Examining the Hydra: Simultaneously Shared Links in Tor and the Effects on its Performance

Sebastian Pahl
Institute of Information Systems (iisys)
Hof University of Applied Sciences
Hof, Germany
sebastian.pahl@hof-university.de

Florian Adamsky
Institute of Information Systems (iisys)
Hof University of Applied Sciences
Hof, Germany
florian.adamsky@hof-university.de

Daniel Kaiser
Status Research & Development GmbH
Zug, Switzerland
dkaiser.lu@protonmail.me

Thomas Engel
University of Luxembourg
Esch-sur-Alzette, Luxembourg
thomas.engel@uni.lu

ABSTRACT

Tor is a popular privacy-enhancing technology that allows anonymous communication using onion routing. However, such technologies are only helpful if used; therefore, performance is an important aspect. One of the main performance bottlenecks of Tor is the cross-circuit interference (CCI) problem. Tor multiplexes multiple circuits over a single Transport Layer Security (TLS) 1.2 connection if they share a path segment (link). Therefore, they have the same congestion window, which can yield unfair bandwidth allocation. However, there has been little work in understanding this problem in more depth.

This paper investigates the number of simultaneously shared links in the current Tor network, which are the root cause of CCI. We developed a novel shared links simulator called SALSA to investigate this problem. Our results show that 3.7 % of active links are shared, and the involving Onion Routers (ORs) have the most common bandwidth capabilities. Additionally, we show that the internal circuits and exit policy influence the CCI problem. Furthermore, we model the number of shared links when the demand grows further and show that the number of shared links can go up to 16 %. Finally, we run Shadow simulations with a 25 % downscaled Tor network and show that a network without shared links is faster.

KEYWORDS

tor, anonymisation networks, cross-circuit interference

1 INTRODUCTION

Anonymisation is fundamental for the right to privacy. Anonymisation networks allow anonymous communication via the Internet. These networks encrypt network traffic multiple times and send it through a network via intermediate relays. Due to this mechanism, these networks are slower than a direct connection. Tor is the most prominent anonymisation network, and according to the NSA/GCHQ leaks from Edward Snowden, *"still the king of high secure, low latency Internet Anonymity"* [1]. It allows users to communicate anonymously by hiding their identities using onion routing. A client chooses a path consisting of three Tor relays, called a *circuit*, and encrypts the traffic for each relay, beginning with the last one. This guarantees that every relay only knows the predecessor and the successor, not the whole circuit. Since its first release in 2002 [2], Tor's size has grown drastically.

The Tor Metrics project [3] estimates around 2 000 000 daily active clients [3], whereas a measurement study from 2018 [4] estimates around 8 000 000 daily active clients. Jansen, Traudt, and Hopper [5] measures the number of *simultaneously* active users up to 792 000 within 10 min. The popularity of Tor is growing, particularly with increasing Internet surveillance and censorship.

The survey from Alsabah and Goldberg [6] summarised Tor's main performance problems and identified cross-circuit interference (CCI) as the main problem. Tor multiplexes several circuits, which correspond to separate communication channels, over a single Transport Layer Security (TLS)-secured TCP connection. These multiplexed circuits share the same congestion window leading to unfair bandwidth allocation. Without Tor, each connection has a separate congestion window and consequently reacts independently loosening the interference between circuits. This is mainly a problem when bulk traffic circuits (e. g. a download) are multiplexed with an interactive session where responsiveness is essential. If bulk traffic causes loss events and congestion control is triggered, packets from the interactive session must also wait. This is a well-known problem called TCP head-of-line blocking (HolB) and can happen when there are simultaneously shared links.

Much research has been conducted to reduce the effect of CCI. One proposal for this problem is to replace the transport protocol [7–10] with a UDP-based protocol or use a separate TCP connection for each circuit. This is a promising direction, but there are still open problems that need to be solved, i. e. lack of hop-by-hop reliability. Jansen, Geddes, Wacek, Sherr, and Syverson [11] and Jansen, Traudt, Geddes, Wacek, Sherr, and Syverson [12] took a different approach and investigated congestion in the kernel socket buffers. They proposed Kernel Informed Socket Transport (KIST), which optimises the amount of data that is written to each socket. KIST has been merged into Tor version 0.3.2.1-alpha [13] and can be seen as an application layer TCP HolB mitigation. Another proposal is to directly integrate congestion control and avoidance mechanism into Tor [14, 15]. Recently, Tor has integrated end-to-end congestion

Proceedings on Privacy Enhancing Technologies 2023(3), 268–285
© 2023 Copyright held by the owner/author(s).
https://doi.org/10.56553/popets-2023-0081

control [16] in version 0.4.7.7. It is unclear what the implications are for the CCI problem.

There is a gap in the scientific literature regarding the number of simultaneously shared links in Tor. Merely, Jansen, Geddes, Wacek, Sherr, and Syverson [11] investigated this issue and estimated 99.775 % of unshared paths in the Tor network. However, this study is from 2014 and in the meantime, the Tor protocol and infrastructure have changed. However, no study focuses entirely on measuring and estimating simultaneously shared links in Tor to the best of our knowledge. Additionally, there has been little work in understanding the performance when Tor's demand grows further. For those reasons, we investigate the probability of shared links in the Tor network and their influence on its performance.

Research Questions (RQs). We conducted our study with the following RQs in mind:

- RQ1 How many simultaneously active shared links[1] exist currently in the network, and what influences this number?
- RQ2 How many simultaneously active shared links[1] exist when the demand grows further?
- RQ3 How does the CCI influence the current network performance in the face of simultaneously shared links?

Contributions. In addition to addressing our RQs, our work makes the following contributions. We

(1) developed a novel simulator called ShAred Links SimulAtor (SALSA) to investigate CCI and compare it with other simulators;
(2) conduct different simulations to find out how many simultaneously shared links per second currently exist in the Tor network and compare them;
(3) run experiments with Shadow in which we simulate a network of 25 % and compare it with the simulations;
(4) use our simulations and experiments to predict the number of simultaneously shared links when the demand is further growing;
(5) run experiments with Shadow in a 25 % network and compare Vanilla Tor with Per-Circuit TCP-over-TLS (PCTLS), including a model with packet loss, showing that this is indeed a promising direction;
(6) run experiments in a local private Tor network with different traffic types and measure their impact on Tor's performance.

Organisation. The rest of the paper is organised as follows. In Section 2, we described Tor, the CCI problem and defined the terminology we use throughout the paper. Next, we describe our experimental setup and our simulations in Section 3. Section 4 evaluates our RQs; verifies and compares SALSA; and presents our results. In Section 5, we discuss the limitations of our study, and in Section 6, we discuss related work. Finally, in Section 7, we conclude and discuss future work.

2 BACKGROUND

This section provides an overview of Tor and its CCI problem.

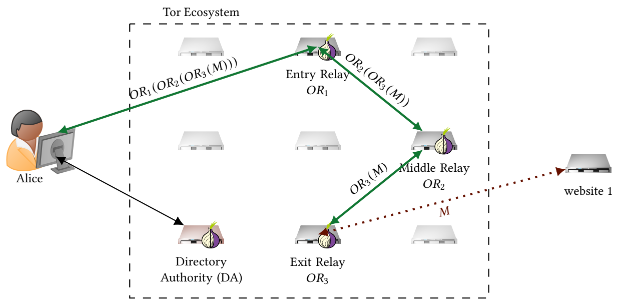


Figure 1: Example of how Tor works. Alice's Tor client first contacts the Directory Authority (DA) to get a list of Onion Routers (ORs). Then it encrypts her message M three times in reverse order and sends it to the first Tor relay, also referred to as the Tor entry relay. The straight green line contains traffic that Tor encrypts, and the dotted red line is traffic that might or might not be encrypted.

2.1 Tor

In the example in Figure 1, Alice uses the Tor client, also referred to as Onion Proxy (OP), and sends a message M anonymously to a website. First, Alice downloads a list of OR descriptors from the DA, which contains each OR's IP address, port number, public key, and bandwidth capability. Second, Alice selects n ORs according to Tor's path selection algorithm. This algorithm chooses each OR with a probability proportional to its bandwidth capabilities. Meaning ORs with higher bandwidth capabilities are chosen more often. The path through the selected ORs is called a *circuit*. In a typical Tor configuration, $n = 3$. Alice builds the circuit telescopically by initiating a link to the first OR and then through the first one, a link to the second OR and so on. This is done by sending an `EXTEND` message to the first OR to build a link to the second OR and then an `EXTEND` message to the second OR to establish a link with the last OR. If there is no link between ORs, then a new link is initiated; otherwise, a previous link is used. For performance reasons, Alice establishes multiple circuits in advance. TLS 1.2 is used as the communication protocol between ORs. This ensures reliability and in-order delivery and mitigates OS fingerprinting attacks since the TCP stack from the OP is not used. Third, after a circuit is created, Alice encrypts the message M in reverse order. This means Alice encrypts the message M for OR_3 and adds the address of *website 1*. This message M is then encrypted for onion router OR_2, followed by adding the address of onion router OR_3. Alice repeats this process until adding the layer for onion router OR_1 and then sends this layered encrypted message to OR_1.

2.2 Cross-circuit Interference (CCI) Problem

The underlying problem of CCI is the HolB problem. This is a general problem and can happen whenever there is a single queue of data and multiple receiver queues. If one packet (the head of the line) is lost, all following packets must wait. Figure 2 shows this problem for Tor.

[1] Measure resolution is one second

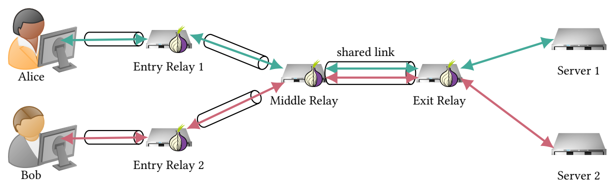


Figure 2: OR can multiplex multiple circuits over a common TLS connection. Green arrows represent an interactive protocol (e. g., SSH, Chat) and red arrows represent a bulk download. The tube represents the TLS 1.2 connection with a single congestion window.

Tor's cryptographic protocols require reliability and in-order delivery; otherwise, cells cannot be decrypted correctly. For this reason, multiple circuits are multiplexed over a single TLS connection. However, in-order delivery of cells is only necessary within a single circuit and not for all circuits over a single connection. When traffic from different circuits are multiplexed over one TLS connection, the problem might occur.

Most modern server and desktop Operating Systems (OSs) use congestion control algorithms that use packet loss to detect congestion and optimise network flow, e. g., TCP CUBIC [17] and TCP NEW RENO [18]. TCP considers a packet *lost* when there is no acknowledgement after the retransmission timeout or after receiving three duplicated acknowledgements. Depending on the connection phase, packet loss has different implications for the connection. In the slow start phase, a loss event halves the window size and starts the congestion avoidance phase. In the congestion avoidance phase, a loss event reduces the window size by a factor that is defined by the congestion control algorithm. TCP CUBIC[2] reduces its window size by a factor of β, set to 0.2 in the original research paper [17]. TCP NEW RENO also halves the window size and enters the fast recovery phase. That means if a loss event occurs on one of the multiplexed connections, TCP's congestion control is triggered, and the other connection with no packet loss slows down or needs to wait. This problem happens only if a circuit shares a link.

