
# Mind the Gap: Towards a Backpressure-Based Transport Protocol for the Tor Network

**Florian Tschorsch and Björn Scheuermann,** ***Humboldt University of Berlin***

https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/tschorsch

**This paper is included in the Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16).**

**March 16–18, 2016 • Santa Clara, CA, USA**

ISBN 978-1-931971-29-4

**Open access to the Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16) is sponsored by USENIX.**

# Mind the Gap: Towards a Backpressure-Based Transport Protocol for the Tor Network

Florian Tschorsch Björn Scheuermann
*Humboldt University of Berlin*

## Abstract

Tor has become the prime example for anonymous communication systems. With increasing popularity, though, Tor is also faced with increasing load. In this paper, we tackle one of the fundamental problems in today's anonymity networks: network congestion. We show that the current Tor design is not able to adjust the load appropriately, and we argue that finding good solutions to this problem is hard for anonymity overlays in general. This is due to the long end-to-end delay in such networks, combined with limitations on the allowable feedback due to anonymity requirements. We introduce a design for a tailored transport protocol. It combines latency-based congestion control per overlay hop with a backpressure-based flow control mechanism for inter-hop signalling. The resulting overlay is able to react locally and thus rapidly to varying network conditions. It allocates available resources more evenly than the current Tor design; this is beneficial in terms of both fairness and anonymity. We show that it yields superior performance and improved fairness—between circuits, and also between the anonymity overlay and concurrent applications.

## 1 Introduction

Tor [15] is currently the first choice to preserve online privacy. Implementing what has become the standard architecture for low-latency anonymity services, it routes application-layer data, packaged into equally-sized *cells*, along a cryptographically secured virtual *circuit* through an overlay network. Clients build a circuit by selecting three *relays* (an entry, a middle, and an exit) and establishing cryptographic key material with each of them. A circuit can carry data from one or more application-layer streams. In every overlay hop, one "skin" of encryption is added (or removed, depending on the direction of communication). Intermediate relays are neither able to read the cell contents nor to link streams to a specific source and destination at the same time. This is the foundation for achieving anonymity.

Unfortunately, the current overlay design faces major performance issues. Previous work on improving this more often than not focused on isolated symptoms: for instance, cells that dwell for a too long time in socket buffers [22, 37], a too rigid end-to-end sliding window mechanism [6, 46], security threats due to unbounded buffer growth [25], or unfairness effects caused by different traffic patterns [5, 24, 42]. We note that all of the named problems boil down to unsuitably chosen or unfavorably parameterized algorithms for congestion control, scheduling and buffer management. While this has been pointed out before [37, 45], a consistent and superior overall protocol design—beyond treating the symptoms—is still missing.

Designing such a protocol raises interesting challenges, because the requirements in anonymity overlays deviate in several important points from those of congestion control in the general Internet. First, anonymity demands that relays used along one circuit should be located in different legislations and autonomous systems. This implies typically long end-to-end latencies. Consequently, end-to-end feedback loops (typically stretching over three overlay hops) are necessarily slow. At the same time, though, relays in an anonymity network are aware of individual circuits, because they perform per-circuit cryptography. Therefore, stateful processing per circuit at relays is easily possible, also for the purpose of congestion/flow control. This motivates a protocol design that leverages active participation of the relays.

Second, anonymity demands that control feedback must not reveal user identities, neither directly nor indirectly. Therefore, feedback—especially end-to-end-feedback—must be limited and well considered. This is seen as a reason why reliability should not be implemented end-to-end, but instead hop-wise; this matches Tor's current approach.

Third, relay operators donate resources, in particular bandwidth, to the anonymity overlay. The anonymity traffic typically competes with other traffic on the dona-

tor's network. To incentivize relay operation, anonymity traffic should therefore not be overly aggressive.

Other desirable properties include the standard set of requirements in networks with multiple independent users, including, in particular, good resource utilization, fairness between users/circuits and reasonable latencies.

Tor currently multiplexes all circuits between a consecutive pair of relays into one joint TCP connection. Circuits carry application-layer data, not transport-layer or network-layer packets; that is, the anonymized TCP connection to the destination server is established by the Tor exit relay, where the payload byte stream is handed over from the circuit to this connection and vice versa. In each relay, cells arriving over one of the inter-relay TCP connections are demultiplexed, kept in per-circuit queues, and then multiplexed again, according to the next outgoing connection for the respective circuits. This design is complemented with an end-to-end sliding window mechanism with a fixed, constant window size. Due to its fixed size, this window, quite obviously, lacks adaptivity. As a result, excessive numbers of cells often pile up in the per-circuit queues and/or in the socket buffers of a relay—that is, in the "gap" between the incoming and outgoing TCP connections. The large number of inter-relay standard TCP connections furthermore results in aggressive aggregate traffic, and thus causes unfairness towards other applications in the same network. Last but not least, multiplexing varying numbers of circuits into one joint TCP connection is also the root of substantial inter-circuit unfairness within Tor [44, 45].

In this paper we propose a new design, which we call *BackTap: Backpressure-based Transport Protocol*. With BackTap, we replace Tor's end-to-end sliding window by a hop-by-hop backpressure algorithm between relays. Through per-hop flow control on circuit granularity, we allow the upstream node to control its sending behavior according to the variations of the queue size in the downstream node. Semantically, the employed feedback implies "I forwarded a cell". The circuit queue in the downstream relay is therefore, in essence, perceived as nothing but one of the buffers along the (underlay) network path between the outgoing side of the local relay and the *outgoing* side of the next relay. This includes circuit queues and socket buffers into the per-hop congestion control feedback loop, yielding responsiveness and adaptivity. At the same time, it couples the feedback loops of consecutive hops along a circuit, thereby closing the above-mentioned gap. The result is backpressure that propagates along the circuit towards the source if a bottleneck is encountered, because each local control loop will strive to keep its "own" queue short, while its outflow is governed by the next control loop downstream.

We stick to Tor's paradigm of hop-by-hop reliability, and also to Tor's design decision to tunnel application layer data. However, we implement it in a slightly different way: relays in our architecture have a choice whether to accept or to drop a cell on a per-circuit basis. To this end, congestion control and reliability decisions are shifted to the overlay layer, instead of using TCP between relays. This architecture also avoids reliability-related security flaws as they have been found in Tor [25].

We also do not use a fixed window size, neither end-to-end nor per hop. Instead, we adjust the per-hop window size using an appropriately adapted delay-based congestion control algorithm. In previous applications of delay-based congestion control, first and foremost in TCP Vegas [11], its properties have often been seen as a weakness [1, 12]: it is less aggressive than its loss-based counterparts and therefore tends to be disadvantaged in competitive situations. In our approach, this weakness becomes a strength, because the aggressiveness of aggregate Tor traffic can be a significant problem otherwise.

BackTap, including all congestion control and reliability mechanisms, can be implemented on the overlay nodes' application layer, based on UDP transport. Consequently, lower-layer changes are not required. A simulation-based evaluation confirms the benefits of the proposed architecture and demonstrates a huge relief of the network regarding congestion.

Our key contributions are (1) identifying the "feedback gap" as the primary cause of Tor's performance problems, (2) a novel approach to flow control for environments where data is forwarded over multiple overlay hops, (3) a hop-by-hop backpressure design that avoids network congestion with quick, local adjustments and is therefore well suited to long-delay overlay paths, and (4) an in-depth evaluation including a simulation framework for Tor with a specific focus on network aspects.

The remainder of this paper is structured as follows. First, we review related work in Section 2. In Section 3, we discuss the problems and the design space and develop the transport protocol. In Section 4, we evaluate the proposed protocol, before we conclude this paper with a summary in Section 5.

## 2 Related Work

Since Tor's introduction more than a decade ago [15], it has received significant attention in the research community (and beyond). For obvious reasons, this attention has focused on security and privacy aspects. In recent years, though, performance aspects of Internet anonymity in general and the awareness for network congestion issues in particular have become part of the research agenda.

Performance enhancements have been proposed, for instance, by considering an alternative circuit selection algorithm [3, 7, 47] or through an adaptive prioritization of circuits [5, 24, 42]. These research directions are orthogonal to our approach and remain applicable.

The authors of [22, 37] find that cells reside in socket buffers for a long time. In [22], it is suggested to fix this by actively querying all sockets before blindly writing to a socket. Thereby the majority of queued cells is kept in the application layer, so that scheduling on circuit granularity becomes possible with a smaller backlog between data leaving the application and leaving the host. This follows the general intentions of [44, 45]. However, it does not solve the fundamental problem of excessive standing queues due to circuit windows that often far exceed the end-to-end bandwidth-delay product—it only moves these queues to a different place.

