PCTCP: Per-Circuit TCP-over-IPsec Transport for Anonymous Communication Overlay Networks

Mashael AlSabah*
Qatar Computing Research Institute
Department of Computer Science and Engineering, Qatar University
malsabah@qu.edu.qa

Ian Goldberg
Cheriton School of Computer Science
University of Waterloo
iang@cs.uwaterloo.ca

ABSTRACT

Recently, there have been several research efforts to design a transport layer that meets the security requirements of anonymous communications while maximizing the network performance experienced by users. In this work, we argue that existing proposals suffer from several performance and deployment issues and we introduce PCTCP, a novel anonymous communication transport design for overlay networks that addresses the shortcomings of the previous proposals. In PCTCP, every overlay path, or *circuit*, is assigned a separate kernel-level TCP connection that is protected by IPsec, the standard security layer for IP.

To evaluate our work, we focus on the Tor network, the most popular low-latency anonymity network, which is notorious for its performance problems that can potentially deter its wider adoption and thereby impact its anonymity. Previous research showed that the current transport layer design of Tor, in which several circuits are multiplexed in a single TCP connection between any pair of routers, is a key contributor to Tor's performance issues.

We implemented, experimentally evaluated, and confirmed the potential gains provided by PCTCP in an isolated testbed and on the live Tor network. We ascertained that significant performance benefits can be obtained using our approach for web clients, while maintaining the same level of anonymity provided by the network today. Our realistic large-scale experimental evaluation of PCTCP shows improvements of more than 60% for response times and approximately 30% for download times compared to Tor. Finally, PCTCP only requires minimal changes to Tor and is easily deployable, as it does not require all routers on a circuit to upgrade.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General— *Data communications*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; C.4 [**Computer Systems Organization**]: Performance of Systems; K.4.1 [**Computers and Society**]: Public Policy Issues—*Privacy*

*Work done while at the University of Waterloo

CCS'13, November 4–8, 2013, Berlin, Germany.

ACM 978-1-4503-2477-9/13/11 ...$15.00.
http://dx.doi.org/10.1145/2508859.2516715.

General Terms

Measurement, Performance, Security

Keywords

Tor, transport design, performance improvement

1. INTRODUCTION

While advances to the Internet have enabled users to easily interact and exchange information online, they have also created several opportunities for adversaries to prey on users' private information. Whether the motivation for data collection is commercial, where service providers sell data for marketers, or political, where a government censors, blocks and tracks its people, or even personal, for cyberstalking purposes, there is no doubt that the consequences of personal information leaks can be severe.

Consequently, several solutions emerged, a key example of which is Tor [14]. Tor is the most widely used privacy-preserving network that empowers people with low-latency anonymous online access. That is, people can surf the Internet without the fear of revealing their identity or location. Since its introduction in 2003, Tor has successfully evolved to support hundreds of thousands of users using approximately 3000 volunteer-operated routers run all around the world. Incidents of sudden increases in Tor's usage, coinciding with global political events, confirm the importance of the Tor network for Internet users today [13].

Despite Tor's increasing popularity, the bitter reality is that it offers anonymity at the expense of intolerable performance costs. Not only do performance problems hinder Tor's wider adoption, but they can have an immense impact on its anonymity. If users are discouraged from Tor's below-mediocre service, the anonymity set of all users would eventually shrink, which in turn reduces the anonymity guarantees obtained from the network today.

For this reason, the Tor research community has been intensively investigating the sources of the performance problems in Tor, as well as proposing remedies to enhance the usability of Tor. First, one major problem in Tor is traffic congestion, which has a number of causes. One cause for congestion is the high client-to-relay ratio which is approximately 165:1. To help reduce the client-to-relay ratio, incentive-based schemes have been introduced to encourage users to donate bandwidth to the network to reduce the traffic pressure on the routers [20, 27, 29].

Congestion is also magnified because a small fraction of users use greedy file-sharing applications that can consume up to 40% of the bandwidth [24]. What adds to the problem is Tor's lack of congestion control and awareness, as Tor only implements an end-to-end window-based flow-control algorithm that does not react to congestion. To address these problems, some congestion control and avoidance techniques have been proposed to reduce congestion [6, 43]. To reduce the effects that greedy applications impose on the network, static and dynamic throttling approaches have been proposed for clients' connections [21, 27].

Regardless of all these intensive efforts, performance problems will continue to persist in Tor, even if the above proposals are employed. A major culprit is Tor's poor transport design, which has been shown to add unnecessary latency [15, 31]. Tor multiplexes circuits (overlay paths established through the Tor network) from different users over the same TCP connection. Reardon and Goldberg [31] observed that since heavy circuits are often multiplexed with light circuits in the same TCP connection, and since heavy circuits have higher loss rates, they result in unfair application of the TCP congestion control of the shared connection on all circuits. As a design solution, Reardon and Goldberg proposed TCP-over-DTLS, where every circuit gets a separate user-level TCP connection, and DTLS is used for encrypting and securing the communication between routers. Unfortunately, TCP-over-DTLS faces the following design drawbacks:

- **Performance:** User-level implementations of TCP provide significantly lower performance than their kernel-level counterparts in terms of throughput and consume substantially more CPU cycles [10, 16], a scarce resource in Tor. Such heavy costs might render any performance benefits moot if a user-level TCP scheme is deployed at a wide scale.
- **Deployability:** First, the unavailability of a reliable user-level TCP stack with a license that is compatible with Tor is a major obstacle facing TCP-over-DTLS.[1] Second, for any pair of routers to use TCP-over-DTLS, both routers need to upgrade their transport design.

Our Approach. In this work, we seek to enhance the performance and usability of the Tor network for interactive application users. We tackle the performance problem in Tor at its roots, and focus on fixing the weaknesses in Tor's transport design. This work is not concerned with the lack of bandwidth resources, as there have been several proposals that address this problem, as we described above. We propose PCTCP, a new transport design for Tor in which a separate kernel-level TCP connection is dedicated to every circuit. To protect and secure communication between routers, we use IPsec, the standard security layer for IP. Our design significantly improves the performance of Tor while maintaining its threat model. Additionally, PCTCP requires only minimal changes to the software. Our design combines the advantages of the previous TCP-over-DTLS proposal, while avoiding its deployment and performance shortcomings, inherent from using a user-level TCP stack. Furthermore, PCTCP does not require all routers on the circuit to upgrade, except for enabling IPsec communication for a pair of routers that wish to use PCTCP. Our design has a significantly easier road to deployment.

Contributions. This is the first work that implements a new transport design, for anonymous communication systems in general and for the Tor anonymity network in particular, and evaluates it with realistic large-scale experiments, as well as live network experiments. In designing and implementing PCTCP, we offer the following contributions:

- We propose and implement PCTCP, a novel transport design for anonymous communication systems in general and for Tor in particular that avoids the deployability and performance drawbacks of previous designs.
- We carry out small-scale experiments on the live Tor network to evaluate our design. Our results show significant reductions in delays observed. At the 75$^{\text{th}}$ percentile, our response times are improved by more than 47% and our download times are improved by 27%.
- We further evaluate our design by performing a series of large-scale experiments on a network emulator with a topology that closely approximates the performance of the live Tor network. Our results show significant performance benefits for the download and response times of web clients.
- Our simple, yet effective, approach is incrementally deployable, as our changes, except for enabling IPsec communication between any pair of routers using PCTCP, are local to individual routers and do not affect their operation with other routers.

[1] Reardon and Goldberg used the Daytona TCP stack for their implementation and measurements. Unfortunately, Daytona cannot be used for the Tor network due to its unavailability for open-source projects.

The rest of the paper is structured as follows. We provide the reader with the necessary background on Tor and IPsec in section 2 and compare our work to previous work in section 3. Then, we elaborate on our design in section 4 and evaluate it in section 5. Finally, we discuss some open issues regarding our design and experiments in section 6 and conclude in section 7.

2. BACKGROUND

In this section, we start by providing an overview of the Tor network and its current transport design. Then, we introduce and explain the basic functionality of IPsec.