2.3 Terminology

This section defines the terminology we use throughout this paper since there is no consistent nomenclature.

Guard relay: An OR that has a set `guard` flag.
Middle relay: An OR with neither a set `guard` nor an `exit` flag.
Circuit Part: A circuit with the following ORs, $A \rightleftarrows B \rightleftarrows C$, consisting of two parts, either $A \rightleftarrows B$ or $B \rightleftarrows C$.
Link: A TLS 1.2 connection between two ORs in which one or multiple circuits can be multiplexed.
Shared Link: A link in which two or more circuits are multiplexed.
Active Circuit (Part): A circuit (part) where one or more Tor cells are transferred in a specific time frame.
Active Link: A link with one or more active circuit parts.
Directed/Undirected: For directed links, the order of the relays matter; for undirected links, it does not.
External/Internal: External circuits target the Internet, and internal circuits target mainly Onion services.

[2]CUBIC is the default congestion control in MacOS, Linux since Kernel 2.6.19, in later Microsoft Windows 10 builds, and Windows Server 2019.

3 EXPERIMENTAL SETUPS

This section describes the experimental setups that we have used in our study.

3.1 Simulators

In this section, we describe the known simulators and our novel simulator SALSA.

3.1.1 Shadow. It is a discrete-event simulator [19] that runs a private Tor network on a single machine. We use Shadow in version 1.13.2 (Nov 2019), the same version used in [20]. To estimate how many active simultaneously shared links per second exits, we will conduct a Shadow simulation with a modified Tor client based on tag `tor-0.3.5.18` (Jan 2022). This modified version logs every second all active circuit parts. Relay cells include but are not limited to circuit `EXTEND` and `SENDME` messages. Unfortunately, Shadow simulations with a higher network scale take multiple days.

3.1.2 Tor Path Simulator (TorPS). It is a path simulator for Tor [19]. It reimplements the path selection algorithm in Python. An initialisation step is necessary to run simulations. For this step, we need consensus and server descriptor documents from CollecTor [21]. Afterwards, simulations can be run repeatedly. Simulations with TorPS run faster than Shadow simulations. We used its simple user model to generate circuits. The simple user model only makes circuits to google.com on port 80, representing web traffic. TorPS only outputs IP addresses for the relays in a circuit, but one IP address can have multiple Tor clients. In our other setups, we used only relay fingerprints. A relay fingerprint always implies a unique combination of IP address and port. Therefore, we modify TorPS to output fingerprints.

3.1.3 Modified Tor Client. We modified the Tor client to generate circuits offline from any consensus document to verify our simulation. We used the consensus document archive from CollecTor [21] to extract the matching documents for a timestamp. The following three documents are necessary: `cached-certs`, `cached-microdesc-consensus`, and `cached-microdescs`. These documents can be extracted from the CollecTor archives. The Tor process needs to change the clock to load these consensus documents. In Linux, we use `faketime` [22], a command-line program that fakes dates and times without modifying the system-wide time.

Any side effects on the live network are undesirable, so we disable Internet access and circuit establishment and only log the generated circuits. Generating many circuits is very slow, even though Tor is written in C.

However, there is a problem while generating circuits from one Tor client. Tor clients sample up to 60 guard relays and stick with them. They use these guards for two months, or until they become unavailable, so generated circuits will only use a small subset of all guard relays. To solve this issue, we disable this specific code path, so guards are always chosen randomly by weight. One caveat, we can only simulate one circuit per client, but this is enough for our purposes.

3.1.4 Shared Link Simulator (SALSA). We developed SALSA, a simplified model of the Tor Path Selection Algorithm. It is not designed to answer questions of a particular relay but to model the overall shared link distribution of the Tor Network. It takes a consensus document and the number of active circuits as input; and; outputs the number of shared links in percentage.

We only implement the necessary parts of the Tor Path Selection Algorithm. This algorithm relies primarily on two principles:

(1) Relays are chosen at random weighted by their bandwidth and;
(2) the bandwidth is weighted by the position in the circuit depending on relay flags.

The consensus chooses the position-dependent weights. Besides these weights, the Tor Path Selection Algorithm does different filtering steps. The two most important steps are:

(1) The consensus allows middle relays to be chosen in the guard position, but only relays with guard flag are chosen for this position.
(2) The consensus allows all relays to be chosen for the exit position, but only relays with a suitable exit policy are chosen.

Our simulator implements these steps. We verify our simulator in Section 4.4. Now, we can take a consensus document and some active circuits and calculate the probability for each relay in each position. If we choose one relay for every position, we have a circuit. We have included an annotated pseudocode of SALSA in Algorithm 1 in the Appendix, and we published our source code on Gitlab [3].

3.2 Environments

In this section, we describe the hardware and software environments in which our experiments and simulations run.

3.2.1 High-Performance Environment. As described in the previous section, Shadow has high hardware requirements for memory and processing power. For this reason, we used a 4 x 16-Core AMD Opteron 6380 with 900 GB of RAM. On this system, we run Debian 11 using Linux Kernel 5.10.106-1. This hardware restricted us from simulating a 25 % Tor network.

3.2.2 Private Tor Setup. Shadow simulations are essential to see network-wide effects of Tor features. However, we built two private Tor setups with network namespaces to isolate parameters and see effects in specific scenarios. For this reason, we used a desktop computer with a 8-core AMD Ryzen 7 3700X with 64 GB of RAM. We run Manjaro with the Linux Kernel 6.0.11-1-MANJARO. Both testbeds systems are build with *mininet* [23, 24] 2.3.0-1.

The first testbed is depicted in Figure 3. For this testbed system, we oriented ourselves on Reardon and Goldberg [9] for a better comparison. It contains eight clients that download bulk data from the server. Two clients are connected to each entry relay. These entry relays are then connected to one middle relay, which connects to the exit relay. The connection between the middle and exit relay is a shared link.

[3]https://gitlab.com/spahl/hydra-popets2023

However, eight clients behind a shared link is an extreme example, as seen in our measurements in Figure 7. For this reason, we build a testbed that only contains two clients, similar to Figure 2.

4 EVALUATION

This section evaluates our RQs with the experimental setups described in the previous section. For RQ1 and RQ2, we will use SALSA, a modified Tor client, and TorPS. However, SALSA needs the number of active circuits per second, and we get this number from experiments with Shadow. Regarding RQ3, we will use Shadow experiments to investigate the performance influence.

4.1 Current Number of Simultaneously Shared Links (RQ1)

This section investigates RQ1, asking how many simultaneously active shared links exist currently in the Tor network and what influences this number beside the number of relays and users. Recall that a link is a TLS connection between two ORs. Only one link exists between a pair of ORs in the current Tor implementation. Over this link, multiple circuits can be multiplexed and then become a *shared link*. We call it an *active circuit* when at least one relay cell is sent during a pre-defined time frame. Here we deviate from Jansen, Traudt, and Hopper [5] because they define an active circuit if more than seven cells are sent during 10 min. That is because it takes seven cells to establish a circuit. We count these cells, too, because they contribute to the problem like any other cell. However, we choose a time frame of 1 s instead of 10 min to have an more accurate snapshot. In theory, simultaneously active shared links mean that we have that many active shared links at any time. This requires a snapshot of the whole network. However, a perfect snapshot is impossible because we always have a discrete-time unit.

The necessary steps to calculate the percentage of shared links are as follows: Foremost, we count the number of circuits and put the first and second links in a list. Every unique member of that list counts as a link. Every unique member of that list that appears two or more times counts as a *shared link*. Then, we divide the number of shared links by the number of links and get the shared link percentage. For directed links, the order of the relays matter; for undirected links, it does not. We consider both and have a detailed explanation in Section 4.1.4.

4.1.1 Measure Active Circuits per Second. To address RQ1, we need to know how many active circuits exist per second in the network. This value is also needed to calculate the number of shared links for every consensus document. We use our Shadow setup described in Section 3.1.1 with a 25 % Tor network to measure the number of active circuits per second. Our Shadow setup uses *tornettools* [25] from [20] which scales the total number of relays and the number of active/inactive circuits linearly with the network scale. Active circuits per second depend on these parameters and will also scale linearly. This allows us to extrapolate to a 100 % Tor network with SALSA. Our Shadow simulation is based on consensus data from November 2021, and it runs for 60 simulation min. Within 10 min, Shadow will simulate $\approx$ 198 k active users and $\approx$ 374 k active circuits. We log all active circuit parts and their fingerprints every second.

Based on this log data, we can calculate the number of active links, active circuits, and the percentage of shared active links. If we

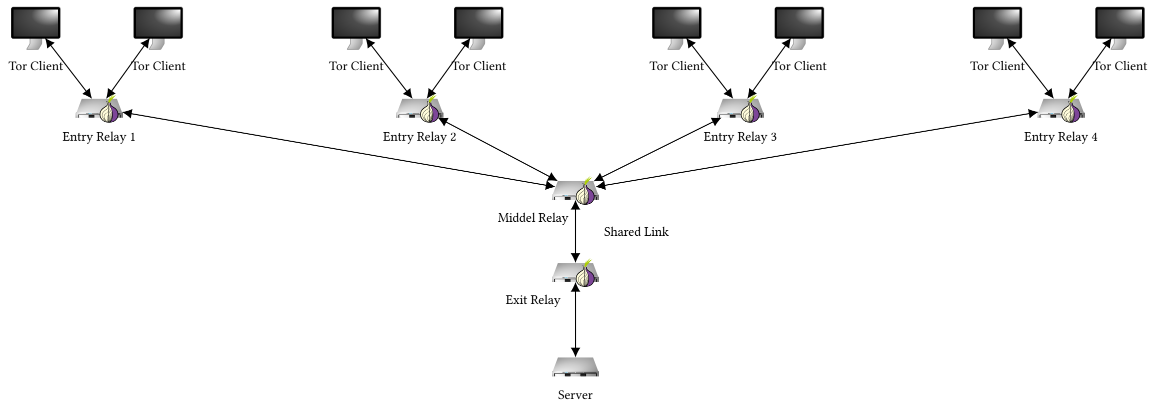


Figure 3: This experimental setup is similar to [9]. It contains eight clients that download bulk data. Always two clients are connected on one entry relay. Each entry relay is connected to a single middle relay, which is again connected to a single exit relay. The link between the middle relay and the exit relay is a share link.

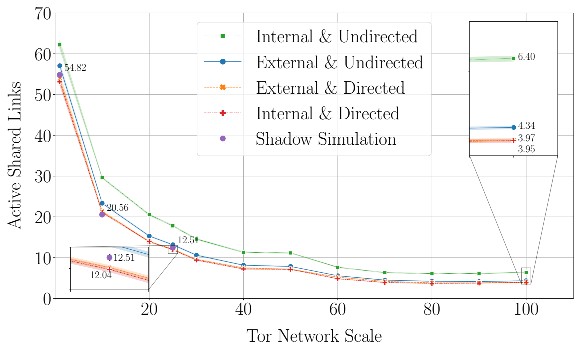


Figure 4: Shows the active shared link in percentage depending on the Tor network scale. All lines are results from SALSA based on consensus documents, one for every day, in November 2021. The orange and blue lines show general purpose exit circuits. The red and green lines show internal circuits. For each network scale, twenty consensus documents are sampled via tornettools. Shadow measurements for 1 %, 10 % and 25 % are shown as points.

group by relay and timestamp, the number of unique fingerprints is the number of active links to the next relay. We only see one part of a circuit because we measure on ORs. For this reason, we divide the number of all circuit parts by two to get the overall number of active circuits. If one link in a circuit is active, the other link will likely be active within one second. Our results: A 25 % Tor network has, on average, 24 219 active circuits per second with a standard deviation of 155. This value comes from ten different 25 % Shadow simulations with vanilla Tor without packet loss.