Transport-related modifications for Tor have been considered before [6, 8, 17, 27, 33, 37, 46]. A comparative summary of most approaches is provided in [31]. Even though each proposal improves individual aspects, most of them [8, 17, 27, 33, 37] still use the same sliding window mechanism and hence inherit the exact same issues.

As observed by [37], a missing TCP segment carrying data from one circuit will also temporarily stall any other circuit on the same connection until the missing segment has been recovered. This results in head-of-line blocking upon TCP segment losses. The manifest remedy is to use separate (loss-based) TCP connections per circuit [8, 37]. However, as we point out in [45], such a modification would largely increase the (already very high) aggressiveness of the traffic, due to the higher number of parallel loss-based TCP connections. It also does not overcome the fundamental problems with the end-to-end window mechanism and the corresponding feedback gap. In this work, we tackle all these issues.

Only [46] and [6] get rid of Tor's sliding window. The author of [46] builds upon UDP to tunnel an end-to-end TCP connection through the entire circuit. However, while this may be considered a very clean design, it soon comes to its limits because of the long round trip times of a full circuit, which impairs the responsiveness of both reliability and congestion control. Using complex protocols like TCP end-to-end also come at a significant risk of leaking identifying attributes and providing a fingerprint (e. g., via the source port or specific combinations of TCP options), so that end-to-end designs are generally not favorable [8, 45].

The authors of [6] substitute the end-to-end window by a hop-by-hop window, with a scheme adapted from congestion control in ATM networks. However, head-of-line blockings and the choice of window parameters remain open issues. Virtually all transport-related approaches continue to use standard TCP with its built-in congestion control. We instead develop a tailored transport design for anonymity overlays, which eliminates the need for an end-to-end window.

Looking beyond the area of anonymity overlays, the design of a Tor relay resembles a Split TCP setting as it also occurs in performance-enhancing proxies (PEPs): data is forwarded from an incoming to an outgoing TCP connection, linked by an application-layer queue. A survey on PEPs, including case studies, can be found in [10]. Split TCP was originally developed in the context of wireless communication and satellites, but nowadays also finds use in content distribution networks [35]. It basically subdivides an end-to-end TCP connection into a sequence of typically two concatenated connections, where a middlebox (e. g., a wireless access point or a router) acts as PEP.

By terminating the connection at the middlebox and acknowledging data before the actual destination received it, Split TCP, in fact, violates TCP's end-to-end semantics. If desired, this can be avoided by acknowledging data upstream only after it has been acknowledged by the downstream node [48]. In the context of anonymity networks, such a strict adherence to TCP semantics is generally considered unnecessary, though (just like for most practically deployed PEPs). Since Split TCP aims for maximizing the utilization of link capacities, PEPs buffer data and hence congestion might become a problem. As it has been noted before [30], using Split TCP in an overlay network poses particular challenges in this and many other regards. Therefore, even though we focus on the case of anonymity networks, some of our results may also be applied in the area of PEPs and for other overlay designs.

## 3 The BackTap Design

BackTap performs reliability, in-order delivery and flow control on circuit granularity on the application layer. It can be encapsulated in UDP transport, so that there is no need for modifications to the operating system; of course, a transport-layer implementation of the same concepts in the kernel is, in principle, likewise conceivable, but not pursued here. In fact, the approach to tunnel tailored transport protocols has become more and more widespread in recent years, the likely best-known examples are $\mu$TP [41] as used in BitTorrent [41] and QUIC [18] designed for HTTP/2. UDP transport can be combined with DTLS [38] or IPsec to provide message integrity and confidentiality, just like Tor currently uses TLS to secure its TCP-based overlay links.

In this section, we motivate and present the building blocks of our transport approach in detail. In order to emphasize the changes that we propose and to point out the major design challenges in anonymity networks, we use the current Tor design as a reference architecture throughout the discussion.

### 3.1 Tor's Feedback Gap

Tor implements another instance of data forwarding and transport functionality on the application layer, i. e., on

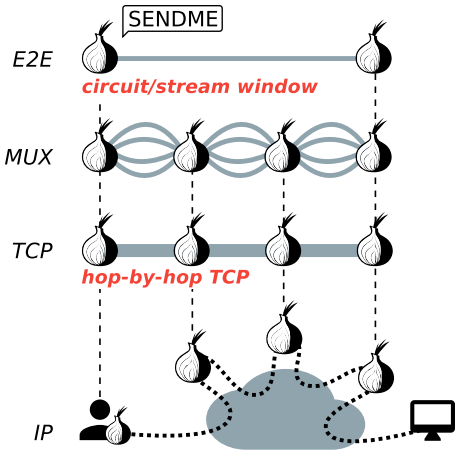



Figure 1: Overview of the layered Tor architecture: relays build a TCP-based overlay, multiplexing circuits, while using an end-to-end sliding window.

top of the existing Internet protocol stack. For this reason, overlay mechanisms will interact with the behavior of underlay protocols and the properties of underlay network paths. This is hardly taken into account in the current design of anonymity overlays in general and of Tor in particular. There are multiple cases where different mechanisms on both layers have overlapping aims. Figure 1 illustrates the various layers of the Tor architecture. The prime example is Tor's end-to-end (E2E) sliding window mechanism between a client's Tor software and an exit relay: it will obviously interact with TCP congestion and flow control, which is used between adjacent overlay nodes. This is also at the heart of the feedback gap in Tor's current design, so that the interplay of these two mechanisms is worth a closer look. This will motivate the key design decisions behind our approach.

Recall, Tor relays forward cells according to the circuit switching principle, but the individual relay does not know about the full path of the circuit. Leaving cryptography aside, relays receive cells over TCP, enqueue them to the respective outgoing circuit queue and then forward them to the downstream node, again via TCP. Between any two adjacent relays, circuits share the same TCP connection. The number of cells in flight for any given circuit is limited by an end-to-end sliding window with a fixed size of 1000 cells (= 512 kB of data).

A node on the receiving side of a Tor circuit signals to send more data by issuing a circuit-level `SENDME` cell for every 100 delivered cells. Receiving such a `SENDME` increments the circuit's transmission window by 100. An additional, analogous mechanism exists on the stream level: a *stream* is Tor's notion for the data carried through a circuit, belonging to one anonymized TCP session. Only the end points of a circuit can associate cells with a stream. Intermediate relays, i. e., entry and middle, only differentiate circuits. The stream-level window's fixed size is 500 cells, and stream-level `SENDME`s worth 50 cells each are used. Due to the end-to-end sliding window there will be no more than 500 cells in flight on a stream, which is capped by 1000 cells in sum on the circuit level. 1000 cells, though, can be significantly more than the bandwidth-delay product of a circuit, so that long queues build up often: excessive queuing is one of the major causes for huge delays, which Tor painfully experiences [13, 27]. In addition, long queues give implicit preference to bulk flows which constantly keep the queue filled, when compared to more interactive flows, like for instance web traffic.

Even if the end-to-end window size were not fixed (a possible modification which, of course, has been taken into consideration before [6]), the end-to-end delay of a circuit is too high to dynamically adjust it with reasonable responsiveness. Given the specific situation in anonymity overlays, it is fortunately also not necessary to find an end-to-end solution: because intermediate nodes are aware of individual circuits anyway, relay-supported hop-by-hop feedback with local readjustments based on perceived local congestion is a reasonable way out.

What happens, now, if the flow control and congestion control mechanisms of the TCP connections between relays come into play? For the inflight traffic permitted by the end-to-end sliding window, TCP will determine the *local* data flow. Congestion control will adapt to the underlay network path between adjacent relays. Flow control will specifically depend on the receiving relay's policy for reading from sockets.

This is where the feedback gap appears, which we illustrate in Figure 2a: Tor relays read from incoming TCP connections regardless of the current fill level of corresponding circuit queues in the relay. Therefore, limited outflow of a circuit does not propagate back to the incoming side of the relay. For this reason, the end-to-end sliding window with its non-adaptive constant size and its long feedback loop is the *only* mechanism that limits the number of cells in flight along the circuit, and it is the only mechanism that will eventually throttle the source.

One may then, of course, ask whether it would suffice to stop reading from a circuit's incoming socket if a queue for that circuit builds up locally. This, however, is infeasible because, as discussed before, circuits are multiplexed over joint TCP connections. A relay therefore cannot selectively read cells from one specific circuit; stopping to read from one socket could result in massive head-of-line blocking for other circuits.

