Congestion Control in TOR

Ambuj Ojha (apo249@nyu.edu), Neha Sharma (ns3786@nyu.edu)
Advisor: Anirudh Sivaraman
New York University, Courant Institute of Mathematical Sciences,
CSCI-GA.2620-001: Project Report

***Abstract*—Tor is the most popular anonymous communication systems by a mile. With increasing popularity, though, Tor is also faced with increasing load and high flow latencies. One of the fundamental reasons for this is flawed network congestion control. As recent work has shown, the current Tor design has a very naive congestion control setup which interacts in unpredictable ways with the underlying TCP transport layer, leading to flows suffering big latencies. Although, finding good solutions to the high latencies problem is hard for anonymity overlays due to the long end-to-end delay in such networks, it is essential in order to increase the popularity of the system and improve its anonymity properties. In this report, we go over the design of the original TOR network and present the problems it suffers from. Then we discuss many of the ideas introduced to solve those problems, with special emphasis on the BackTap protocol, a recent design which outperforms others in simulations. Finally, we identify certain issues with BackTap and propose an improved version of the same called Marut which we analyze theoretically and through simulations. Marut has latency-based end-to-end congestion control with feedback derived from inter-hop signalling. The rest of the design including hop-by-hop reliability is same as BackTap. The resulting overlay is able to get congestion feedback from anywhere in the network to the necessary end-hosts much quicker than BackTap and thus react rapidly to varying network conditions. We show that it allocates available resources (between bulk and web circuits) more evenly than BackTap and the current Tor design which is beneficial in terms of both fairness and anonymity.**

I. Introduction

The Internet often conveys the impression of providing an anonymous communication channel, but in fact the underlying infrastructure was not designed with anonymity in mind: IP addresses serve as identifiers and are generally accessible by the recipient of a message as well as anybody with access to the communication channel along the route. These address can be mapped to a particular Internet Service Provider (ISP), and this ISP can then provide information about what customer that IP address was leased to.

However, the ability to conduct anonymous communication using Internet is critical in this age. For instance, people in a country with a repressive political regime may want to use Internet-based anonymity servers in other countries to avoid persecution for their political opinions. Similarly there are people who are activists or whistle blowers but being dependent on an organization, or afraid of revenge, may divulge serious misuse, which should be revealed. Anonymous tips can be used as an information source by newspapers, as well as by police departments, soliciting tips aimed at catching criminals. In effect, the digital communication should be able to provide an option to protect personal privacy of users, as well as their freedom of speech and ability to conduct confidential communication.

The Tor network has become the prime example for anonymity overlays to preserve online anonymity. It implements *onion-routing* [14], the standard architecture for low latency anonymous communication systems which routes application layer data along a cryptographically secured virtual circuit through an overlay network. Unfortunately, Tor's current overlay design suffers from severe performance issues. This is a significant issue because low latency is what gives Tor its edge over other high-latency anonymous communication platforms. Performance is also critical as better performance would ensure adoption by more users which will further enhance anonymity.

In our report, we discuss the current design of Tor and the congestion control measures implemented in it [2]. We then analyze various reasons [1], [3] of why these congestion control measures don't work so well and what can be done to improve the performance in Tor. In particular, we try to understand the feedback gap that exists between the outflow and the inflow at a Tor relay and the reasons as to why this feedback gap is one of the main causes of congestion. We then discuss solutions proposed over the years to this problem and concentrate on Backtap algorithm, which outperforms all proposed designs in simulations. BackTap is a Back-pressure based Transport Protocol [1] and represents the state of the art research in the field of congestion control in Tor.

Finally we propose a modification to BackTap that we derived from the theory of Congestion Control as developed in seminal papers like BBR [6]. Our protocol, Marut, makes the congestion control end-to-end and we show that the new design performs even better than BackTap over latency and fairness metrics.

Designing a congestion control protocol for anonymity overlay raises interesting challenges[1]. 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 and slower feedback loops. Second, anonymity demands that control feedback must not reveal user identities, neither directly nor in- directly. Therefore, feedback must be limited and well considered. Third, relay operators donate resources, in particular bandwidth, to the anonymity overlay. To incentivize relay operation, anonymity traffic should therefore not be overly aggressive.

