DefenestraTor: Throwing out Windows in Tor

Mashael AlSabah[1], Kevin Bauer[2], Ian Goldberg[1], Dirk Grunwald[2],
Damon McCoy[3], Stefan Savage[3], and Geoffrey Voelker[3]

[1] University of Waterloo
{malsabah,iang}@cs.uwaterloo.ca
[2] University of Colorado
{bauerk,grunwald}@colorado.edu
[3] University of California—San Diego
{dlmccoy,savage,voelker}@cs.ucsd.edu

Abstract. Tor is the most widely used privacy enhancing technology for achieving online anonymity and resisting censorship. While conventional wisdom dictates that the level of anonymity offered by Tor increases as its user base grows, the most significant obstacle to Tor adoption continues to be its slow performance. We seek to enhance Tor's performance by offering techniques to control congestion and improve flow control, thereby reducing unnecessary delays.

To reduce congestion, we first evaluate small fixed-size circuit windows and a dynamic circuit window that adaptively resizes in response to perceived congestion. While these solutions improve web page response times and require modification only to exit routers, they generally offer poor flow control and slower downloads relative to Tor's current design. To improve flow control while reducing congestion, we implement *N23*, an ATM-style per-link algorithm that allows Tor routers to explicitly cap their queue lengths and signal congestion via back-pressure. Our results show that N23 offers better congestion and flow control, resulting in improved web page response times and faster page loads compared to Tor's current design and the other window-based approaches. We also argue that our proposals do not enable any new attacks on Tor users' privacy.

1 Introduction

Tor [9] is a distributed circuit-switching overlay network consisting of over two-thousand volunteer-run *Tor routers* operating around the world. Tor clients achieve anonymity by source-routing their traffic through three Tor routers using onion routing [13].

Context. Conventional wisdom dictates that the level of anonymity provided by Tor increases as its user base grows [7]. Another important, but often overlooked, benefit of a larger user base is that it reduces suspicion placed on users simply because they use Tor. Today, there are an estimated 150 to 250 thousand daily Tor users [18]. However, this estimate has not increased significantly since 2008. One of the most significant road blocks to Tor adoption is its excessively high and variable delays, which inhibit interactive applications such as web browsing.

Many prior studies have diagnosed a variety of causes of this high latency (see Dingledine and Murdoch [10] for a concise summary). Most of these studies have noted that the queuing delays often dominate the network latencies of routing packets through the three routers. These high queuing delays are, in part, caused by bandwidth bottlenecks that exist along a client's chosen circuit. As high-bandwidth routers forward traffic to lower-bandwidth downstream routers, the high-bandwidth router may be able to read data faster than it can write it. Because Tor currently has no explicit signaling mechanism to notify senders of this congestion, packets must be queued along the circuit, introducing potentially long and unnecessary delays for clients. While recent proposals seek to re-engineer Tor's transport design in part to improve its ability to handle congestion [16,24,31], these proposals face significant deployment challenges.

Improving Congestion and Flow Control. To reduce the delays introduced by uncontrolled congestion in Tor, we design, implement, and evaluate two classes of congestion and flow control. First, we leverage Tor's existing end-to-end window-based flow control framework and evaluate the performance benefits of using small fixed-size circuit windows, reducing the amount of data in flight that may contribute to congestion. We also design and implement a dynamic window resizing algorithm that uses increases in end-to-end circuit round-trip time as an implicit signal of incipient congestion. Similar solutions are being considered for adoption in Tor to help relieve congestion [5], and we offer a critical analysis to help inform the discussion. Window-based solutions are appealing, since they require modifications only to exit routers.

Second, we offer a fresh approach to congestion and flow control inspired by standard techniques from Asynchronous Transfer Mode (ATM) networks. We implement a per-link credit-based flow control algorithm called N23 [17] that allows Tor routers to explicitly bound their queues and signal congestion via back-pressure, reducing unnecessary delays and memory consumption. While N23 offers these benefits over the window-based approaches, its road to deployment may be slower, as it may require all routers along a circuit to upgrade.

Evaluation. We conduct an holistic experimental performance evaluation of the proposed algorithms using the ModelNet network emulation platform [30] with realistic traffic models. We show that the window-based approaches offer up to 65% faster web page response times relative to Tor's current design. However, they offer poor flow control, causing bandwidth under-utilization and ultimately resulting in poor download time. In contrast, our N23 experiments show that delay-sensitive web clients experience up to 65% faster web page responses and a 32% decrease in web page load times compared to Tor's current design.

2 Tor Background

The Tor network is a decentralized circuit-switching overlay consisting of volunteer-run Tor routers hosted around the world. Tor offers anonymity to clients by employing a layered encryption scheme [13] with three Tor routers. All data is sent in fixed-sized 512-byte units called cells. In general, the client selects routers

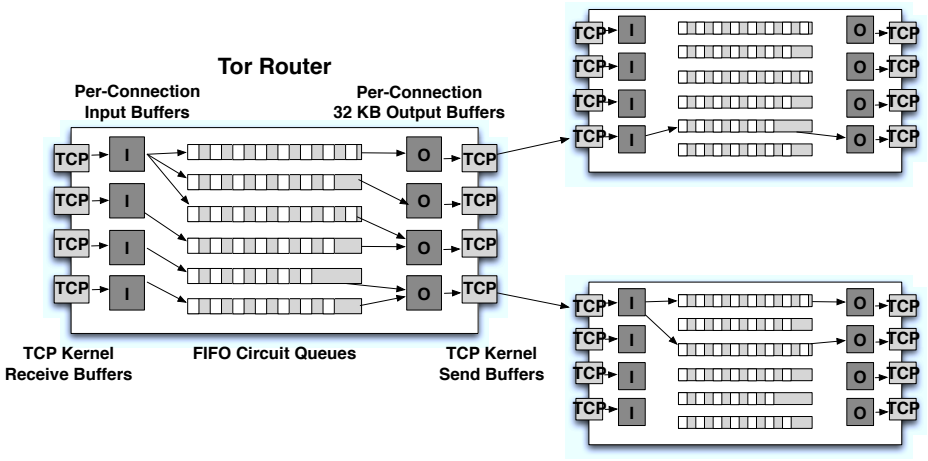


Fig. 1: A Tor router's queuing architecture

to use on a circuit taking into account their bandwidth capacities, in order to balance the traffic load over the available router bandwidth. The first router on a circuit (called an entry guard) is chosen carefully to reduce the threat of profiling and the predecessor attack [33]. Upon receiving a cell, the router removes its layer of encryption and forwards the cell to the next router on the circuit. Once the final (exit) router in the circuit removes its layer of encryption, the client's traffic is forwarded to the destination. A prior study found that the majority of Tor traffic by connection is interactive HTTP [19], and most of this traffic volume flows from the destination to the client. More details about Tor can be found in its design document [9] and its evolving protocol specification [8].