Using separate standard, loss-based TCP connections per circuit is also not a good design avenue: this would result in excessive numbers of parallel connections, and therefore in very aggressive traffic and high packet loss. In accordance with TCP models such as [34], we ar-

gue that more (loss-based) TCP connections imply a smaller rate per connection and thus inevitably a higher packet loss probability *per connection* [45]. In addition, Bufferbloat phenomena [16] cause long reaction times due to excessively large buffers. Thus, approaches such as [8, 37] still suffer from the feedback gap in the same way as Tor does.

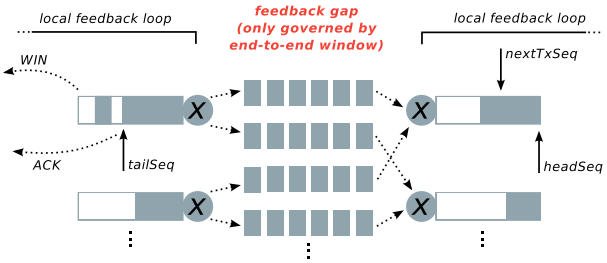
These observations motivate our design based on *delay-based per-circuit congestion control loops*, which can be expected to be much less aggressive than a corresponding loss-based design.

## 3.2 Realizing Backpressure

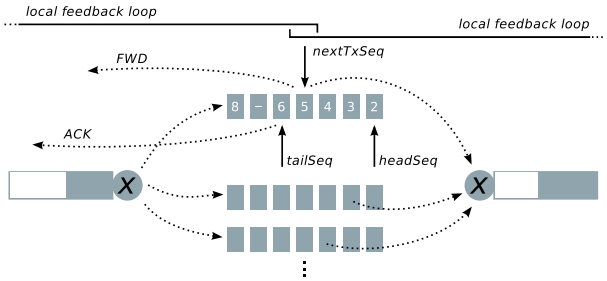
A naive realization of the ideas sketched so far—with separate transport-layer connections per circuit, each with delay-based congestion control—would now likely proceed as follows: if an application-layer queue builds up for one circuit, the inflow might be throttled for that circuit by ceasing to read from its incoming connection. The incoming connection's input buffer would consequently fill up, so that the flow control window is not reopened; a zero window would be triggered. This would, in turn, throttle the outflow of the upstream node, so that the outgoing socket buffer fills up. The outgoing socket in the upstream node would then no longer be writable, an application-layer queue would build up there, and so on. Thereby, congestion feedback would propagate indirectly through backpressure.

However, this implies that upstream of the bottleneck, in each relay there must be enough queued data to fill up a) the outgoing socket buffer, b) the application-layer circuit buffer, and c) the incoming socket buffer. Even keeping technical difficulties related to sizing and management of socket buffers in various operating systems aside, incoming and outgoing socket buffers must at least be sufficiently large to cover the bandwidth-delay product of the respective link, in order not to waste performance. Together with the additional application-layer buffer, the total amount of queued data per overlay hop and circuit would once again have to be very significant, and feedback propagation would once again be slow.

To mitigate these effects, we follow a somewhat different, more consequent path: our solution also performs congestion control per circuit, and it likewise does so without multiplexing circuits into joint connections. However, we virtually extend the network into and through the application layer, by emitting flow control feedback only when a cell has been *forwarded* out of the local relay. The application-layer circuit queues in our design therefore take the role of a fused version of the respective ingress and egress socket buffers. Such a queue is illustrated in Figure 2b, and contrasted with the design that is currently followed in Tor, shown in Figure 2a. The feedback gap in the latter is clearly visible, whereas the




(a) Tor's queuing mechanism with cell multiplexing and a feedback gap between ingress and egress, i. e., TCP sockets.




(b) Fused circuit queue triggers flow control feedback (`FWD`) not until a cell has been forwarded to the successor to achieve backpressure.

Figure 2: Comparison of feedback loops.

local feedback loops in our protocol are directly coupled so that backpressure can build up and propagate immediately upon a deterioration of the available bandwidth.

In BackTap, arriving cells from the predecessor are read from the UDP socket and processed as usual; that is, in particular the cryptographic operations demanded by the anonymity overlay are performed. The cell is subsequently enqueued in the respective circuit queue. The variable *tailSeq* points to the last cell that has been received in order. *tailSeq* is updated when new cells are received. Cells received out of order may also be queued, with respective gaps in the buffer.

On the other end of the queue, *headSeq* points to the frontmost unacknowledged cell. As soon as we learn that the successor has successfully received the cell, *headSeq* is incremented and the respective cell may be discarded from the buffer.

The third pointer, *nextTxSeq*, is incremented when a cell is forwarded to the downstream relay. The key point that distinguishes our design is: this forwarding at the same time also triggers the transmission of corresponding flow control feedback upstream. In the practical implementation this event triggers the transmission of a `FWD` message. Similar to an `ACK`, an `FWD` carries a sequence number that refers to a cell. The upstream node can make use of `FWD`s to determine a *sending window* (*swnd*) based on the provided feedback. It is allowed to keep at most *swnd* cells in the transmission pipe.

The resulting design is a hybrid between flow control and congestion control: the *swnd* adjustment strategy follows a delay-based approach, based on the latency

experienced before receiving an FWD. It will therefore adjust both to the outflow in the downstream node (because only then the FWD is issued) and to the conditions of the network path between consecutive relays (because this path, too, will influence the delays). In essence, it therefore turns the application-layer circuit buffer into yet another buffer along the network path, without a special role from the perspective of the load feedback.

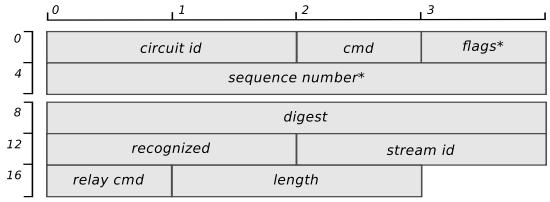
Moreover, tying FWD transmissions to the forwarding of the corresponding cell yields tight feedback coupling between consecutive overlay hops: if the *swnd* adjustment control loop of one overlay hop in a circuit results in a throttled outflow of cells, the FWD arrival delay over the preceding overlay hop will increase accordingly within a one-way local-hop delay. *swnd* can therefore be adjusted quickly. This way, hop-by-hop feedback emerges, and backpressure propagates back to the source. Because delay-based congestion control strives to maintain very short queues, the emerging queues will be small, while available capacity can be fully utilized.

## 3.3 Reliable Transfer

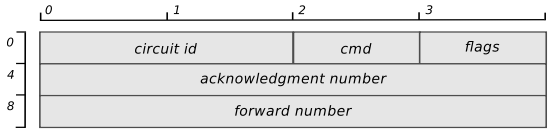
Tor circuits—or, more precisely, Tor streams—carry application-layer data that expects TCP-like reliable bytestream service. The anonymity overlay design relies on each intermediate hop to ensure reliable in-order delivery. That is, there is no end-to-end ARQ (i. e., reliability/acknowledgment/retransmission) scheme. Reliability on the individual hop in Tor uses the per-hop TCP connections' reliability mechanism; relays are not allowed to drop or re-order cells residing in their per-circuit queues.

We stick to this model also in our proposed transport protocol, i. e., we implement reliability on a per-hop basis. To this end, we use cell sequence numbers to determine the order of cells and to detect losses. The mechanisms generally adhere closely to those employed by TCP. The sender infers, either by a timeout or by duplicate acknowledgments, that cells have likely been lost and retransmits them. The key point where we deviate from TCP's mechanism is where the circuit queue in the downstream node and the coupling between consecutive hop feedback loops comes into play.

The most consequent version of the philosophy of taking the application-layer circuit queue as "yet another network buffer" would use the FWD packets as acknowledgments. This might actually be expected to work reasonably well under many circumstances. However, we argue that it entails a pitfall: after all, when a cell has arrived at the next relay, it is already under the control of the downstream application-layer instance, but reliability feedback is not yet generated. This creates a risk for spurious timeouts, and it might take unnecessarily long to recognize and fix losses.




(a) Extended Cell Header (*new header field).




(b) New Feedback Cell.

Figure 3: Cell structure.

