous section Student's t-test cannot be used to evaluate the significance of this change in the round-trip time. Therefore again a Mann-Whitney U test is applied with a significance level of 99 %. The resulting p-value is also shown in the table. The null hypothesis, stating that both samples are from the same distribution, can be rejected.

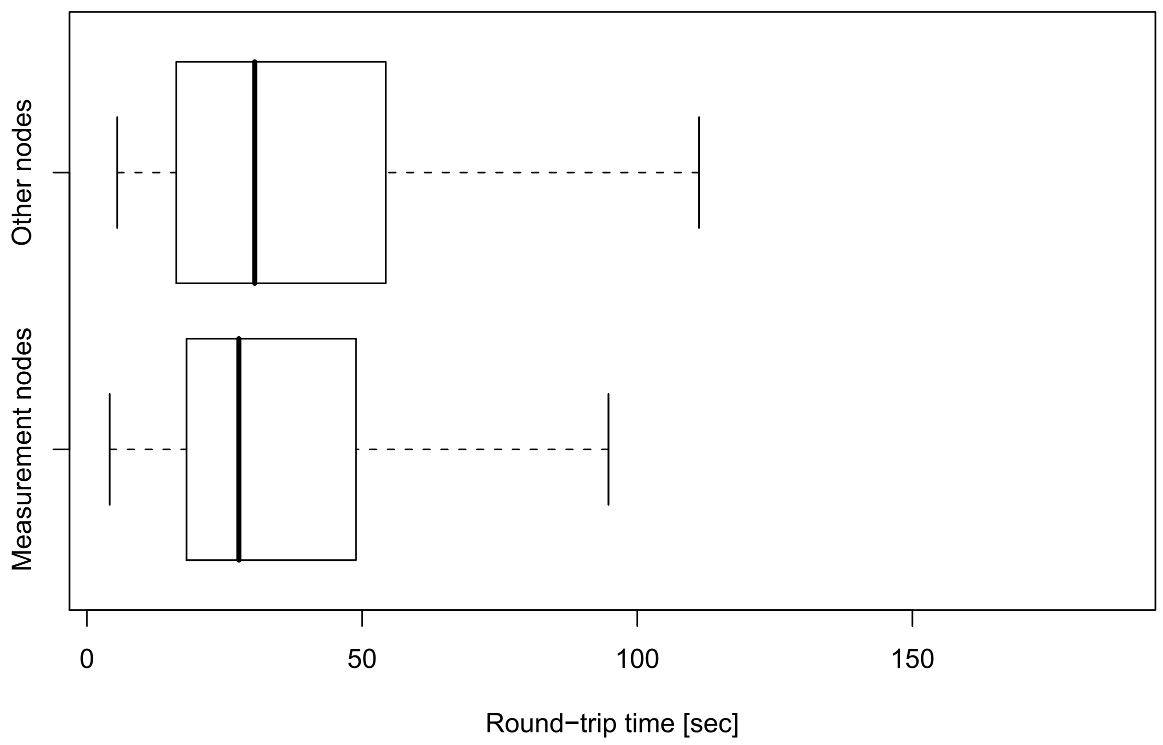

Figure 16: Distribution of attempts using measurement nodes and other nodes

But the mean and median for the time until the first introduction circuit is opened are interesting. They are also listed in the table. While the median went up by 0.069 seconds the mean went down by 0.387 seconds. Applying the same test to the distribution of these values results in a p-value of 0.6952. This means that the null hypothesis cannot be rejected. Both samples are from the same distribution.

The same is true for the rendezvous circuit. The median of the rendezvous circuit is nearly zero, because the rendezvous circuit usually is instantaneously open after picking the circuit. This is because an existing 3-hop circuit is selected, which is usually pre-built and therefore open.

The missing significance for both sub steps is important, because these sub steps are the only ones that can profit from the implemented change. Even if the round-trip time reduced significantly, this is not because of the change of the protocol.

6.5 Evaluation of Opening Two Introduction Circuits

In the second implemented change two circuits were extended to the same introduction point. Table 2 presents changes of the round-trip time and the time until the first introduction circuit was open.

For the round-trip time both mean and median decrease by 6.190 and 7.220 seconds. The

