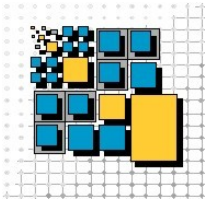

University of Bamberg

Distributed and Mobile Systems Group

Paper

on the Seminar

Tor Research

Topic:

Performance Evaluation of Tor Hidden Services

Presented by:

Christian Wilms

Supervisor: Karsten Loesing

Bamberg, Summer term 2007

Contents

List of Figures

List of Tables

List of Abbreviations

CEST	Central European Summer Time
HTTP	Hyper Text Transfer Protocol
IP	Internet Protocol
IPO	Introduction Point
IT	Information Technology
RPO	Rendezvous Point
RSD	Rendezvous Service Descriptor
RTT	Round Trip Time
SVN	Subversion
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URL	Uniform Resource Locator

1 Motivation

In large companies whistleblowing can often help to prevent fraud. But even if employees have knowledge of malpractice, for example of their superiors, and are willing to report it to protect the company, another condition must be given to enable them to do the final step and report without being afraid of losing their job or fearing other personal consequences.

Anonymity enables people to take action without revealing their identity. The common form of anonymity is the anonymity of a sender during communication. That means that the receiver of a message is known to the sender, but the receiver doesn't know who sent the message.

1.1 The Tor Network

The low-latency anonymity network Tor uses a technique called onion routing to provide sender anonymity. This enables a user to connect to servers for example for a website request without revealing their identity. All the user needs is the Tor application with a standard configuration as a client. The client connects to a random node within the network consisting of hundreds of nodes and creates a so-called circuit of a few onion routers to an exit node that connects to the requested service. It does this by negotiating TLS secured TCP connections by exchanging relay cells secured by several layers of encryption, one for every hop on the circuit, hence it is called onion routing. At the end of this process a TCP stream is established between the client and the exit node. The service only sees the exit node requesting a website for example, but not the user who originally initiated the request.[1]

1.2 Tor Hidden Services

Sender anonymity as described above can help an employee to file a complaint in a company's online suggestions website anonymously. But what can he do to collect details of fraud over time and for example publish them on a web log? In this case he is the one who offers a service and others need to know where it is, that means they must know a URL or an IP address to access it. But the company might also be able to identify the employee by knowing IP or URL.

The Tor network includes a feature called hidden services, that enables responder anonymity in addition to sender anonymity. With hidden services one can offer a service on the web, like a web site or an IRC channel, that is accessible for anyone using an onion proxy without revealing the real location, i.e. the IP address of the service.

Tor uses rendezvous points to establish a TCP stream between client and hidden service. The hidden service is identified by an onion address, like *http://6sxoyfb3h2nvok2d.onion/*. That is the only information users have to know to access the service. The sequences necessary to access a hidden service are shown in figure 1.

When the onion router providing the hidden service is started, it selects three other onion

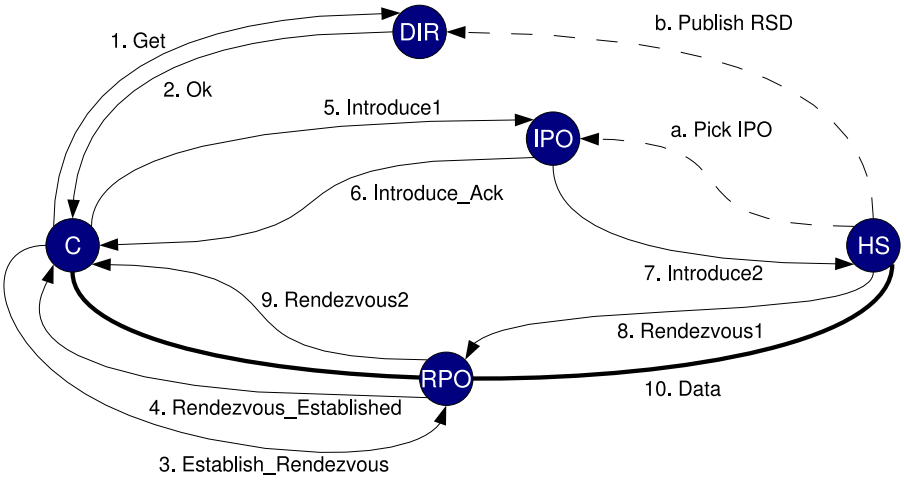

Figure 1: Hidden Service Message Exchange

routers in the network as introduction points (a.). The information about the introduction points is published as rendezvous service descriptor on several directory servers (b.). A more technical description including the involved messages is not given here, because this paper focuses on accessing a hidden service, not setting it up.

When a client application tries to access an onion address the Tor application on the client recognizes it and receives the rendezvous service descriptor from a directory server (1. 2.). It also establishes a circuit to a node within the network as its rendezvous point. An `Establish_Rendezvous` cell is sent to the rendezvous point (3.) and the rendezvous point confirms by answering with a `Rendezvous_Established` cell back to the client (4.). The client extracts the information about introduction points from the received rendezvous service descriptor and builds a circuit to one of them. When the circuit is established the client sends an `Introduce1` cell to the introduction point, including information about the rendezvous point (5.). The introcution point confirms with an `Introduce_Ack` cell (6.). It also notifies the hidden service of the request with an `Introduce2` cell (7.). Now the hidden service establishes a circuit to the rendezvous point and sends a `Rendezvous1` cell (8.). The rendezvous point connects the two circuits and notifies the client with a `Rendezvous2` cell (9.). When receiving the `Rendezvous2` cell the client's Tor application informs the client application that the connection is open and ready to exchange data (10.).