For this reason, as an optimization, we separate reliability on the one hand and congestion/flow control on the other hand in terms of feedback. We provide reliability feedback as early as possible, namely upon arrival of a cell, by sending a corresponding ACK. The calculation of the retransmission timeout (RTO) and the fast retransmit mechanism follow RFCs 6298 [36] and 5681 [4], respectively. Both ACKs and FWDs are cumulative. Handshakes upon circuit establishment and teardown can likewise closely follow their respective counterparts in TCP, and can easily be integrated with Tor's handshakes.

Implementing reliability on the application layer makes it possible to drop arriving cells by a deliberate decision (only before the respective ACK has been sent, of course). This opens up new ways out of a difficult problem: the fact that Tor relays in the current overlay design can be attacked by overloading them with cells which they are not allowed to drop [25]. Dropping excessive cells for a given circuit is a much cleaner and simpler solution than the heuristics that are currently used to relieve Tor from this threatening attack vector.

In an extended cell structure, we introduce new header fields: a sequence number (4 Byte) and a field for flags (1 Byte). They fulfill comparable roles to the respective fields in the TCP header. However, since cells have a fixed size for anonymity reasons, sequence numbers refer to cells rater than bytes. The extended cell header is illustrated in Figure 3a.

For FWDs and ACKs, we introduce a separate message format, much smaller than a Tor cell. Smaller feedback messages can be considered safe and per se do not affect anonymity, because they occur in TCP anyway. Moreover, regular cells and feedback messages—not necessarily for the same circuits—can be encapsulated in one UDP packet traveling between two relays. In a typical MTU up to two regular cells and a number of FWD/ACK messages fit in. The freedom to combine FWDs/ACKs with cells also from other circuits (or, of course, to send them separately if no cells travel in the opposite direction) corresponds to a generalized variant of piggybacking.

Since the Tor protocol already exchanges hop-by-hop control messages and supports variable cell lengths and the negotiation of protocol versions [14], our modifications of the cell structure are resonably easy to integrate. Generally speaking, hop-by-hop feedback messages of any kind and size are allowable under Tor's attacker model, i. e., a local adversary with a partial view of the network. Moreover, our modifications affect the cell preamble only. The preamble is not part of the onion encryption and therefore remains unencrypted on the application layer. Likewise, `FWD`/`ACK` messages are not application-layer encrypted. However, the DTLS encryption between consecutive relays shields the preamble from observers on the wire, and also `FWD`s and `ACK`s. The BackTap design therefore provides *additional* protection in comparison to the current TCP-based transport: when using kernel-level TCP as in Tor today, TCP flow control and ACKs are not encrypted by TLS.

### 3.4 Window Adjustment

In the proposed protocol design, each node determines the size of its local *swnd* based on the `FWD` feedback from the next hop downstream. `ACK`s are used for reliability, but do not influence the window adjustment.

Most transport protocols, and in particular most TCP variants, use packet loss as an indicator of congestion and therefore as a basis for adjusting their window size or transmission rate; details highly depend on the TCP flavor [1]. Here, we use a delay-based approach as originally used in TCP Vegas [11]. Delay-based congestion control uses latency variations rather than packet losses as a signal for congestion. If queues start to build up—that is, before losses due to exceeded buffers occur—such a control algorithm re-adjusts the congestion window. Thus, they are less aggressive in the sense that they do not force losses and do typically not fully utilize buffers in intermediate nodes.

This reduced aggressiveness constitutes a significant benefit for an anonymity overlay. The Tor overlay at this time is formed by more than 6000 relays (with increasing trend [43]) in a fully connected topology. All currently active connections to other relays compete for the available capacity. The resulting traffic, in sum, is very aggressive and inevitably provokes significant packet loss—also for other traffic traversing the same bottleneck. One may expect that this can significantly be reduced by using delay-based controllers.

Following the ideas of TCP Vegas, we calculate the difference between expected and actual window size as

$$diff = swnd \cdot \frac{actualRtt}{baseRtt} - swnd,$$

where *actualRtt* and *baseRtt* are the RTT with and without load. In the literature they are also referred to as the "experienced RTT" and the "real RTT". We sample the RTT based on the flow control feedback by measuring the time difference between sending a cell and receiving the respective `FWD`. The *actualRtt* is estimated by taking the smallest RTT sample during the last RTT. This reduces the effect of outliers due to jitter on the network path. The *baseRtt* is the minimum over all RTT samples of *all* circuits directed to the *same* relay. Hence, the individual *diff* calculations per circuit use a joint *baseRtt* estimate. This mitigates potential intra-fairness issues of delay-based approaches.

Depending on the value of *diff*, we adjust the sending window every RTT as follows:

$$swnd' = \begin{cases} swnd + 1 & \text{if } diff < \alpha \\ swnd - 1 & \text{if } diff > \beta \\ swnd & \text{otherwise.} \end{cases} \quad (1)$$

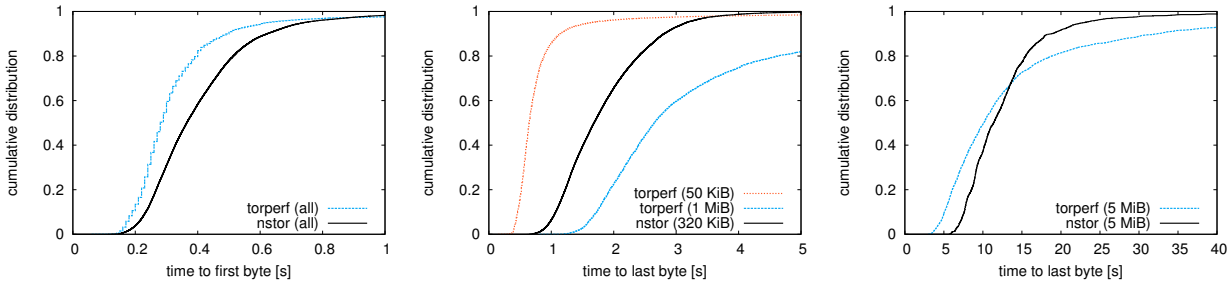
Since *swnd* changes by at most one, it follows an additive increase additive decrease (AIAD) policy. Typically $\alpha$ and $\beta$ are chosen as 2 and 4 (here measured in cells). Therefore, one may expect that *swnd* does not exceed the bandwidth-delay product by much. This is sufficient to achieve full utilization of the available capacities.

Combined with a locally operating scheduling algorithm that round robins all circuits, this adjustment scheme yields a rate allocation that achieves global max-min fairness between circuits [44], because it aims for maintaining a non-empty queue at the bottleneck. In addition, prioritization heuristics such as [5, 24, 42] can be applied, if a prioritization of certain traffic types and patterns is desired. End-to-end windows and corresponding feedback (in Tor: `SENDME`s) are no longer necessary.

## 4 Evaluation

A deployment in a real-world anonymity overlay will only be realistic after very thorough preceding evaluations and in-depth discussion in the research community—a process which we hope to initiate with this work. Even deployments in an emulated or testbed-based anonymity network, are also notoriously hard to analyze—because the anonymity itself, of course, prohibits in-depth traceability and measureability. We therefore evaluated the proposed protocol in a large-scale simulation study.

In fact, setting up such a simulation study is a challenging task by itself. As it turned out, there is a missing link in the tool chain when it comes to experimenting with network protocols for Tor under thorough consideration of protocol behavior below the application layer. Some tools focus only on specific aspects, such as the Tor Path Simulator (TorPS) for circuit paths [26]. Others, such as Shadow [23] and ExperimenTor [9], run real




(a) Bulk and web circuits. (b) Web circuits. (c) Bulk circuits.

Figure 4: Calibration of the simulation evironment.

Tor code, which results in a high degree of realism regarding the application logic, but at the same time requires extensive development efforts to evaluate experimental protocols [28]. While all approaches have their benefits and drawbacks [40], all miss the "noise" of the real Internet and have no real user behavior. Therefore, assumptions about traffic patterns and user behavior are inevitable anyway. The opportunities, though, to vary parameters, scale the setting, and prototype experimental protocols are much more favorable with advanced network simulators such as ns-3 [19].

Therefore, as a further contribution of this work, we introduce *nstor*, a Tor module for ns-3. It is modeled along the lines of the original Tor software, but clearly focuses on the network aspects. In particular, it includes the transport protocol, the cell transmission scheduler, the traffic shaper (Tor includes a token bucket which can be used by a relay operator to constrain the rate and burst that is used), and the multiplexing. First and foremost, it allows direct and reproducible comparisons of protocol design alternatives in both toy examples and larger scenarios. In addition, with ns-3 the Linux kernel can be hooked, so that practically deployed and widely used transport protocol implementations can be used in the simulations, for additional realism. The code of nstor is publicly available on Github[1].