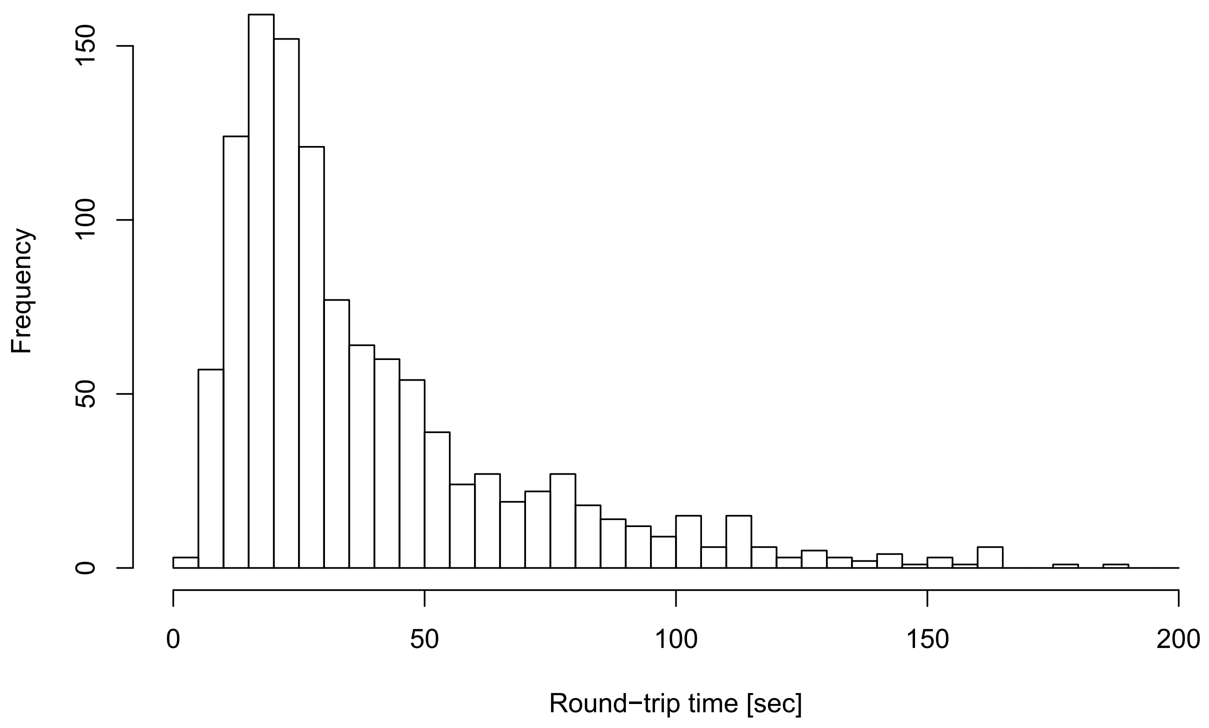

Figure 17: Histogram of all attempts using the original protocol

Table 2: Changes due to opening two introduction circuits

	RTT		IC open	
	Median	Mean	Median	Mean
Regular protocol	28.460	39.240	1.793	7.784
Changed protocol	22.270	32.020	2.472	6.305
Difference	-6.190	-7.220	+0.679	-1.479
P value	5.995e-15		1.557e-09	

p-value of the test allows to reject the null hypothesis, stating that both samples are from the same distribution. Therefore the decrease is significant. Figure 19 shows the boxplot diagram comparing the changed protocol with the regular one.

Also interesting are the values for the introduction circuit. While the median increases by 0.697 seconds, the mean decreases by 1.479 seconds. Unlike the first protocol change, this change of the value is significant with a very low p-value.

The intention of this protocol change was not to make the fast circuits even faster, but to lower the probability of high values, i.e. circuits open after a long time, because of problems preventing a circuit from opening and a second attempt finally opening successfully. This is, why the mean is more interesting, because it considers the heavy tail of the distribution better than the median does.

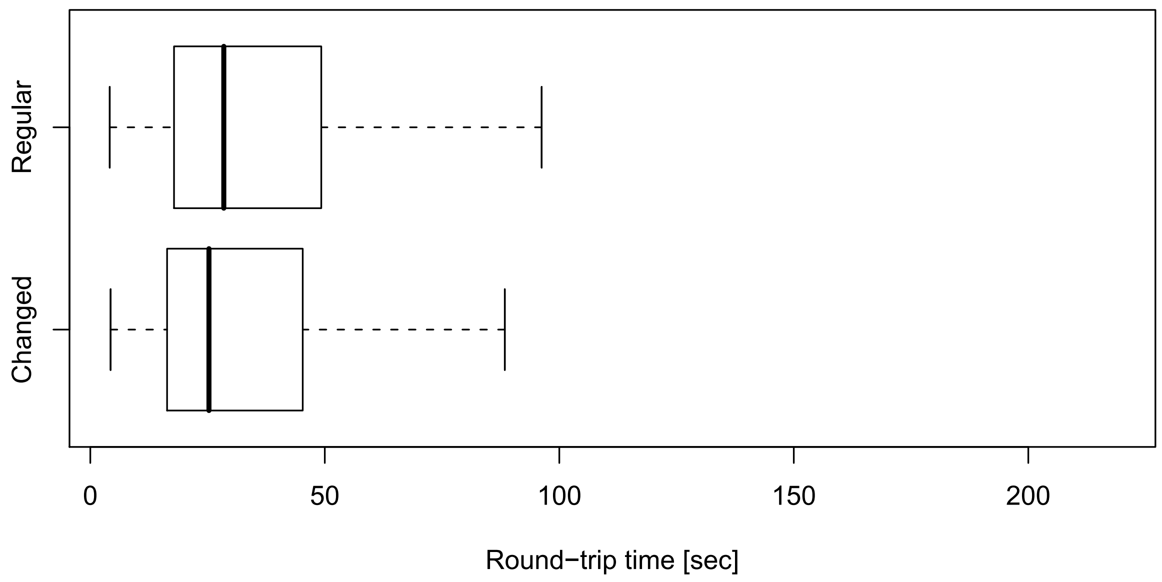

Figure 18: Round-trip time distribution for using more internal circuits

Table 3: Changes due to combining introduction and rendezvous circuits

	RTT	
	Median	Mean
Regular protocol	28.460	39.240
Changed protocol	23.000	33.100
Difference	-5.460	-6.140
P value	5.375e-11	

6.6 Evaluation of Combining Introduction and Rendezvous Circuits

Contacting the introduction point by extending the rendezvous circuit is the third protocol change that was implemented. The resulting round-trip times are listed in Table 3.