4.1.2 Active Shared Links in the Tor Network. Figure 4 depicts the number of active shared links in the Tor network run with SALSA verified selectively with Shadow. The number of active shared links in Figure 4 with Shadow for a 1 %, 10 % and 25 % network is 54.82 %, 20.56 % and 12.51 %, respectively. We repeated the 25 % Shadow simulations ten times, and on average, the network has 12.5 % shared links with a standard deviation of 0.2 %, close to 12.0 % from SALSA. Figure 4 shows these values in the bottom left. We repeated the 1 % and 10 % Shadow simulations once since they only serve us to check SALSA validity.

These results are comparable to the *External & Directed* by SALSA (orange line). We have sampled five consensus documents from November 2021 using *tornettools* for each network scale. This tool multiplies the number of relays and users from a 100 % network times the network scale. The confidence intervals per line are tiny due to how tornettools samples. The bandwidth distribution for every sample is very similar, leading to a similar number of shared links. The only variation comes from the simulation itself.

4.1.3 Influence of Exit Policies on Shared Links. Besides web traffic, file-sharing protocols, such as BitTorrent, are often used over Tor [26]. Relay operators do not want to deal with abuse issues from file-sharing; for this reason, Tor has implemented exit policies. An exit policy filters outgoing connections according to specific rules, which gets propagated to the Tor client. The Tor client then avoids choosing exit relays that do not allow the intended network traffic. As a result, other network traffic can have more simultaneously shared links since the number of exit relays that allows that type of traffic is lower.

In 2021, all exit relays allowed HTTP/S traffic with very few exceptions. The modified Tor client and TorPS can consider exit policies, but we use them only for HTTP/S traffic. Our shadow setup does not consider exit policies. Other protocols are at least included in the average number of active circuits within 10 min, reflecting the live network.

However, with our simulation, we can filter exit relays with specific exit policies to investigate the influence of these policies on the number of shared links. We will use the same traffic classification from [26], which we have added for completion in Table 1. Figure 5 compares the different file-sharing classes depending on the number of shared links. All classes have the same number of active circuits as a 100 % Tor network. This allows us to compare these classes as if the whole network would only serve one of those classes exclusively.

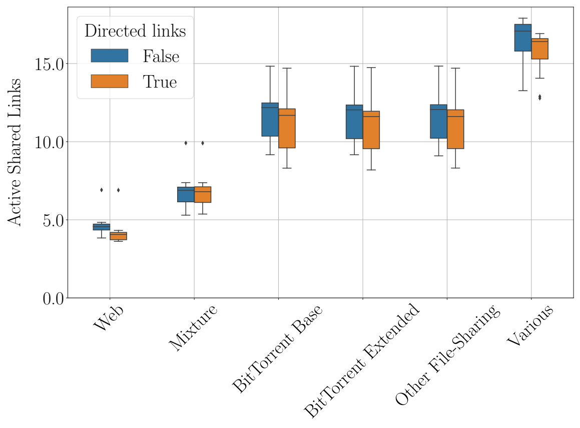


Figure 5: Shows a comparison of different file-sharing classes based on [26] versus regular web traffic and the influence on shared links per second. We compare here only extreme cases. Each box plot is based on consensus documents from November 2021 with a 100 % Tor network. File-sharing exit policies can have a significant influence on shared links.

The results show that the number of shared links for file-sharing protocols are higher compared to regular web traffic due to fewer exit relays. We compare web traffic, file-sharing and a mixture class. The mixture class mixes the other classes with measured ratios from the live network. We describe this in more detail in Section 4.1.7. Fewer and fewer exit relays are available for the different classes. On average, for Web or Mixture is 1348, for Various is 45, and for the remaining classes are around 200 exit relays available. This leads to a growing number of shared links between middle and exit relays. We have an average value of 4.3 % for the Web, 6.7 % for Mixture, 11.3 % for BitTorrent Base, 11.2 % for BitTorrent Extended, 11.2 % for other file-sharing protocols, and 16.2 % for various.

Key Insight: The exit policy influences the number of shared links and increases the CCI problem, particularly for ORs that allow file-sharing protocols.

4.1.4 Influence of Link Direction on Shared Links. We consider directed and undirected links. Suppose we have a link between OR_1 and OR_2 and two circuits. The first circuit uses a path from OR_1 to OR_2 and the other vice versa. An undirected link would be shared since both circuits use the TCP connection between the two ORs. A directed link would not consider a shared link because the circuits have different directions. Each OR has its congestion window, so in theory, they do not affect each other. However, congestion in one direction could affect the other. Since we have two-way communication, a delayed or lost packet (both are feedback signals for congestion) in one direction impacts the congestion window from the other side. In two-way communication, TCP often piggybacks acknowledgements on data packets. Also, if a router in-between is congested, it might affect both sides, depending on the cause of congestion. A router running out of memory will affect both directions; if a send or receive queue is full, it may only affect one direction. For these reasons, we consider both directed and undirected links.

Figure 4 shows that if we view links as undirected, we have more shared links. This is mainly due to guard relays which can show up in the entry and middle positions, in the internal case, and also in the exit position. It is more likely that a middle relay is picked for this position if it has the same bandwidth weight as a guard, but guards have a higher bandwidth weight on average. So two guards will likely be picked for both positions. That is why we see more shared links. We included link direction in all our figures where it makes sense.

Traffic Class	TCP Ports
Normal Web Traffic	80, 443
BitTorrent Base	[6881–6880]
BitTorrent Extended	[6890–6999]
Other File-Sharing	1214,[4661–4666],[6346–6429],6699
Various	25,119,[135-139],445,563

Table 1: Distinguishing features of the traffic classes based on [26].

4.1.5 Influence of Internal Circuits on Shared Links. Recall that internal circuits stay within the Tor network; they do not connect to an external server. They are used for Onion services, measuring bandwidth, testing and directory connections. Both client and service connect to a Rendezvous Point (RP), which acts as a proxy to exchange data with an Onion service. The RP is chosen as a middle relay by the client who wants to connect to a service. A client builds a three-hop circuit, where the last relay is the RP. A service builds a three-hop circuit to the RP. The last relay before the RP is also chosen as a middle relay. There is more complexity involved with Onion services, but we only focus on established connections. We refer interested readers to [27] for a detailed description of the rendezvous specification. Both client and service now have a three-hop circuit that is joined on both ends.

The difference between internal and external circuits exit relays are chosen like middle relays. To integrate these circuits in our simulation, we build three-hop internal circuits, where the last relay is chosen as a middle relay. There are more middle than exit relays. Therefore, we could expect fewer shared links because there are more possible links overall. However, relays with set guard flags can also be chosen in the middle position, which leads can lead to more shared links between guard and middle OPs as well as middle and exit OPs. This can even increase the number of shared links; see the green line in Figure 4.

If we first focus on the directed case, both internal and external are on the same level. This is due to the consensus data from November 2021. Nevertheless, for the whole year, see Figure 8, internal is always above external. We will discuss Figure 8 later in Section 4.2. The difference between both cases is visible for the undirected case, which is also more critical. We have around a 1 % increase in shared links.

These figures show extreme cases where all circuits are either external or internal. The real network has a mixture of both cases between these extremes. For example, Jansen, Traudt, and Hopper

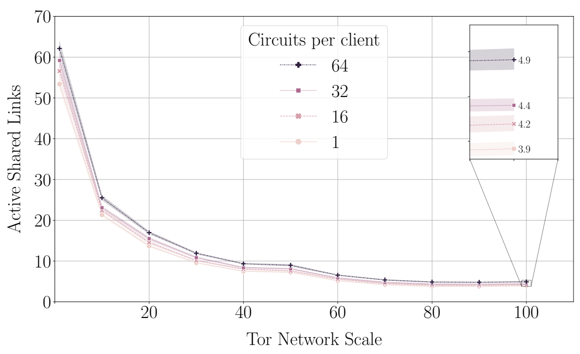


Figure 6: Shows the number of active shared links in percentage depending on the Tor network scale. Each line represents a client (OP) with 1, 16, 32 and 64 active circuits. We highlighted the values for 100 % to show the differences.

[5] found ten times fewer circuits on exit relays than guard relays, suggesting more internal than external circuits.

Key Insight: Internal circuits have more shared links than external circuits.

4.1.6 Influence of Guards Relays on Shared Links. Onion Proxies sample up to 60 guard relays and stick with them. They use these guards for two months or until they become unavailable. Thus, generated circuits for one OP will only use a small subset of all guard relays. Further, clients have a list of three primary guards chosen from all sampled guards. These primary guards are selected randomly. In the live network, 84 % of OPs use up to 5.0 active circuits [5] within 10 min. We did not measure how many active clients per second were active; we have only the number of active circuits per second. Therefore, we do not have the average active circuits per client, but they are most likely between 1.0 to 5.0. Remember that 84 % of OPs use up to 5.0 active circuits.

If OPs only switch between three guard relays, it increases the probability of a shared link for their circuits. If we keep the number of active circuits constant and increase the number of active circuits per client to 64, then the shared link percentage only increases from 4 % to 5 %. As shown in Figure 6, the number of shared links grows very slowly, even for more significant values. The total number of circuits is constant for every network scale. The circuits per client go up to 64, which does not reflect the current network.

We use only one guard per client in SALSA, so that this change would be even more minor in the live network. Furthermore, we expect the average number of active circuits per second and per client to be very close to 1.0, so this increase is neglectable.

Key Insight: Limiting the number of guard relays per client does not significantly influence the number of shared links.

4.1.7 Results. Our measurements show that the number of active shared links in a 100 % Tor network range from 3.7 % to 6.9 % in 2021. According to Tor's path selection algorithm, the involved ORs in these shared links have the most common bandwidth capabilities.

In November 2021, the number of active shared links ranges from 3.9 % to 6.4 %. Between guard and middle relays, we have 2.9 % to 3.7 %; between middle and exit relays we have 2.5 % to 5.0 % active shared links.

In the previous sections, we looked only at isolated influences to establish boundaries between shared links. To estimate how many shared links are in the live network, we take recent privacy-preserving measurements to get realistic ratios for our influences. Link direction can only be on or off, we include both cases. Guards have an influence on shared links, but we estimate it to be very close to 1.0 active circuits per second per client, even if it is bigger, it does not have much influence.

Jansen, Traudt, and Hopper [5] measured ten times more circuits on entries than on exits, which can be attributed, among other things, to internal Tor circuits. We can take this value to calculate how many internal against external circuits should be included in our simulation.

Exit policies influence shared links negatively. Jansen and Johnson [26] show that file-sharing protocols are the most common, second to HTTP/s, but only a few exit relays support those protocols. They also measure how many active *web* circuits and active *other* circuits exist in the current network. We will use this to split our external circuits into *other* and *web* circuits. There are 1.7 times more *other* than *web* circuits. The *other* circuits will be further split by traffic class.