The key point in setting up an environment for a valid evaluation of Tor is to model the overlay appropriately. The Tor model from [21] serves as a guideline here. In our simulations, we use a star topology for simple, easy-to-analyze toy scenarios, and a dumbbell topology for larger-scale, more realistic experiments. Since approximately 93 % of all Tor relays are currently hosted in North America or Europe [43], the dumbbell topology can be thought to approximate the geographical clustering. For this reason, we adjusted the delay according to the iPlane [29] RTT measurements and the client access rates according to Akamai's state of the Internet report [2] by inverse transform sampling, i. e., generating random samples from its cumulative distribution function (CDF). In addition, we scaled and sampled the Tor consensus (as of 2015/08/04) and generated a large set of circuit paths by feeding this consensus to TorPS [26]. Unless otherwise specified, we assumed neither the physically available bandwidth of the relays' access link nor the Internet backbone to be a bottleneck, but that the relay capacity is bounded by the operators using the abovementioned token bucket rate limiter.

In accordance to the model proposed in [21], we deliberately distinguish only two types of circuits, bulk and web circuits. Bulk circuits continuously transfer 5 MiB files, i. e., after completing such a download they immediately request another one. Web circuits request 320 KiB files with a random "think time" of 1 to 20 seconds between consecutive requests. Although apparently being very simplistic, it is the common approach used by the Tor community and hence increases the comparability to related research. As [21] stresses, the ratio of simulated web and bulk circuits in relation to the number of relays requires calibration to produce network characteristic that approximate Tor. Therefore, we used the publicly available torperf data set [43], which consists of measurements of various file downloads over the live Tor network. The time-to-last-byte (TTLB) and time-to-first-byte (TTFB) results (as of August 2015) are shown in Figure 4 as CDF plots. For our analysis in a larger setting, we observed that a scenario with 100 relays and 375 circuits with 10 % bulk circuits approximates Tor's performance reasonably well (cf. Figure 4). This configuration corresponds to one of Shadow's example scenarios (as of Shadow v1.9.2). In this setting, we simulated a period of 300 seconds (simulation time), started the clients at random times during the first 30 seconds and left the system another 30 seconds lead time before evaluating. For statistically sound results, all simulations in this paper were repeated with varying random seeds and are presented either with 95 % confidence intervals or as cumulative distribution functions.

In addition to "vanilla" Tor and our approach, BackTap, we also implemented the N23 protocol as proposed

[1] `https://github.com/tschorsch/nstor`

in [6] and PCTCP [8] (which is conceptually identical to TCP-over-DTLS [37]). This constitutes the first qualitative comparison among alternative transport design proposals for Tor. It also underlines the flexibility of nstor, our ns-3-based simulation module.

## 4.1 Steady State

First, we take a look at the steady state behavior, i. e., when long-term flows reach equilibrium. For the analysis of the steady state, we focus on the cumulative amount of delivered data of a circuit: by $W(t)$ we denote the amount of payload data delivered to the client up to time $t$. The counterpart on the sender side is $R(t)$, which denotes the cumulative amount of data injected into a circuit up to time $t$. Obviously, both functions are non-negative, non-decreasing and $R(t) \geq W(t)$ must hold true at all times.

Given $R$ and $W$, the end-to-end backlog can be defined as $R(t) - W(t)$, the end-to-end delay as $t_2 - t_1$ for $t_1 \leq t_2$ and $R(t_1) = W(t_2)$, and the achieved end-to-end data delivery rate during an interval $[t_1, t_2]$ as

$$\frac{W(t_2) - W(t_1)}{t_2 - t_1}.$$

Intuitively, these are the vertical difference, the horizontal difference and the slope of the respective functions. For our simulation, we sampled $R(t)$ and $W(t)$ at the sender side (in Tor often called the "packaging edge") and the receiver side (the "delivering edge") of a circuit every 10 ms (simulation time). After the steady state is reached, we performed a linear regression on our data points and calculated the rate, backlog and delay accordingly. The results for a single circuit with a bottleneck rate of 1 500 kB/s (enforced through an application layer limit at the middle relay) and varying end-to-end RTT are given in Figure 5 as a mean of 20 runs with 95 % confidence intervals.

Since Tor has a fixed window size that it will fully utilize, the results with the standard Tor protocol heavily depend on how this window size relates to the bandwidth-delay product (BDP), and thus to the end-to-end RTT. In our example, the circuit window size matches the BDP at an RTT of approximately

$$1\,000 \cdot \frac{512\,\text{B}}{1\,500\,\text{kB/s}} \approx 341\,\text{ms}.$$

Before this point, the backlog significantly increases the delivery delay; for higher RTTs, the download rate drops and asymptotically converges to zero, because the window does not suffice to fully utilize the available bandwidth. This clearly demonstrates Tor's fundamental problem: on the end-to-end level, the only control mechanism is the fixed window, which, however, does not adapt to the network path.

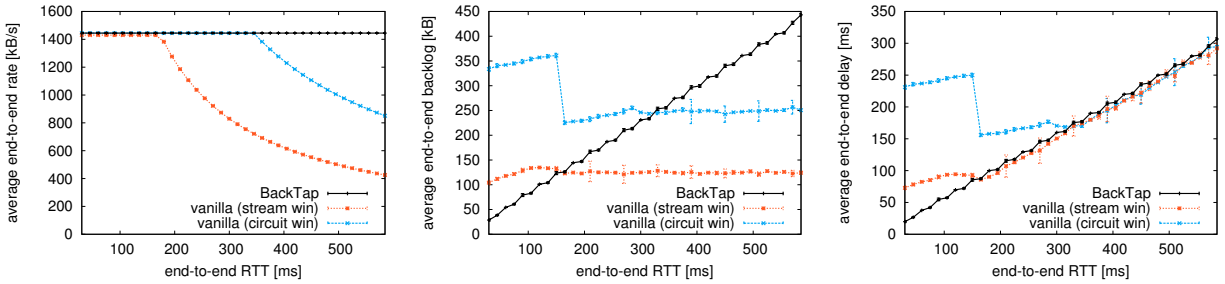
There are some noteworthy phenomena that might be confusing at first sight. In a first approximation according to theory, one would expect that half of a circuit window's worth of data (i. e., approx. 250 kB) is travelling in downstream direction, while the other half of the window is on its way back in the form of SENDME cells. The end-to-end backlog (as defined above: the difference between the amount of sent and received data at a given point in time) should therefore be approximately 250 kB. However, recall that the rate limit is enforced on the application layer by a token bucket. Our model follows the implementation in Tor, where this token bucket is refilled periodically every 100 ms. The bottleneck operates at its capacity limit, always draining its bucket and sending corresponding cell bursts. Thus, about every 100 ms approximately $1\,500\,\text{kB/s} \cdot 100\,\text{ms} = 150\,\text{kB}$ (300 cells) arrive at the client, consequently triggering three SENDMEs. As a result, as long as the RTT is lower than the 100 ms refill interval, only three SENDMEs are on the way back, so that the upstream amount of data is correspondingly higher (approx. 350 kB). For higher RTTs, the observed backlog approaches the theoretical limit without this effect, i. e., 250 kB. Both levels, 350 kB and 250 kB, can be observed in Figure 5b for vanilla Tor ("circuit win").

The respective end-to-end delay, as seen in Figure 5c, behaves according to the built up backlog. That is, while the circuit window is larger than the BDP, there is a noticable delay. Ideally, the end-to-end delay should be half the end-to-end RTT, though.

It is important to note that with a fixed window size there is only one sweet spot, i. e., the BDP. If this point is not met, either the backlog and hence the delay increases or the circuit becomes underutilized. A heterogeneous and volatile network such as Tor is condemned to yield poor performance when employing a static mechanism.