1.3 Performance of Hidden Services

The Tor network makes it possible to add anonymity features to existing communication protocols like instant messaging. Using Socks all kinds of applications are able to send their information through the Tor network. But to offer an incentive to users and developers the network needs to provide a certain performance.

The performance of already established connections through the network can be expressed by throughput and latency. If only small packets with information need be send through the network this kind of performance can be neglected. But for protocols when communication takes place between many different and changing communication partners, it is way more important to establish new connections in a quick manner. The former kind of performance is not subject of this paper but the latter is.

2 Testing Setup

2.1 Testing Environment

To get realistic performance data we had to use the existing Tor network with nodes spread all over the world instead of setting up an own private network. The advantage of a private network is that you have full control over all involved communication partners, i.e. client, onion nodes, hidden service and directory servers.

Using the global Tor network for testing we still tried to control as many roles in the process of establishing a connection to a hidden services as necessary. This includes client, hidden service, introduction point and rendezvous point. It does not include other entry nodes, middle nodes, exit nodes for the circuits as well as directory servers.

Therefore we used a virtual root server located in Germany and set up two instances of Tor version 0.2.0.6 alpha (r11276). One was configured to provide a hidden service and one was configured to be a middle node that could be selected as introduction and rendezvous point. The former instance was started one hour after the latter, to make sure that the introduction point could be found. The first client was started with another delay of one hour for the same reason.

2.2 Tor Configuration

We used the default configuration of the 0.2.0.6 alpha (r11276) version for both the hidden service and the rendezvous/introduction point except for the following options:

- Set `SocksPort 0` to disable local application connections
- Comment `SocksListenAddress` out
- Enable logging on Info level
- Disable safe logging with `SafeLogging 0`
- To run Tor as a daemon set `RunAsDaemon 1`
- Specify an own data directory with `DataDirectory`
- Set `Nickname` and `ContactInfo`
- Limit bandwidth usage with `BandwidthRate 20 KB` and `BandwidthBurst 80 KB`
- Set `ORPort` and reject all exits with `ExitPolicy reject *:*`

2.3 Hidden Service Specific Configuration

The hidden service uses all configuration options listed above plus the following options:

- Uncomment and specify `HiddenServiceDir` and `HiddenServicePort`
- Recommend to use own introduction point with `HiddenServiceNodes <IntroID>`

<IntroID> is the unique public key hash of the hidden service node.

2.4 PuppeTor Configuration for Client

To measure the performance of hidden services over time individual independent clients should try to access the same service. Right after startup the clients should not have any information about other nodes already, because cached information from previous runs could accelerate the access and falsify the results.

To simulate independent clients accessing the hidden service, the Java framework Puppetor[1] was used. After creating a network and a single proxy in the network, configuration options were added to that proxy. The options specify the location of the log files and recommend a certain rendezvous node using the `RendNodes` option.

The network was started and waits until the first circuit is built. After getting the notice that Tor is running it waits another 120 seconds before a client application is created to access the hidden service. The client performs its request and waits for receiving the reply. Right after receiving the reply the PuppeTor network containing only the client is immediately shut down.

The cron daemon of the virtual root server is configured to start the PuppeTor class twelve times per hour, at h:00, h:05, h:10, h:15, h:20, h:25, h:30, h:35, h:40, h:45, h:50, and h:55 for 24 hours, resulting in 24 * 12 = 288 measurements.

2.5 Code Changes at Client Instance

We tried to limit changes on Tor source code as much as possible, to ease repetition of the test with default configurations. But it was necessary to disable the random generator that randomly chooses one of the three IPOs to connect to and instead always use the first router in the RSD. It can be found in the `rend_client_get_random_intro()` function in `rendclient.c`.

3 Problems

When setting up the testing environment and collecting data we experienced several problems. Some of these problems could be solved, others could not.

[1] `https://tor-svn.freehaven.net/svn/puppetor`

3.1 Unknown Router

If the onion proxy has no router descriptor of an introduction point, it cannot connect to it and chooses another one to connect to. If all three introduction points cannot be found, accessing the hidden service fails and it is unavailable. In the beginning of the tests, we tried to request the hidden service right after getting the log notice that Tor has successfully opened a circuit. But the number of unknown routers was pretty high. In some cases the first IPO was unknown and the second wasn't and ten minutes later it was vice versa. All three IPOs were marked as available in the directories.

A workaround for the problem could be found by waiting 120 seconds after getting the log notice that Tor is running until the hidden services was requested. Sometimes a router is still unknown, but the frequency is pretty low. The log notice that the client functionality seems to work doesn't mean that all possible router descriptors are already fetched. The probability is reasonable, that at least one of three IPOs is available at the beginning, but if one wants to choose a specific one, some delay is necessary.

3.2 Unused Rendezvous Node

The Tor client configuration option to choose a specific onion router as rendezvous point (`RendNodes`) only works if a new circuit is build to connect to that router. If an existing circuit is cannibalized, the option is ignored. Without the above mentioned 120 seconds delay sometimes a new circuit was built and the specified rendezvous point was used, probably because there weren't enough existing circuits. With the delay cannibalization was used in all accessing attempts during the three 24 hour runs, without exception. Therefore we had no control over the rendezvous node, because other onion routers were chosen as RPO instead of the specified one and we had no access to the logs.