If we consider link direction, guards, internal circuits and exit policies, we come up with around 3.70 % to 6.05 % shared links. The first value is directed, and the other value is undirected.

To get a better understanding of how shared links affect users. We need to infer the number of affected users from the number of actively shared circuit parts, but this is not obvious. The number of actively shared circuits is a good indicator of getting halfway there. We measured this number with ten SALSA simulations in a 100 % network. The results show there are 14 812 ± 138 (15 %) active shared circuits per second, this value is a sum of circuits with only one shared link 13 805 ± 125 (14 %) and circuits with two shared links 1006 ± 31 (1 %). This value is close to the number of 15 828 ± 166 active shared circuit parts.

The configuration based on directed links with only external circuits has a circuit parts over links distribution in Figure 7. The distribution follows an expected exponential decay. Mainly two circuits are multiplexed over one link, with 3.5 %, and the rest has 0.3 %.

4.2 Number of Simultaneously Shared Links with Growing Demand (RQ2)

This section evaluates our second RQ, asking how many simultaneously active shared links exist when the demand grows. There are numerous ideas to integrate Tor directly into web browsers. Brave [28], for example, has already integrated Tor in its private browsing mode. For Firefox, there are proposals either to have an extra *Super Private Browsing Mode* [29] or to replace the already integrated private browsing mode [30]. However, researchers [20, 31] are cautious since Firefox has more than 200 000 000 monthly active

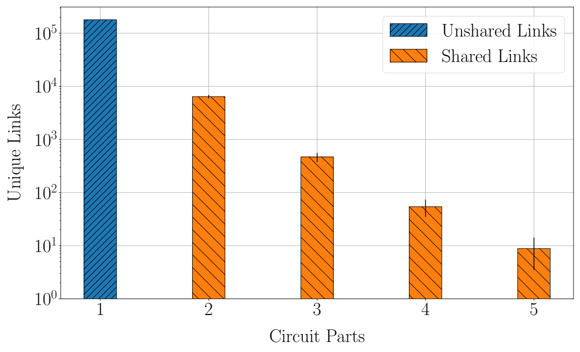


Figure 7: Shows the distribution of circuit parts for the current Tor network. The red rectangle in Figure 8 is only for external and directed circuits. The diagram shows how many unique links exist with how many multiplexed active circuit parts per second. If there is only one circuit per link, it is an unshared link (blue); otherwise a shared link (orange). Error bars show the standard deviation. Data is based on all consensus documents from 2021. For every consensus, we did one simulation in SALSA.

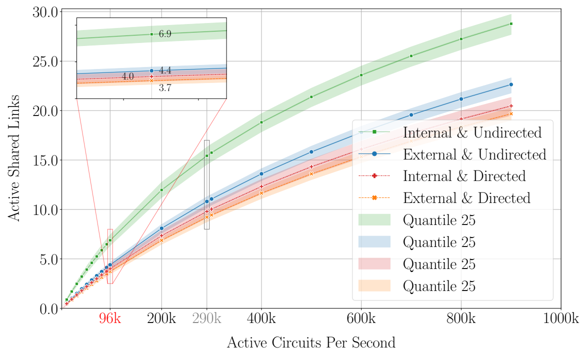


Figure 8: Forecast of the number of shared links depending on the number of active circuits per second. Data reflects the average of all consensus documents from 2021. The red rectangle marks the current Tor network. The grey rectangle is the maximum capacity of the current Tor network. The orange and blue lines are general-purpose exit circuits. The red and green lines are internal circuits.

users [32], and this might overload the capacity of the Tor infrastructure. For this reason, estimating the number of simultaneously shared links is crucial when the number of circuits grows.

Section 4.1 discussed the different influences on the number of shared links. However, the number of relays and users has the most significant influence. We focus, for now, on the growth of one or both of these values. If the number of relays and users grows linearly, we have the same situation as Figure 4; the number of shared links will decrease exponentially. On the other hand, if only the number of relays grows, the number of shared links will also decrease. In the following, we will focus only on the growth of the number of users and, therefore, the number of active circuits.

We look at the advertised and consumed bandwidth to estimate how many users the Tor network could handle. Currently, the network has 600 Gbit/s advertised and consumes 200 Gbit/s of this bandwidth [21]. Therefore, if we assume a linear relationship between simultaneously active users and consumed bandwidth, Tor can handle approximately three times the number of users at the moment. Even though the number of relays and users has been stable since 2015, the consumed bandwidth, on the other hand, keeps growing [21]. We used our simulator from Section 3.1.4 to create a forecast. Figure 8 depicts our forecast of the number of shared links when Tor would have a broader deployment.

The current network with 96 000 active circuits, marked with the red rectangle, has 3.7 % to 6.9 % active shared links. These numbers are different from the upper ones, referring to the year 2021 and not only to November 2021. The maximum capacity will have approximately 8 % to 16 % active shared links marked with the grey rectangle. Generally, internal circuits have more shared links than external circuits and undirected links are above directed links.

Key Insight: If the Tor infrastructure reaches its capacity with around 290 k active circuits per second, the number of shared links amounts to around 8 % to 16 %.

4.3 Influence of the CCI with Simultaneously Shared Links (RQ3)

This section evaluates our third RQ, asking how the CCI problem influences the current network performance in the face of simultaneously shared links. We divide network performance into circuit built time and file transfer time.

4.3.1 File Transfer Time. There is already supporting evidence [7–10] that CCI influences the performance of Tor. However, previous studies did not evaluate this RQ appropriately. For example, Reardon and Goldberg [9] used a small network, i. e. six relays and eight clients, to evaluate the CCI problem's impact. Other studies [7, 8, 10] measured only once, did not indicate confidence intervals, and still used a small network compared to the current network and simulation capabilities. Additionally, the Tor protocol has changed since it integrated KIST [13] as its new traffic scheduler. See Table 5 for a better comparison. For this reason, there is a need to investigate the performance impact of the CCI problem in a more extensive simulation.

AlSabah and Goldberg [7] published Per-Circuit TCP-over-IPsec (PCTCP), a proposal to mitigate the CCI problem. Their idea was to use a separate TCP connection for each circuit and wrap all these connections in an Internet Protocol Security (IPsec) tunnel. We will use a similar approach to use a TLS connection per circuit. We indicate this approach as PCTLS. Our simulations use a newer Tor version which already has integrated KIST [11, 12], an application-level mitigation for the CCI problem.

PCTLS and the other CCI mitigations have two advantages, (1) preventing TCP HolB and (2) fairer bandwidth allocations due to multiple congestion windows.

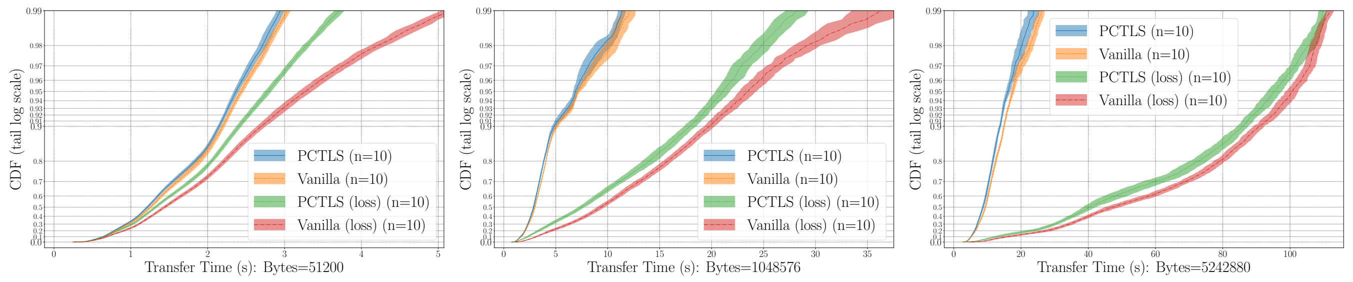


Figure 9: Time to last byte for a transfer of 50 kB, 1 MB and 5 MB. Comparison between Vanilla and PCTLS with Shadow in a 25 % network with the Shadow 201801 model [33] with and without packet loss. The experiments were repeated ten times per configuration, and the CDF is plotted with tail-logarithmic y-axes.

This paper shows that KIST, in conjunction with PCTLS, performs better than without PCTLS. Our contribution is a reevaluation of the impact of CCI with a recent Shadow simulation with a 25 % scale, considering the latest research methods from Jansen, Tracey, and Goldberg [20]. We compare Vanilla and PCTLS with the Shadow 201801 model [33] with and without packet loss based on Tor 0.3.5.18[4]. We measure every configuration ten times.

Each simulation has around 198 clients that measure performance when transferring 50 kB, 1 MB or 5 MB files. These transfers are not evenly distributed, instead 50 kB is 80 %, 1 MB is 13 %, and 5 MB is 7 % likely. There is a one-minute pause between each transfer. `Tornettools` preconfigure these clients. Figure 9 shows the time to complete a 50 kB, 1 MB and 5 MB transfer compared to Vanilla Tor.

The simulation in Figure 9 with PCTLS has no active shared links since it creates a separate TLS connection for each circuit. On the other hand, the simulation with Vanilla has 12.5 % active shared links on average.

Without packet loss for 50 kB, 1 MB and 5 MB, PCTLS is, on average, 39 ms, 150 ms and 703 ms faster. With packet loss, PCTLS is, on average, 172 ms, 1764 ms and 5696 ms faster. With ten measurements for both models, it confirms the impact of shared links.

Key Insight: Our Shadow simulations show that 12.5 % shared links results in an 8 % performance loss.

4.3.2 Circuit Built Time. Performance can be seen as how fast a file transfer is and how long it takes to create the circuit. Unfortunately, the latter is also time a user has to wait at startup until the circuit is built. Figure 10 shows the result of our Shadow simulations regarding the circuit built time.

Our results in Figure 10 show that PCTLS has longer circuit build times than Vanilla Tor. This is no surprise considering that every circuit has its own TLS connection, and it costs time to establish this connection. Which also leads to more extensive memory consumption. Overall it is, on average, 725 ms slower without and 495 ms slower with packet loss. However, if we compare PCTLS directly, it is not much slower with packet loss. PCTLS uses around 100 GB more RAM, PCTLS uses 527 GB and Vanilla uses 430 GB on average.

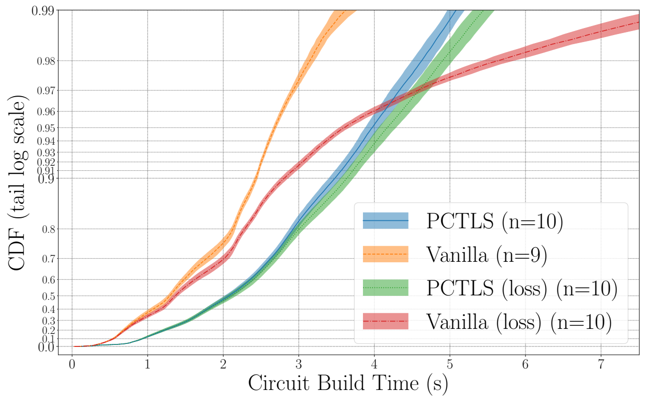


Figure 10: Circuit built time in seconds. Comparison between Vanilla and Vanilla+PCTCP with the Shadow 201801 lossless model, and; Vanilla and Vanilla+PCTCP with the Shadow 2018001 loss model in a 25 % network. The experiments were repeated ten times per configuration, and the CDF is plotted with tail-logarithmic y-axes.