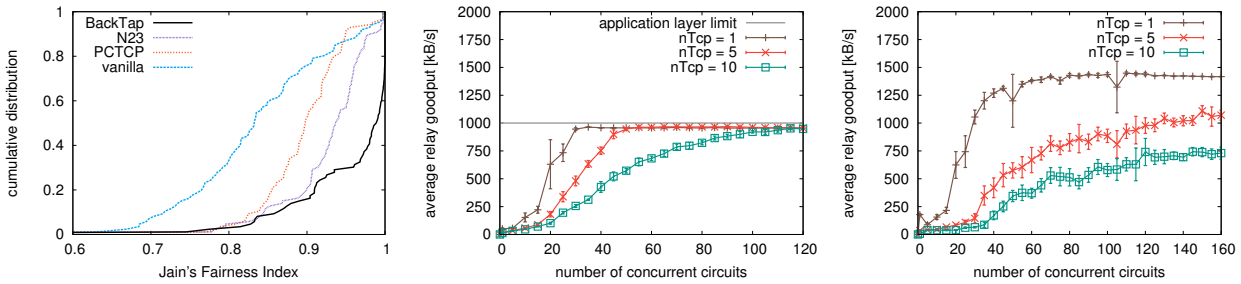
Of course, the same applies to simulations where the (smaller) stream window is the limiting factor: the rate drops much earlier, at $500 \cdot 512\,\text{B} / 1\,500\,\text{kB/s} \approx 171\,\text{ms}$. While the end-to-end RTT is less than 100 ms, the three SENDMEs in upstream direction cause a backlog of about 100 kB, this time slightly less than half the window size. Beyond this point, the results meet theory and the backlog levels at half the stream window, that is 125 kB.

We also observed that Nagle's algorithm [32] can interfere with Tor's window mechanism. In a nutshell, Nagle's algorithm suspends transmission for a short period and tries to combine small chunks of data to reduce the overhead. This behavior causes extra delays upon transmission of SENDMEs, and thereby artificially increases the experienced RTT. As a consequence, the rate drops much earlier and the backlog settles at a lower level accordingly, because a larger fraction of the window is spent on the upstream (SENDME) direction (not shown in the figure). However, as soon as scenarios become more




(a) End-to-end rate. (b) End-to-end backlog. (c) End-to-end delay.

Figure 5: Single circuit scenario clearly demonstrates Tor's fundamental problem and the benefits of our approach.




(a) Intra-fairness. (b) Inter-fairness (application layer bottleneck). (c) Inter-fairness (access link bottleneck).

Figure 6: Fairness evaluation.

complex including more traffic flows, the effect vanishes. By default Nagle is enabled in today's deployments and hence also in Tor. Therefore, we disabled it only to make the previous simulations more easily comprehensible; in all our following simulations Nagle will be enabled. Nevertheless, either with or without Nagle enabled or with the stream or circuit window in place, a fixed-size window is not able to adapt and obviously comes at a severe cost in performance.

In contrast, our approach is able to adjust to the network in all situations. It maintains the rate, while the backlog increases linearly with the RTT (and thus with the BDP). As a result, we achieve an end-to-end delay that always just slightly exceeds the physical RTT. This is the behavior a good transport protocol should exhibit.

## 4.2 Fairness

For those readers familiar with delay-based congestion control, a number of typical issues will likely come to mind. In particular, they relate to intra-fairness and inter-fairness. We therefore now assess these aspects.

**Intra-Fairness** Delay-based congestion control depends on accurate RTT measurements. In particular, "late coming" circuits may suffer from an overestimated *baseRTT*. This leads to intra-fairness issues, i. e., to drawbacks in the competition with other delay-based circuits. We mitigate this issue by sharing *baseRTT* information among circuits directed to the same successor. Thus, circuits established later will still base their calculations on sound *baseRTT* measurements. This is a feature of our approach that becomes possible, because the transport logic is implemented in the application layer.

Furthermore, our approach enables cell scheduling on circuit granularity. This avoids fairness issues due to varying numbers of active circuits multiplexed into one transport layer connection, as described in [44]. Figure 6a shows Jain's fairness index [20] calculated over per-circuit goodputs at the respective bottlenecks. This index quantifies fairness as a value between zero and one, where one means perfect fairness. For this simulation, a star topology with 50 relays and 100 circuits generated according to the real-world Tor consensus were used. We started infinite large downloads (i. e., bulk traffic) over each circuit, where the starting times were randomly distributed during the first 30 seconds. We let the simulation settle for another 60 seconds to reach a steady state before evaluating the mean per-circuit end-to-end rates. The results of 20 runs are given as a cumulative distribution plot. Our approach, in fact, achieves a much fairer distribution than all other protocols, which the larger fraction of higher fairness indices confirms.

In these simulations, we also investigated the overhead by comparing the ratio of the achieved goodput (on the application layer) and the actually transmitted bytes (on the MAC layer), i. e., the throughput. The results, as seen in Table 1, show an insignificant difference of approxi-

Table 1: Overhead and backlog comparison

| | protocol | $\frac{\text{goodput}}{\text{throughput}}$ ratio | avg. backlog |
|---|---|---|---|
| 100 circs | BackTap | 0.89 | 29 kB |
| | N23 | 0.86 | 112 kB |
| | PCTCP | 0.90 | 181 kB |
| | vanilla | 0.90 | 184 kB |

Table 2: Completed downloads (#dwnlds) and mean rate

| | | bulk | | web | |
|---|---|---|---|---|---|
| | protocol | #dwnlds | avg. rate | #dwnlds | avg. rate |
| 375 circs | BackTap | 7 503 | 587 kB/s | 52 102 | 357 kB/s |
| | N23 | 4 563 | 378 kB/s | 49 065 | 215 kB/s |
| | PCTCP | 5 426 | 424 kB/s | 49 513 | 223 kB/s |
| | vanilla | 5 493 | 426 kB/s | 49 522 | 228 kB/s |
| 800 circs | BackTap | 12 108 | 439 kB/s | 110 142 | 302 kB/s |
| | N23 | 9 067 | 346 kB/s | 104 641 | 204 kB/s |
| | PCTCP | 10 388 | 376 kB/s | 105 288 | 207 kB/s |
| | vanilla | 10 491 | 382 kB/s | 105 276 | 217 kB/s |

mately 1 % compared to vanilla Tor. Note that Tor regulalry sends at least one `SENDME` (512 Byte) cell every 250 kB and produces constantly `ACK`s on the transport layer, while our appraoch sends a comparable amount of `ACK`s but emits much smaller flow control feedback messages with a higher frequency. As the results suggest the overhead approximately balances out. We also found that our approach largely reduces the number of in-flight cells in the network: the total backlog is about three (for N23) to six times (for vanilla and PCTCP) lower.

**Inter-Fairness** One of the most prominent caveats of delay-based approaches is that they are "over-friendly" to concurrent loss-based connections. Basically, they reduce the sending rate before loss-based approaches do, because they detect congestion earlier. In some cases this is an intended behavior (cf. LEDBAT [39]), while in the case of TCP Vegas this was generally perceived as an issue [1, 12]. However, if a number of delay-based sessions come together, they are in sum able to compete well [12]. We exploit the properties of delay-based congestion control, because it allows the anonymity overlay to compete more reasonably with other applications (using loss-based TCP) in the relay operators' networks.

We simulated a scenario with a varying number of parallel circuits (on the $x$ axis) and a likewise varying number of competing loss-based TCP connections (*nTcp*). The TCP connections represent downloads that are performed on the same machine as the Tor relay. In a first setting, we limited the anonymity relay bandwidth to 1 MB/s (by the token bucket), while the access link has twice that capacity. In a second setting, we left Tor virtually unlimited (token bucket configured to 10 MB/s) and let the access link become the bottleneck. For small numbers of circuits, the results in Figure 6b and 6c clearly demonstrate in both settings the over-friendly behavior of the delay-based controller, relative to the number of TCP connections. A higher number of active circuits still leaves a good fraction of the total 2 MB/s for the competing non-anonymity connections. For typical relays today one may expect between a few hundred and several thousand concurrently open circuits [8]; of course, not all of them are active all the time.

We believe that the over-friendly inter-fairness of BackTap constitutes an important incentive for relay operators to donate more bandwidth. Typically, relay operators use Tor's token bucket to impose conservative bandwidth limits on their relays. If Tor, however, will appropriately reduce its bandwidth consumption while another application's traffic demand temporarily increases, relay operators will be more willing to operate a Tor relay with less restrictive bandwidth limits. In addition performance penalties of loss-based protocols in environments like Tor [45] will be mitigated.