3.3 Missing Log Statements

Standard log statements should be used to identify events of interest to enable repeating the test with an out-of-the-box Tor version. But during early testing some critical statements were missing and made it impossible to distinguish between several actions, especially after the hidden service has built a circuit to the rendezvous point.

Therefore, no data was available concerning the hidden service contacting the rendezvous point and the latter notifying the client, i.e. the `Rendezvous1` and `Rendezvous2` cells, as well as the HTTP request of PuppeTor and the the according reply. Only the begin and end of those four actions combined could be determined by observing the shutdown of the onion proxy initiated by PuppeTor right after receiving the HTTP reply.

With revision 11074 those statements were added to the Tor SVN and could be used in this paper.

3.4 Server Latency

The German IT magazine *iX* published an article during the tests that the response time of the virtual server hoster *1blu*[2], that provided the root server used for the tests, had a high variability with an average of 42 ms and a maximum of 200 ms. The packet loss was two per cent.[2] This might have an influence on the measurements.

4 Results

Three 24 hour runs were realized, to determine if the runs produce similar results. The times when the three 24 hour tests took place can be found in table 1, as well as general information concerning the runs. The values in the *Successful* row are the number of successful attempts, i.e. a connection to the hidden service could be established and the client received a reply from the service.

Table 1: General Information about Runs

	Run 1	Run 2	Run 3
Access attempts	288	288	288
Start date	Mon, 8/27	Tue, 8/28	Wed, 8/29
Start time	2 am CEST	2 am CEST	2 am CEST
Successful	270	281	281
Successful (%)	93.75	97.57	97.57
Own IPO	247	249	250

The following values could be measured and will be presented in this paper:

- Complete round-trip time
- Descriptor round-trip time
- Establishing rendezvous point
- Building introduction circuit
- `Introduce1` cell
- Acknowledging introduction
- Contacting hidden service
- Build rendezvous circuit
- `Rendezvous1` & `Rendezvous2` cells
- Data round-trip time

[2] `http://www.1blu.de/`

4.1 Complete Round-trip Time

The complete round-trip time was measured from an external point of view. It is the time between sending an HTTP request by the client and receiving the reply to this request. The frequencies of complete round trip are shown in figure 2. Table 2 lists the median (50%), upper quartile (75%), quartile difference and ninth decile (90%) for the value in all three runs.

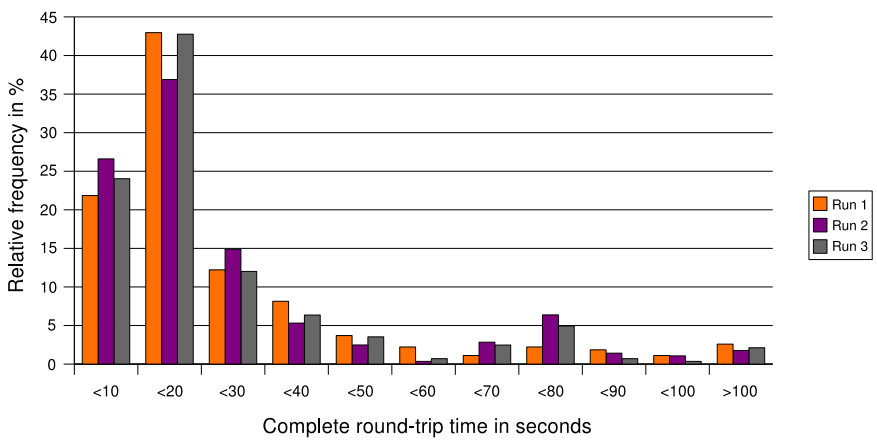

Figure 2: Relative Frequencies of Complete Round-trip Time

Table 2: Distribution of Complete Round-trip Time (s)

	Run 1	Run 2	Run 3
Median	15.600	14.915	15.418
Upper Quartile	28.393	26.829	25.590
Quartile Difference	17.658	17.114	15.262
Ninth Decile	56.103	70.200	64.491
Maximum	139.056	132.794	151.853

Figure 3 shows the complete round-trip times for all individual attempts of run 1 exemplarily. Corresponding graphs for runs 2 and 3 can be found in the appendix.

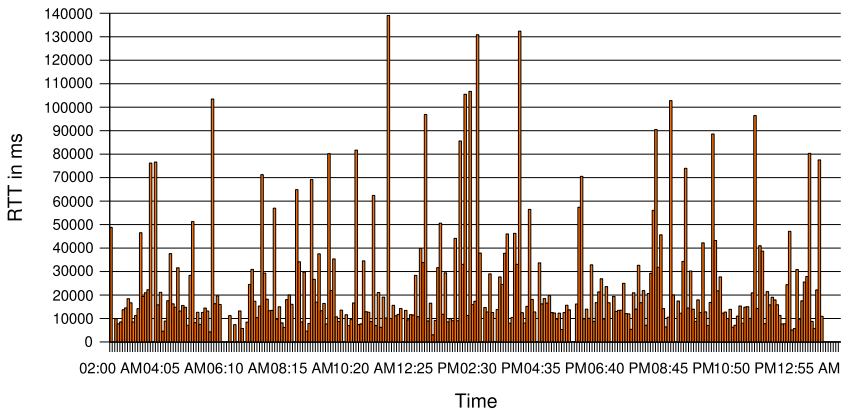

Figure 3: Complete Round-trip Time for all Run 1 Attempts

4.2 Descriptor Round-trip Time

We had no control over the directory server, therefore only the round-trip time of requesting and receiving a rendezvous service descriptor could be measured. Measurement started when the RSD was requested and stopped, when it was received. The relative frequencies can be seen in figure 4 and statistical values in table 3.