3 Tor's Approach to Congestion and Flow Control

Since the Tor network consists of volunteer-run routers from across the world, these routers have varying and often limited amounts of bandwidth available to relay Tor traffic. Consequently, as clients choose their circuits, some routers have large amounts of bandwidth to offer, while others may be bandwidth bottlenecks. In order for Tor to offer the highest degree of performance possible, it is necessary to have effective mechanisms in place to ensure steady flow control, while also detecting and controlling congestion. In this section, we discuss the many features that directly or indirectly impact congestion and flow control in Tor.

3.1 Congestion and Flow Control Mechanisms

Pairwise TCP. All packets sent between Tor routers are guaranteed to be delivered reliably and in-order by using TCP transport. As a result of using TCP, communications between routers can be protected with TLS link encryption. However, several circuits may be multiplexed over the same TCP connections, which could result in an unfair application of TCP's congestion control [24].

Tiered Output Buffers. Each Tor router's internal queuing architecture is illustrated in Figure 1. When a Tor router receives a cell on one of its TCP connections, the cell is first copied from the connection's receive kernel buffer into an application-layer input buffer to be decrypted. Next, the cell is pushed onto a FIFO *circuit queue* for the cell's respective circuit. For each outgoing TCP connection, a FIFO *output buffer* is maintained. The output buffer has a fixed size of 32 KiB, while the circuit queue has no explicit bound, but the circuit window size restricts how many cells may be in flight (described below). Since multiple circuits are often multiplexed over the same TCP connection, when there is space available in the outgoing connection's respective output buffer, the router must choose which circuits' cells to copy onto the output buffer. Initially, cells were chosen by round-robin selection across circuits. Recently, circuit prioritization has been proposed to give burstier circuits that likely correspond to interactive traffic priority over long-lived, bulk circuits [29].

Circuit and Stream Windows. Tor uses two layers of end-to-end window-based flow control between the exit router and the client to ensure steady flow control. First, a *circuit window* restricts how many cells may be in flight per circuit. By default, Tor uses a fixed 500 KiB (1000 cell) circuit window. For every 50 KiB (100 cells) received, an acknowledgment cell called a `SENDME` is sent, informing the sender that they may forward another 100 cells to the receiver.[1]

Within each circuit window there is a *stream window* of 250 KiB (500 cells) to provide flow control (or fairness) within a circuit. The receiver replies with a stream-level `SENDME` for every 25 KiB (50 cells) received. On receiving a stream-level `SENDME`, the sender may forward another 50 cells.

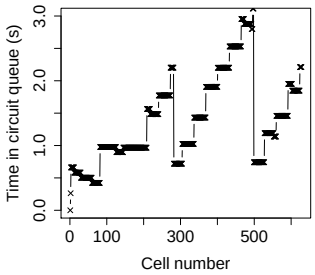


Fig. 2: The exit router's circuit queue delays for a 300 KiB download

Both the stream-level and circuit-level windows are relatively large and static. To illustrate how this can degrade performance, consider the following scenario. Suppose a client downloads a file through a circuit consisting of 10 MiB/s entry and exit routers and a 128 KiB/s middle router. Since the exit router can read data from the destination server faster than it can write it to its outgoing connection with the middle router, and the reliable TCP semantics preclude routers from dropping cells to signal congestion, the exit router must buffer up to one full circuit window (500 KiB) worth of cells. Furthermore, as shown in Figure 2, these cells often sit idly for several seconds while the buffer is slowly emptied as `SENDME` cells are received. Since cells may travel down a circuit in large groups of up to 500 KiB followed by periods of silence while the exit router waits for `SENDME` replies, Tor's window-based flow control does not always keep a steady flow of cells in flight.

[1] Due to a bug, clients running `Tor 0.0.0–0.2.1.19` erroneously reply with circuit-level `SENDME` cells after receiving 101 cells (rather than 100 cells).

Token Bucket Rate Limiting. In order to allow routers to set limits on the amount of bandwidth they wish to devote to transiting Tor traffic, Tor offers token bucket rate limiting. Briefly, a router starts with a fixed amount of tokens, and decrements their token count as cells are sent or received. When the router's token count reaches zero, the router must wait to send or receive until the tokens are refilled. To reduce Tor's CPU utilization, tokens are refilled only once per second. It has been previously observed that refilling the tokens so infrequently contributes in part to Tor's overall delays [4].

3.2 Alternate Proposals to Reduce Congestion

There have been several recent proposals aimed specifically at reducing Tor's congestion. First, Tor has incorporated adaptive circuit-building timeouts that measure the time it takes to build a circuit, and eliminate circuits that take an excessively long time to construct [3]. The intuition is that circuits that build slowly are highly congested, and would in turn offer the user poor performance. While this approach likely improves the users' quality of service in some cases, it does not help to relieve congestion that may occur at one or more of the routers on a circuit *after* the circuit has been constructed.

In addition, user-level rate limiting has been proposed to throttle over-active or bulk downloading users. Here, the idea is to reduce the overall bandwidth consumption by bulk downloaders by using per-connection token bucket rate limiting at the entry guard. Early experiments indicate an improvement in download time for small file downloaders (the majority of Tor users), while harming bulk downloaders [6].

4 Improving Tor's Congestion and Flow Control

Our primary goal is to improve Tor's performance, specifically by better understanding and improving Tor's congestion and flow control. We consider two broad classes of solutions. First, we wish to understand how much improvement is possible simply by adjusting Tor's existing end-to-end window-based flow control mechanisms to reduce the amount of data in flight, and thereby mitigate congestion. We also evaluate an end-to-end congestion control technique that enables exit Tor routers to infer incipient congestion by regarding increases in end-to-end round-trip time as a congestion signal. Second, we consider a fresh approach to congestion and flow control in Tor, eliminating Tor's end-to-end window-based flow control entirely, and replacing it with ATM-style, per-link flow control that caps routers' queue lengths and applies back-pressure to upstream routers to signal congestion.