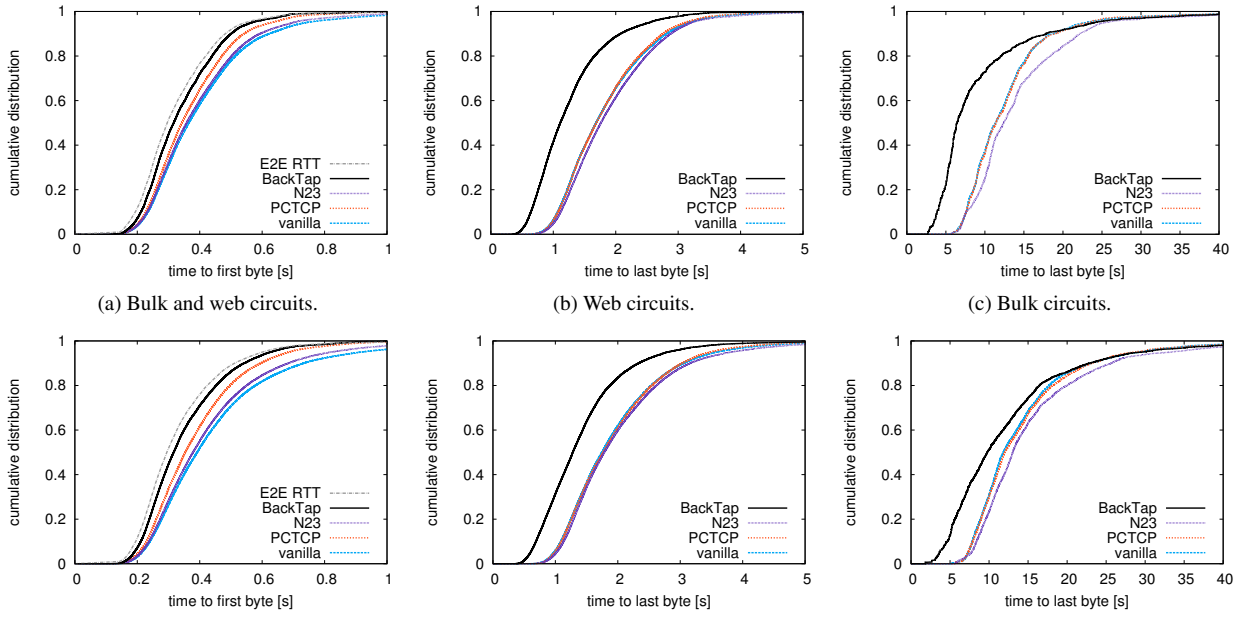
## 4.3 Larger-Scale Analysis

For an analysis in a larger setting, we simulated scenarios with a dumbbell topology and paths generated according to the real-world Tor consensus, as described above. The time-to-first-byte and time-to-last-byte results of the calibrated setting are shown in Figure 7 (a)–(c) as CDF plots.

In Figure 7a, we show the TTFB results for web and bulk traffic. Virtually all initial byte sequences of answers to requests are delivered faster with BackTap than with any other protocol. In fact, BackTap's TTFB results are very close to the optimum, i. e., the network's physical end-to-end RTT (denoted as "E2E RTT" in the plot). TTFB is an important measure for the interactivity and has a significant impact on the overall user experience. The lower achieved TTFB would likely result in an increased user satisfaction, due to increased reactivity.

The performance gain of our approach becomes apparent when looking at the TTLB results in Figures 7b and 7c. While the download times for web requests typically vary between 1 and 3 s, we achieve significantly better performance, where almost half of all the requests are already completed in less than 1 s. Also the bulk transfers yield better results, i. e. approximately 30 % more bulk downloads are completed in less than 10 s.

In order to assess the performance of our approach in a very congested network, we additionally simulated a scenario with 800 circuits. The results are shown in Figure 7 (d)–(e). Also in this "stress test" scenario, BackTap is able to achieve reasonable results, which in all cases yield shorter download times. Particularly a look at Figure 7f provides a deeper explanation for these results. It shows that the CDF of our approach is closer to the other protocols and "flattens" quicker than in Figure 7c, i. e., more bulk downloads take longer to finish. As a con-




(a) Bulk and web circuits. (b) Web circuits. (c) Bulk circuits.

(d) Bulk and web circuits. (e) Web circuits. (f) Bulk circuits.

Figure 7: Time to download files over Tor (10 runs, 100 relays, (a)–(c) 375 circuits, (d)–(f) 800 circuits).

sequence, bulk traffic is prevented from "clogging" the network as with the other protocols. However, this does not mean that bulk traffic is treated unfairly: quite in contrast, all of the circuits and flows are treated equally. This is an important feature of our approach: it gives all circuits, web and bulk, a fair share of the network capacity, without the need for (complex, error-prone) explicit traffic pattern analysis and prioritization.

Another perspective on the performance of the various protocols is provided by Table 2. There, we summarize the number of completed downloads (within the simulation time) and the mean download rate for both larger-scale simulation scenarios. In the stress test with 800 circuits, BackTap is able to complete approximately 5 % and 15 % more web and bulk requests, respectively, compared to vanilla Tor. Eventually, the mean download rate is in all cases higher as well. On a more general level, we note that vanilla Tor shows, particularly for the web traffic, a much higher variance of TTFB and TTLB. There is, for instance, always a non-negligible fraction of connections that takes far longer than average. This observation is in line with practical experiences of Tor users and the results presented in [21,43]. Our approach, according to the results presented here, typically reduces the overall variance by more than 17 %.

## 5 Conclusion

Aware of Tor's fundamental problems and the specific requirements of anonymity overlays, we developed a tailored transport protocol, namely BackTap. In particular, we presented a novel way to couple the local feedback loops for congestion and flow control. It builds upon backpressure between consecutive application-layer relays along a circuit, and a delay-based window size controller. We showed that this can bring a huge relief regarding network congestion by closing the gap between local controllers, so that the need for slow end-to-end control vanishes. In packet level simulations we confirmed the expected improvement.

Besides, there are good reasons why our approach also makes Tor more resilient. First, due to the backpressure, congestion-based attacks will have less influence on other circuits. Second, the much fairer resource allocation makes circuits "look" more "similar", thereby improving the cover traffic properties of concurrent circuits. However, the trade-off between anonymity and performance needs further investigation. In particular, the use of delayed and aggregated feedback to impede traffic confirmation is on our agenda for future work. Generally, we believe that an advanced network traffic control can make Tor's degree of anonymity stronger.

Overall, our approach shows new ways for designing suitable transport mechanisms for anonymity overlays.

## Acknowledgements

We thank our students Manuel Rüger and Tobias Schall for valuable discussions. We also like to thank the anonymous reviewers and our shepherd Paul Francis for their constructive, detailed and very helpful feedback.

## References


[1] AFANASYEV, A., TILLEY, N., REIHER, P. L., AND KLEINROCK, L. Host-to-host congestion control for TCP. *IEEE Communications Surveys and Tutorials 12*, 3 (2010), 304–342.

[2] AKAMAI. State of the Internet report, Q1 2015.

[3] AKHOONDI, M., YU, C., AND MADHYASTHA, H. V. LASTor: A low-latency AS-aware Tor client. In *SP '12: Proceedings of the 33th IEEE Symposium on Security and Privacy* (May 2012).

[4] ALLMAN, M., PAXSON, V., AND BLANTON, E. TCP Congestion Control. RFC 5681 (Draft Standard), Sept. 2009.

[5] ALSABAH, M., BAUER, K., AND GOLDBERG, I. Enhancing Tor's performance using real-time traffic classification. In *CCS '12: Proceedings of the 19th ACM Conference on Computer and Communications Security* (Oct. 2012), pp. 73–84.

[6] ALSABAH, M., BAUER, K., GOLDBERG, I., GRUNWALD, D., MCCOY, D., SAVAGE, S., AND VOELKER, G. DefenestraTor: Throwing out windows in Tor. In *PETS '11: Proceedings of the 11th Privacy Enhancing Technologies Symposium* (July 2011).

[7] ALSABAH, M., BAUER, K. S., ELAHI, T., AND GOLDBERG, I. The path less travelled: Overcoming tor's bottlenecks with traffic splitting. In *PETS '13: Proceedings of the 13th Workshop on Privacy Enhancing Technologies* (July 2013), pp. 143–163.

[8] ALSABAH, M., AND GOLDBERG, I. PCTCP: per-circuit tcp-over-ipsec transport for anonymous communication overlay networks. In *CCS '13: Proceedings of the 20th ACM Conference on Computer and Communications Security* (Oct. 2013), pp. 349–360.

[9] BAUER, K. S., SHERR, M., AND GRUNWALD, D. Experimentor: A testbed for safe and realistic tor experimentation. In *CSET '11: Proceedings of the 4th Workshop on Cyber Security Experimentation and Test* (Aug. 2011).

[10] BORDER, J., KOJO, M., GRINER, J., MONTENEGRO, G., AND SHELBY, Z. Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations. RFC 3135 (Informational), June 2001.