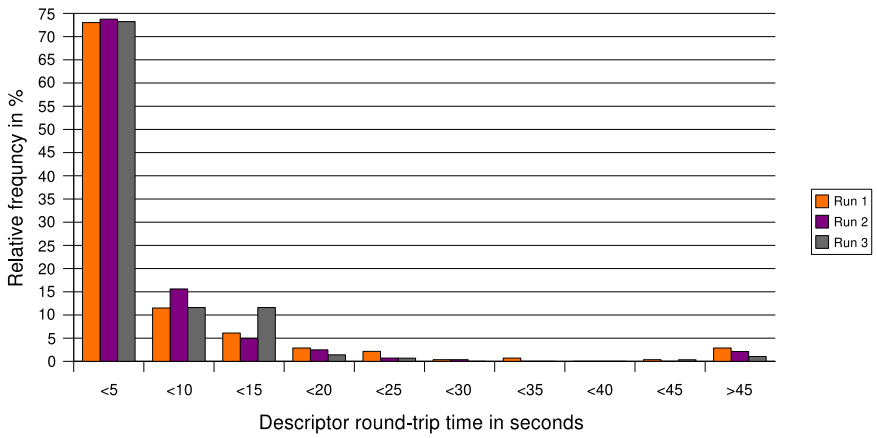

Figure 4: Relative Frequencies of Descriptor Round-trip Time

Table 3: Distribution of Descriptor Round-trip Time (s)

	Run 1	Run 2	Run 3
Median	2.498	2.597	2.525
Upper Quartile	5.741	5.370	5.314
Quartile Difference	4.802	4.224	4.436
Ninth Decile	13.537	11.690	11.266
Maximum	104.133	81.759	60.982

4.3 Establishing Rendezvous Point

Due to the problems to use the own controlled rendezvous point, round-trip times could be measured only. The measured value describes the time between the client sending the rendezvous cell `Establish_Rendezvous` and receiving the acknowledge `Rendezvous_Established`. The data can be found in figure 5 and table 4.

The circuit that was used to exchange those two cells was immediately open after receiving the RSD, because of cannibalization.

Table 4: Distribution of Duration to Establish RPO (s)

	Run 1	Run 2	Run 3
Median	0.577	0.434	0.494
Upper Quartile	1.665	1.282	1.438
Quartile Difference	1.375	1.039	1.216
Ninth Decile	3.761	2.942	2.566
Maximum	52.755	55.006	39.317

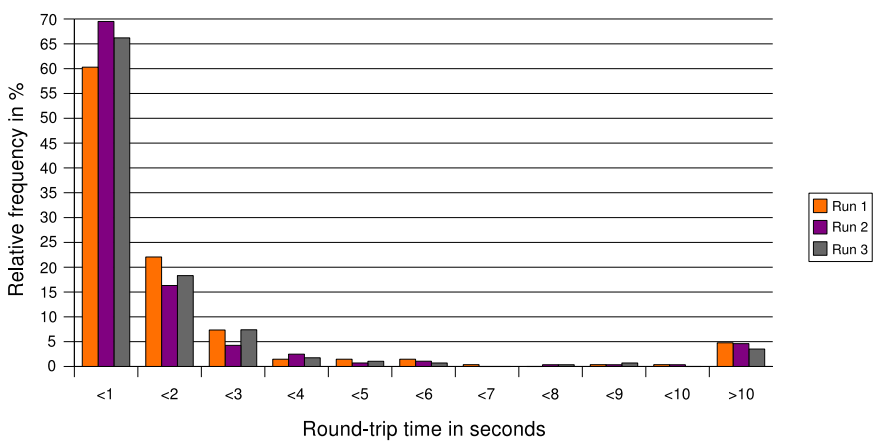

Figure 5: Relative Frequencies of Duration to Establish Connection to RPO

4.4 Building Introduction Circuit

The introduction circuit is the first circuit that had to be extended in the process of accessing a hidden service, because the introduction points of a service are not known until receiving the RSD. This value is measured between receiving the RSD and the log notice that the introduction circuit is open. It is shown in figure 6 and table 5.

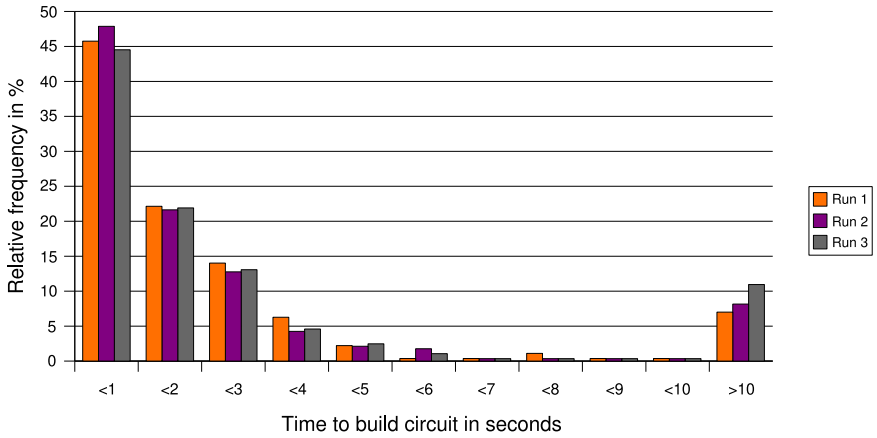

Figure 6: Relative Frequencies of Duration to Build Introduction Circuit

Table 5: Distribution of Duration to Build Introduction Circuit (s)