2.1 Tor

Tor is a low-latency anonymization network that is based on the concept of onion routing. The network consists of approximately 3000 volunteer-operated relays [39], known as *Onion Routers* (ORs). Each OR creates a *router descriptor* that contains its contact information, such as its IP address, ports, public keys, and its bandwidth capabilities, and sends the descriptor to *directory authorities*. Tor clients, nicknamed *Onion Proxies* (OPs), download the router descriptors from directories to build paths, referred to as *circuits*, through the network before they can communicate with their Internet destinations. Each circuit usually consists of three ORs, which are referred to as the *entry guard*, *middle*, and *exit* OR, according to their position in the circuit. ORs in a circuit are connected by TCP connections and TLS [12] is used to provide hop-by-hop authenticity, data integrity and confidentiality.

Circuit Construction. For performance reasons, an OP preemptively creates a number of spare circuits for its user applications. When the OP receives a new TCP stream from a user application, it attaches it to an appropriate pre-established circuit. If no such circuit exists, the OP builds a new circuit by first selecting three routers, X_i, according to Tor's bandwidth-weighted router selection algorithm. Next, to start establishing the circuit, the OP sends a *create_fast* command to X_1, which responds with a *created_fast* reply. To extend the Diffie-Hellman (DH) channel, the OP sends an *extend* command to X_1, containing in its payload a *create* command and the first half of the DH handshake for router X_2 encrypted to X_2's public key. Router X_1 forwards this *create* command to router X_2, and when it receives a *created* cell back from router X_2, it forwards its payload in an *extended* cell to the OP to finish the client's DH handshake with router X_2. The same procedure is carried out for each subsequent OR added to the circuit.

The OP acts as a SOCKS proxy to communicate with user applications. The OP divides the user's data into 512-byte fixed-sized

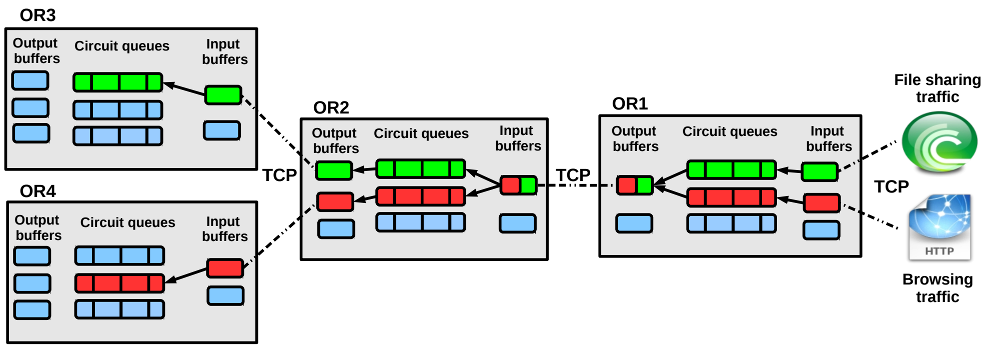


Figure 1: The cross-circuit interference problem: the figure demonstrates the cross-circuit interference problem when a single TCP connection is shared between a loud and a quiet circuit. OR1, acting as an exit for both circuits, receives file-sharing data and web browsing data on two different connection input buffers. The cells then are pushed to their circuit queues. Since the next hop for each circuit is OR2, both circuits share the same connection output buffer. Since the file-sharing circuit is expected to drop more data on the connection between OR1 and OR2, the web browsing circuit experiences more delays due to the unfair application of the TCP congestion control on the shared connection.

cells, adds a layer of encryption for every node on the forward path, and then cells are source-routed through the established circuits. Every hop, on receiving a relay cell, looks up the corresponding circuit, decrypts the relay header and payload with the session key for that circuit, replaces the circuit ID of the header, and forwards the decrypted cell to the next OR. When the exit OR receives the cell, it removes the last layer of the encryption, and establishes the connection on behalf of the user to the intended destination.

Threat Model. Anonymity is maintained for Tor's users because only the entry OR receives a direct connection from a user, and only the exit OR forms a direct connection to the destination. Therefore, no single entity can link users to their destinations. The threat model in Tor assumes a local active adversary that can watch part of the network. The anonymity of a Tor circuit is compromised if the adversary can watch the two ends, the entry and exit, of the circuit.

Cross-Circuit Interference Problem. Tor's OPs and ORs communicate with each other using TCP connections. Every OR-to-OR TCP connection multiplexes circuits from several users. Reardon [31] pointed out that this design can potentially hinder the performance of interactive circuits. This problem is illustrated in Figure 1. The connection between OR1 and OR2 in the figure depicts a scenario where a noisy circuit, carrying BitTorrent traffic for example, is multiplexed with a circuit carrying interactive web browsing traffic. In this case, TCP congestion control would be unfairly applied on both circuits whenever the noisy circuit triggers congestion, due to lost or dropped packets, on the shared TCP connection. Since the amount of data transmitted by file sharing applications is significantly larger than that by interactive applications, it is expected that bulk application circuits trigger congestion control more often than interactive circuits. However, TCP congestion control would apply on all circuits equally and would result in extended queueing times for data cells in TCP output buffers and thereby, longer delays observed by clients.

Tor's Queuing Architecture Tor uses a tiered buffer architecture to manage cells traveling through circuits, as also shown in Figure 1. When an OR receives a cell from an external server or from another OR or OP, the cell is passed from the kernel TCP receive buffer to a corresponding 32 KiB connection-level input buffer in Tor. After the cell is encrypted or decrypted, it is placed on the appropriate FIFO circuit queue. Since several circuits share the same connection output buffer, a scheduler is used to retrieve cells from the circuit queues to be placed on a 32 KiB output buffer. Finally, the cells are sent to the kernel TCP send buffer which flushes them to the next OR or OP.

2.2 IPsec

IP security (IPsec) [22] is a collection of standards that provides security at the network (IP) layer. It defines several protocols that enable authenticating and/or encrypting IP data packets. It consists of mainly two sub-protocols: *Authentication Header* (AH) and *Encapsulating Security Payload* (ESP). We next briefly describe each sub-protocol and their modes of operation.

The AH protocol allows two communicating points to authenticate, and protect the integrity of the data they exchange. Although the AH protocol guards against spoofing and replay attacks, it does not encrypt the data traveling between the two ends, so an eavesdropper can view the contents of the data packets.

The ESP protocol, on the other hand, enables both authentication and encryption, which provides confidentiality of the transferred data. The two communicating ends need to have secret keys to decrypt the packets. IPsec provides a variety of key-exchange and authentication algorithms.

For both protocols, there are two modes of IPsec operation: either the *transport* or the *tunnel* mode. Transport mode is used to secure the connection, consisting of the traffic from different applications, between two hosts. The payload of the IP packet, which typically contains TCP or UDP data, is encrypted or authenticated and an ESP or an AH header is added to the packet. The original IP header also remains in the packet.

Tunnel mode, on the other hand, secures not only host-to-host communication, but it also can be used to protect communication between subnets to subnets or hosts to subnets. In this mode, the whole IP packet is encrypted or authenticated and a new IP header is added to the encrypted packet in addition to the AH or ESP header. Using ESP in tunnel mode provides the strongest security for communication at the expense of a few extra bytes per packet as an overhead. However, when only host-to-host communication is required, ESP protocol in transport mode suffices.

In the next section, we present previous work on anonymous communication transport design for Tor. After that, we introduce our proposed anonymous communication transport for Tor and how we use IPsec to secure communication between Tor ORs.

3. RELATED WORK

Since Tor was introduced around a decade ago, it has received a great amount of attention. Several aspects of Tor's design have been intensively investigated including Tor's routing [4,33,35], scalability [25,26] and enhancing its awareness and handling of congestion [6,17,21,37,43]. There are also several proposals that aim to increase the total number of ORs using incentive schemes [20,27,29].

New transport designs for Tor have also been investigated and considered by several previous proposals [31,40,42]; Murdoch [28] provides a summary and compares all these previous possible transport designs. He categorizes the available designs into three different architectures: hop-by-hop reliability, initiator-to-exit reliability or initiator-to-server reliability. Although Murdoch does not experimentally evaluate these design choices, he expects that a hop-by-hop reliability approach will be the most promising approach. Next, we summarize the first two design categories and contrast them with our design. For more details on the initiator-to-server design architecture, we refer the reader to Freedom [9] and Murdoch's summary [28].