Vanilla has only nine measurements because the first measurement did not record the circuit build time.

Key Insight: Avoiding active shared links by creating a separate TLS connection for all circuits will increase the circuit build time.

4.4 Verification and Comparision of SALSA

To confirm that SALSA represents the Tor Path Selection Algorithm, we verify it with Shadow simulations, modified Tor client, and TorPS. We compare the modified Tor client and TorPS separately from our Shadow simulations. First we verify the accuracy of our simulator and later we compare the simulators besides accuracy.

With ShAred Links SimulAtor we can easily isolate effects on the number of simultaneously shared links and scale the number of relays. To scale a Network we use `tornettools` [20]. Isolation will be explained in more detail later in this section.

As mentioned in Section 3.1.4 we do not implement the complete Path Algorithm.

[4]Commit ID: `1628af235e5fe06536ad6ec32bf87f2225c82d32`

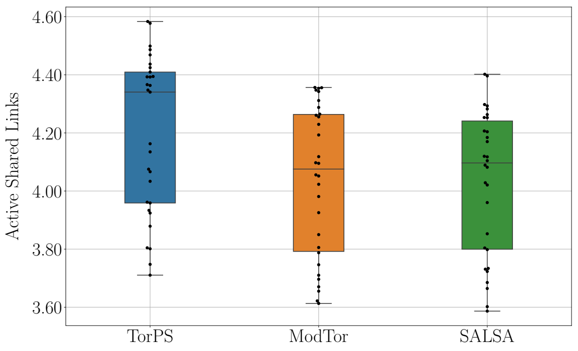


Figure 11: Boxplot that compares the number of active shared links for TorPS, a modified Tor client, and SALSA. Beware that the y-axis does not start at 0; instead, at 3.5 %. Every point per tool is based on a daily consensus document from November 2021.

For example, instead of using the exit policy to determine an exit relay, which requires additionally relay microdescriptors, we only use the exit flag. But only very few relays have no exit flag, but a suitable exit policy. A relay can be chosen more than once in the same circuit, but this happens very rarely, under 300 times in 100 000 circuits. We also do not consider other filtering steps in the path selection algorithm, this includes relay and IP address subnet families.

These simplification are justified if we compare the modified Tor client with SALSA in Figure 11. Both are very similar and the modified Tor client does all filtering steps and considers relay families.

In Figure 11, every point is simulated as a 100 % Tor network. Every point per tool is based on one consensus document every day in November 2021. SALSA's results match the modified Tor client. Only TorPS is slightly shifted upwards by 0.2 %. TorPS selects on average less relays in the guard (-145 ± 40) and middle (-40 ± 16) position, this leads automatically to more shared links. This hints at the guard filtering in TorPS as source of this upward shift. In the modified Tor client we disable long term entry guard selection to be able to simulate many circuits from one client. But this does not fully explain the difference.

Even if left out filtering steps have more influence or TorPS is more accurate we would underestimate the problem. Because we would chose more relays in one or more circuit positions which leads to less shared links.

To verify our simulation with Shadow, we simulate networks at 1 %, 10 % and 25 % scale. For a 25 % network, we have five measurements; otherwise, one measurement. Figure 4 depicts our measurements, and the 25 % network is in the bottom left showing. The average shared links and their difference to SALSA are for 1 %; 54.82 % with a difference of 0.97 %, for 10 %; 20.56 % with a difference of −0.63 % and for 25 %; 12.51 % with a difference of 0.48 %.

In the remaining section we will show other differences from SALSA to the modified Tor Client and previous simulators. In addition, we evaluate the simulators in terms of runtime speed, memory consumption and scalability and describe how different parameter can be isolated. For an in-depth description of the simulators, we refer the reader to Section 3.1.

Shadow [34] is the most prominent simulator, which is highly accepted by the scientific community to reflect the real Tor network. However, Shadow is slow and needs high-computing capabilities. For example, one simulation of a 25 % network on the hardware described in Section 3.2.1 takes around 3 d, consumes 600 GB of memory at the peak and stores around 110 GB. Shadow also has other shortcomings, e. g. no support for entry guards, onion services[5], and exit policies. Additionally, Shadow cannot be used to make forecasts.

TorPS, on the other hand, requires consensus documents and server descriptors. Shadow can generate these documents in conjunction with `tornettools` [20]. However, that takes around 10 min to 20 min of simulation. Building these documents ourselves is possible but error-prone. TorPS reimplements the Tor Path Selection Algorithm and simulates path generation over time based on historical data. The simple user model takes samples of circuit paths at predefined time intervals. Before TorPS can run simulations, it needs to convert the documents into an internal format. For example, a two-month simulation consumes ≈ 50 GB of memory and takes ≈ 50 min due to this extra conversion step. A simulation that takes a sample of 1 M circuits per day consumes 6 GB and take ≈ 40 min per day. For these reasons, using TorPS for scaled networks is cumbersome.

Our modified Tor also requires consensus documents and server descriptors. Additionally, it needs valid directory authority certificates, but only when simulating a network smaller than full scale. The modified Tor client has the same disadvantages as TorPS, except it does not need a conversion step and consumes only ≈ 50 MB of memory. However, it takes 2 h to generate 1 M circuits. This can be mitigated by starting multiple Tor instances in parallel.

We developed SALSA to eliminate most of the disadvantages of the previous simulators and optimise it for speed and scalability. Table 2 summarises the differences between the simulators. Another advantage of SALSA is that we can easily isolate specific parameters. However, this is also possible with the other simulators but requires more effort. In the following, we describe how to isolate specific parameters.

Exit Policies: Considering the modified Tor client and TorPS as black boxes, one could try every TCP port from the traffic classes in Table 1. However, this process is not feasible, and in this case, it would be better to modify the consensus documents directly, as mentioned above. On the other hand, if we consider both simulators as white boxes, we could modify the source code to only include exits in the traffic classes. For TorPS, it would also be possible to modify the internal format and remove unwanted exits. For SALSA, we filter out all unwanted exits in the consensus C before we give it to the simulator.

Direction: This has nothing to do with the simulator. All simulator outputs are circuit paths; we look at the links as directed or undirected.

[5]Shadow in conjunction with `tornettools` version ≤ 1.1.0.

Simulator	Runtime	Memory	Scalable
Shadow 1.13 [34]	≈ 3 d	≈ 600 GB	✓
TorPS [19]	≈ 40 min	≈ 6 GB	✗
Modified Tor Client	≈ 2 h	≈ 50 MB	✗
SALSA	≈ 5 s	≈ 1 GB	✓

Table 2: Comparison between different simulators to estimate the number of shared links in Tor. Shadow is evaluated for a 25 % network and the other simulators to generate 1 M circuits. A 25 % network is creating around 2 M circuits.

Internal Circuits: With the modified Tor client, we can set a flag for an internal circuit, and it will launch one. The same is possible for TorPS. For example, for SALSA, we can set $i = 1$ to generate internal circuits.

Guards relays: With SALSA, we can set the k parameter to how many guards we want per client. It is impossible for the modified Tor client to have many guard "sessions" with one client. This is the same problem Shadow has [20]; it simulates many users per client as optimization and disables entry guards. In our case, we could launch many clients and generate k circuits per client until we have as many circuits as we need. TorPS already has a parameter called `num_guards`, which determines the number of guards per client.

4.5 Different Traffic Types and their Impact

This section investigates different traffic types and their impact on Tor's performance. We use both of our setups depicted in Figure 3 and Figure 2. Table 3 summaries the different setups and scenarios. In the second evaluation, we build on the results of this study.

As already mentioned, a shared link with eight multiplexed circuit parts is unlikely in the current network. Moreover, a wholly shared circuit is even more unlikely. Eight multiplexed circuit parts would only occur in the future if the number of relays stays constant and the number of users grows. In the current network, the most common case is two shared circuit parts over one link. In our second setup, we have only two clients, two guard relays, one middle relay and one exit relay; the link between the middle and exit relay is shared. The two clients build only one circuit, corresponding to two circuit parts over one link. We use the following traffic models:

- A 50 kB/s audio stream model, 400 kbit/s is very high-quality audio (see Figure 14a);
- A 1 MB/s video stream model, this simulates an High Definition video stream with 30 fps (see Figure 14b); and;
- A 2 MB web model, the median website size [35] in 2022, if it is done with one transmission, it will wait for 10 s and then start another (see Figure 14c).
- A 5 MB bulk model, if it is done with one transmission it starts immediately another (see Figure 14d);

We use these models in three different scenarios. In all scenarios, one client always uses the bulk model and the other client one of the four traffic models. We measure each model in each scenario for 5 min and repeat the measurements multiple times. The first scenario is similar to *SharedDropping*; it has a delay of 50 ms between entry and exit, 0.1 % packet loss between the middle and exit relay, and; 300 Mbit/s bandwidth. But only with two clients and guards. It is a point of reference for the previous setup. The second scenario has 300 Mbit/s bandwidth and no delay, but the link between the middle and exit is congested with four TCP bulk streams. The third scenario has a bandwidth limit of 2 Mbit/s on all links between relays, with no delay and the link between middle and exit is also congested with four TCP bulk streams. The four TCP streams download data in the client's direction.

These scenarios highlight the two advantages of PCTLS; (1) preventing TCP HolB and (2) fairer bandwidth allocations due to two congestion windows. The first one is covered by scenario *One*. The second one is covered by scenario *Two* and *Three*. In both, PCTLS should have an advantage over Vanilla.

The results are depicted in Figure 14 in the Appendix. In all scenarios, the circuit round-trip time is always better for PCTLS. The two TLS connections can explain this. In the case of PCTLS, only one connection is affected by packet loss, but Vanilla is always affected. For the former, it is more likely for one packet to be sent with the previous sending rate. As a result, PCTLS is always faster for all models in scenarios *One* and *Three*. The longer the transmission times, the more advantage PCTLS has.

For scenario *Two*, the audio (Figure 14a) and video (Figure 14b) models show nearly no differences. Note that in both cases, the sending rate is limited to one second, so the lines start shortly after one second. Surprisingly, for the web model (Figure 14c), Vanilla is faster. This scenario has no packet loss, which is comparable to *NoDropping*. Due to time-delayed transmissions in the web model, the congestion window gets reseted with PCTLS. But Vanilla has a constant congestion window because of the bulk stream that goes over the same link. This explains why PCTLS is slower and has a greater variation. This scenario is not likely in the live network because the Shadow measurement, which have a similar sending pattern, does not show an advantage for Vanilla overall. It could suggest that reusing connections for thus sending patterns is better. The other traffic models have a constant congestion window for Vanilla and PCTLS, but PCTLS is faster because it uses two connections which allows a bigger combined congestion window than a single connection.

In scenario *Three*, we have no measurements for the web model due to its limited bandwidth. The web model starts transmissions every 10 s without considering if the previous transmission has finished. This causes the measurements for this model to be aborted. The results of our local measurements are comparable to the results of our Shadow setup. Scenario *Two* is the best case and scenario *Three* is the worst case for congestion window scaling. In scenario *Two* the sending congestion window is around 120 segments and in scenario *Three* ten segments or more. Ten segments is the minimum congestion window of TCP. This shows that even in the worst case PCTLS can sent ten segments or more per circuit.