4.1 Improving Tor's Existing End-to-End Flow Control

We first consider whether adjusting Tor's current window-based flow control can offer significant performance improvements. Keeping Tor's window-based mechanisms is appealing, as solutions based on Tor's existing flow control framework

may be deployed immediately, requiring modifications only to the exit routers, not clients or non-exit routers.

Small Fixed-size Circuit Windows. The smallest circuit window size possible without requiring both senders and receivers to upgrade is 50 KiB (100 cells, or one circuit-level `SENDME` interval). We evaluate how fixed 50 KiB circuit windows impact clients' performance.[2]

Dynamic Circuit Windows. We next consider an algorithm that initially starts with a small, fixed circuit-window and dynamically increases the window size (*e.g.*, amount of unacknowledged data allowed to be in flight) in response to positive end-to-end latency feedback. Inspired by latency-informed congestion control techniques for IP networks [2,32], we propose an algorithm that uses increases in perceived end-to-end circuit round-trip time (RTT) as a signal of incipient congestion.

The algorithm works as follows. Initially, each circuit's window size starts at 100 cells. First, the sender calculates the circuit's end-to-end RTT using the circuit-level `SENDME` cells, maintaining the minimum RTT (rtt_{min}) and maximum RTT (rtt_{max}) observed for each circuit. We note that rtt_{min} is an approximation of the base RTT, where there is little or no congestion on the circuit. Next, since RTT feedback is available for every 100 cells,[3] the circuit window size is adjusted quickly using an additive increase, multiplicative decrease (AIMD) window scaling mechanism based on whether the current RTT measurement (rtt) is less than the threshold T, defined in Equation 1. This threshold defines the circuit's tolerance to perceived congestion.

$$T = (1 - \alpha) \times rtt_{min} + \alpha \times rtt_{max} \tag{1}$$

Choosing a small α value ensures that the threshold is close to the base RTT, and any increases beyond the threshold implies the presence of congestion along the circuit.[4] For each RTT measurement (*e.g.*, each received circuit-level `SENDME`), the circuit window size (in cells) is adjusted according to Equation 2.

$$new\_window(rtt) = \begin{cases} old\_window + 100 & \text{if } rtt \leq T \\ \lfloor old\_window/2 \rfloor & \text{otherwise} \end{cases} \tag{2}$$

Finally, we explicitly cap the minimum and maximum circuit window sizes at 100 and 1000 cells, respectively.[5]

4.2 ATM-style Congestion and Flow Control for Tor

Because Tor's flow control works at the circuit's edges—the client and the exit router—we seek to improve performance by implementing per-link flow con-

[2] Due to the aforementioned bug, in practice, the window size should be 101 cells.

[3] Similar to the 50 KiB windows, `SENDME` cells may be available after 101 cells.

[4] For our experiments, we use $\alpha = 0.25$.

[5] Note that a selfish Tor client could attempt to increase their circuit window by preemptively acknowledging data segments before they are actually received. Prior work in mitigating similar behavior in selfish TCP receivers may be applied here [25,27].

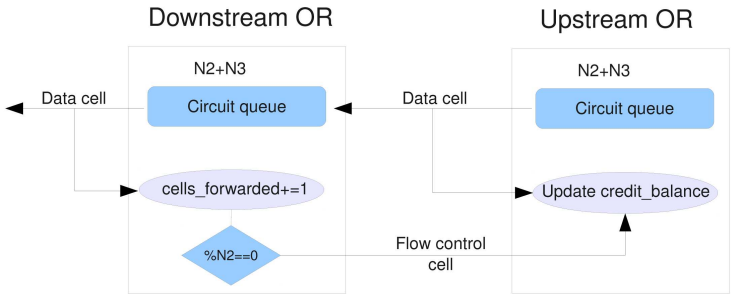


Fig. 3: Credit-based flow control in Tor (N23 scheme)

trol to ensure a steady flow of cells while reducing congestion at the intermediate routers. Implementing per-link flow control in Tor resembles the problem of link-by-link flow control (LLFC) in ATM networks. While the goals of Tor and ATM are certainly different, there are many similarities. Both networks are connection-oriented, in the sense that before applications can send or receive data, virtual circuits are constructed across multiple routers or switches, and both have fixed-sized cells. Furthermore, it has been shown that ATM's credit-based flow control approaches, such as the N23 scheme, eliminate cell loss due to buffer overflows [15], a feature that makes such approaches similar to Tor, where no packets may be dropped to signal congestion.

Figure 3 depicts the N23 scheme that we integrated into Tor, and it works as follows. First, when a circuit is built, each router along the circuit is assigned an initial *credit balance* of $N2+N3$ cells, where $N2$ and $N3$ are system parameters. When a router forwards a cell, it decrements its credit balance by one for that cell's circuit. Each router stops forwarding cells if its credit balance reaches zero. Thus, routers' circuit queues are bounded by $N2 + N3$ cells, and congestion is indicated to upstream routers through this back-pressure. Next, for every $N2$ cells that a router forwards, it sends a flow control cell to the upstream router that contains credit information reflecting the amount of circuit queue space available. On receiving a flow control cell, the upstream router updates its credit balance for the circuit and is allowed to forward more cells if the credit balance is greater than zero.

The algorithm as described assumes a static $N3$. We also developed an adaptive algorithm that adjusts the $N3$ value when there is downstream congestion, which is detected by monitoring the delay that cells experience in the connection's output buffer. When the congestion subsides, $N3$ can increase again. The value of $N3$ is updated periodically and is bounded by a minimum and a maximum value (100 and 500 cells, respectively).