TCP-over-DTLS is an example of the hop-by-hop reliability design, which is also the same design approach we adopt in PCTCP. The TCP-over-DTLS proposal advocates for using a user-level TCP connection to manage every user circuit over DTLS—the datagram alternative to TLS—to provide confidentiality and authenticity of Tor's traffic. Since every circuit is managed by its own TCP connection, every circuit is guaranteed reliability and in-order delivery of cells. Furthermore, congestion control is performed at the circuit level, which solves the cross-circuit interference problem. Several differences separate PCTCP from TCP-over-DTLS. First, PCTCP uses mature IPsec protocols to hide TCP/IP header information, whereas TCP-over-DTLS uses the relatively rare DTLS for the same purpose. Also, TCP-over-DTLS introduces deployment and performance issues that hinder its adoption (as highlighted in Section 1). PCTCP avoids these problems by using the kernel-level TCP stack, and by having an easier path to deployment. Second, while initial experiments performed on a localhost private Tor network showed slightly less degraded latency results, as compared to Tor, when packet drop rates increased, there is still a need for further realistic large-scale experiments in order to obtain conclusive results of the potential benefits.[2]

UDP-OR [42] is an example of an initiator-to-exit reliability design. In this design, an OP and the exit OR of the circuit maintain a TCP connection, while intermediate ORs communicate using UDP, an unreliable transport protocol. While this design significantly simplifies the operations of the intermediate routers, it still suffers from several problems. The first problem is that since hop-by-hop communication is unreliable, there will be a need to change the cryptographic protocols that are implemented in Tor as the current circuit encryption scheme depends on in-order delivery of cells. Another problem is that this design uses the OP's host TCP stack, rather than a user-level one, which opens the door for OS fingerprinting attacks [23] in which the exit node can learn information about the client. Second, since a circuit's round trip time is large, it would take the TCP endpoints a significant amount of time before congestion is triggered. Also, with the high variability of circuit performance in Tor, a non-trivial amount of tuning for TCP parameters, including congestion timers, thresholds and windows, may be required for the TCP endpoints; see section 4 for more details.

Torchestra [17] was recently proposed to enhance the performance of interactive application users of Tor. In that proposal, two TCP connections are used for OR-to-OR communication. One TCP connection is dedicated for light circuits and another is dedicated for heavy circuits. An Exponentially Weighted Moving Average (EWMA) algorithm of the number of cells sent on a circuit, originally proposed by Tang and Goldberg [37], is used to classify circuits into light and heavy categories. Previous work [5] suggested that this metric alone is not enough to distinguish circuits.[3] Also, Torchestra has not been examined using large-scale experimentations to understand the system-level effects of utilizing it. Finally, to benefit from Torchestra, all ORs on the circuits need to upgrade, as two TCP connections, as well as a new command cell type, are needed between every pair of ORs in a circuit.

Tschorsch *et al.* [40] consider the impact of several proposed transport designs for Tor on throughput, packet loss, delay and fairness. For their analysis, the authors use a TCP performance model proposed by Padhye *et al.* [30]. They examine the performance of several proposed transport designs for Tor using a discrete-event simulator, and conclude that they expect that a joint congestion control that detects loss rates and congestion for all circuits traversing an overlay node would be a good direction. The authors ruled out the use of parallel TCP connections, such as in PCTCP, as a design option, as more connections traversing a bottleneck may result in higher packet losses, which reduces throughput. We argue that packet losses mainly occur for the connections carrying bulk traffic, as they send significantly more data than connections carrying interactive applications. We also demonstrate through comprehensive emulation and live-network experiments that our approach is effective.

4. PROPOSED TRANSPORT

Before embarking on the description of PCTCP, we first ask ourselves, why not adopt and implement an end-to-end TCP approach, which has been proposed as a possible transport design for Tor. We start by explaining why we avoided such an approach, and then we elaborate on our design.

4.1 Why not end-to-end TCP?

One transport design that has received some positive speculation in the Tor research community is the end-to-end TCP design. This design is inspired by many previous proposals [9,11,42]. The basic idea of this design is that a TCP connection is maintained by the two ends of the circuit. In the context of Tor, one end is the client and the other end can be the exit OR or the destination server. Communication between intermediate ORs is carried out using a datagram protocol, such as UDP. We next point out some weaknesses in this design choice.

Tuning Parameters TCP is a reliable transport. If a packet gets dropped or lost due to congestion or routing problems in the underlying IP network, TCP's congestion control algorithm is triggered and the sender retransmits the lost packet. Also, TCP ensures that the Tor process, residing at the application layer, receives data in the order they were sent. This functionality significantly simplifies the task of data processing for Tor. By contrast, a datagram protocol like UDP, or its secure DTLS alternative, do not implement reliability or in-order delivery.

[2] One difficulty is that TCP-over-DTLS is implemented in a 5-year-old version of Tor (0.2.0.25). Since Tor's data structures, queuing and networking have changed over time, a direct comparison between PCTCP and TCP-over-DTLS is meaningless. Also, Reardon *et al.* reported that they found many bugs in both Daytona and OpenSSL's DTLS implementation, which affected their results.

[3] Unfortunately, the classification accuracy was not discussed in Torchestra [17].

In the end-to-end TCP design for Tor, it is assumed that reliable in-order delivery is maintained only by the end points. There are several shortcomings with this design that might worsen the experience of Tor users. The biggest challenge is how to best tune the TCP parameters to yield a reasonable performance for Tor. TCP relies on duplicate acknowledgement packets sent by the receiver to detect congestion which signals that several out of order packets have been received at the destination. Moreover, TCP also relies on retransmission timers at the sender to detect loss of packets.

Typically, retransmission timers should be equal to the round-trip-time (RTT) between a source and a destination. In a network like Tor, where the RTT of circuits can be several seconds long, it can be easily seen that a client would detect congestion very late. Of course, the client can set a smaller retransmission timer to detect congestion faster; however, one should be careful not to send redundant packets too quickly, as this might cause even further congestion. Striking a good balance between how fast we want to detect congestion and how careful we should be before we decide we are experiencing congestion is a very difficult problem. Also, considering the timing characteristics of Tor circuits, which are notorious for their highly variable performance, one soon realizes that an end-to-end TCP solution for Tor is unwise.

Interoperability and Anonymity. An important aspect of any new transport design for Tor is to ensure that it can be smoothly integrated to work with the existing Tor network infrastructure without disrupting the operation of the network and its users. Recall that Tor today currently has thousands of ORs and hundreds of thousands of users. The network has not experienced significant downtime since its deployment in 2003. Using a drastically different transport design such as end-to-end TCP would require the network to pause its operation while ORs and users update. As a workaround, it might be possible for ORs upgraded with end-to-end TCP to coexist with unmodified ORs; however, this might open the door for fingerprinting or partitioning attacks. For example, an upgraded malicious exit can reduce the anonymity set of the entry guard used on a circuit from the set of all entry guards in the network to the smaller set of upgraded entry guards. Therefore, one shortcoming of upgrading to an end-to-end TCP design is possibly hindering the anonymity provided by the network.

Cryptographic Protocols. An inherent consequence of allowing an unreliable transport is for the Tor process to expect lost packets. Since Tor uses the Advanced Encryption Standard (AES) in counter mode for encrypting and decrypting cells at ORs, lost or dropped cells will cause subsequent cells to be unrecognized. Therefore, adopting an end-to-end TCP approach requires changing the cryptographic protocols that are currently used in Tor; this is another obstacle facing such a design.

4.2 PCTCP

The aim of this work is to address the shortcomings of the transport design in Tor. In particular, our goal is to reduce the impact of the cross-circuit interference problem which hinders the experience of interactive application users. Based on our discussion in section 4.1, we believe that reliability should be maintained on a per-hop basis for Tor circuits. Therefore, in this work, we advocate for maintaining TCP connections between each adjacent pair of ORs that comprise a circuit. In particular, we propose two key design changes to Tor's transport.