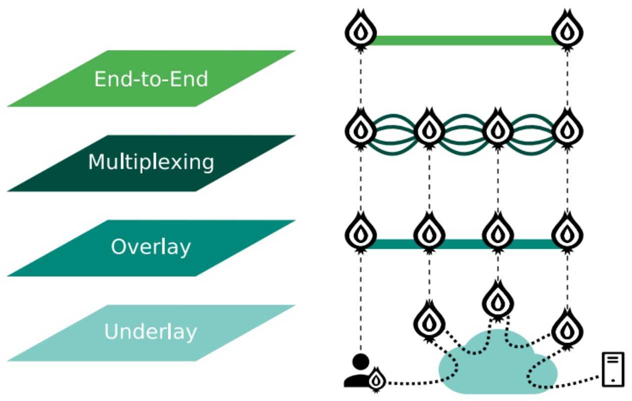


Fig. 1. Overview of the layered Tor architecture: relays build a TCP-based overlay, multiplexing circuits, while using an end-to-end sliding window.

II. Current TOR Design

A. Overview

The Tor network is an overlay network; each onion router (OR), also referred as relays, runs as a normal user-level process without any special privileges. Each onion router maintains a TLS connection to every other onion router. Each user runs local software called an onion proxy (OP) to fetch directories, establish circuits across the network, and handle connections from user applications. These onion proxies accept TCP streams and multiplex them across the circuits. The onion router on the other side of the circuit connects to the requested destinations and relays data. The Fig. 1 shows the layers in the TOR architecture.

We first explain three terms that we will use throughout the report. **Connections** are TCP/TLS links between neighboring relays (any two adjacent relays uses only one TCP connection and hence multiplexes data of different circuits within this connection. **Circuits** are logical paths (consisting of ORs) through the network chosen by the clients in order to route the data to achieve anonymity. A circuit traverses multiple overlay hops and multiple connections and each node (or ”onion router” or ”OR”) in the path knows its predecessor and successor, but no other nodes in the circuit. **Stream** is Tor's notion for the application layer data carried through a circuit, belonging to one anonymized TCP session (only end points of a circuit can associate cells with a stream)

Traffic passes along the TCP connections between relays in fixed-size cells (512 bytes). There are two types of cells: First type are *Control Cells* which are interpreted by the node that receives them. Control cell commands are: padding, create or created, and destroy. The second type are *Relay Cells* that carry end-to-end stream data. Relay commands are: relay data, relay begin, relay end, relay teardown, relay connected, relay extend and relay extended, relay truncate and relay truncated, relay sendme and relay drop. Tor relays forward cells according to the circuit switching principle, but the individual relay does not know about the full path of the circuit. 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. 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.

We now provide a quick overview of how circuits are formed in Tor followed by how data transfer happens over the circuit. We will refer to the Fig. 2 for our discussion.

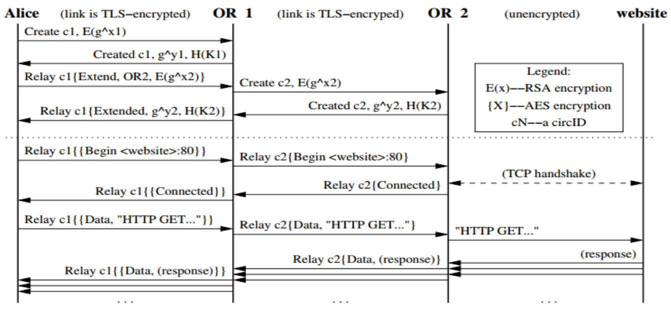


Fig. 2. Alice builds a two-hop circuit and begins fetching a web page.

A users OP constructs circuits incrementally, negotiating a symmetric key with each OR on the circuit, one hop at a time. To begin creating a new circuit, the OP (call her Alice) sends a create cell to the first node in her chosen path (call him Bob). This create cell accomplishes the Diffie-Hellman handshake between Alice and once the circuit has been established, Alice and Bob can send one another relay cells encrypted with the negotiated key. To extend the circuit further, Alice sends a relay extend cell to Bob, specifying the address of the next OR (call her Carol), and an encrypted key for Diffie-Hellman handshake in the payload for her. Bob copies the half-handshake into a create cell, and passes it to Carol to extend the circuit. (Bob chooses a new circID not currently used on the connection between him and Carol. Alice never needs to know this circID;) When Carol responds with a created cell, Bob wraps the payload into a relay extended cell and passes it back to Alice. Now the circuit is extended to Carol, and Alice and Carol share a common key. In the same way, circuit can be extended to a third node or beyond.

B. Congestion Control

If enough users choose the same OR-to-OR connection for their circuits, that connection can become saturated. Tor uses end-to-end (E2E) sliding window mechanism between a clients Tor software and an exit relay in order to control the number of cells in flight at any given time. This mechanism of throttling works at both circuit level and stream level.

- Circuit-level throttling: 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 circuits transmission window by 100.