The N23 algorithm has two important advantages over Tor's current flow control. First, the size of the circuit queue is explicitly capped, and guaranteed to be no more than $N2 + N3$ cells. This also ensures steady flow control, as routers always have cells available to forward. The current flow control algorithm in Tor allows the circuit queue of a circuit's intermediate routers to grow up to one circuit window in size, which not only wastes memory, but also results in

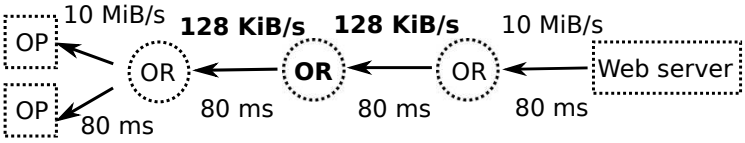


Fig. 4: A simple topology with a middle router bandwidth bottleneck

unnecessary delays due to congestion. In contrast, for typical parameter values ($N3 = 500$ and $N2 = 10$), N23 ensures a strict circuit queue bound of 510 cells, while these queues currently can grow up to 1000 cells in length.

The second advantage is that the adaptive N3 aspect of N23 reacts to congestion within a single link RTT. If congestion occurs at any router in the circuit, the preceding router will run out of credit and must stop forwarding cells until it receives a flow control cell.

5 Experiments and Results

In order to empirically demonstrate the efficacy of our proposed improvements, we evaluate our congestion and flow control algorithms in isolation, using a network emulation platform called ModelNet [30]. Briefly, ModelNet enables the experimenter to specify realistic network topologies annotated with bandwidth, delay, queue length, and other link characteristics, and run real code atop the emulated network.

Our evaluation focuses on performance metrics that are particularly important to the end-user's quality of service. First, we measure *time-to-first-byte*, which is how long the user must wait from the time they issue a request for data until they receive the first byte. The time-to-first-byte is two end-to-end circuit RTTs: one RTT to connect to the destination web server, and a second RTT to issue a request for data (*e.g.*, HTTP `GET`) and receive the first byte of data in response.[6] Second, we measure *overall download time* (including time-to-first-byte). For all experiments, we use the latest development branch of the Tor source code (version `0.2.3.0-alpha-dev`).[7]

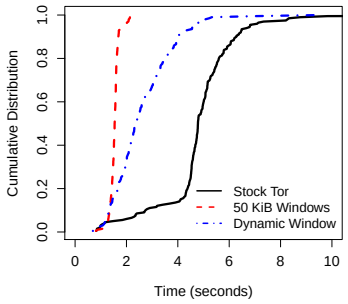
5.1 Small-scale Analysis

Setup. We emulate the topology depicted in Figure 4 on ModelNet where two Tor clients compete for service on the same set of routers with a bandwidth bottleneck at the middle router.[8] One client downloads 300 KiB files, which roughly correspond to the size of an average web page [23]. The second client, a bulk

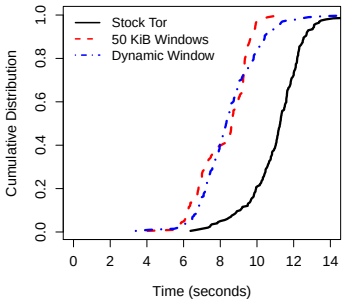
[6] Note that there is a proposal being considered to eliminate one of these RTTs [12].

[7] In our evaluation, we refer to unmodified Tor version `0.2.3.0-alpha-dev` as *stock Tor*, 50 KiB (100 cell) fixed windows as *50 KiB window*, the dynamic window scaling algorithm as *dynamic window*, and the N23 algorithm as *N23*.

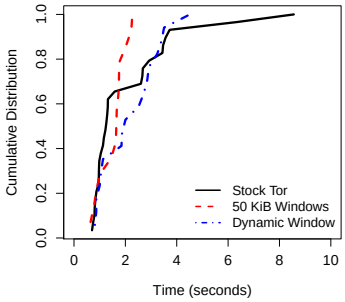
[8] Note that a 128 KiB/s router corresponds to the 65th percentile of routers ranked by observed bandwidth, as reported by the directory authorities. Thus, it is likely to be chosen fairly often by clients.



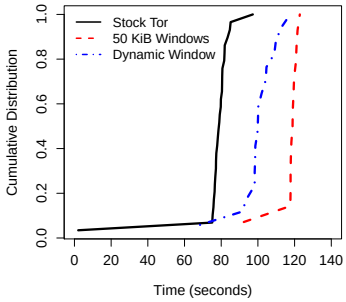
(a) Web client's time-to-first byte



(b) Web client's download time

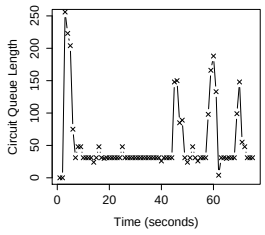


(c) Bulk client's time-to-first-byte

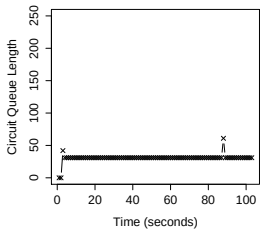


(d) Bulk client's download time

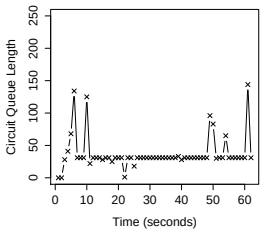
Fig. 5: Performance comparisons for window approaches in a bottleneck topology



(a) Stock Tor



(b) 50 KiB window

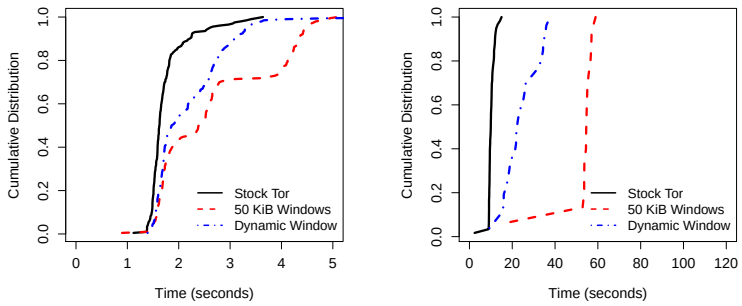


(c) Dynamic window

Fig. 6: Bulk client's circuit queues at the exit router over the course of a download

downloader, fetches 5 MiB files. Both clients pause for a random amount of time between one and three seconds, and repeat their downloads. Each experiment concludes after the web client completes 200 downloads.