4.2.1 *Kernel-mode per-circuit TCP*

We propose using a separate kernel-mode TCP connection for each circuit for Tor. Our design is similar to the TCP-over-DTLS design that was introduced by Reardon and Goldberg in the sense that reliable in-order delivery of data is implemented between every two communicating ORs. Also, both designs ensure that congestion control is performed at the circuit granularity. The elimination of connection-sharing among circuits ensures that we isolate the effects of loud circuits on the quiet ones; a cell dropped or lost from one circuit will only affect that particular circuit.

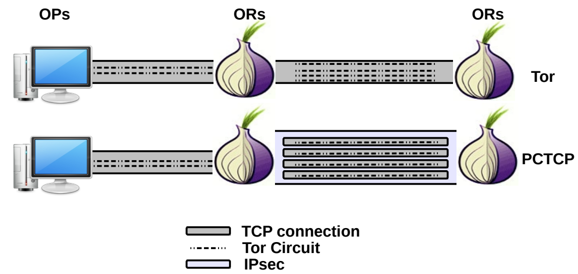


Figure 2: Design comparison between Tor and PCTCP.

However, one key difference between PCTCP and TCP-over-DTLS is that for circuit management, PCTCP uses kernel-mode TCP connections for every circuit, while TCP-over-DTLS uses a user-space TCP implementation. The lack of availability of a reliable open-source user-level TCP stack whose license is compatible with that of Tor hinders the deployability of the TCP-over-DTLS solution. Furthermore, PCTCP uses IPsec to protect the communication between ORs whereas TCP-over-DTLS uses DTLS. One issue that is inherent from using DTLS is that it is rarely used today on the Internet. IPsec, on the other hand, is increasingly common, as it is utilized in many implementations of Virtual Private Network (VPNs) [34]. Consequently, the rarity of DTLS makes it easier to be blocked by censors without fearing side effects. Blocking IPsec would be more problematic, as blocking it may interrupt the operation of legitimate businesses and organizations.

We next describe how we modify the behaviour of Tor to support PCTCP. Recall that during the circuit construction process, every time an OP attempts to extend the circuit by one more hop, it sends an *extend* command cell to the current last OR on the partially constructed circuit. When an OR X_i receives an *extend* cell to another OR X_j, X_i checks if it has a current TCP connection with X_j. If a connection exists, X_i uses that connection to send the *create* cell; otherwise, it creates a new TCP connection to X_j before a *create* cell is sent.

In PCTCP, when an OR X_i receives an *extend* command cell to X_j, PCTCP always establishes a new TCP connection from X_i to X_j. In PCTCP, we maintain the same queueing architecture of Tor, except that our design eliminates the contention that occurs among circuits when they share the same connection output buffer, as each circuit queue is mapped to a single output and a single input connection buffer. When a circuit is torn down, its corresponding TCP connections are closed.

Figure 2 visualizes a design comparison between Tor and PCTCP. As the figure shows, between an OP and an OR, PCTCP, like Tor, maintains a single TCP connection, which can multiplex several circuits from the same user. However, PCTCP dedicates a separate TCP connection for each circuit between any two ORs.

This design has the advantage that it does not require clients to upgrade, as each client in our design continues to maintain a single TCP connection with each of its entry guards. Moreover, the modifications proposed in PCTCP are only local to each OR. This means that not all ORs in the circuit need to upgrade to benefit from PCTCP. For example, if the middle and exit ORs are the only

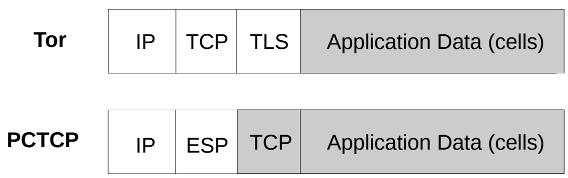


Figure 3: Packet headers for current Tor and for PCTCP. The grey shaded area depicts the encrypted part of the packet. The upper figure shows the design of the Tor packets at the network (IP) layer. TLS is used to encrypt the TCP payload, but not the TCP header. The lower figure depicts the packet format when PCTCP is used. The whole IP payload, which contains the TCP segment, is encrypted. An ESP header is added between the encrypted data and the IP header.

ORs upgraded with PCTCP on a circuit, that pair of ORs will use PCTCP for their communication even if the entry guard is not upgraded. Nevertheless, more performance gains should be obtained when more ORs on the circuit upgrade.

4.2.2 Replace TLS with IPsec

One issue that arises with our design so far is that it allows an adversary monitoring a relay to easily count the total number of circuits that are currently serviced by the monitored relay. Furthermore, the adversary can perform traffic analysis to infer the activity of each circuit [5]. While it is not clear how this extra information can be beneficial for a non-global adversary,[4] there is no doubt that such a design reduces the overall anonymity of the system and its users. To alleviate this problem, we propose using the ESP protocol of IPsec in transport mode to encrypt and protect the traffic between the ORs using PCTCP. Since IPsec can encrypt the IP packet payload, TCP connection ports will be encrypted and hidden from an eavesdropper. This makes it more difficult for an adversary to perform traffic analysis on TCP connections between routers. Figure 3 compares the format of PCTCP and Tor data packet headers.

Using ESP makes the TLS encryption redundant for PCTCP for OR-to-OR communication, as ESP can provide the hop-by-hop authenticity and data confidentiality that is currently provided by TLS in Tor. Furthermore, like TLS, ESP provides perfect forward secrecy for the data on connections, and prevents an attacker from modifying data. For two ORs to authenticate each other, they can use a certificate-based authentication method that is provided by IPsec. Since ORs issue a long-term identity key that they use to sign their descriptors, they can use the same identity key to sign their IPsec certificates.

Alternatively, ORs can use a public-key authentication approach. An OR could publish its IPsec public key with its signed descriptor to the directory authorities. Then, when other ORs download the descriptors, they can find each other's public keys and use them to start the IPsec connections. Communication between ORs and directory authorities or OPs can continue to use the traditional TLS connections that are used in Tor today.

Ideally, a user-mode IPsec implementation integrated with Tor would be the best option. First, OR operators would not have to deal with the details of setting up IPsec. Second, for user-mode IPsec to operate, superuser privileges are not needed. However,

[4]The threat model of Tor assumes an active local adversary.

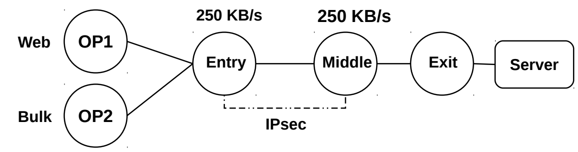


Figure 4: Network setup for the live experiment

with the lack of an available user-space IPsec implementation, we default to the kernel-mode IPsec option. Luckily, installing IPsec is a one-time operation which typically should not require periodic maintenance. To facilitate this operation for non-expert OR operators, Tor should ship with scripts for IPsec configurations.

5. EXPERIMENTS

To evaluate the performance benefits possible with PCTCP, we have implemented our proposed transport in a stable release of the Tor source code (0.2.2.39). Our implementation, which changes fewer than 20 lines of code in the Tor OR application, can be easily turned on or off using a configuration option for any OR. We first performed small-scale experiments on the live Tor network. We also performed a series of large-scale experiments on an isolated testbed using different traffic loads. As evaluation metrics, we use the *download time*, the time needed for a client to finish downloading a file over a Tor circuit after issuing a request, and the *time-to-first-byte*, which is the time it takes the client to receive the first chunk of the file data after issuing a download request.

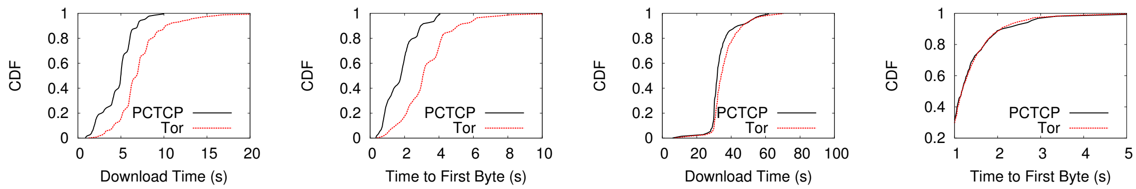
5.1 Live Experiments

To test our new proposed design, we first conducted some experiments on the live Tor network in October and November 2012. We next describe our experimental setup and then present our results.