	Run 1	Run 2	Run 3
Median	1.169	1.146	1.214
Upper Quartile	2.244	2.266	2.542
Quartile Difference	1.682	1.831	2.043
Ninth Decile	4.795	5.635	12.786
Maximum	106.555	110.274	76.276

4.5 Introduce1 Cell

Using the open introduction circuit an `Introduce1` cell is sent to the IPO, containing information about the rendezvous point. This is not a round-trip time, because we controlled both sender and receiver of this message. The time measured between the client

sending the cell and the introduction point receiving it can be found in figure 7 and table 6.

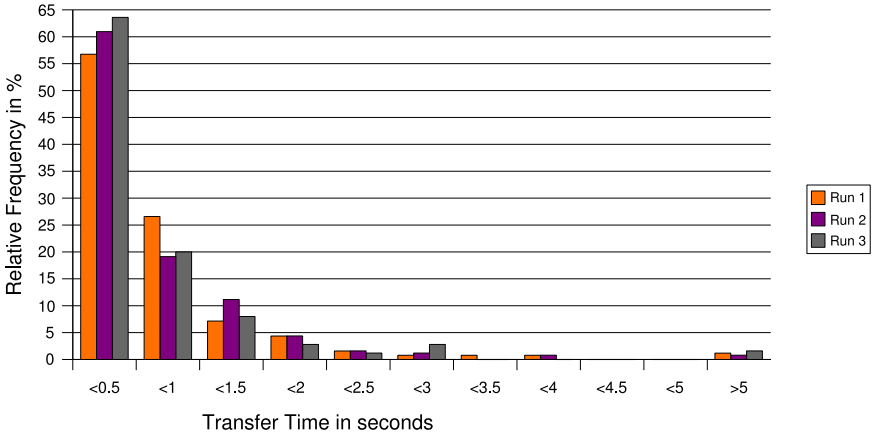

Figure 7: Relative Frequencies of Transfer Time of `Introduce1` Cell

Table 6: Distribution of Transfer Time of `Introduce1` Cell (s)

	Run 1	Run 2	Run 3
Median	0.350	0.249	0.272
Upper Quartile	0.933	0.895	0.848
Quartile Difference	0.771	0.763	0.703
Ninth Decile	1.364	1.398	1.273
Maximum	23.880	61.104	32.427

4.6 Acknowledging Introduction

The introduction point replies with an `Introduce_Ack` cell sent back to the client. The same circuit is used as for the `Introduce1` cell. The transfer time is measured between sending the cell at the introduction point and receiving it at the client. Values are shown in figure 8 and table 7.

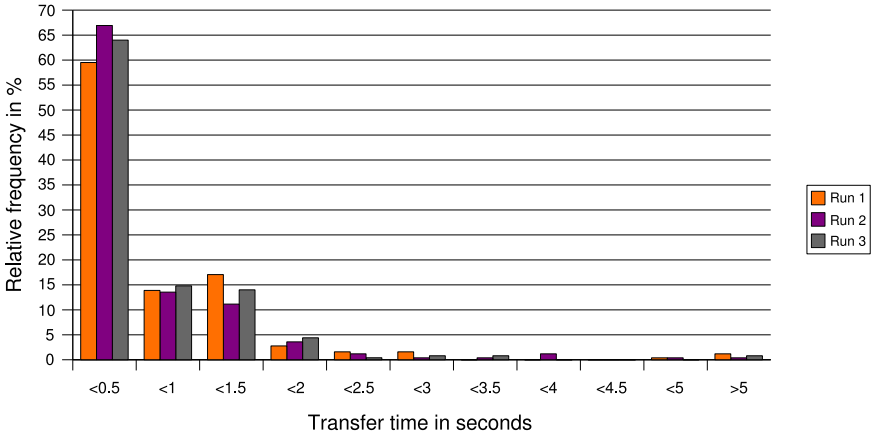

Figure 8: Relative Frequencies of Transfer Time of `Introduce_Ack`

Table 7: Distribution of Transfer Time of `Introduce_Ack` (s)

	Run 1	Run 2	Run 3
Median	0.267	0.231	0.274
Upper Quartile	1.001	0.861	0.899
Quartile Difference	0.839	0.744	0.747
Ninth Decile	1.317	1.293	1.301
Maximum	6.901	7.631	7.648

4.7 Contacting Hidden Service

The introduction point copies the information of the `Introduce1` cell in an `Introduce2` cell and sends it to the hidden service. The circuit used for this message was established at the very beginning, when the hidden services was started. The data shown in figure 9 and table 8 is the time between the introduction point sending the introduction request and the hidden service receiving the cell.