Both median and mean decrease by 5.460 and 6.140 seconds. The p-value indicates that the change of the values is significant. A boxplot diagram comparing the round-trip time is shown in Figure 20. It is important to mention, that the valet node concept is necessary not to reduce the anonymity by applying this protocol change. But the concept was not implemented for the measurements. The additional valet nodes will increase the latency a little, because a new circuit needs to be build between valet node and introduction point.

6.7 Evaluation of Direct Hidden Service Usage

The protocol change completely omitting the rendezvous point results in the biggest savings of all protocol changes considered in this thesis. The round-trip times are listed in Table 4. The median decreases by 8.510 seconds, the mean even by 10.25 seconds. The change is also

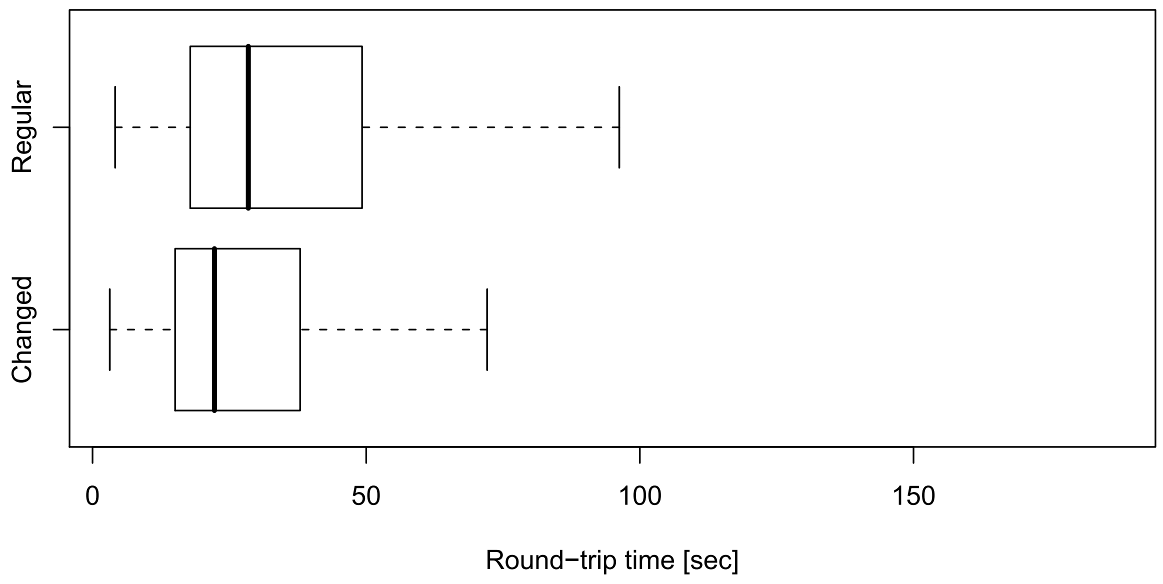

Figure 19: Round-trip time distribution for opening two introduction circuits

Table 4: Changes due to direct hidden service usage

	RTT	
	Median	Mean
Regular protocol	28.460	39.240
Changed protocol	19.950	28.990
Difference	-8.510	-10.25
P value	< 2.2e-16	

significant, because the null hypothesis of the Mann-Whitney U test can be rejected due to a very low p-values. Like for the previous change the valet node concept is also necessary in this protocol change for the same reasons, which will increase latency.

Figure 21 shows the boxplot diagram for this change. The upper quartile decreased from 49.230 to 33.250 seconds. That means that in 75 % of all successful attempts of the changed protocol a reply from the hidden service was received after at most 33 seconds, compared to 49 seconds for the attempts using the original protocol.

7 Conclusion

In this section a summary of this thesis and its achievements is presented. Furthermore next steps founding on this work are proposed.