Experimental Setup. Our setup is shown in Figure 4.We configured an IPsec connection, using OpenSwan [3], between our two ORs, entry and middle, which we deployed on the live Tor network. Our entry implements PCTCP which can be enabled as a configuration option only for our clients, so as not to affect other users of the network. Our middle OR runs an unmodified Tor process, but, as above, has an IPsec connection configured. For gathering Tor measurements, we simply turned off the option to enable PCTCP from the configuration of the entry and disabled the IPsec connection. Both ORs have been configured with a bandwidth rate of 250 KB/s. Our ORs obtained the FAST flag by the authority directories, which allows them to be selected by the network clients for their circuits.[5] To protect the privacy of other users, we configure both ORs to belong to the same *Tor family*, which prevents other users' unmodified Tor clients from choosing them both on one circuit. Also, we do not disable TLS in order to avoid risking other users' privacy in case of an accidental misconfiguration. We next describe our two experiments and present our results.

In our live experiment, we run two clients. One client acts as the bulk traffic generator by continuously downloading a 5 MB file without pausing between downloads. The second client is an interactive web browsing client that downloads a 300 KB file and pauses randomly for 3 to 30 seconds between downloads. We have also

[5]Our ORs did not achieve the STABLE flag because they were not running continuously between our experiments. Note that the STABLE status is mainly used for services that need long lived connections (such as SSH on port 22), whereas a FAST status, which our ORs obtained, is requird for most services like web browsing, which comprises the majority of the network traffic.



(a) Web client (b) Web client (c) Bulk client (d) Bulk client

Figure 5: Performance of the web and bulk clients in the live experiment

implemented the *MeasureMe* [5] cell. Briefly, this is a new command cell type that is sent by our clients to any OR on a circuit they create to inform the OR to gather statistics only for the respective circuit.

Results. Figure 5(a) depicts the download time performance for Tor and PCTCP for the web client.[6] With PCTCP, it takes 4.9 seconds to finish downloading, while Tor takes 6.8 seconds at the median. The improvements become more visible for the 4th quartile, as download times show a 26% improvement when PCTCP is used. Figure 5(b) shows the time-to-first-byte results for PCTCP and Tor. Again, the results consistently show strong improvements that are magnified at the third and 4th quartiles. For instance, at the 75th percentile, the time-to-first-byte for Tor clients is approximately 4 seconds, whereas for PCTCP clients, it is only 2.1 seconds, which is a more than 47% improvement.

Finally, Figure 5(c) demonstrates that the PCTCP bulk client exhibited slightly better performance than the Tor bulk client. Note that in this experiment, the introduction of the bulk downloader consumes the majority of the available bandwidth between entry and middle. Nevertheless, PCTCP still maintains the performance advantage for web clients compared to Tor. In Figure 5(d), both PCTCP and Tor produced very similar fast time-to-first-byte results as the light web traffic did not introduce congestion to the bulk client.

5.2 Large-scale experiments

Emulation Tools. In order to understand the system-level effects of our proposed transport, we use ExperimenTor [8], a Tor network emulation-based testbed that is based on the Modelnet network emulation platform [41]. Modelnet offers the ability to evaluate large-scale distributed networked systems using commodity hardware and OSes. Briefly, our Modelnet setup consists of two machines, an emulator node and a virtual node. The virtual node runs the Tor network, which consists of directory authorities, ORs and OPs. The virtual node also runs the destination servers. Communication among the different nodes on the Tor network and the destination servers is routed through the emulation node, which provides the underlying IP network emulation. Several network parameters such as the bandwidth, propagation delay and drop rate can by configured on the network topology deployed on the emulator node to provide a realistic underlying network emulation. In our experiments, we use the network and Tor topology models that were recently proposed by Jansen *et al.* [18] in order to accurately produce a scaled-down Tor network that that closely approximates the performance of the live network.

Configuring IPsec. One challenging task in our experiments is to enable an IPsec connection between any two ORs in the network. We found that the IPsec implementation we used (Linux Openswan U2.6.23/K2.6.38 (klips)) does not start the IPsec tunnel between two virtual interfaces that reside in the same machine. To overcome this problem, we introduced an intermediary node between our virtual and emulator nodes. We set up an IPsec tunnel between the virtual node and the intermediary, whose only purpose is to receive packets on the IPsec tunnel and forward them to the emulator. The emulator performs the network emulation and then forwards the packets back to their destinations in the virtual node. That way, a packet between any two ORs is forced to go through an IPsec connection. For stock Tor performance experiments, we disabled the IPsec tunnel between the virtual and the intermediary nodes.

Underlying Network Topology. We use the network topology that was produced and published[7] by Jansen *et al.* in an effort to methodically model the Tor network for ExperimenTor and Shadow [19]. Briefly, the authors form a complete network graph consisting of vertices that correspond to different locations (countries, American states and Canadian provinces) with upstream, downstream and packet loss properties that they obtained from the Ookla Net Index dataset [2]. All vertices are connected by edges with approximated latency,[8] jitter, and packet loss properties.

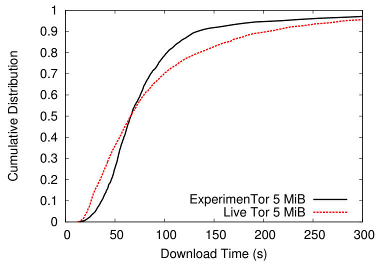
Overlay Tor Topology. We follow the footsteps of Jansen *et al.* and create a scaled-down topology that consists of 500 Tor clients (OPs), 50 Tor ORs, and 50 HTTP servers. Of the 50 ORs, 5 work as directory authorities. Our ORs are assigned bandwidth values that are sampled from the bandwidth distribution of the live Tor network ORs. We create two client types: web clients and bulk clients. Our client model is based on a previous study of the exit Tor traffic by McCoy *et al.* [24]. The study found that 93% of connections that exited the Tor network are HTTP connections which consumed approximately 60% of traffic volume. They also found that file sharing applications consumed approximately 40% of the bandwidth in Tor. During our experiments, our web clients continuously fetch fixed-sized 320 KiB files, and pause randomly for 1 to 30 seconds between fetches. Our bulk clients continuously download 5 MiB files without pausing. Finally, we first use a web-to-bulk client ratio of 19:1, as recommended by Jansen *et al.*(Figures 7(a)–7(d)). In addition, we also repeated our experiments on the same network topology where we lower the web-to-bulk client ratio to 9:1 in order to test PCTCP with different traffic loads and with increased congestion (Figures 7(e)–7(h)).

Model Accuracy. Before we present our results, we first compare the performance of our stock Tor bulk and web clients, which we obtained from our testbed, to the performance of the live Tor network published by the Tor metrics portal [38]. This comparison step was also carried out by Jansen *et al.* The purpose of this step is to confirm that our testbed measurements can indeed approxi-

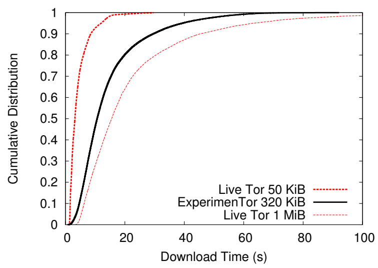
[6] Note that the stair-step pattern is a consequence of Tor's token bucket algorithm which flushes data once per second. This pattern becomes more visible with increased congestion. In versions of Tor more recent than the stable version we used, this flushing has been increased to ten times per second.

[7] The model files are available for download from the authors' websites (`http://www.mit.edu/~ke23793/misc/tormodel_exptor.tar.gz`).

[8] The authors use iPlane [1] RTTs to approximate latency.



(a) Large files



(b) Small files

Figure 6: Comparison between the performance of torperf (Live Tor), and our scaled-down testbed Tor network (ExperimenTor)

mate the measurements taken from the live network, even though our network is significantly scaled down.