Key Insight: The CCI problem interferes more with Tor performance when packet loss occurs and more circuits are multiplexed.

Scenario Name	# of Clients	BW Limit	Delay	Packet Loss	Traffic Type
NoDropping (ND)	8	300 Mbit/s	50 ms	no	bulk
SharedDropping (SD)	8	300 Mbit/s	50 ms	0.1 % (middle and exit)	bulk
DroppingRemaining (DR)	8	300 Mbit/s	50 ms	0.1 % (client, guard, and middle)	bulk
One	2	300 Mbit/s	50 ms	0.1 % (middle and exit)	bulk × {bulk, audio, video, web}
Two	2	300 Mbit/s	no	4 TCP streams (middle and exit)	bulk × {bulk, audio, video, web}
Three	2	2 Mbit/s	no	4 TCP streams (middle and exit)	bulk × {bulk, audio, video, web}

Table 3: We used different scenarios to conduct experiments in our private Tor setup. The first three scenarios are inspired by [9], and the last three are our own. The traffic type from the last three is bulk combined with either bulk, audio, video or web traffic.

4.6 Repetition of the Performance Experiments from 2009

In this evaluation, we repeat the performance experiments from [9] to have a comparison. We constantly compare Vanilla against PCTLS. In these local setups, we use TCP CUBIC as a congestion control algorithm, in contrast to our Shadow setup, which uses TCP New Reno. In the first setup, we measure three different scenarios. All scenarios have a delay of 50 ms between entry and exit and 300 Mbit/s bandwidth. This bandwidth limitation is unlimited from Tor's point of view as it cannot fully utilize the bandwidth capacity. The first scenario has no packet loss and is called *NoDropping*; the second scenario has 0.1 % packet loss between middle and exit, which is called *SharedDropping*; and; the third scenario has 0.1 % packet loss on links between client, guards and middles, and is called *DroppingRemaining*. These are based on the experiments of Reardon and Goldberg [9]. The packet loss is uniformly distributed. On every client, we run TGen [36], which receives 5 MB repeatedly for 5 min. We measure the circuit round trip time as well as the time to transfer 5 MB.

The results are depicted in Figure 13 in the Appendix. In scenario *NoDropping*, Vanilla is faster than PCTLS, but not by much. In scenario *DroppingRemaining*, Vanilla is a bit slower than PCTLS. In scenario *SharedDropping*, Vanilla is much slower than PCTLS. The circuit round trip time is generally lower for PCTLS. This confirms that the problem is still relevant even with recent improvements.

5 DISCUSSION

5.1 Limitations

We think the number of external shared links is higher than our measurements; this is why we declare them as conservative estimates. In our experiments, we only count the number of active links per second in one direction, i. e. from OP to the exit relay. Therefore we will miss active links in which a Tor cell was sent in the other direction, i. e. from the exit relay to the OP. This direction has the most cells, but both directions are similar in overlapping active links. A similar problem exits with internal shared links. We do not count the link between RP and the last relay of the Onion service circuit. This is because both circuits are independent but should be connected on their ends. The real number of simultaneously shared links for internal circuits is slightly higher than our estimates. In addition, we only model some of the complexity involved with Onion services. For example, we do not simulate the link between RPs of the client side and the last relay of the service side in Section 4.1.5. We also do not consider vanguards, which protect against guard discovery attacks. They introduce one additional relay into the circuit before the RP and restrict relay selection for the middle positions. Finally, we do not consider relay and IP address subnet families. Relays with the same family can not be in the same circuit, reducing the possible relays. Again, this can increase the number of internal and external shared links.

Shadow only implemented TCP New Reno [37] for its congestion control. As described in Section 2.2, most modern OS use TCP CUBIC as their default congestion control. This could lead to minor performance variations in our Shadow simulations. However, the differences will be insignificant since both, Vanilla and PCTCP are affected. In our local experimental environment, we used TCP CUBIC.

After we finished our Shadow simulations without packet loss, we found a race condition in the PCTLS code, which led to only one connection being used for multiple circuits. However, this race condition only happens when a relay *simultaneously* extends multiple circuits to another relay. This condition is unlikely in Shadow since the creation of circuits happens asynchronously. As far as we know, this race condition affects the original PCTCP code and the QuicTor implementation of PCTCP. This bug does not affect our measurements with Vanilla or the local experiments.

Our Shadow setup builds besides general purpose circuits, other circuits, i. e. around 150 one hop, 800 internal and 400 timeout measurement circuits. We did not measure the influence of these circuits on the CCI problem.

5.2 Ethical Principles

One missing measurement in our study is the measurement of the actual Tor network. However, we decided against measurements on the live network because of ethical and privacy concerns.

6 RELATED WORK

Reardon and Goldberg [9] did a comparable study in 2009. As mentioned before, we rebuilt their private testbed system with a newer Tor version. This comparison can be found in Table 4. The latency degradation is less pronounced for PCTLS but comparable for Vanilla. One possible explanation is that we have a different solution for CCI and 50 ms delay instead of 400 ms. For throughput degradation, it is the other way around, it is comparable for PCTLS but for Vanilla it is much worse. We measure the transmission time

Version	Scenario	Circuit Throughput	TP Deg.	TP Deg. [9]	Average Latency	Lat. Deg.	Lat. Deg. [9]
PCTLS	*NoDropping* (ND)	1.97 MB/s ± 0.04 MB/s	0 %	0 %	75.0 ms ± 8.8 ms	0 %	0 %
PCTLS	*DroppingRemaining* (DR)	1.79 MB/s ± 0.09 MB/s	9 %	8 %	76.0 ms ± 8.2 ms	2 %	20 %
PCTLS	*SharedDropping* (SD)	1.86 MB/s ± 0.08 MB/s	6 %	4 %	76.0 ms ± 8.5 ms	1 %	7 %
Vanilla	*NoDropping* (ND)	2.00 MB/s ± 0.04 MB/s	0 %	0 %	88.0 ms ± 13.7 ms	0 %	0 %
Vanilla	*DroppingRemaining* (DR)	1.37 MB/s ± 0.19 MB/s	31 %	6 %	91.0 ms ± 14.9 ms	4 %	5.4 %
Vanilla	*SharedDropping* (SD)	0.37 MB/s ± 0.04 MB/s	81 %	17 %	97.0 ms ± 15.1 ms	10 %	12.9 %

Table 4: Comparison of our results with the results from Reardon and Goldberg [9]. Throughput (TP) and Latency (Lat.) degradations (deg.) for different scenarios. Columns five and eight are the results from Reardon and Goldberg [9], where a similar table exists.

for 5 MB and calculate throughput afterwards. It could be that the problem worsened or the measurement method is different. This comparison indicates that the CCI problem could have gotten worse with the newer Tor version.

Jansen, Geddes, Wacek, Sherr, and Syverson [11] and Jansen, Traudt, Geddes, Wacek, Sherr, and Syverson [12] investigated congestion in Tor and discovered two significant problems: First, Tor writes data sequentially to the socket and second, it writes as much data to the socket as possible. They proposed KIST, which dynamically calculates the amount of data to write on each socket based on real-time kernel information. KIST has been merged in Tor version 0.3.2.1-alpha and can be seen as an application layer TCP HolB mitigation. However, KIST is most effective if there are many unshared links. According to the paper the probability of an unshared link is 99.775 %. That can be explained by the fact that the study is from 2014. At that time, the Tor infrastructure was different from today.

AlSabah and Goldberg [7] replaced the TLS connection between ORs with an IPsec connection. By securing the connection between ORs on Layer 3, each circuit can have its kernel-level TCP connection and congestion window. According to their measurements, it will improve the response time by 60 % and reduce the download time by 30 %. However, Geddes, Jansen, and Hopper [10] found that PCTCP is vulnerable to socket exhaustion attacks. An attacker can create arbitrary circuits on a specific path until the OR runs out of TCP connections since it has a soft and hard limit. Therefore, they proposed novel connection schedulers that prevent socket exhaustion attacks while still having the performance advantages of PCTCP.

A similar idea from Basyoni, Erbad, Alsabah, Fetais, Mohamed, and Guizani [8] is QuicTor, in which they use QUIC instead of IPsec. QUIC is defined in RFC 9000 [38] and is a secure and reliable transport protocol based on UDP. In contrast to TCP, QUIC supports multiple streams in a single connection and avoids the HolB problem. A packet loss in one stream only affects the same stream, while other streams can continue exchanging packets. Their performance evaluation seems promising; however, they have yet to evaluated it in a large-scale environment, as seen in Table 5.

Name	Year	Relays	Clients	Evaluation
TCPoDTLS [9]	2009	6	8	-
PCTCP [7]	2013	50	500	ExperimenTor [39]
KIST [11]	2014	3600	13 800	Shadow
IMUX [10]	2014	500	1800	Shadow
KIST [12]	2018	2000	49 800	Shadow
QuicTor [8]	2021	50	350	NetMirage
Our Study	2022	1750	200 000	Shadow (n=10)

Table 5: Comparison of different performance measurements from previous studies.

7 CONCLUSION AND FUTURE WORK

This paper examined the number of simultaneously active shared links in the Tor network, which is the root cause of CCI. We developed a novel simulator that we verified with Shadow simulations to investigate this problem. According to our measurements, the current Tor network has around 3.9 % simultaneously active shared links. The involved ORs have the highest bandwidth capabilities. An influence on this number has the exit policy and if the Tor network is used to address internal or external services. Surprisingly, ignoring entry guards and the number of active circuits per client has little influence on the number of simultaneously active shared links. We used the simulator and modelled what happens if the number of active circuits increases up to the limit of the Tor network. When this happens, the number of simultaneously active shared links can grow to 16 %. We highlight that our estimates are conservative. Finally, we run experiments with Shadow with a 25 % network with Vanilla and PCTLS. Our results show that the CCI problem directly influences the performance of the Tor network.

In future work, we want to investigate the implications of the CCI problem with the latest version of the Tor client that has end-to-end congestion control integrated. Additionally, we want to compare the different solutions that have been proposed to mitigate the CCI problem in Shadow.

ACKNOWLEDGMENTS

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors. We thank Richard Göbel for the provided testbed system, Adrian Wöltche for the administrative support; and Ian Goldberg, Mashael Alsabah and Lamiaa Basyoni for sharing source code with us.

REFERENCES

[1] J. Ball, B. Schneier, and G. Greenwald. "NSA and GCHQ target Tor network that protects anonymity of web users," the Guardian. (), [Online]. Available: http://www.theguardian.com/world/2013/oct/04/nsa-gchq-attack-tor-network-encryption (visited on 10/28/2020).

[2] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router:" Tech. Rep., 2004. [Online]. Available: https://svn-archive.torproject.org/svn/projects/design-paper/tor-design.pdf (visited on 05/30/2022).

[3] Tor Project. "Tor Metrics!" (2022), [Online]. Available: https://metrics.torproject.org/ (visited on 02/16/2022).

[4] A. Mani, T. Wilson-Brown, R. Jansen, A. Johnson, and M. Sherr, "Understanding Tor Usage with Privacy-Preserving Measurement," in *Proceedings of the Internet Measurement Conference 2018*, Boston MA USA: ACM, Oct. 31, 2018. DOI: 10.1145/3278532.3278549.