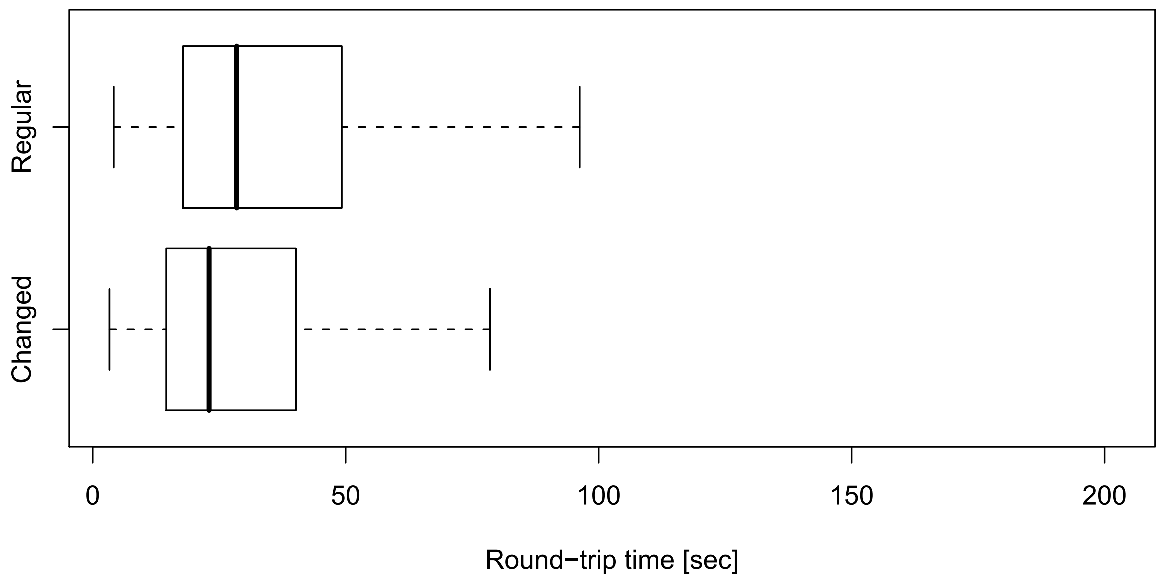

Figure 20: Round-trip time distribution for combined circuits

7.1 Summary

After motivating the usage and research of anonymity systems and their performance different systems to provide location-hidden services were reviewed. It was shown that either broadcast or multicast on the one hand or a variation of onion routing on the other hand can be found in most approaches. The onion routing concept was implemented on different protocol levels like the IP level or the TCP level.

Tor, the anonymity system used for the implementation in this thesis, was described in detail to understand how Tor provides anonymity to clients accessing a public service. Also the regular protocol to offer location-hidden services was analyzed, to build a foundation to improve the protocol in order to increase its performance.

A measurement environment was developed and implemented to measure the performance of hidden services in Tor using the global Tor network. In the environment the central roles involved in accessing a hidden service were controlled to learn from log events, how long certain sub steps of the process take.

Four protocol changes were developed and implemented. The changes covered a broad range of complexity, from simply changing the number of pre-built internal circuits over simultaneous circuit extension attempts to multiple use of the same circuit and omitting a central role of the hidden service concept of Tor.

The protocol changes were measured over a time period of three weeks, using the measurement environment, resulting in 1,200 access attempts per protocol. This data was used to evaluate the impact of the protocol changes on performance empirically. The result is that increasing the number of pre-built internal circuits from two to five does not have a significant impact. The other three changes do have an impact. The mean access time of the regular protocol is 39.240

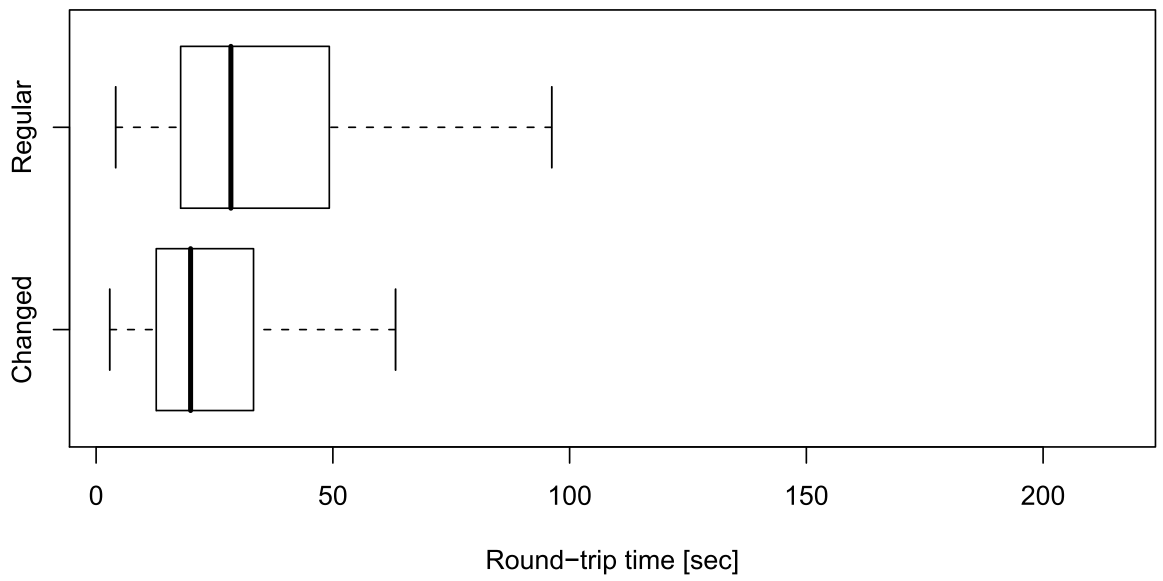

Figure 21: Round-trip time distribution of direct hidden service usage

seconds. Extending two client circuits to the introduction point simultaneously decreases the mean by 7.220 seconds. Extending the rendezvous circuit to the introduction point instead of building an independent introduction circuit results in a decrease of the mean by 6.140 seconds. Completely omitting the rendezvous point and responding over the introduction circuit decreases the mean round-trip time by 10.25 seconds. For the last two changes it is important to mention, that the valet node concept, which is necessary not to reduce anonymity provided by the network, was not implemented for the measurements and will reduce performance a little.

The intention of this thesis to add performance enhancing changes to an existing location-hidden service protocol can be considered as success. Three of the four changes reduced the round-trip time of accessing a hidden service significantly, at most by 25 %. Nevertheless the evaluation is rather short due to time restrictions. This is all the more disappointing because of the huge amount of data collected during the measurements. Although not used in this thesis this data will help to further improve location-hidden services and the regular Tor protocol in the future.