Figure 6(a) compares the distribution of the download times of our testbed bulk downloaders and those measured by torperf, a tool that measures download performance on the live Tor network. The two distributions display comparable performance and they indeed intersect at the median. That is, 50% of the 5 MiB downloads take 65 seconds or less on the live network, and the same is true on our testbed. Figure 6(b) compares the results of our 320 KiB downloads and torperf's 50 KiB, and 1 MiB downloads.[9] As expected, the distribution of download times for our web clients fits between the distributions of download times between torperf's 1 MiB and 50 KiB file downloads.

Results. Now that we have verified that our Tor model closely approximates the performance of the Tor network, we next shift attention to our results.[10] Figure 7(a) compares the download time observed by web clients when stock Tor and PCTCP are used. The figure shows significant improvement for the slowest 50% of the downloads, especially for the 4th quartile of the download times. For example, the download times for Tor range from 17–90 seconds, whereas for PCTCP, the download times range from 14–56 seconds.

In Figure 7(b), significant time-to-first-byte improvements can be observed when PCTCP is used, as compared to Tor. At the median, it takes Tor clients 3.6 seconds before the browser starts changing for them, whereas PCTCP clients only wait for 1.6 seconds, which is a 55% improvement. For the 75th percentile response times, the time-to-first-byte is only 2 seconds for PCTCP users, whereas Tor clients experience delays of up to 6 seconds. This increases the observed improvements to 66%.

[9] Torperf only maintains the results of 5 MiB, 1 MiB and 50 KiB file downloads.

[10] For simplicity, we have not disabled TLS in PCTCP experiments even though it is not needed in our design due to the use of IPsec.

We observe in Figure 7(c) that download times for bulk clients are slightly improved when PCTCP is used. The improvement is roughly 20% for 80% of the requests. For example, the median download time for stock Tor is 65 seconds, whereas for PCTCP, the median download time is approximately 51 seconds. Also, the time-to-first-byte results are significantly improved for the bulk downloaders, as can be seen in Figure 7(d). This suggests that congestion is vastly reduced in the network.

Under heavier traffic loads, the amount of available bandwidth in the network decreases, which affects the benefits we observe for the download time. The download time comparison between PCTCP and Tor for web and bulk clients when the ratio of web-to-bulk clients is 9:1, depicted in Figures 7(e) and 7(g), shows that PCTCP improves the long tail of the distribution for the web clients by approximately 20%. The reason for the improvement is that, despite the lack of spare bandwidth, PCTCP allows each circuit at the transport layer to get its fair share of the bandwidth and forces the bulk downloads present in the system to back off whenever they attempt to get more than their allocated bandwidth, as evidenced by the degradation of the bulk client performance shown in Figure 7(g). With PCTCP, heavy circuits might observe more delays because such circuits are expected to drop more cells, and their respective TCP connections would back off more frequently as a result of the separate TCP congestion control. However, performance improvements can be observed even for bulk clients if more bandwidth was available. For example, we have observed significant improvements for both web and bulk clients in the higher-bandwidth experiments we report in Appendix A.

Figures 7(f) and 7(h) show the significant time-to-first-byte improvements for both the web clients and the bulk downloaders. The improvements at the 75th percentile are more than 60% for both the web and bulk clients.

Summary of results. Based on our observations, we conclude that PCTCP produces significant performance benefits that can certainly be perceived by clients. The download time improvements depend on the amount of available bandwidth in the network. If the network has spare bandwidth to offer, PCTCP will improve the experience of all users in the system. When the network operates at its capacity, web clients will notice download time enhancements, while bulk clients will observe degraded performance. However, in all experimental scenarios, PCTCP significantly improves the response times in the network for all clients.

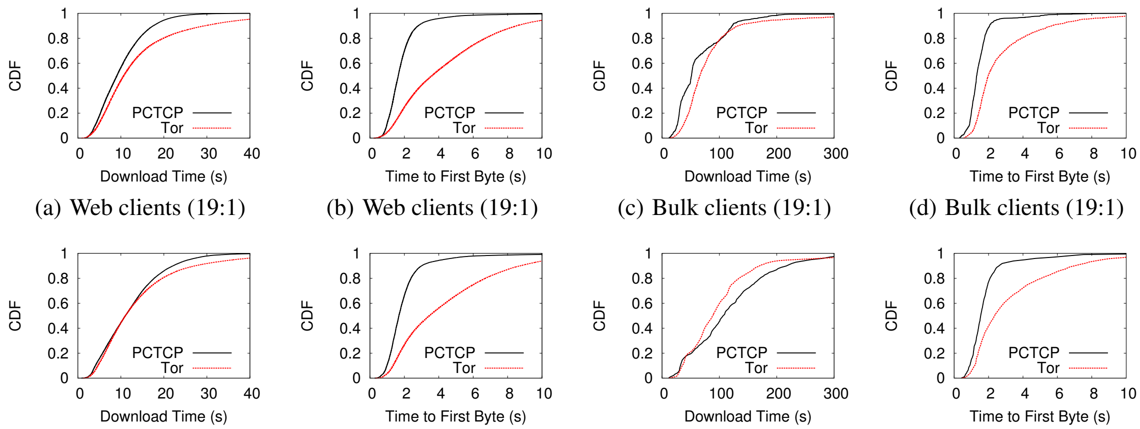
To maximize the benefits of using PCTCP, it should be used in combination with previous proposals that aim to increase the amount of available bandwidth in the network, such as traffic classification [5], throttling approaches [21,27] or approaches aimed to incentivize clients to run ORs [20,29].

6. DISCUSSION

We next discuss a variety of open issues regarding PCTCP.

6.1 Anonymity Implications

Since our transport proposal is designed for Tor, an anonymity network, it is essential to consider the anonymity implications of our design. In particular, it is important to ensure that our new design does not add new vulnerabilities to the Tor network. Recall that the anonymity of a circuit is compromised in Tor if its two ends, the entry and exit, are compromised. Therefore, one issue to consider is whether using PCTCP can reduce the anonymity set of the ORs used in a circuit. For example, can an exit OR reduce



(a) Web clients (19:1) (b) Web clients (19:1) (c) Bulk clients (19:1) (d) Bulk clients (19:1)

(e) Web clients (9:1) (f) Web clients (9:1) (g) Bulk clients (9:1) (h) Bulk clients (9:1)

Figure 7: Performance of all clients in the large-scale experiment under different traffic loads. Figures 7(a)–7(d) show the performance of all clients when the web-to-bulk client ratio is 19:1, whereas Figures 7(e)–7(h) show the performance when we increase congestion by setting the web-to-bulk client ratio to 9:1.

the anonymity set of the entry OR used on a circuit because of PCTCP?[11]

First, with exception of the IPsec connections, the changes that are imposed by PCTCP on any OR are local. That is, our design does not introduce a new cell type or require other ORs on the circuit to upgrade. If an entry OR uses PCTCP, then only the middle OR will notice because the middle has to agree to establish the IPsec connection with entry and because it receives more than one TCP connection from the upgraded entry. Those changes do not affect the exit OR in the circuit; therefore, the exit would not be able to know if entry belongs to the set of upgraded ORs or not. Even if the exit learns from router descriptors that middle is an upgraded OR, the exit would still not be able to know if entry is upgraded or not.

Furthermore, one might wonder if dedicating separate TCP connections might open the door to timing attacks. First, a connection between the OP and the OR is very similar for Tor and PCTCP. Second, because the communication between ORs is protected using IPsec, it would be difficult for the adversary to extract specific circuit information even though each circuit uses a separate TCP connection. Therefore, PCTCP does not introduce any new threats to the Tor network.

6.2 Incremental Deployment

One advantage of PCTCP is that it is incrementally deployable in two steps. The first step towards deployment is enabling IPsec communication among ORs. Basically, ORs need to advertise in their descriptors that they are willing to accept IPsec connections. Then, IPsec-enabled ORs can try to establish IPsec connections proactively among each other. When OR1 wishes to use PCTCP with OR2, it can check if it has an existing IPsec connection with OR2,[12] in which case OR1 can proceed with using PCTCP. If OR1 detects no IPsec connection with OR2, it uses the default Tor TLS connection with OR2 and multiplexes the circuits in the same connection.

[11] It is important to prevent the two ends of the circuit from learning about each other to prevent active or legal attacks.