End-to-end Window-based Solutions. We first present the results for the window-based flow control solutions. Figure 5(a) shows that the time-to-first-byte for a typical web client using stock Tor is 4.5 seconds at the median, which



(a) Web client's download time (b) Bulk client's download time

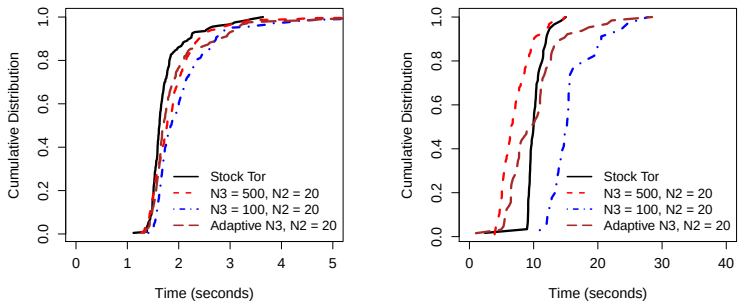
Fig. 7: Performance comparisons for window approaches in a non-bottleneck topology

is unacceptably high for delay-sensitive, interactive web users who must incur this delay for each web request. In addition, stock Tor's circuit queues fluctuate in length, growing up to 250 cells long, and remaining long for many seconds, indicating queuing delays, as shown in Figure 6(a). Reducing the circuit window size to 50 KiB (*e.g.*, one circuit `SENDME` interval) offers a median time-to-first-byte of less than 1.5 seconds, and dynamic windows offer latencies of two seconds at the median. In Figure 5(b), we see that the web client's overall download time is influenced significantly by the high time-to-first-byte, and is roughly 40% faster with 50 KiB and dynamic windows relative to stock Tor. Also, the circuit queues are smaller with the 50 KiB and dynamic windows (see Figures 6(b) and 6(c)).

The bulk client experiences significantly less time-to-first-byte delays (in Figure 5(c)) than the web client using stock Tor. This highlights an inherent unfairness during congestion: web clients' traffic is queued behind the bulk traffic and, consequently, delay-sensitive clients must wait longer than delay-insensitive bulk downloaders to receive their first byte of data. Using a small or dynamic window reduces this unfairness, since the bound on the number of unacknowledged cells allowed to be in flight is lower.

However, Figure 5(d) indicates that the bulk client's download actually takes significantly longer to complete with 50 KiB windows relative to stock Tor. Thus, 50 KiB windows enhance performance for web clients at the cost of slower downloads for bulk clients. The bulk clients experience slower downloads because they keep less data in flight and, consequently, must incur additional round-trip time delays to complete the download. Dynamic windows offer a middle-ground solution, as they ameliorate this limitation by offering an improvement in download time for web clients while penalizing bulk clients less than small windows, but bulk clients are still penalized relative to stock Tor's performance.

We next consider the same topology shown in Figure 4, except we replace the bottleneck middle router with a 10 MiB/s router. In such a topology, congestion is minimal, as evidenced by a median time-to-first-byte of 0.75 s for both the web



(a) Web client's download time (b) Bulk client's download time

Fig. 8: Download time comparison for Tor and N23 in a non-bottleneck network

and bulk clients (regardless of the window size). However, because the 50 KiB and dynamic windows generally keep less data in flight, these solutions offer slower downloads relative to stock Tor, as shown in Figures 7(a) and 7(b).[9]

Despite the improvements in time-to-first-byte in the presence of bandwidth bottlenecks, we find that smaller circuit windows tend to under-utilize the available bandwidth and the dynamic window scaling algorithm is unable to adjust the window size fast enough, as it receives congestion feedback infrequently (only every 100 cells). Furthermore, small windows offer even worse flow control than Tor's current design, since only one window worth of cells is allowed to be in flight, and the exit router must wait for one full circuit RTT until more data can be read and sent down the circuit.

While we elucidate the drawbacks of Tor's current window-based congestion and flow control, we argue that in order to achieve an improvement in both time-to-first-byte and download speed, it is necessary to re-design Tor's fundamental congestion and flow control mechanisms. We next offer an evaluation of a per-link approach implemented in Tor.

Per-link Congestion and Flow Control. We implemented N23 first with fixed values of N2 and N3 (*static N23*) and then with N3 values that react to network feedback (*adaptive N3*). We disabled Tor's window-based flow control, so that exit routers ignored `SENDME`s they received from clients. We discuss the results of adaptive N3 with our large-scale experiments. In this section, we present the results of static N23 for both the bottleneck and non-bottleneck topologies discussed earlier.

For the non-bottleneck circuit experiments, we see in Figure 8(b) that N23 provides a substantial improvement in download time for the 5 MiB downloads compared to stock Tor only for higher values of N3 — 500 cells, comparable to stock Tor's stream window size. The graph shows that there is a 25% decrease in delay for 50% of the bulk downloads when N23 is used. Since the maximum

[9] We also consider the effect of manipulating Tor's circuit windows in combination with circuit-level prioritization. These results are available in Appendix A.

throughput is bounded by W/RTT, where W is the link's TCP window size and RTT is the round-trip time, and since N23's per-link RTT is significantly smaller than a stock Tor's complete circuit RTT, throughput is increased when N23 is used. This improvement suggests that in non-bottleneck scenarios, bulk traffic data cells are unnecessarily slowed down by Tor's flow control at the edges of the circuit. For bursty web traffic, both Tor's current flow control and N23 have similar performance for both fixed and adaptive N3, as shown in Figure 8(a). Also, the time-to-first-byte is the same for the web and bulk experiment, with a median of 0.75 seconds.

Second, for bottleneck scenarios, our results show that smaller values of N3 improve both the download time and time-to-first-byte for the bursty web traffic as shown in Figures 10(a) and 10(b). For example, the web browsing client experiences a 20% decrease in download time for 80% of the requests when N23 is used. Also, the web client's time-to-first-byte is only two seconds for 90% of the requests, whereas for stock Tor's client, 80% of web requests take more than four seconds to receive the first byte. Figure 9 shows that the circuit queue length is upper bounded by $N2 + N3 = 90$ cells.