7.2 Future Work

The upcoming work founding on this thesis can be separated in three main categories. The first category considers the changes proposed in Section 5. In the second category the data collected during the measurements needs to be subject of further research. The third category includes the measurement environment presented in Section 4.

The changes presented in this thesis need to be discussed with the developers and community of the Tor network. The implementation shown focused on measuring the impact of the changes. To write a proposal, before implementing the changes into the Tor source code, a clean implementation has to be done.

The work done for this thesis provides a good foundation for further research. This is because the data collected during the three weeks of measurements is much more detailed then what is presented in the results in Section 6.

The building time for individual hops of circuits should be analyzed. The result can be combined with the work of Panchenko et al., who performed dedicated measurements with additional cells to select faster circuits in the path selection. If there is a significant correlation between the building time of a circuit and the time for cannibalization as well as the message transfer time using the circuit, circuits can be rated in terms of expected performance. This would work without additional cells, because the cells for circuit building are sent anyways but their times are not evaluated in the current implementation.

The circuit building timeout of 60 seconds seems very high, but the data collected in this thesis bases on clients installed on a virtual root server with a broadband connection to the Internet. Such a connection cannot be assumed to be representative for all clients. There are clients with much lower bandwidth using modem connection especially in the developing countries. The measurement environment presented in this work should be extended to clients using different bandwidths, e.g. using cell phone connection like the Universal Mobile Telecommunications System (UMTS) or the much slower Global System for Mobile communications (GSM), to evaluate the impact of bandwidth on circuit building times. After those measurements lowering the timeouts might be considered.

References

[1] Sabine Helmers. A brief history of anon.penet.fi - the legendary anonymous remailer. *CMC Magazine*, 1997.

[2] Roger Dingledine and Nick Mathewson. Anonymity loves company: Usability and the network effect. In Ross Anderson, editor, *Proceedings of the Fifth Workshop on the Economics of Information Security (WEIS 2006)*, Cambridge, UK, June 2006.

[3] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In *Information Hiding*, 1996.

[4] Ian Goldberg. *A Pseudonymous Communications Infrastructure for the Internet.* PhD thesis, UC Berkeley, December 2000.

[5] Michael J. Freedman, Emil Sit, Josh Cates, and Robert Morris. Introducing tarzan, a peer-to-peer anonymizing network layer. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02)*, Cambridge, MA, March 2002.

[6] Anon. I2p technical introduction. Website. Available online at `http://www.i2p2.de/techintro.html`; visited on 23 May 2008.

[7] Rob Sherwood and Bobby Bhattacharjee. A protocol for scalable anonymous communication, 2002.

[8] Vincent Scarlata, Brian Levine, and Clay Shields. Responder anonymity and anonymous peer-to-peer file sharing, 2001.

[9] Stefan Köpsell. Low latency anonymous communication - how long are users willing to wait? In *ETRICS, volume 3995 of Lecture Notes in Computer Science, Springer*, 2006.

[10] Rolf Wendolsky, Dominik Herrmann, and Hannes Federrath. Performance comparison of low-latency anonymisation services from a user perspective. In *Privacy Enhancing Technologies*, 2007.

[11] Andriy Panchenko, Lexi Pimenidis, and Johannes Renner. Performance Analysis of Anonymous Communication Channels Provided by Tor. In *Proceedings of the Third International Conference on Availability, Reliability and Security (ARES 2008)*, Barcelona, Spain, March 2008.

[12] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router, 2004.

[13] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6), 1976.

[14] Karsten Loesing, Werner Sandmann, Christian Wilms, and Guido Wirtz. Performance measurements and statistics of tor hidden services. In *Proceedings of the 2008 International Symposium on Applications and the Internet (SAINT 2008)*, July 2008.

[15] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A System Approach, 2nd Edition.* Morgan Kaufman, 1999.

[16] Lasse Øverlier and Paul Syverson. Improving efficiency and simplicity of tor circuit establishment and hidden services. In Nikita Borisov and Philippe Golle, editors, *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007)*, Ottawa, Canada, June 2007. Springer.

[17] Lasse Øverlier and Paul Syverson. Valet services: Improving hidden servers with a personal touch. In *Proceedings of the Sixth Workshop on Privacy Enhancing Technologies (PET 2006).* Springer, June 2006.

[18] Michael C. Fleming and Joseph G. Nellis. *Principles of Applied Statistics.* Routledge & Kegan Paul, London, 1994.

[19] Samuel S. Shapiro and Martin B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 3(52), 1965.

[20] Henry B. Mann and Donald R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1), 1947.