[12] For Openswan, the visibility of IPsec for an application can be established using the libwhack API.

6.3 Experimental Limitations

To be able to faithfully test and evaluate our new transport proposal, we ran a series of testbed experiments on different network topologies using different traffic models and loads. Regardless of our efforts, we recognize that our large-scale experiments were conducted on an isolated experimental testbed. We were unable to experiment with larger topologies because we are limited by our CPU, bandwidth and memory resources.

However, to ensure that we report accurate results, we followed the methodology of Jansen *et al.* [18] to produce an accurate model of the Tor network. We also used their published topology files in order to avoid biased results that might be obtained using a different experimental setup. Finally, we carried out additional experiments on the live Tor network to confirm our results.

6.4 IPsec through NATs

One challenge that IPsec faced in the past is its inability to connect to hosts behind NATs. As a result, NAT-Traversal [36] (NAT-T) has evolved to address this problem. NAT-T can be used when two hosts detect that if they are behind a NAT. In the context of Tor, this problem is currently irrelevant as most Tor ORs are publicly reachable; however, there are some efforts to enable the operation of ORs from behind NATs [7]. In this case, IPsec can still benefit from NAT-T.

6.5 File Descriptor and Memory Usage

One issue to consider is how this work affects the very busy routers on the live network. Since Tor uses a weighted-bandwidth OR selection algorithm where ORs are selected in proportion to their bandwidth, some high-bandwidth ORs service thousands of circuits at the same time. This means that, with PCTCP, such routers are expected to maintain thousands of file descriptors at the same time. One might wonder if such a requirement might raise memory usage concerns due to the TCP buffer space allocated in the kernel for each file descriptor.

To get an idea of how many file descriptors would be needed when PCTCP is used, we examined a fast exit OR on the live network configured with a bandwidth of 100 Mb/s, which puts it among the fastest 6% of the network routers. This fast exit OR used roughly 10,000 file descriptors for its communication with other

ORs and with destination servers. Since an exit OR uses one file descriptor for each *stream* within a circuit, the number of circuits it is handling is certainly less than the number of file descriptors it is using. Note that intermediate routers are currently expected to use a number of file descriptors that is equal to the number of ORs in the network, which is approximately 3000. We therefore expect that other intermediate ORs, such as middles or entries that have the same bandwidth capabilities as the fast exit, to use between 3000 and 10,000 file descriptors if they use PCTCP. In short, file descriptor and memory usage should not be a problem with PCTCP, as even the busiest entry and middle ORs running PCTCP should consume fewer of these resources than the existing Tor network requires exit ORs to support today.

However, one possible way to reduce the number of file descriptors is to use a threshold algorithm. For instance, every OR can use PCTCP up to a certain threshold of the number of circuits being serviced, which can be configured by the relay operator. If the number of circuits serviced exceeds the threshold, the OR can multiplex new incoming circuits in existing connections in a round-robin manner. Even better, an OR can use a classification approach similar to Torchestra [17], where the OR avoids multiplexing loud circuits with the quiet ones. In short, we do not believe memory management issues are detrimental to PCTCP, as several solutions exist to address them.

6.6 Future Work

One important area for future investigation is to implement other transport proposals such as TCP-over-DTLS and UDP-OR, in order to compare their performance to that of PCTCP in large-scale network emulation. Tor's forthcoming transport abstraction layer [32] should greatly facilitate this task.

Another area for future work is to consider an alternative queueing design for Tor that reduces the number of times cells are copied. Indeed, our design eliminates the need for circuit queues as every input buffer corresponds to single output buffer, which means that data can be copied immediately from the input buffer to the output buffer after being encrypted or decrypted.

7. CONCLUSION

In this work, we recognize the importance of the Tor network as a privacy-preserving tool online and seek to enhance its performance for interactive application users. To this end, we propose PCTCP, a new anonymous communication transport design for Tor which allows every circuit to use a separate kernel-level TCP connection protected by IPsec. Our design is easily deployable and requires minimal changes to routers. PCTCP avoids the deployability and performance problems that were introduced by TCP-over-DTLS due to the use of a user-level TCP stack. Furthermore, experimental evaluation of PCTCP shows vast improvement gains, while maintaining the threat model of the Tor network. Our realistic large-scale experiments show that it is possible to obtain improvements of more than 60% for response times and approximately 30% for download times when PCTCP is used, as compared to Tor.

Acknowledgements.

We thank the anonymous reviewers for their helpful comments and suggestions. We thank NSERC, ORF, and The Tor Project, Inc. for funding this research.

8. REFERENCES

[1] iPlane: Data. `http://iplane.cs.washington.edu/data/data.html`. Accessed Feb. 2013.

[2] Net Index Dataset. `http://www.netindex.com/source-data/`. Accessed Feb. 2013.

[3] OpenSwan. `https://www.openswan.org/projects/openswan/`. Accessed Feb. 2013.

[4] M. Akhoondi, C. Yu, and H. V. Madhyastha. LASTor: A Low-Latency AS-Aware Tor Client. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 476–490, Washington, DC, USA, 2012. IEEE Computer Society.

[5] M. AlSabah, K. Bauer, and I. Goldberg. Enhancing Tor's performance using real-time traffic classification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 73–84. ACM, 2012.

[6] M. AlSabah, K. Bauer, I. Goldberg, D. Grunwald, D. McCoy, S. Savage, and G. M. Voelker. DefenestraTor: Throwing out Windows in Tor. In *11th Privacy Enhancing Technologies Symposium*, pages 134–154, July 2011.

[7] J. Appelbaum. Tor and NAT devices increasing bridge & relay reachability or enabling the use of NATâĂŞPMP and UPnP by defaults. `https://trac.torproject.org/projects/tor/attachment/ticket/4960/tor-nat-plan.pdf`, August 2012. Accessed Feb. 2013.

[8] K. Bauer, M. Sherr, D. McCoy, 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)*, pages 51–59, August 2011.

[9] P. Boucher, A. Shostack, and I. Goldberg. Freedom Systems 2.0 Architecture. White paper, Zero Knowledge Systems, Inc., December 2000.

[10] T. Braun, C. Diot, A. Hoglander, and V. Roca. An Experimental User Level Implementation of TCP. Technical Report RR-2650, INRIA, Sept. 1995.

[11] Z. Brown. Pragmatic IP Anonymity. `http://www.cypherspace.org/cebolla/cebolla.pdf`, June 2002. Accessed Feb. 2013.

[12] T. Dierks and E. Rescorla. RFC 5246âĂŤThe Transport Layer Security (TLS) Protocol Version 1.2. `http://www.ietf.org/rfc/rfc5246.txt`, August 2008.

[13] R. Dingledine. Tor and Circumvention: Lessons Learned. In *Proceedings of the 31st Annual Conference on Advances in Cryptology (CRYPTO)*, pages 485–486, August 2011.

[14] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, August 2004.

[15] R. Dingledine and S. Murdoch. Performance Improvements on Tor or, Why Tor is Slow and What We're Going to Do about It. `http://www.torproject.org/press/presskit/2009-03-11-performance.pdf`, March 2009.

[16] A. Edwards and S. Muir. Experiences implementing a high performance TCP in user-space. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '95, pages 196–205. ACM, 1995.

[17] D. Gopal and N. Heninger. Torchestra: Reducing Interactive Traffic Delays over Tor. In *Proceedings of the 2012 ACM*

Workshop on Privacy in the Electronic Society (WPES 2012), pages 31–42. ACM, 2012.

[18] R. Jansen, K. Bauer, N. Hopper, and R. Dingledine. Methodically Modeling the Tor Network. In *Proceedings of the USENIX Workshop on Cyber Security Experimentation and Test (CSET 2012)*, August 2012.

[19] R. Jansen and N. Hopper. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In *Proceedings of the 19th Network and Distributed Security Symposium*, February 2012.

[20] R. Jansen, N. Hopper, and Y. Kim. Recruiting New Tor Relays with BRAIDS. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 319–328. ACM, 2010.

[21] R. Jansen, P. Syverson, and N. Hopper. Throttling Tor Bandwidth Parasites. In *21st USENIX Security Symposium*, August 2012.