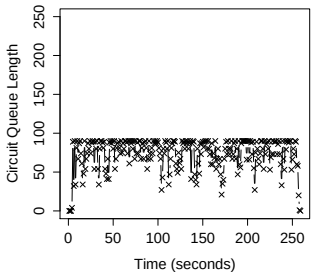
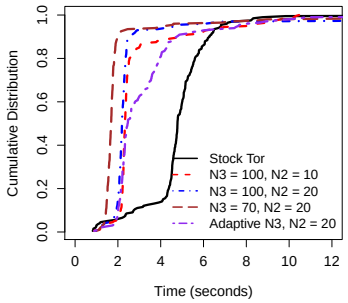


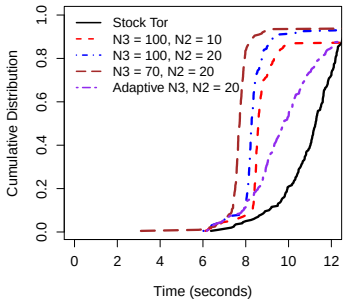
Fig. 9: Circuit queue length with bottleneck: N3 = 70, N2 = 20

To understand how N23 performs with different $N2$ values, we repeated the bottleneck experiments while varying that parameter. Although a higher value for $N2$ has the undesirable effect of enlarging the circuit buffer, it can be seen in Figures 10(a) and 10(b) that when $N3$ is fixed at 100 cells, increasing $N2$ to 20 cells slightly improves both download time and time-to-first-byte. It can be observed from Figure 10(a) that time-to-first-byte is significantly improved by keeping a smaller $N3 = 70$ and a larger $N2 = 20$. Decreasing $N3$ to 70 cells makes up for the increase in the $N2$ zone of the buffer, which means we gain the benefits of less flow control overhead, and the benefits of a small buffer of $N2 + N3 = 90$ cells. While performance is improved for the web client, the bulk client's time-to-first-byte is not affected greatly, as seen in Figure 10(c), but its downloads generally take longer to complete, as we see in Figure 10(d). In addition, adaptive N3 offers improved time-to-first-byte and download times for the web client, while slowing downloads for the bulk client. By N23 restricting the amount of data in flight, the bandwidth consumed by bulk clients is reduced, improving time-to-first-byte and download time for delay-sensitive web clients.

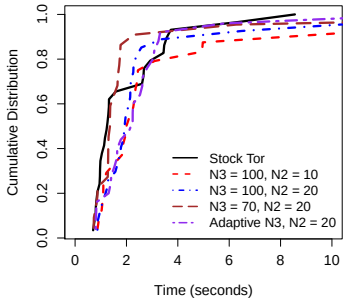
Finally, the bandwidth cost associated with the N23 scheme is relatively low. For instance, with $N2 = 10$, a flow control cell must be sent by each router on the circuit for every 10 data cells forwarded, which requires a 10% bandwidth overhead per router. For $N2 = 20$, a flow control cell is sent for every 20 data cells, which is only a 5% overhead per router. While this cost is higher than Tor's



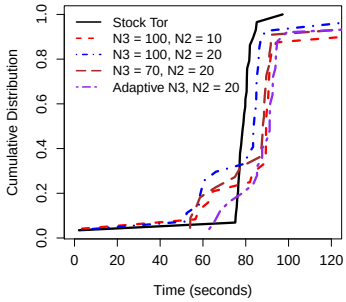
(a) Web client's time-to-first-byte



(b) Web client's download time



(c) Bulk client's time-to-first-byte



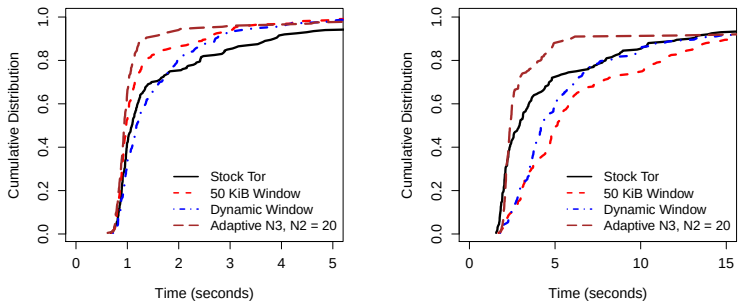
(d) Bulk client's download time

Fig. 10: Performance comparisons for Tor and N23 in a bottleneck topology

window-based flow control (*e.g.*, one stream-level `SENDME` for every 50 data cells is only a 2% overhead per circuit), the cost of N23 is nonetheless modest.

5.2 Larger-scale Experiments

Setup. We next evaluate our proposed congestion and flow control algorithms in a more realistic network topology. We deploy 20 Tor routers on a random ModelNet topology whose bandwidths are assigned by sampling from the live Tor network. Each link's latency is set to 80 ms. Next, to generate a traffic workload, we run 200 Tor clients. Of these, ten clients are bulk downloaders who fetch files between 1–5 MiB, pausing for up to two seconds between fetches. The remaining 190 clients are web clients, who download files between 100–500 KiB (typical web page sizes), pausing for up to 30 seconds between fetches. This proportion of bulk-to-non-bulk clients approximates the proportion observed on the live Tor network [19]. Circuit-level prioritization is disabled for this experiment.
Results. For web clients, Figure 11(a) shows that both the 50 KiB fixed and dynamic windows still offer improved time-to-first-byte. However, both algorithms perform worse than stock Tor in terms of overall download time (Figure 11(b)).



(a) Web client's time-to-first-byte (b) Web client's download time

Fig. 11: Performance results for large-scale experiments

This is because smaller windows provide less throughput than larger windows when there is no bottleneck. Thus, non-bottlenecked circuits are under-utilized.

N23 with the adaptive N3 algorithm, in contrast, has the ability to react to congestion quickly by reducing routers' queue lengths, causing back pressure to build up. Consequently, Figures 11(a) and 11(b) show that N23 offers an improvement in both time-to-first-byte *and* overall download time. This experiment again highlights the potential negative impact of 50 KiB and small dynamic windows, since even in a larger network with a realistic traffic load, smaller windows offer worse performance for typical delay-sensitive web requests relative to Tor's current window size. Thus, to achieve maximal improvements, we suggest that Tor adopt N23 congestion and flow control.

6 Discussion

Having empirically evaluated our proposed congestion and flow control approaches, we next discuss a variety of open issues.