A List of previous University of Bamberg reports

Bamberger Beiträge zur Wirtschaftsinformatik

Nr. 1 (1989)	Augsburger W., Bartmann D., Sinz E.J.: Das Bamberger Modell: Der Diplom-Studiengang Wirtschaftsinformatik an der Universität Bamberg (Nachdruck Dez. 1990)
Nr. 2 (1990)	Esswein W.: Definition, Implementierung und Einsatz einer kompatiblen Datenbankschnittstelle für PROLOG
Nr. 3 (1990)	Augsburger W., Rieder H., Schwab J.: Endbenutzerorientierte Informationsgewinnung aus numerischen Daten am Beispiel von Unternehmenskennzahlen
Nr. 4 (1990)	Ferstl O.K., Sinz E.J.: Objektmodellierung betrieblicher Informationsmodelle im Semantischen Objektmodell (SOM) (Nachdruck Nov. 1990)
Nr. 5 (1990)	Ferstl O.K., Sinz E.J.: Ein Vorgehensmodell zur Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM)
Nr. 6 (1991)	Augsburger W., Rieder H., Schwab J.: Systemtheoretische Repräsentation von Strukturen und Bewertungsfunktionen über zeitabhängigen betrieblichen numerischen Daten
Nr. 7 (1991)	Augsburger W., Rieder H., Schwab J.: Wissensbasiertes, inhaltsorientiertes Retrieval statistischer Daten mit EISREVU / Ein Verarbeitungsmodell für eine modulare Bewertung von Kennzahlenwerten für den Endanwender
Nr. 8 (1991)	Schwab J.: Ein computergestütztes Modellierungssystem zur Kennzahlenbewertung
Nr. 9 (1992)	Gross H.-P.: Eine semantiktreue Transformation vom Entity-Relationship-Modell in das Strukturierte Entity-Relationship-Modell
Nr. 10 (1992)	Sinz E.J.: Datenmodellierung im Strukturierten Entity-Relationship-Modell (SERM)
Nr. 11 (1992)	Ferstl O.K., Sinz E. J.: Glossar zum Begriffsystem des Semantischen Objektmodells
Nr. 12 (1992)	Sinz E. J., Popp K.M.: Zur Ableitung der Grobstruktur des konzeptuellen Schemas aus dem Modell der betrieblichen Diskurswelt
Nr. 13 (1992)	Esswein W., Locarek H.: Objektorientierte Programmierung mit dem Objekt-Rollenmodell
Nr. 14 (1992)	Esswein W.: Das Rollenmodell der Organsiation: Die Berücksichtigung aufbauorganisatorische Regelungen in Unternehmensmodellen
Nr. 15 (1992)	Schwab H. J.: EISREVU-Modellierungssystem. Benutzerhandbuch
Nr. 16 (1992)	Schwab K.: Die Implementierung eines relationalen DBMS nach dem Client/Server-Prinzip
Nr. 17 (1993)	Schwab K.: Konzeption, Entwicklung und Implementierung eines computergestützten Bürovorgangssystems zur Modellierung von Vorgangsklassen und Abwicklung und Überwachung von Vorgängen. Dissertation
Nr. 18 (1993)	Ferstl O.K., Sinz E.J.: Der Modellierungsansatz des Semantischen Objektmodells

Nr. 19 (1994)	Ferstl O.K., Sinz E.J., Amberg M., Hagemann U., Malischewski C.: Tool-Based Business Process Modeling Using the SOM Approach
Nr. 20 (1994)	Ferstl O.K., Sinz E.J.: From Business Process Modeling to the Specification of Distributed Business Application Systems - An Object-Oriented Approach -. 1st edition, June 1994
	Ferstl O.K., Sinz E.J. : Multi-Layered Development of Business Process Models and Distributed Business Application Systems - An Object-Oriented Approach -. 2nd edition, November 1994
Nr. 21 (1994)	Ferstl O.K., Sinz E.J.: Der Ansatz des Semantischen Objektmodells zur Modellierung von Geschäftsprozessen
Nr. 22 (1994)	Augsburger W., Schwab K.: Using Formalism and Semi-Formal Constructs for Modeling Information Systems
Nr. 23 (1994)	Ferstl O.K., Hagemann U.: Simulation hierarischer objekt- und transaktionsorientierter Modelle
Nr. 24 (1994)	Sinz E.J.: Das Informationssystem der Universität als Instrument zur zielgerichteten Lenkung von Universitätsprozessen
Nr. 25 (1994)	Wittke M., Mekinic, G.: Kooperierende Informationsräume. Ein Ansatz für verteilte Führungsinformationssysteme
Nr. 26 (1995)	Ferstl O.K., Sinz E.J.: Re-Engineering von Geschäftsprozessen auf der Grundlage des SOM-Ansatzes
Nr. 27 (1995)	Ferstl, O.K., Mannmeusel, Th.: Dezentrale Produktionslenkung. Erscheint in CIM-Management 3/1995
Nr. 28 (1995)	Ludwig, H., Schwab, K.: Integrating cooperation systems: an event-based approach
Nr. 30 (1995)	Augsburger W., Ludwig H., Schwab K.: Koordinationsmethoden und -werkzeuge bei der computergestützten kooperativen Arbeit
Nr. 31 (1995)	Ferstl O.K., Mannmeusel T.: Gestaltung industrieller Geschäftsprozesse
Nr. 32 (1995)	Gunzenhäuser R., Duske A., Ferstl O.K., Ludwig H., Mekinic G., Rieder H., Schwab H.-J., Schwab K., Sinz E.J., Wittke M: Festschrift zum 60. Geburtstag von Walter Augsburger
Nr. 33 (1995)	Sinz, E.J.: Kann das Geschäftsprozeßmodell der Unternehmung das unternehmensweite Datenschema ablösen?
Nr. 34 (1995)	Sinz E.J.: Ansätze zur fachlichen Modellierung betrieblicher Informationssysteme - Entwicklung, aktueller Stand und Trends -
Nr. 35 (1995)	Sinz E.J.: Serviceorientierung der Hochschulverwaltung und ihre Unterstützung durch workflow-orientierte Anwendungssysteme
Nr. 36 (1996)	Ferstl O.K., Sinz, E.J., Amberg M.: Stichwörter zum Fachgebiet Wirtschaftsinformatik. Erscheint in: Broy M., Spaniol O. (Hrsg.): Lexikon Informatik und Kommunikationstechnik, 2. Auflage, VDI-Verlag, Düsseldorf 1996
Nr. 37 (1996)	Ferstl O.K., Sinz E.J.: Flexible Organizations Through Object-oriented and Transaction-oriented Information Systems, July 1996