- Stream-level throttling: On the stream level also, an analogous mechanism exists. The stream-level windows fixed size is 500 cells, and stream-level SENDMEs 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.

C. Problems with Original Design

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. This is where the feedback gap appears, which is illustrated in the Fig. 3 also. In the current Tor design, 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 eventually throttle the source. However, such an approach of throttling in the application layer is not sufficient for good network performance.

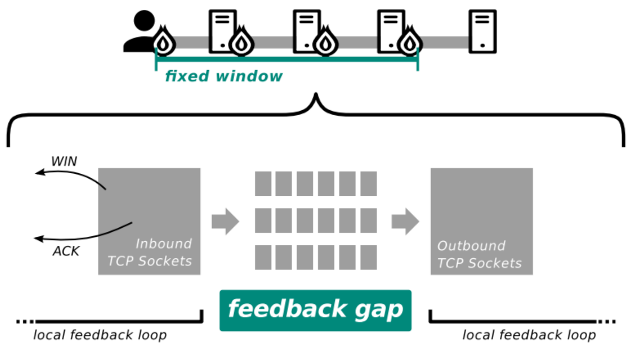


Fig. 3. Feedback gap visible in current Tor design.

Due to the fixed size of these windows, Tor lacks adaptivity. The 1000 cells can be significantly more than the bandwidth-delay product of a circuit, so that long queues build up often either 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. 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. 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. Since circuits are multiplexed over joint TCP connections, a relay therefore cannot selectively throttle cells from one specific circuit. Stopping to read from one socket could result in massive head-of-line blocking for other circuits.

III. Survey of Research

Since Tor's introduction more than a decade ago, it has received significant attention in the research community (and beyond). Here we discuss two proposed solutions that we surveyed along with the BackTap protocol that we discuss in the next section.

A. N23 Algorithm

The authors of this paper suggest a per-link flow control algorithm that works as follows: 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 cells circuit. Each router stops forwarding cells if its credit balance reaches zero. Thus, routers circuit queues are upper bounded by N2 + N3 cells, and congestion is indicated to upstream routers through this back-pressure. Next, for every N2 cells forwarded, the downstream router sends a flow control cell to the upstream router that contains credit information reflecting its available circuit queue space. On receiving a flow control cell, the upstream router updates the circuits credit balance and may forward cells only if the credit balance is greater than zero. This algorithm does substitute the end-to-end window by a back pressure based hop-by-hop window for congestion control but head-of-line blocking and the choice of window parameters remain open issues.

B. PCTCP Algorithm

In this paper, the authors propose an algorithm based on creating a separate kernel-mode TCP connection for each circuit for Tor. This algorithm is conceptually identical to TCP-over-DTLS design that was introduced by Reardon and Goldberg. In particular, reliable in-order delivery of data is implemented between every two communicating ORs and congestion control is performed at the circuit granularity. The main difference with current Tor design is that when an OR receives an extend command cell to another OR, PCTCP always establishes a new TCP connection between these two relays. The queueing architecture is same as that of Tor. This helps to eliminate the contention that occurs among circuits when they share the same connection output buffer, as each circuit queue is mapped to a single output and a single input connection buffer. When a circuit is torn down, its corresponding TCP connections are closed. This algorithm however 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.

IV. BackTap Design

A. Overview & Congestion Control

BackTap performs reliable, in-order delivery and flow control on circuit granularity in the application layer. Since reliability is pulled into the application layer, TCP connections between Tor Nodes are replaced by UDP transport. In BackTap, arriving cells from the predecessor are read from the UDP socket and the cryptographic operations demanded by the anonymity overlay (adding encryption or peeling off encryption) are performed. The cell is subsequently enqueued in the respective circuit queue. Queue management is done as follows

1) *tailSeq* points to the last cell that has been received in order. When a new cell is received *tailSeq* is updated and an ACK cell is sent upstream.
2) *headSeq* points to the front-most unacknowledged cell. Upon receiving acknowledgement from the downstream node (ACK cell) it is incremented and the cell freed up.
3) *nextTxSeq* is incremented when a cell is forwarded to the downstream relay. This forwarding also triggers the transmission of corresponding flow control feedback cell (FWD) upstream.

The upstream node makes use of FWD cells to determine a sending window (*cwnd*) based on the provided feedback. It is allowed to keep at most *cwnd* cells in the transmission pipe.

The *cwnd* adjustment strategy follows a delay-based approach, based on the latency experienced before receiving an FWD. It therefore adjusts both to the outflow in the downstream node (because only then is FWD issued) and to the conditions of the network path between consecutive relays (because this path, too, will influence the delays).