[22] S. Kent and R. Atkinson. RFC 2401âĂŤSecurity Architecture for the Internet Protocol. `http://www.ietf.org/rfcs/rfc2401.txt`, November 1998.

[23] T. Kohno, A. Broido, and K. C. Claffy. Remote Physical Device Fingerprinting. *IEEE Trans. Dependable Secur. Comput.*, 2(2):93–108, Apr. 2005.

[24] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, and D. Sicker. Shining Light in Dark Places: Understanding the Tor Network. In *Proceedings of the 8th Privacy Enhancing Technologies Symposium*, pages 63–76, July 2008.

[25] J. McLachlan, A. Tran, N. Hopper, and Y. Kim. Scalable Onion Routing with Torsk. In *Proceedings of the 16th ACM conference on Computer and Communications Security*, CCS '09, pages 590–599. ACM, 2009.

[26] P. Mittal, F. Olumofin, C. Troncoso, N. Borisov, and I. Goldberg. PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval. In *Proceedings of the 20th USENIX Security Symposium*, August 2011.

[27] W. B. Moore, C. Wacek, and M. Sherr. Exploring the Potential Benefits of Expanded Rate Limiting in Tor: Slow and Steady Wins the Race with Tortoise. In *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC)*, pages 207–216, December 2011.

[28] S. J. Murdoch. Comparison of Tor Datagram Designs. *Tor Project Technical Report*, November 2011.

[29] T.-W. J. Ngan, R. Dingledine, and D. S. Wallach. Building Incentives into Tor. In *Proceedings of Financial Cryptography*, pages 238–256, January 2010.

[30] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: a simple model and its empirical validation. In *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '98, pages 303–314. ACM, 1998.

[31] J. Reardon and I. Goldberg. Improving Tor Using a TCP-over-DTLS Tunnel. In *Proceedings of the 18th USENIX Security Symposium*, August 2009.

[32] A. Shepard. Build abstraction layer around TLS. `https://trac.torproject.org/projects/tor/ticket/6465`. Accessed Feb. 2013.

[33] M. Sherr, M. Blaze, and B. T. Loo. Scalable Link-Based Relay Selection for Anonymous Routing. In *PETS '09: Proceedings of the 9th International Symposium on Privacy Enhancing Technologies*, pages 73–93, Berlin, Heidelberg, 2009. Springer-Verlag.

[34] C. Shue, Y. Shin, M. Gupta, and J. Y. Choi. Analysis of IPSec overheads for VPN servers. In *Proceedings of the First international conference on Secure network protocols*, NPSEC'05, pages 25–30, Washington, DC, USA, 2005. IEEE Computer Society.

[35] R. Snader and N. Borisov. A Tune-up for Tor: Improving Security and Performance in the Tor Network. In *Proceedings of the Network and Distributed Security Symposium (NDSS)*, February 2008.

[36] A. H. T. Kivinen, B. Swander and V. Volpe. RFC 3947—Negotiation of NAT-Traversal in the IKE. `http://www.ietf.org/rfc/rfc3947.txt`, January 2005.

[37] C. Tang and I. Goldberg. An Improved Algorithm for Tor Circuit Scheduling. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS)*, pages 329–339, October 2010.

[38] The Tor Project. Tor Metrics Portal: Data. `https://metrics.torproject.org/data.html#performance`. Accessed Feb. 2013.

[39] The Tor Project. Tor Metrics Portal: Network. `http://metrics.torproject.org/network.html`. Accessed Feb. 2013.

[40] F. Tschorsch and B. Scheurmann. How (not) to Build a Transport Layer for Anonymity Overlays. In *Proceedings of the ACM Sigmetrics/Performance Workshop on Privacy and Anonymity for the Digital Economy*, June 2012.

[41] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and Accuracy in a Large-scale Network Emulator. *SIGOPS Oper. Syst. Rev.*, 36(SI):271–284, Dec. 2002.

[42] C. Viecco. UDP-OR: A Fair Onion Transport Design. `http://www.petsymposium.org/2008/hotpets/udp-tor.pdf`, 2008. Accessed Feb. 2013.

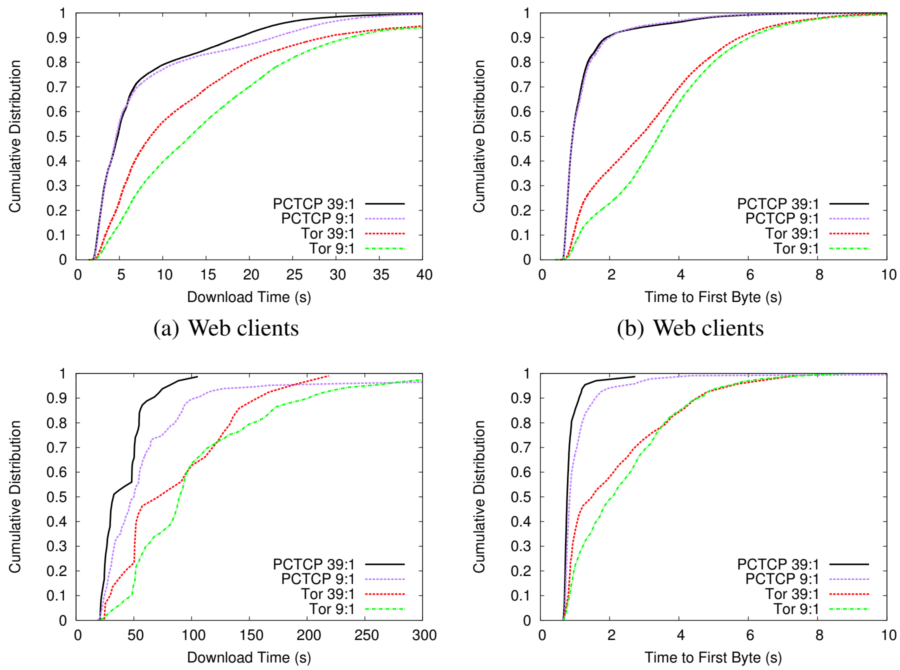
[43] T. Wang, K. Bauer, C. Forero, and I. Goldberg. Congestion-aware Path Selection for Tor. In *Proceedings of Financial Cryptography and Data Security (FC'12)*, February 2012.

APPENDIX

A. LARGE-SCALE EXPERIMENTS USING A HIGHER-BANDWIDTH TOPOLOGY

To observe the effect of PCTCP in a potential future Tor network with more available bandwidth, we construct an experiment on ExperimenTor using a higher-bandwidth underlying Modelnet network topology. Our overlay Tor network is a scaled-down network in which we run 400 clients and 20 Tor routers. The ORs are assigned bandwidth capabilities that are sampled from the bandwidth distribution of the live Tor network ORs. We test the performance of PCTCP in this topology using a light traffic load of 39:1 web-to-bulk client ratio, and using a high traffic load of 9:1 web-to-bulk client ratio.

We also experiment with PCTCP on ExperimenTor using a higher-bandwidth underlying Modelnet network topology. Our overlay Tor network is a scaled-down network in which we run 400 clients and 20 Tor routers. The ORs are assigned bandwidth capabilities that are sampled from the bandwidth distribution of the live Tor network ORs. We test the performance of PCTCP in this topology



(a) Web clients (b) Web clients

(c) Bulk clients (d) Bulk clients

Figure 8: Performance of the web clients in a high-bandwidth network of 400 clients and 20 routers. Compare to Figure 7.

using a light traffic load of 39:1 web-to-bulk client ratio, and using a high traffic load of 9:1 web-to-bulk client ratio.

Figure 8 shows the download time and time-to-first-byte comparisons for Tor and PCTCP using the different traffic loads for the web and bulk clients. The figures show that for Tor clients, the performance degrades faster, compared to PCTCP clients, as we increase the traffic load in the network by decreasing the web-to-bulk client ratio. For example, for the web client, the median time-to-first-byte remains 0.9 seconds for PCTCP under the low and high traffic loads, whereas the corresponding value in Tor degrades by approximately 20%. This is also true for the download time distribution. The median download time for PCTCP remains the same as we increase the load (though the 4^{th} quartile is slightly degraded), whereas the median download time for Tor clients degrades by more than 30%.