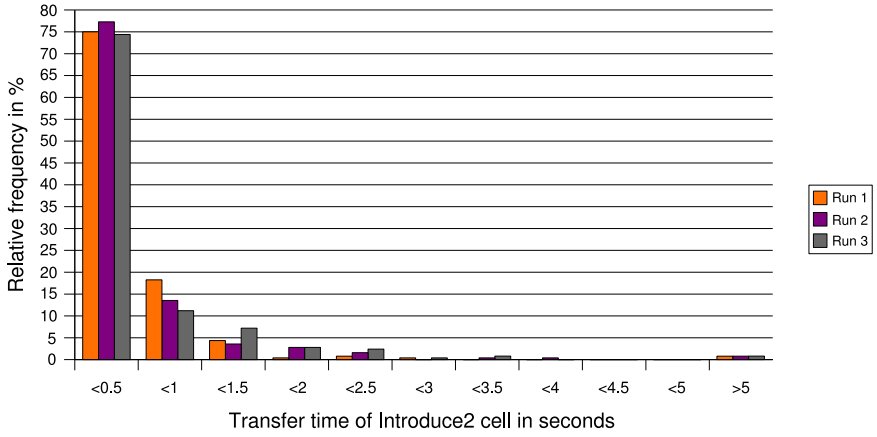

Figure 9: Relative Frequencies of Transfer Time of `Introduce2` Cell

Table 8: Distribution of Transfer Time of `Introduce2` cell (s)

	Run 1	Run 2	Run 3
Median	0.144	0.144	0.146
Upper Quartile	0.498	0.416	0.509
Quartile Difference	0.355	0.273	0.365
Ninth Decile	0.866	0.981	1.119
Maximum	14.619	21.505	27.472

4.8 Building Rendezvous Circuit

The hidden service needs to extend a circuit to contact the rendezvous point, because it received the necessary information about the RPO of this specific request in the `Introduce2` cell. The measurement took place between accepting the `Introduce2` cell and the log notice that the circuit has been opened. The results can be seen in figure 10 and table 9.

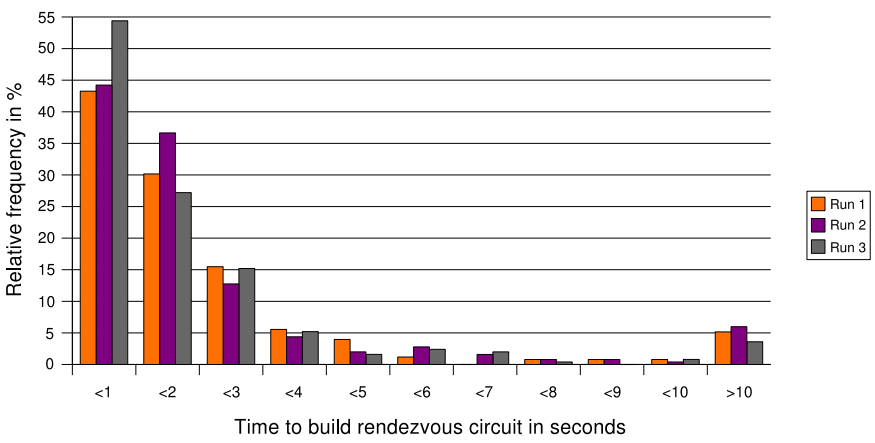

Figure 10: Relative Frequencies of Duration to Build Rendezvous Circuit

Table 9: Distribution of Duration to Build Rendezvous Circuit (s)

	Run 1	Run 2	Run 3
Median	1.339	1.213	1.011
Upper Quartile	2.244	2.115	2.027
Quartile Difference	1.643	1.531	1.527
Ninth Decile	4.266	5.466	3.735
Maximum	27.784	48.386	54.580

4.9 Rendezvous1 and Rendezvous2 cells

Using the newly created circuit the hidden service sends a `Rendezvous1` cell to the rendezvous point. The rendezvous point subsequently sends a `Rendezvous2` cell to the client. Because we had no control over the rendezvous point, these two cell could be measured in combination only. Therefore the measurement took place between the hidden service sending the `Rendezvous1` cell and the client receiving the `Rendezvous2` cell.The results can be seen in figure 11 and table 10.

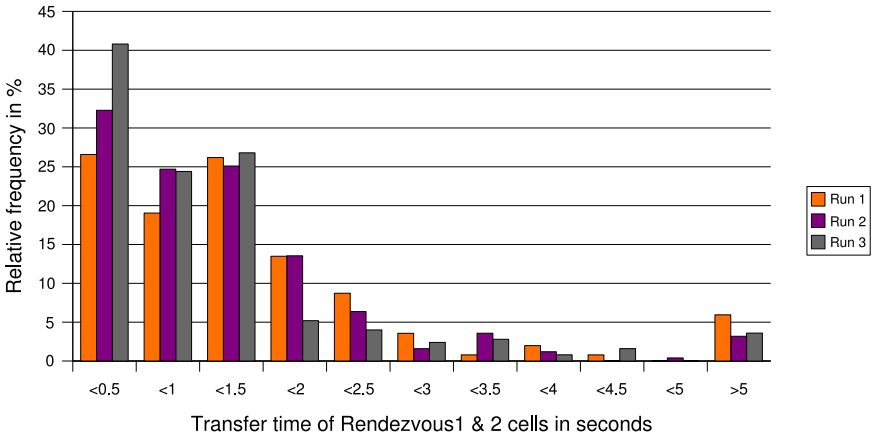

Figure 11: Relative Frequencies of Transfer Times of `Rendezvous1` and `Rendezvous2` cells

Table 10: Distribution of Transfer Time of of `Rendezvous1` and `Rendezvous2` cells (s)

	Run 1	Run 2	Run 3
Median	1.132	0.967	0.895
Upper Quartile	1.767	1.618	1.322
Quartile Difference	1.260	1.181	0.947
Ninth Decile	2.875	2.372	2.440
Maximum	27.918	20.369	56.531

4.10 Data Round-trip Time

The round-trip time of the HTTP request and reply could be measured by using the log statements about retrieving the `Rendezvous2` cell and retrieving the shutdown signal only. The results are shown in figure 12 and table 11.