Tying transmissions to the forwarding of the corresponding cell yields tight feedback coupling between consecutive overlay hops (Fig. 4): if the *cwnd* adjustment control loop of one overlay hop in a circuit results in a throttled outflow of cells, the arrival delay over the preceding overlay hop will increase accordingly within a one-way local-hop delay. This way, hop-by-hop feedback emerges, and backpressure propagates back to the source.

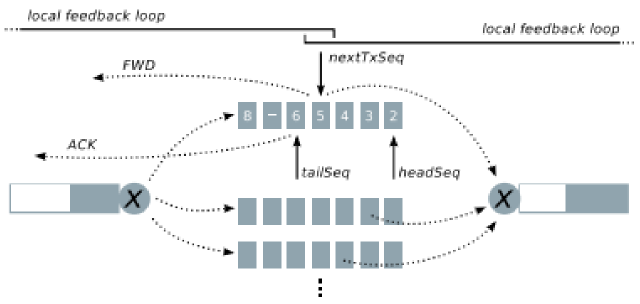


Fig. 4. Feedback between hops is tightly coupled gap in BackTap

Each node determines the size of its local *cwnd* based on the feedback from the next hop downstream. Following the ideas of TCP Vegas [16], *diff*, the difference between expected and actual window size is calculated as

$$diff = cwnd * \frac{actualRtt}{baseRtt} - cwnd \tag{1}$$

RTTs are sampled based on the flow control feedback by measuring the time difference between sending a cell and receiving the respective FWD. The *actualRtt* is estimated per circuit by taking the smallest RTT sample during the last RTT. 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*, the sending window, *cwnd*, is adjusted every RTT as follows:

$$cwnd = \begin{cases} cwnd + 1 & if\, diff < \alpha \\ cwnd - 1 & if\, diff > \beta \end{cases} \tag{2}$$

Since *cwnd* changes by at most one, it follows an additive increase additive decrease (AIAD) policy. Typically α and β are chosen as 2 and 4 (here measured in cells). Therefore, one may expect that *cwnd* does not exceed the bandwidth-delay product by much. 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, because it aims for maintaining a non-empty queue at the bottleneck.

B. Reliability

BackTap implements reliability on a per-hop basis. Reliability feedback is provided as early as possible, namely upon arrival of a cell, by sending a corresponding ACK upstream. Cell sequence numbers are used 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 BackTap reliability scheme differs from original TOR's TCP mechanism is where the circuit queue in the downstream node and the coupling between consecutive hop feedback loops comes into play. 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 [15]. 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.

C. Anonymity Properties

BackTap adds new header fields: a sequence number (4 Byte) and a field for flags (1 Byte) to the TOR relay cell. For ACKs and FWDs, a separate message format, much smaller than a Tor cell is introduced. 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, from different 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 messages fit in.

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. BkTap modifications affect the cell preamble only. The DTLS encryption between consecutive relays can shield the preamble from observers on the wire.

D. *Problems with BackTap*

- **Slow Congestion Update**: At each relay, the queues need to fill up in order for upstream node to detect congestion and decrease sending window. Effectively, queues at all the hops end up filling before the sender becomes aware of congestion.
- **Head of Line Blocking**: If two streams share a circuit, both get affected by the queues building up due to back pressure even when only one of the streams may be bulky.
- **Throughput Unfairness**: Relays perform round robin scheduling between competing circuits, which can lead to significant unfairness at the circuit level between bulk (which fillup queues more) and web circuits (which fillup queues less)

V. MARUT

All the three problems described above result from the the queues which still build up in BackTap at every TOR node in any circuit. Admittedly the queues in BackTap will be shorter in practice than in the original TOR design but a protocol whose congestion control does not depend on queues building up at all by operating at the intersection of the application-limited and bandwidth-limited regions [6] should in theory perform better than one which operates within the bandwidth limited layer.

Essentially what we want is to move the congestion and queue buildup from the inside of the TOR-network to its edges. If TOR end-nodes could detect the optimum window size that they should maintain given the congestion along any circuit we would decrease the queue build up at every node. However, unlike in BackTap, in order to detect the congestion along any circuit as compared to another, it would not make sense to compare their RTTs since two circuit starting from the same TOR-node could have completely different paths (Fig. 5). Two circuits RTTs can only be compared to the extent that they share a hop (connection) between TOR nodes. Hence, since a circuit consists of multiple hops the design we propose intends to compute the congestion of each hop and then relay it to the upstream node which constructs a cumulative feedback to be sent on further upstream. This way the TOR end-node gets a cumulative congestion feedback for every circuit that starts from it and using this feedback, it modifies the *cwnd* for that circuit accordingly (Fig. 6).