6.1 Incremental Deployment

In order for our proposed congestion and flow control mechanisms to be practical and easily deployable on the live Tor network, it is important that any modifications to Tor's router infrastructure be incrementally deployable. Any solutions based on Tor's existing window-based flow control require upgrades only to the exit routers; thus they can be slowly deployed as router operators upgrade. N23 may also be deployed incrementally, however, clients may not see substantial performance benefits until a large fraction of the routers have upgraded.

6.2 Anonymity Implications

A key question to answer is whether improving Tor's performance and reducing congestion enables any attack that was not previously possible. It is well

known that Tor is vulnerable to congestion attacks wherein an attacker constructs circuits through a number of different routers, floods them with traffic, and observes if there is an increase in latency on a target circuit, which would indicate a shared router on both paths [20]. More recent work has suggested a solution that would mitigate bandwidth amplification variants of this attack, but not the shared router inference part of the attack [11]. We believe that by reducing congestion (and specifically, by bounding queue lengths), our proposed techniques may increase the difficulty of mounting congestion attacks.

However, if only a fraction of the routers upgrade to our proposals and if clients only choose routers that support the new flow control, then an adversary may be able to narrow down the set of potential routers that a client is using. Thus, it is important to deploy any new flow control technique after a large fraction of the network has upgraded. Such an incremental deployment can be controlled by setting a flag in the authoritative directory servers' consensus document, indicating that it is safe for clients to use the new flow control.

Another well-studied class of attack is end-to-end traffic correlation. Such attacks endeavor to link a client with its destination when the entry and exit points are compromised, and these attacks have been shown to be highly accurate [1,21,22,26,28]. Reducing latency might improve this attack; however, Tor is already highly vulnerable, so there is little possibility for additional risk.

Finally, previous work has shown that round-trip time (RTT) can be used as a side channel to infer a possible set of client locations [14]. By decreasing the variance in latency, we might expose more accurate RTT measurements, thus improving the effectiveness of this attack. However, reducing congestion does not enable a new attack, but rather may potentially increase the effectiveness of a known attack. To put this attack in perspective, Tor's design has already made many performance/anonymity trade-offs, and thus, we believe that our performance improvements outweigh any potential decrease in anonymity brought about by reducing the variance in latency.

7 Conclusion

We seek to improve Tor's performance by reducing unnecessary delays due to poor flow control and excessive queuing at intermediate routers. To this end, we have proposed two broad classes of congestion and flow control. First, we tune Tor's existing circuit windows to effectively reduce the amount of data in flight. However, our experiments indicate that while window-based solutions do reduce queuing delays, they tend to suffer from poor flow control, underutilizing the available bandwidth, and consequently, smaller windows provide slower downloads than unmodified Tor.

To solve this problem, we offer a fresh approach to congestion and flow control in Tor by designing, implementing, and experimentally evaluating a per-link congestion and flow control algorithm from ATM networks. Our experiments indicate that this approach offers the promise of reduced web page response times and faster overall web page downloads.

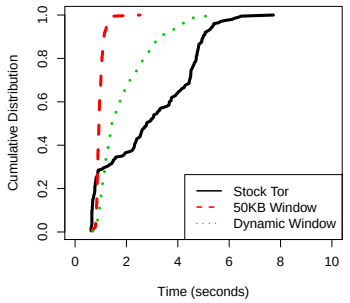
References

1. Bauer, K., McCoy, D., Grunwald, D., Kohno, T., Sicker, D.: Low-resource routing attacks against Tor. In: Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2007). Washington, DC, USA (October 2007)
2. Brakmo, L.S., O'Malley, S.W., Peterson, L.L.: TCP Vegas: New techniques for congestion detection and avoidance. In: Proceedings of the conference on Communications architectures, protocols and applications. pp. 24–35. SIGCOMM '94, ACM, New York, NY, USA (1994), http://doi.acm.org/10.1145/190314.190317
3. Chen, F., Perry, M.: Improving Tor path selection. https://gitweb.torproject.org/torspec.git/blob_plain/HEAD:/proposals/151-path-selection-improvements.txt (July 2008)
4. Dhungel, P., Steiner, M., Rimac, I., Hilt, V., Ross, K.W.: Waiting for anonymity: Understanding delays in the Tor overlay. In: Peer-to-Peer Computing. pp. 1–4. IEEE (2010)
5. Dingledine, R.: Prop 168: Reduce default circuit window. https://gitweb.torproject.org/torspec.git/blob_plain/HEAD:/proposals/168-reduce-circwindow.txt (August 2009)
6. Dingledine, R.: Research problem: adaptive throttling of Tor clients by entry guards. https://blog.torproject.org/blog/research-problem-adaptive-throttling-tor-clients-entry-guards (September 2010)
7. Dingledine, R., Mathewson, N.: Anonymity loves company: Usability and the network effect. In: Workshop on the Economics of Information Security (June 2006)
8. Dingledine, R., Mathewson, N.: Tor Protocol Specificiation. https://gitweb.torproject.org/tor.git/blob_plain/HEAD:/doc/spec/tor-spec.txt (2010)
9. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: Proceedings of the 13th USENIX Security Symposium (August 2004)
10. Dingledine, R., Murdoch, S.: 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)
11. Evans, N., Dingledine, R., Grothoff, C.: A practical congestion attack on Tor using long paths. In: Proceedings of the 18th USENIX Security Symposium (August 2009)
12. Goldberg, I.: Prop 174: Optimistic data for Tor: Server side. https://trac.torproject.org/projects/tor/ticket/1795
13. Goldschlag, D.M., Reed, M.G., Syverson, P.F.: Hiding routing information. In: Proceedings of Information Hiding: First International Workshop. Springer-Verlag, LNCS 1174 (May 1996)
14. Hopper, N., Vasserman, E.Y., Chan-Tin, E.: How much anonymity does network latency leak? In: Proceedings of CCS 2007 (October 2007)
15. Jain, R.: Congestion control and traffic management in ATM networks: Recent advances and a survey. COMPUTER NETWORKS AND ISDN SYSTEMS 28, 1723–1738 (1995)
16. Kiraly, C., Bianchi, G., Cigno, R.L.: Solving performance issues in anonymiziation overlays with a L3 approach. University of Trento Information Engineering and Computer Science Department Technical Report DISI-08-041, Ver. 1.1 (September 2008)