Nr. 38 (1996)	Ferstl O.K., Schäfer R.: Eine Lernumgebung für die betriebliche Aus- und Weiterbildung on demand, Juli 1996
Nr. 39 (1996)	Hazebrouck J.-P.: Einsatzpotentiale von Fuzzy-Logic im Strategischen Management dargestellt an Fuzzy-System-Konzepten für Portfolio-Ansätze
Nr. 40 (1997)	Sinz E.J.: Architektur betrieblicher Informationssysteme. In: Rechenberg P., Pomberger G. (Hrsg.): Handbuch der Informatik, Hanser-Verlag, München 1997
Nr. 41 (1997)	Sinz E.J.: Analyse und Gestaltung universitärer Geschäftsprozesse und Anwendungssysteme. Angenommen für: Informatik '97. Informatik als Innovationsmotor. 27. Jahrestagung der Gesellschaft für Informatik, Aachen 24.-26.9.1997
Nr. 42 (1997)	Ferstl O.K., Sinz E.J., Hammel C., Schlitt M., Wolf S.: Application Objects – fachliche Bausteine für die Entwicklung komponentenbasierter Anwendungssysteme. Angenommen für: HMD – Theorie und Praxis der Wirtschaftsinformatik. Schwerpunkheft ComponentWare, 1997
Nr. 43 (1997):	Ferstl O.K., Sinz E.J.: Modeling of Business Systems Using the Semantic Object Model (SOM) – A Methodological Framework - . Accepted for: P. Bernus, K. Mertins, and G. Schmidt (ed.): Handbook on Architectures of Information Systems. International Handbook on Information Systems, edited by Bernus P., Blazewicz J., Schmidt G., and Shaw M., Volume I, Springer 1997
	Ferstl O.K., Sinz E.J.: Modeling of Business Systems Using (SOM), 2nd Edition. Appears in: P. Bernus, K. Mertins, and G. Schmidt (ed.): Handbook on Architectures of Information Systems. International Handbook on Information Systems, edited by Bernus P., Blazewicz J., Schmidt G., and Shaw M., Volume I, Springer 1998
Nr. 44 (1997)	Ferstl O.K., Schmitz K.: Zur Nutzung von Hypertextkonzepten in Lernumgebungen. In: Conradi H., Kreutz R., Spitzer K. (Hrsg.): CBT in der Medizin – Methoden, Techniken, Anwendungen -. Proceedings zum Workshop in Aachen 6. – 7. Juni 1997. 1. Auflage Aachen: Verlag der Augustinus Buchhandlung
Nr. 45 (1998)	Ferstl O.K.: Datenkommunikation. In. Schulte Ch. (Hrsg.): Lexikon der Logistik, Oldenbourg-Verlag, München 1998
Nr. 46 (1998)	Sinz E.J.: Prozeßgestaltung und Prozeßunterstützung im Prüfungswesen. Erschienen in: Proceedings Workshop „Informationssysteme für das Hochschulmanagement". Aachen, September 1997
Nr. 47 (1998)	Sinz, E.J.:, Wismans B.: Das „Elektronische Prüfungsamt". Erscheint in: Wirtschaftswissenschaftliches Studium WiSt, 1998
Nr. 48 (1998)	Haase, O., Henrich, A.: A Hybrid Respresentation of Vague Collections for Distributed Object Management Systems. Erscheint in: IEEE Transactions on Knowledge and Data Engineering
Nr. 49 (1998)	Henrich, A.: Applying Document Retrieval Techniques in Software Engineering Environments. In: Proc. International Conference on Database and Expert Systems Applications. (DEXA 98), Vienna, Austria, Aug. 98, pp. 240-249, Springer, Lecture Notes in Computer Sciences, No. 1460