[5] R. Jansen, M. Traudt, and N. Hopper, "Privacy-Preserving Dynamic Learning of Tor Network Traffic," in *Proceedings of the 25th ACM Conference on Computer and Communications Security*, Toronto Canada: ACM, Oct. 15, 2018, pp. 1944–1961, ISBN: 978-1-4503-5693-0. DOI: 10.1145/3243734.3243815.

[6] M. Alsabah and I. Goldberg, "Performance and Security Improvements for Tor: A Survey," *ACM Computing Surveys*, vol. 49, no. 2, pp. 1–36, Sep. 21, 2016, ISSN: 03600300. DOI: 10.1145/2946802.

[7] M. AlSabah and I. Goldberg, "PCTCP: Per-circuit TCP-over-IPsec transport for anonymous communication overlay networks," in *Proceedings of the 20th ACM Conference on Computer & Communications Security (CCS' 13)*, ser. CCS '13, New York, NY, USA: Association for Computing Machinery, Nov. 4, 2013, pp. 349–360. DOI: 10.1145/2508859.2516715.

[8] L. Basyoni, A. Erbad, M. Alsabah, N. Fetais, A. Mohamed, and M. Guizani, "QuicTor: Enhancing tor for real-time communication using QUIC transport protocol," *IEEE Access*, 2021, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3059672.

[9] J. Reardon and I. Goldberg, "Improving Tor using a TCP-over-DTLS tunnel," in *Proceedings of the 18th USENIX Security Symposium*, USA: USENIX Association, Aug. 10, 2009, pp. 119–134, ISBN: 978-1-71380-400-0. [Online]. Available: https://www.usenix.org/legacy/event/sec09/tech/full_papers/reardon.pdf (visited on 05/31/2022).

[10] J. Geddes, R. Jansen, and N. Hopper, "IMUX: Managing tor connections from two to infinity, and beyond," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, ser. WPES '14, New York, NY, USA: Association for Computing Machinery, Nov. 3, 2014, pp. 181–190, ISBN: 978-1-4503-3148-7. DOI: 10.1145/2665943.2665948.

[11] R. Jansen, J. Geddes, C. Wacek, M. Sherr, and P. Syverson, "Never Been KIST: Tor's Congestion Management Blossoms with Kernel-Informed Socket Transport," in *Proceedings of the 23rd USENIX Security Symposium*, 2014, p. 16.

[12] R. Jansen, M. Traudt, J. Geddes, C. Wacek, M. Sherr, and P. Syverson, "KIST: Kernel-Informed Socket Transport for Tor," *ACM Transactions on Privacy and Security (TOPS)*, no. 1, pp. 1–37, 2018. DOI: 10.1145/3278121.

[13] tommy. "KIST and Tell: Tor's New Traffic Scheduling Feature | Tor Project." (Oct. 3, 2017), [Online]. Available: https://blog.torproject.org/kist-and-tell-tors-new-traffic-scheduling-feature/ (visited on 05/29/2022).

[14] M. AlSabah, K. Bauer, I. Goldberg, D. Grunwald, D. McCoy, S. Savage, and G. M. Voelker, "DefenestraTor: Throwing Out Windows in Tor," in *Proceedings of the 11th on Privacy Enhancing Technologies Symposium (PETS)*, ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 134–154. DOI: 10.1007/978-3-642-22263-4_8.

[15] T. Wang, K. Bauer, C. Forero, and I. Goldberg, "Congestion-Aware Path Selection for Tor," in *Proceedings of the 16th International Conference on Financial Cryptography and Data Security (FC)*, A. D. Keromytis, Ed., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2012, pp. 98–113, ISBN: 978-3-642-32946-3. DOI: 10.1007/978-3-642-32946-3_9.

[16] M. Perry. "Congestion Control Arrives in Tor 0.4.7-Stable! | Tor Project." (May 4, 2022), [Online]. Available: https://blog.torproject.org/congestion-contrl-047/ (visited on 05/07/2022).

[17] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, Jul. 2008, ISSN: 0163-5980. DOI: 10.1145/1400097.1400105.

[18] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC Editor, RFC 6582, Apr. 2012. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6582.txt.

[19] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson, "Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries," in *Proceedings of the 20th ACM Conference on Computer & Communications Security - CCS '13*, Berlin, Germany: ACM Press, 2013, pp. 337–348, ISBN: 978-1-4503-2477-9. DOI: 10.1145/2508859.2516651.

[20] R. Jansen, J. Tracey, and I. Goldberg, "Once is Never Enough: Foundations for Sound Statistical Inference in Tor Network Experimentation," in *Proceedings of the 30th USENIX Security Symposium*, 2021. [Online]. Available: https://neverenough-sec2021.github.io (visited on 05/31/2022).

[21] K. Loesing, S. J. Murdoch, and R. Dingledine, "A Case Study on Measuring Statistical Data in the Tor Anonymity Network," in *Proceedings of the Workshop on Ethics in Computer Security Research (WECSR 2010)*, ser. LNCS, Tenerife, Canary Islands, Spain: Springer, Jan. 2010. DOI: 10.1007/978-3-642-14992-4_19.

[22] Hommel, Wolfgang. "libfaketime." (2022), [Online]. Available: https://github.com/wolfcw/libfaketime (visited on 12/11/2022).

[23] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics

in Networks, ser. Hotnets-IX, New York, NY, USA: Association for Computing Machinery, Oct. 20, 2010, pp. 1–6, ISBN: 978-1-4503-0409-2. DOI: 10.1145/1868447.1868466.

[24] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible Network Experiments Using Container-Based Emulation," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12, New York, NY, USA: Association for Computing Machinery, Dec. 10, 2012, pp. 253–264, ISBN: 978-1-4503-1775-7. DOI: 10.1145/2413176.2413206.

[25] Rob Jansen and Justin Tracey and Ian Goldberg. "tornettools." (2022), [Online]. Available: https://github.com/shadow/tornettools (visited on 12/11/2022).

[26] R. Jansen and A. Johnson, "Safely Measuring Tor," in *Proceedings of the 23rd ACM Conference on Computer and Communications Security*, ser. CCS '16, New York, NY, USA: Association for Computing Machinery, Oct. 24, 2016, ISBN: 978-1-4503-4139-4. DOI: 10.1145/2976749.2978310.

[27] "Tor Rendezvous Specification - Version 3," The Tor Project, Tech. Rep., Feb. 17, 2022. [Online]. Available: https://github.com/torproject/torspec/blob/main/rend-spec-v3.txt (visited on 05/31/2022).

[28] "Brave Web Browser." (), [Online]. Available: https://httparchive.org/reports/page-weight (visited on 09/13/2022).

[29] C. Cimpanu. "Mozilla Offers Research Grant for a Way to Embed Tor inside Firefox," ZDNet. (May 9, 2019), [Online]. Available: https://www.zdnet.com/article/mozilla-offers-research-grant-for-a-way-to-embed-tor-inside-firefox/ (visited on 05/24/2022).

[30] A. Færøy. "Wiki from the Torproject: Firefox Tor Uplift And To rMode AddOn," GitLab. (Jun. 14, 2020), [Online]. Available: https://gitlab.torproject.org/legacy/trac/-/wikis/org/meetings/2019Stockholm/Notes/FirefoxTorUpliftAndTorModeAddOn (visited on 05/24/2022).

[31] B. Fabian, F. Goertz, S. Kunz, S. Müller, and M. Nitzsche, "Privately Waiting - A Usability Analysis of the Tor Anonymity Network," in *Proceedings of the 16th Americas Conference on Information Systems (AMCIS)*, vol. 58, Aug. 12, 2010, p. 258, ISBN: 978-3-642-15140-8. DOI: 10.1007/978-3-642-15141-5_6.

[32] "Firefox Public Data Report," Mozilla. (May 14, 2022), [Online]. Available: https://data.firefox.com/dashboard/user-activity (visited on 05/24/2022).

[33] Jansen, Rob. "Network graph file for use with Shadow v1.13." (Sep. 17, 2018), [Online]. Available: https://github.com/tmodel-ccs2018/tmodel-ccs2018.github.io/blob/master/data/shadow/network/atlas.201801.shadow113.graphml.xml.xz (visited on 09/13/2022).

[34] R. Jansen and N. Hooper, "Shadow: Running Tor in a Box for Accurate and Efficient Experimentation:" in *Proceedings of the 19th Symposium on Network and Distributed System Security (NDSS)*, Internet Society, Feb. 2012. DOI: 10.21236/ADA559181.

[35] "HttpArchive: Page Weight." (Aug. 1, 2022), [Online]. Available: https://httparchive.org/reports/page-weight (visited on 09/13/2022).

[36] "GitHub: TGen." (Jul. 22, 2022), [Online]. Available: https://github.com/shadow/tgen (visited on 09/13/2022).

[37] "GitHub: Shadow: shd-tcp-cong-reno.c." (Oct. 31, 2019), [Online]. Available: https://github.com/shadow/shadow/blob/v1.13.2/src/main/host/descriptor/shd-tcp-cong-reno.c (visited on 09/13/2022).

[38] J. Iyengar and M. Thomson, "RFC 9000: QUIC: A UDP-Based Multiplexed and Secure Transport," IETF, Request for Comments RFC 9000, May 2021, 151 pp. DOI: 10.17487/RFC9000.

[39] K. Bauer, D. McCoy, M. Sherr, and D. Grunwald, "ExperimenTor: A Testbed for Safe and Realistic Tor Experimentation," in *Proceedings of the 4th USENIX Workshop on Cyber Security Experimentation and Test (CSET)*, USENIX Association, Aug. 2011, p. 8. [Online]. Available: https://www.usenix.org/conference/cset11/experimentor-testbed-safe-and-realistic-tor-experimentation (visited on 05/30/2022).