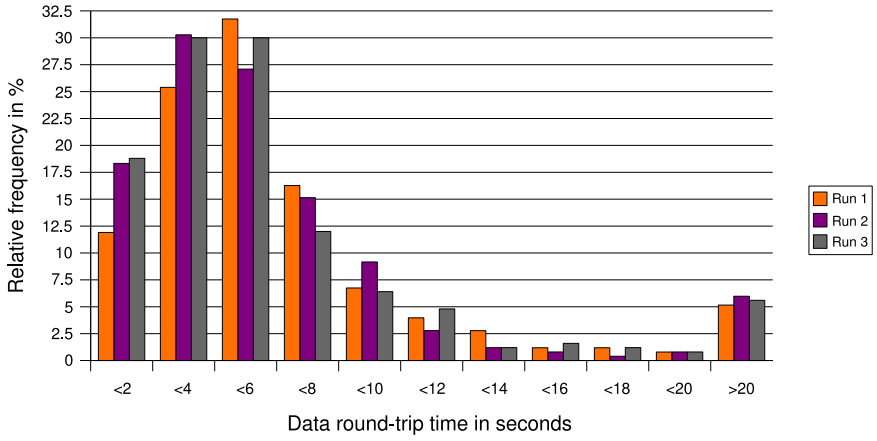

Figure 12: Relative Frequencies of Data Round-trip Time

Table 11: Distribution of Data Round-trip Time (s)

	Run 1	Run 2	Run 3
Median	5.041	4.086	4.292
Upper Quartile	7.139	6.878	6.899
Quartile Difference	4.332	4.520	4.598
Ninth Decile	12.066	10.075	11.058
Maximum	73.593	81.435	99.653

5 Interpretation

5.1 Time not Affecting Performance

Analyzing the three complete round-trip time graphs, patterns cannot be identified. Extremely high values seem to occur randomly over the whole 24 hour period. Attempts also failed randomly in all three runs.

5.2 Circuit Building

During accessing a hidden service new circuits are built twice. One is built by the client to contact the introduction point and one by the hidden service to contact the rendezvous point. In both cases the information necessary to build the circuit is received right before. The statistical values for both circuits are pretty similar in all runs.

The third situation, when another node is contacted initially, is not considered here, because the client only cannibalizes an existing circuit to create a stream to the rendezvous point.

In five per cent of all successful attempts per 24 hour run new rendezvous and introduction circuits were built by the client after both circuits were open already, leading to values greater than 60 seconds.

5.3 Cell Transfer

The transfer of cells using established circuits is a lot faster than establishing the circuit. Nonetheless there is a variation, and higher values occur.

Interesting is a comparison of the `Introduce1` cell and the `Introduce_Ack` cell, because they are using the same stream, but in different directions right after each other. The statistical values for the `Introduce1` cell are pretty similar to the values of the `Introduce_Ack` cell. To determine if the two values correlate, figure 13 shows all single attempts of run three, except for some outliers. We can see a slight correlation for low values, but for values greater than 0.5 s the relation seems random.

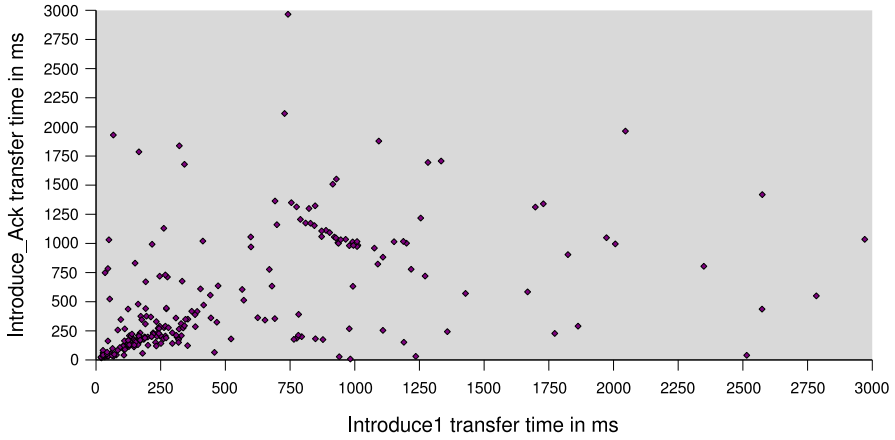

Figure 13: Transfer Times of `Introduce1` and Corresponding `Introduce_Ack` Cells

The values for the third single cell measured, the `Introduce2` cell, are a little lower than the two mentioned before. The two cells to establish a rendezvous point could be measured in combination only, as well as the `Rendezvous1` and `Rendezvous2` cells.