[11] BRAKMO, L. S., O'MALLEY, S. W., AND PETERSON, L. L. TCP vegas: New techniques for congestion detection and avoidance. In *SIGCOMM '94: Proceedings of the 1994 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (Aug. 1994), pp. 24–35.

[12] BUDZISZ, L., STANOJEVIC, R., SCHLOTE, A., BAKER, F., AND SHORTEN, R. On the fair coexistence of loss- and delay-based TCP. *IEEE/ACM Transactions Networking 19*, 6 (2011), 1811–1824.

[13] DHUNGEL, P., STEINER, M., RIMAC, I., HILT, V., AND ROSS, K. Waiting for anonymity: Understanding delays in the tor overlay. In *P2P '10: Proceedings of the 10th IEEE International Conference on Peer-to-Peer Computing* (Aug. 2010).

[14] DINGLEDINE, R., AND MATHEWSON, N. Tor protocol specification. `https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt`.

[15] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The second-generation onion router. In *USENIX Security '04: Proceedings of the 13th USENIX Security Symposium* (Aug. 2004), pp. 303–320.

[16] GETTYS, J., AND NICHOLS, K. Bufferbloat: Dark buffers in the internet. *Queue 9*, 11 (Nov. 2011).

[17] GOPAL, D., AND HENINGER, N. Torchestra: reducing interactive traffic delays over tor. In *WPES '12: Proceedings of the ACM Workshop on Privacy in the Electronic Society* (Oct. 2012), pp. 31–42.

[18] HAMILTON, R., IYENGAR, J., SWETT, I., AND WILK, A. QUIC: A UDP-based secure and reliable transport for HTTP/2. IETF Internet Draft, 2016.

[19] HENDERSON, T. R., LACAGE, M., RILEY, G. F., DOWELL, C., AND KOPENA, J. Network simulations with the ns-3 simulator. In *SIGCOMM '08: Proceedings of the 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (Aug. 2008).

[20] JAIN, R., CHIU, D., AND HAWE, W. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. DEC Research Report TR-301, Digital Equipment Corporation, Maynard, MA, USA, Sept. 1984.

[21] JANSEN, R., BAUER, K., HOPPER, N., AND DINGLEDINE, R. Methodically modeling the Tor network. In *CSET '12: Proceedings of the 5th Workshop on Cyber Security Experimentation and Test* (Aug. 2012).

[22] JANSEN, R., GEDDES, J., WACEK, C., SHERR, M., AND SYVERSON, P. F. Never been KIST: tor's congestion management blossoms with kernel-informed socket transport. In *USENIX Security '14: Proceedings of the 23rd USENIX Security Symposium* (Aug. 2014), pp. 127–142.

[23] JANSEN, R., AND HOPPER, N. Shadow: Running Tor in a box for accurate and efficient experimentation. In *NDSS '12: Proceedings of the Network and Distributed System Security Symposium* (Feb. 2012).

[24] JANSEN, R., SYVERSON, P. F., AND HOPPER, N. Throttling tor bandwidth parasites. In *USENIX Security '12: Proceedings of the 21th USENIX Security Symposium* (Aug. 2012), pp. 349–363.

[25] JANSEN, R., TSCHORSCH, F., JOHNSON, A., AND SCHEUERMANN, B. The sniper attack: Anonymously deanonymizing and disabling the tor network. In *NDSS '14: Proceedings of the Network and Distributed System Security Symposium* (Feb. 2014).

[26] JOHNSON, A., WACEK, C., JANSEN, R., SHERR, M., AND SYVERSON, P. F. Users get routed: traffic correlation on tor by realistic adversaries. In *CCS '13: Proceedings of the 20th ACM Conference on Computer and Communications Security* (Oct. 2013), pp. 337–348.

[27] KIRALY, C., BIANCHI, G., AND LO CIGNO, R. Solving performance issues in anonymization overlays with a L3 approach. Tech. Rep. DISI-08-041, Ver. 1.1, Univ. degli Studi di Trento, Sept. 2008.

[28] LOESING, K., MURDOCH, S. J., AND JANSEN, R. Evaluation of a libutp-based Tor datagram implementation. Tech. Rep. 2013-10-001, The Tor Project, Oct. 2013.

[29] MADHYASTHA, H. V., KATZ-BASSETT, E., ANDERSON, T., KRISHNAMURTHY, A., AND VENKATARAMANI, A. iplane: An information plane for distributed services. `http://iplane.cs.washington.edu`.

[30] MAKI, I., HASEGAWA, G., MURATA, M., AND MURASE, T. Performance analysis and improvement of tcp proxy mechanism in tcp overlay networks. In *ICC '05: Proceedings of the IEEE International Conference on Communications* (May 2005), pp. 184–190.

[31] MURDOCH, S. J. Comparison of Tor datagram designs. Tech. rep., Nov. 2011.

[32] NAGLE, J. Congestion Control in IP/TCP Internetworks. RFC 896, Jan. 1984.

[33] NOWLAN, M. F., WOLINSKY, D. I., AND FORD, B. Reducing latency in tor circuits with unordered delivery. In *FOCI '13: Proceedings of the USENIX Workshop on Free and Open Communications on the Internet* (Aug. 2013).

[34] PADHYE, J., FIROIU, V., TOWSLEY, D., AND KUROSE, J. Modeling TCP throughput: A simple model and its empirical validation. In *SIGCOMM '98: Proceedings of the 1998 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (Aug. 1998), pp. 303–314.

[35] PATHAK, A., WANG, A., HUANG, C., GREENBERG, A. G., HU, Y. C., KERN, R., LI, J., AND ROSS, K. W. Measuring and evaluating TCP splitting for cloud services. In *PAM '10: Proceedings of the 11th International Conference on Passive and Active Measurement* (Apr. 2010), pp. 41–50.

[36] PAXSON, V., ALLMAN, M., CHU, J., AND SARGENT, M. Computing TCP's Retransmission Timer. RFC 6298 (Proposed Standard), June 2011.

[37] REARDON, J., AND GOLDBERG, I. Improving tor using a TCP-over-DTLS tunnel. In *USENIX Security '09: Proceedings of the 18th USENIX Security Symposium* (Aug. 2009).

[38] RESCORLA, E., AND MODADUGU, N. Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard), Jan. 2012.

[39] SHALUNOV, S., HAZEL, G., IYENGAR, J., AND KUEHLEWIND, M. Low Extra Delay Background Transport (LEDBAT). RFC 6817 (Experimental), Dec. 2012.

[40] SHIRAZI, F., GOEHRING, M., AND DÍAZ, C. Tor experimentation tools. In *SPW '15: Proceedings of the 36th IEEE Symposium on Security and Privacy Workshops* (May 2015), pp. 206–213.

[41] STRIGEUS, L., HAZEL, G., SHALUNOV, S., NORBERG, A., AND COHEN, B. BEP 29: $\mu$Torrent transport protocol, June 2009.

[42] TANG, C., AND GOLDBERG, I. An improved algorithm for Tor circuit scheduling. In *CCS '10: Proceedings of the 17th ACM Conference on Computer and Communications Security* (Oct. 2010), pp. 329–339.

[43] THE TOR PROJECT. Tor Metrics Portal. `https://metrics.torproject.org`.

[44] TSCHORSCH, F., AND SCHEUERMANN, B. Tor is unfair – and what to do about it. In *LCN '11: Proceedings of the 36th Annual IEEE International Conference on Local Computer Networks* (Oct. 2011), pp. 432–440.

[45] TSCHORSCH, F., AND SCHEUERMANN, B. How (not) to build a transport layer for anonymity overlays. *ACM SIGMETRICS Performance Evaluation Review 40* (Mar. 2013), 101–106.

[46] VIECCO, C. UDP-OR: A fair onion transport design. In *HotPETS '08: 1st Workshop on Hot Topics in Privacy Enhancing Technologies* (July 2008).

[47] WANG, T., BAUER, K. S., FORERO, C., AND GOLDBERG, I. Congestion-aware path selection for tor. In *FC '12: Proceedings of the 16th International Conference on Financial Cryptography and Data Security* (Mar. 2012), pp. 98–113.

[48] XIE, F., JIANG, N., HO, Y. H., AND HUA, K. A. Semi-split TCP: maintaining end-to-end semantics for split TCP. In *LCN '07: Proceedings of the 32nd Annual IEEE International Conference on Local Computer Networks* (Oct. 2007), pp. 303–314.