Nr. 50 (1999)	Henrich, A., Jamin, S.: On the Optimization of Queries containing Regular Path Expressions. Erscheint in: Proceedings of the Fourth Workshop on Next Generation Information Technologies and Systems (NGITS'99), Zikhron-Yaakov, Israel, July, 1999 (Springer, Lecture Notes)
Nr. 51 (1999)	Haase O., Henrich, A.: A Closed Approach to Vague Collections in Partly Inaccessible Distributed Databases. Erscheint in: Proceedings of the Third East-European Conference on Advances in Databases and Information Systems – ADBIS'99, Maribor, Slovenia, September 1999 (Springer, Lecture Notes in Computer Science)
Nr. 52 (1999)	Sinz E.J., Böhnlein M., Ulbrich-vom Ende A.: Konzeption eines Data Warehouse-Systems für Hochschulen. Angenommen für: Workshop „Unternehmen Hochschule" im Rahmen der 29. Jahrestagung der Gesellschaft für Informatik, Paderborn, 6. Oktober 1999
Nr. 53 (1999)	Sinz E.J.: Konstruktion von Informationssystemen. Der Beitrag wurde in geringfügig modifizierter Fassung angenommen für: Rechenberg P., Pomberger G. (Hrsg.): Informatik-Handbuch. 2., aktualisierte und erweiterte Auflage, Hanser, München 1999
Nr. 54 (1999)	Herda N., Janson A., Reif M., Schindler T., Augsburger W.: Entwicklung des Intranets SPICE: Erfahrungsbericht einer Praxiskooperation.
Nr. 55 (2000)	Böhnlein M., Ulbrich-vom Ende A.: Grundlagen des Data Warehousing. Modellierung und Architektur
Nr. 56 (2000)	Freitag B, Sinz E.J., Wismans B.: Die informationstechnische Infrastruktur der Virtuellen Hochschule Bayern (vhb). Angenommen für Workshop "Unternehmen Hochschule 2000" im Rahmen der Jahrestagung der Gesellschaft f. Informatik, Berlin 19. - 22. September 2000
Nr. 57 (2000)	Böhnlein M., Ulbrich-vom Ende A.: Developing Data Warehouse Structures from Business Process Models.
Nr. 58 (2000)	Knobloch B.: Der Data-Mining-Ansatz zur Analyse betriebswirtschaftlicher Daten.
Nr. 59 (2001)	Sinz E.J., Böhnlein M., Plaha M., Ulbrich-vom Ende A.: Architekturkonzept eines verteilten Data-Warehouse-Systems für das Hochschulwesen. Angenommen für: WI-IF 2001, Augsburg, 19.-21. September 2001
Nr. 60 (2001)	Sinz E.J., Wismans B.: Anforderungen an die IV-Infrastruktur von Hochschulen. Angenommen für: Workshop „Unternehmen Hochschule 2001" im Rahmen der Jahrestagung der Gesellschaft für Informatik, Wien 25. – 28. September 2001

Änderung des Titels der Schriftenreihe *Bamberger Beiträge zur Wirtschaftsinformatik* in *Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik* ab Nr. 61

Note: The title of our technical report series has been changed from *Bamberger Beiträge zur Wirtschaftsinformatik* to *Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik* starting with TR No. 61

Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik

Nr. 61 (2002)	Goré R., Mendler M., de Paiva V. (Hrsg.): Proceedings of the International Workshop on Intuitionistic Modal Logic and Applications (IMLA 2002), Copenhagen, July 2002.
Nr. 62 (2002)	Sinz E.J., Plaha M., Ulbrich-vom Ende A.: Datenschutz und Datensicherheit in einem landesweiten Data-Warehouse-System für das Hochschulwesen. Erscheint in: Beiträge zur Hochschulforschung, Heft 4-2002, Bayerisches Staatsinstitut für Hochschulforschung und Hochschulplanung, München 2002
Nr. 63 (2005)	Aguado, J., Mendler, M.: Constructive Semantics for Instantaneous Reactions
Nr. 64 (2005)	Ferstl, O.K.: Lebenslanges Lernen und virtuelle Lehre: globale und lokale Verbesserungspotenziale. Erschienen in: Kerres, Michael; Keil-Slawik, Reinhard (Hrsg.); Hochschulen im digitalen Zeitalter: Innovationspotenziale und Strukturwandel, S. 247 – 263; Reihe education quality forum, herausgegeben durch das Centrum für eCompetence in Hochschulen NRW, Band 2, Münster/New York/München/Berlin: Waxmann 2005
Nr. 65 (2006)	Schönberger, Andreas: Modelling and Validating Business Collaborations: A Case Study on RosettaNet
Nr. 66 (2006)	Markus Dorsch, Martin Grote, Knut Hildebrandt, Maximilian Röglinger, Matthias Sehr, Christian Wilms, Karsten Loesing, and Guido Wirtz: Concealing Presence Information in Instant Messaging Systems, April 2006
Nr. 67 (2006)	Marco Fischer, Andreas Grünert, Sebastian Hudert, Stefan König, Kira Lenskaya, Gregor Scheithauer, Sven Kaffille, and Guido Wirtz: Decentralized Reputation Management for Cooperating Software Agents in Open Multi-Agent Systems, April 2006
Nr. 68 (2006)	Michael Mendler, Thomas R. Shiple, Gérard Berry: Constructive Circuits and the Exactness of Ternary Simulation
Nr. 69 (2007)	Sebastian Hudert: A Proposal for a Web Services Agreement Negotiation Protocol Framework . February 2007
Nr. 70 (2007)	Thomas Meins: Integration eines allgemeinen Service-Centers für PC-und Medientechnik an der Universität Bamberg – Analyse und Realisierungs-Szenarien. Februar 2007
Nr. 71 (2007)	Andreas Grünert: Life-cycle assistance capabilities of cooperating Software Agents for Virtual Enterprises. März 2007
Nr. 72 (2007)	Michael Mendler, Gerald Lüttgen: Is Observational Congruence on µ-Expressions Axiomatisable in Equational Horn Logic?
Nr. 73 (2007)	Martin Schissler: to be announced
Nr. 74 (2007)	Sven Kaffille, Karsten Loesing: Open chord version 1.0.4 User's Manual. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 74, Bamberg University, October 2007. ISSN 0937-3349.