A APPENDIX

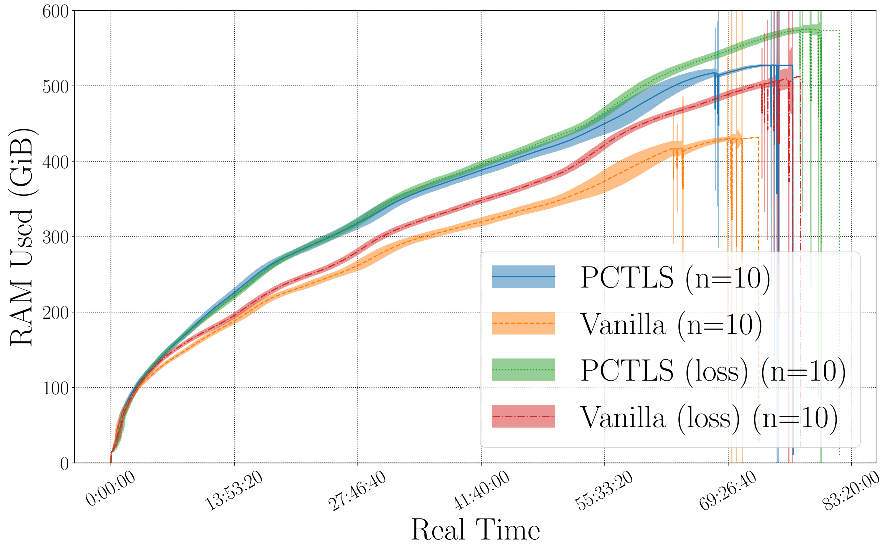
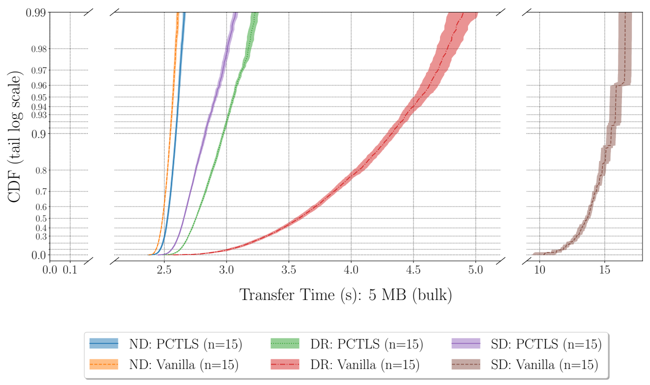
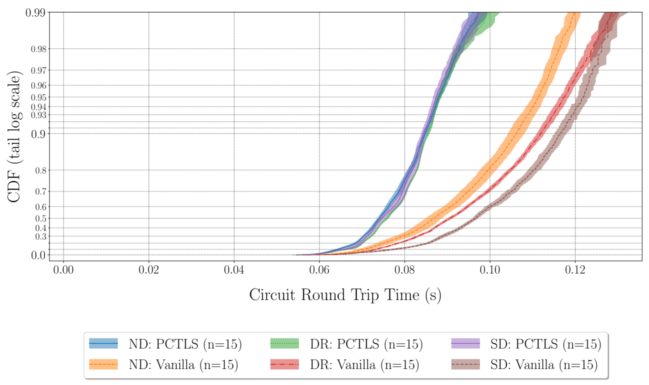


Figure 12: Shows the memory consumption of all shadow simulations over real time. At the peak, the simulation needs around 600 GB of RAM.

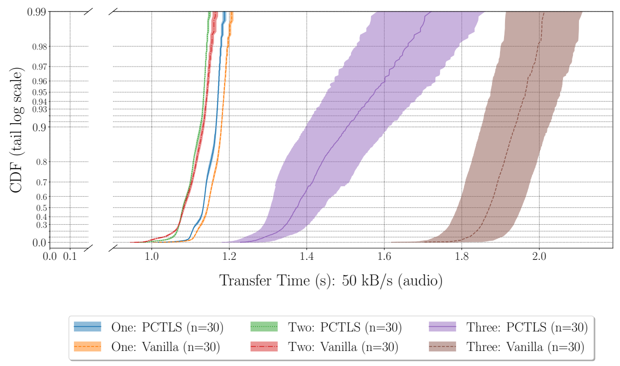


(a)

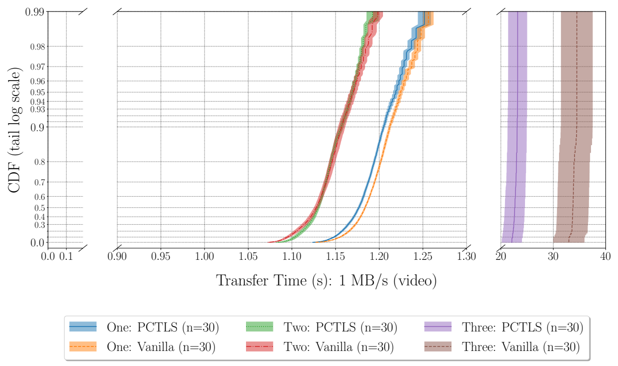


(b)

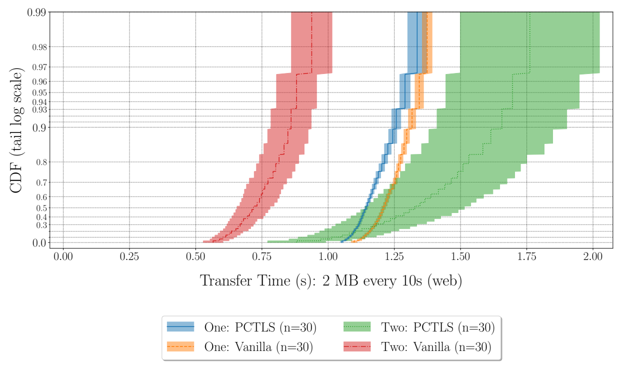
Figure 13: Results for the local testbed setup depicted in Figure 3. We compare Vanilla with PCTLS in three scenarios. Links have 300 Mbit/s bandwidth and 50 ms delay between entry and exit. (a) shows the transfer time of 5 MB and (b) shows the circuit build time. (ND) stands for *NoDropping*, (DR) stands for *DroppingRemaining* and (SD) stands for *SharedDropping*.



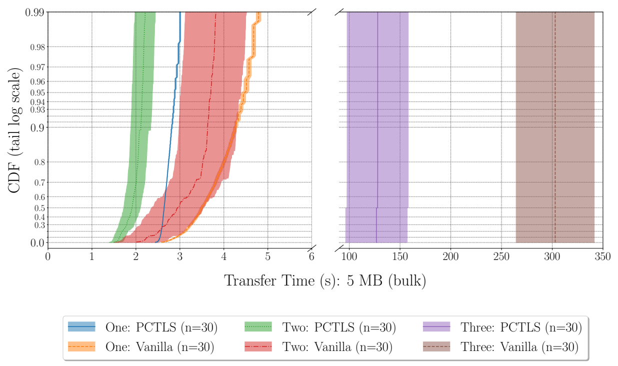
(a)



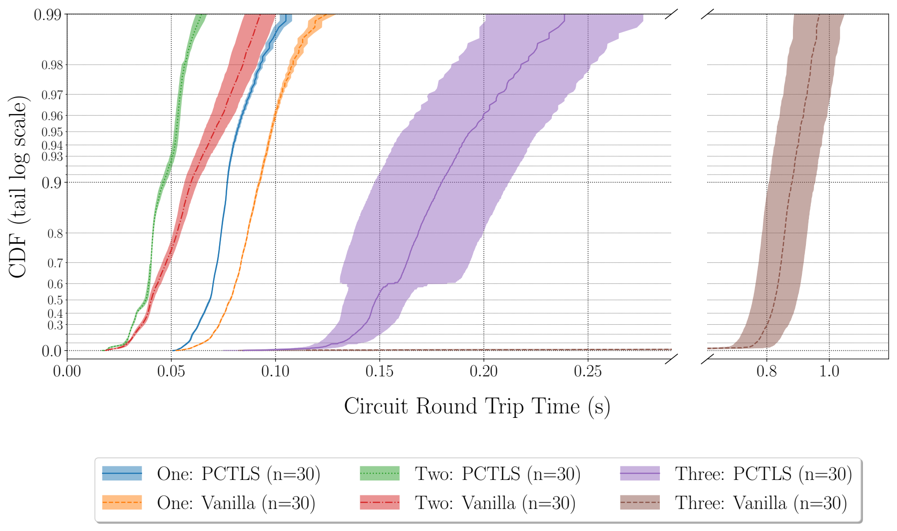
(b)



(c)



(d)



(e)

Figure 14: Results for the second local testbed setup depicted in Figure 2. We compare Vanilla with PCTLS in three scenarios. For more details about the scenarios see Section 4.5. For 14a and 14b the sending rate is limited to 50 kB/s and 1 MB/s respectively. Scenario One limits bandwidth, has delay and packet loss. It is as point of reference to [9]; Scenario Two has practically no limitations and shares the bandwwidth with 4 outside TCP streams; and; Scenario Three sets a low bandwidth limit and shares the bandwidth with 4 outside TCP streams (see Table 3). Beware that the x-axis is broken up and the parts have different scaling factors.

Algorithm 1 SALSA – A simplified model of the Tor Path Selection Algorithm

```
function CONSENSUS($C$)                                   ▷ Prepares consensus
    $N \leftarrow$ number of relays in $C$
    $W_{\text{guards}}[1 \dots N] \leftarrow 0$
    $W_{\text{middles}}[1 \dots N] \leftarrow 0$
    $W_{\text{exits}}[1 \dots N] \leftarrow 0$
    for all relays $r$ with index $i$ in consensus $C$ do
        if $r$ has GUARD or EXIT flag then
            $W_{\text{guards}}[i] \leftarrow r_{\text{bandwidth}} \times$ get_weight($C$, "position 1", $r_{\text{flags}}$)
        end if
        $W_{\text{middels}}[i] \leftarrow r_{\text{bandwidth}} \times$ get_weight($C$, "position 2", $r_{\text{flags}}$)
        if $r$ has EXIT flag then
            $W_{\text{exits}}[i] \leftarrow r_{\text{bandwidth}} \times$ get_weight($C$, "position 3", $r_{\text{flags}}$)
        end if
    end for
    return $N$, $W_{\text{guards}}$, $W_{\text{middles}}$, $W_{\text{exits}}$
end function
function COUNT($L$, $R$, $d$)        ▷ $L$ and $R$ are arrays of relays between links exists, $d$ the link is directed or undirected
    if $d = 0$ then                                       ▷ The order does *not* matter
        $u, s \leftarrow$ Count all unique and shared links between $L$ and $R$. Every link that appears more than once is a shared link.
    else                                                  ▷ The order does matter
        $u, s \leftarrow$ Count all unique and shared links between $L$ and $R$. Every link that appears more than once is a shared link.
    end ifreturn $u, s$
end function
function SALSA($C$, $N_{\text{circuits}}$, $k$, $i$, $d$)▷ $C$ is the consensus document, $N_{\text{circuits}}$ is the number of circuits to generate, $k$ number of circuits
per client, $i = 1$ generate internal circuits, $d = 1$ directed links
    $N, W_{\text{guards}}, W_{\text{middles}}, W_{\text{exits}}$ = CONSENUS($C$)
    $R_{\text{entries}} \leftarrow$ randomly_chose_n_nodes_by_weight($\frac{N_{\text{circuits}}}{k}$, $1 \dots N$, $W_{\text{guards}}$)
    $R_{\text{entries}} \leftarrow$ repeat($R_{\text{entries}}$, $k$)
    $R_{\text{middles}} \leftarrow$ randomly_chose_n_nodes_by_weight($N_{\text{circuits}}$, $1 \dots N$, $W_{\text{middles}}$)
    if $i = 1$ then
        $R_{\text{middles}} \leftarrow$ randomly_chose_n_nodes_by_weight($N_{\text{circuits}}$, $1 \dots N$, $W_{\text{middles}}$)
    else
        $R_{\text{exits}} \leftarrow$ randomly_chose_n_nodes_by_weight($N_{\text{circuits}}$, $1 \dots N$, $W_{\text{exits}}$)
    end if
    $L \leftarrow R_{\text{entries}} | R_{\text{middles}}$                    ▷ Concat both arrays together
    $R \leftarrow R_{\text{middles}} | R_{\text{exits}}$                      ▷ Concat both arrays together
    $u, s = \frac{\text{COUNT}(L,R,d)}{N}$
    return $\frac{s}{u} \times 100$                                  ▷ Percent shared links
end function
```