5.4 Adding the Steps

In a last step we added the single steps of an attempt up and compared them with the complete round-trip time. Figure 14 shows all measured values with solid lines and their dependencies. It is obvious that the transfer time of the `Introduce_Ack` cell is not critical, because the parallel steps need much more time until the stream is ready for data.

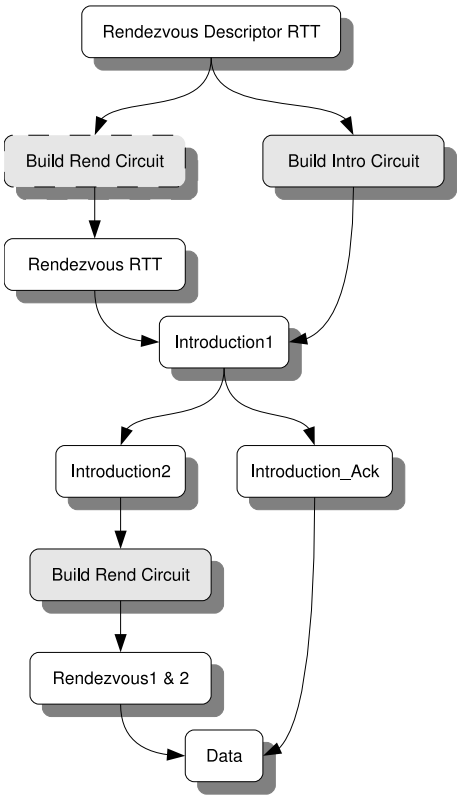

Figure 14: Hidden Service Flow Chart

But the other parallel steps are more interesting. Both building the introduction circuit and the round-trip time of the `Establish_Rendezvous` and `Rendezvous_Established` cells are critical in some cases.

As mentioned before we added all critical steps and compared them with the complete round trip time. For attempts when building the introduction circuit took longer than the two rendezvous cells, and therefore was critical, the median of the difference between all critical steps and the complete round-trip time was about 500 ms. For attempts, when the two rendezvous cells took longer than building the circuit, the median was only 8 ms for all three runs. Upper quartile and ninth decile were about 750 ms and 925 ms in the former case and 20 ms and 23 ms in the latter.

The significant difference can be explained because the arrival of the `Rendezvous_Established` cell is observed by an event listener, while the open introduction circuit is observed by a one second busy waiting loop. If we base the second case on events also about 0.5 s on average can be saved.

6 Future Work

It is necessary to find out what causes the high variability of opening circuits.

A bugfix of the *RendNodes* config option, that makes it work if cannibalizing, can add data collected at the rendezvous node.

Another interesting question left is why the cell transfer over existing circuits sometimes takes so long. It may depend on the nodes chosen for the circuit. And why is there such a difference beween the `Introduce1` and `Introduce_Ack` cells, which are using the same circuit right after each other?

A Appendix

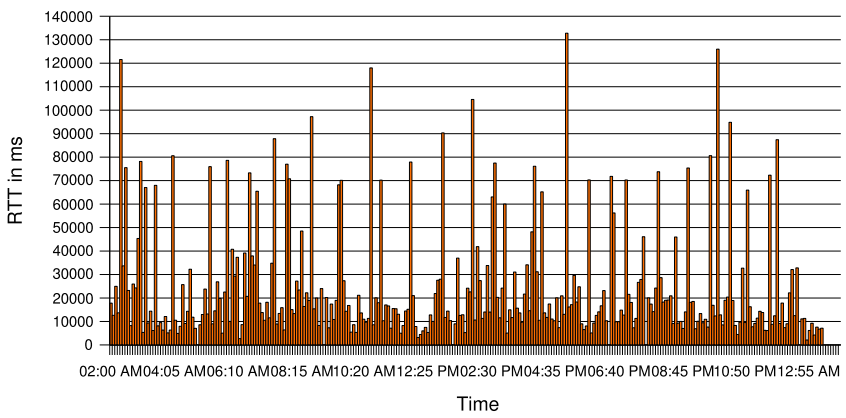

Figure 15: Complete Round-trip Time for all Run 2 Attempts

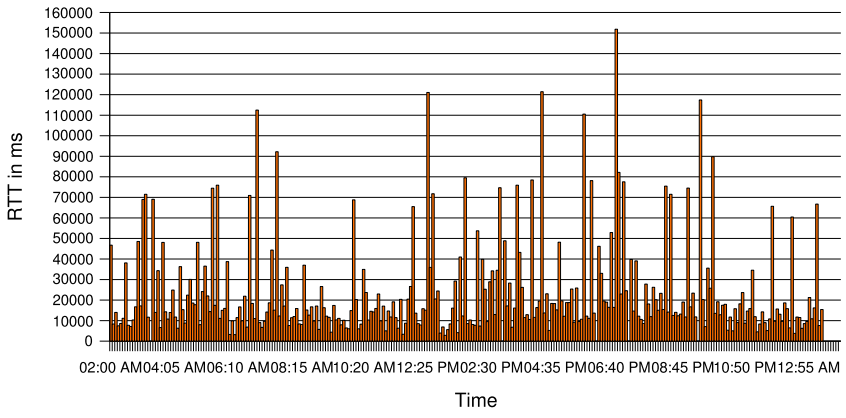

Figure 16: Complete Round-trip Time for all Run 3 Attempts

References

[1] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," 2004.

[2] M. Proest, "Teilerfolg - virtuelle server anonym getestet," *iX*, vol. 9, 2007.