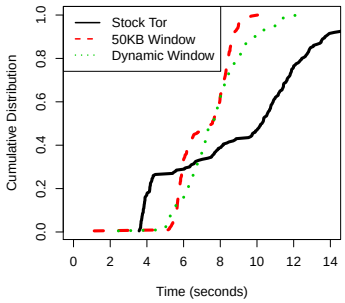
17. Kung, H.T., Blackwell, T., Chapman, A.: Credit-based flow control for ATM networks: credit update protocol, adaptive credit allocation and statistical multiplexing. SIGCOMM Comput. Commun. Rev. 24, 101–114 (October 1994), http://doi.acm.org/10.1145/190809.190324
18. Loesing, K.: Measuring the Tor network: Evaluation of client requests to the directories. Tor Project Technical Report (June 2009)
19. McCoy, D., Bauer, K., Grunwald, D., Kohno, T., Sicker, D.: Shining light in dark places: Understanding the Tor network. In: Proceedings of the 8th Privacy Enhancing Technologies Symposium (July 2008)
20. Murdoch, S.J., Danezis, G.: Low-cost traffic analysis of Tor. In: Proceedings of the 2005 IEEE Symposium on Security and Privacy. IEEE CS (May 2005)
21. Murdoch, S.J., Zieliński, P.: Sampled traffic analysis by Internet-exchange-level adversaries. In: Proceedings of Privacy Enhancing Technologies Workshop (PET 2007) (June 2007)
22. Øverlier, L., Syverson, P.: Locating hidden servers. In: Proceedings of the 2006 IEEE Symposium on Security and Privacy. IEEE CS (May 2006)
23. Ramachandran, S.: Web metrics: Size and number of resources. https://code.google.com/speed/articles/web-metrics.html
24. Reardon, J., Goldberg, I.: Improving Tor using a TCP-over-DTLS tunnel. In: Proceedings of the 18th USENIX Security Symposium (August 2009)
25. Savage, S., Cardwell, N., Wetherall, D., Anderson, T.: TCP congestion control with a misbehaving receiver. SIGCOMM Comput. Commun. Rev. 29, 71–78 (October 1999)
26. Serjantov, A., Sewell, P.: Passive attack analysis for connection-based anonymity systems. In: Proceedings of ESORICS 2003 (October 2003)
27. Sherwood, R., Bhattacharjee, B., Braud, R.: Misbehaving TCP receivers can cause Internet-wide congestion collapse. In: Proceedings of the 12th ACM conference on Computer and communications security. pp. 383–392. CCS '05, ACM, New York, NY, USA (2005)
28. Shmatikov, V., Wang, M.H.: Timing analysis in low-latency mix networks: Attacks and defenses. In: Proceedings of ESORICS 2006 (September 2006)
29. Tang, C., Goldberg, I.: An improved algorithm for Tor circuit scheduling. In: Keromytis, A.D., Shmatikov, V. (eds.) Proceedings of the 2010 ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010. ACM (2010)
30. Vahdat, A., Yocum, K., Walsh, K., Mahadevan, P., Kostić, D., Chase, J., Becker, D.: Scalability and accuracy in a large-scale network emulator. SIGOPS Oper. Syst. Rev. 36, 271–284 (December 2002), http://doi.acm.org/10.1145/844128.844154
31. Viecco, C.: UDP-OR: A fair onion transport. HotPETS (July 2008)
32. Wang, Z., Crowcroft, J.: Eliminating periodic packet losses in the 4.3-Tahoe BSD TCP congestion control algorithm. SIGCOMM Comput. Commun. Rev. 22, 9–16 (April 1992), http://doi.acm.org/10.1145/141800.141801
33. Wright, M.K., Adler, M., Levine, B.N., Shields, C.: The predecessor attack: An analysis of a threat to anonymous communications systems. ACM Trans. Inf. Syst. Secur. 7(4), 489–522 (2004)

A Effects of Circuit Prioritization

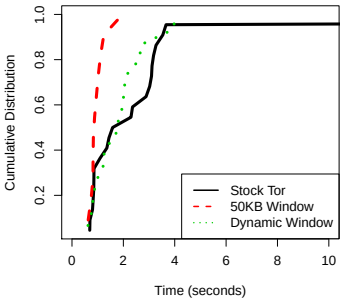
To mitigate the unfairness that may exist when bursty web circuits compete with bulk transfer circuits for router bandwidth, circuit-level prioritization has been



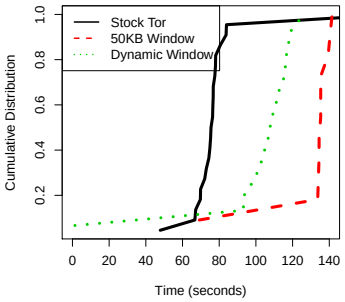
(a) Web client's time-to-first byte



(b) Web client's download time



(c) Bulk client's time-to-first-byte



(d) Bulk client's download time

Fig. 12: Performance comparisons for window-based congestion control in combination with circuit scheduling prioritization

proposed [29] to enable routers to process bursty circuits ahead of bulk circuits. Here, we combine small and dynamic circuit window with circuit scheduling prioritization.[10] For the web client using stock Tor, the time-to-first-byte is reduced from 4.5 seconds to 3 seconds, and the time-to-first-byte for 50 KiB and dynamic windows are roughly the same. However, as shown in Figure 12(a), roughly 25% of requests experience no significant improvement when using small or dynamic circuit windows. For these same requests, stock Tor's large window allows more data in flight without acknowledgment and, as shown in Figure 12(b), induces faster overall downloads. However, for the remaining 75%, small and dynamic windows offer faster downloads. The bulk client's time-to-first-byte and overall download times are not significantly altered by the circuit prioritization, as shown in Figures 12(c) and 12(d), relative to non-prioritized circuit scheduling (see Figures 5(c) and 5(d)). This is consistent with the claims made by Tang and

[10] For this experiment, we set `CircuitPriorityHalflifeMsec` to 30 seconds, the current value used on the live Tor network.

Goldberg [29] that priority-based circuit scheduling does not significantly effect bulk clients' performance.

We found that circuit-level prioritization offered no noticeable change in performance when using N23 flow control.