A. *Design Details*

We propose **Marut**, with the following design choices

- end-to-end congestion control in the application layer
- hop-to-hop reliability implemented in application layer
- use UDP channels between TOR nodes same as BkTap
- overload FWD cells from BackTap to send circuit congestion parameters upstream

Marut allows congestion updates to reach the end-host within one RTT for the circuit. For any circuit

- Each TOR node calculates the congestion parameter,'*diff*', which defines congestion till the next node. Same as in Eq (1)

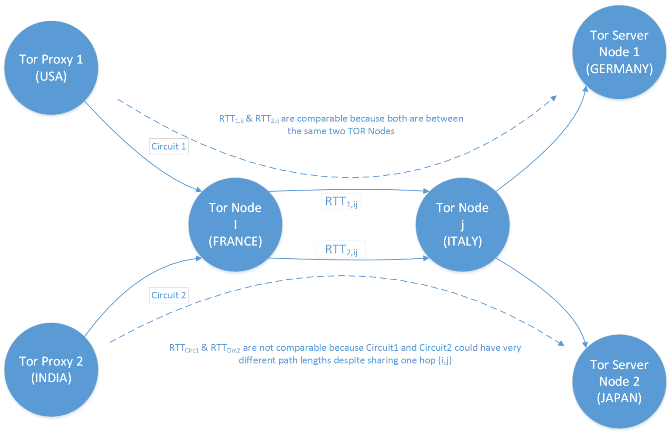

Tor Proxy 1
(USA)
Tor Server
Node 1
(GERMANY)
$RTT_{1,ij}$ & $RTT_{2,ij}$ are comparable because both are between the same two TOR Nodes
Circuit 1
Tor Node
I
(FRANCE)
$RTT_{1,ij}$
$RTT_{2,ij}$
Tor Node
j
(ITALY)
Circuit 2
RTT_{Circ1} & RTT_{Circ2} are not comparable because Circuit1 and Circuit2 could have very different path lengths despite sharing one hop (i,j)
Tor Proxy 2
(INDIA)
Tor Server
Node 2
(JAPAN)

Fig. 5. End-to-end RTTs for circuits are not comparable

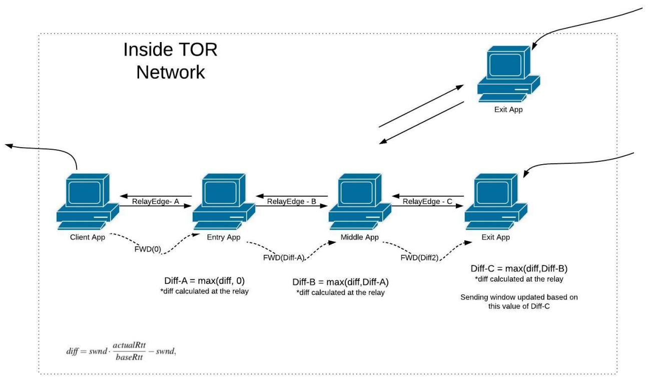

Inside TOR
Network
Exit App
RelayEdge- A
RelayEdge - B
RelayEdge - C
Client App
Entry App
Middle App
Exit App
FWD(0)
FWD(Diff-A)
FWD(Diff2)
Diff-A = max(diff, 0)
*diff calculated at the relay
Diff-B = max(diff,Diff-A)
*diff calculated at the relay
Diff-C = max(diff,Diff-B)
*diff calculated at the relay
Sending window updated based on
this value of Diff-C
$diff = swnd \cdot \frac{actualRtt}{baseRtt} - swnd,$

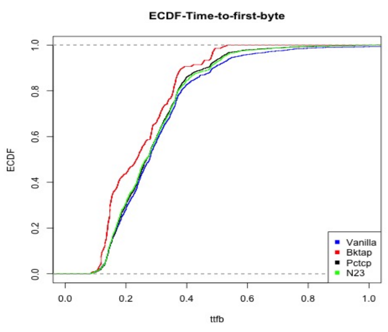
Fig. 6. Flow of feedback cell FWD: information about congestion in downstream is encapsulated in the FWD cell sent to the upstream relay.

- FWD cell sent by any Tor node now contains a new field, '*c_diff*', which encapsulates information on congestion as observed to the downstream of this node
- $CumulativeCongestion = max(diff, c\_diff)$
- The FWD cell current TOR node sends upstream would have $c\_diff = CumulativeCongestion$

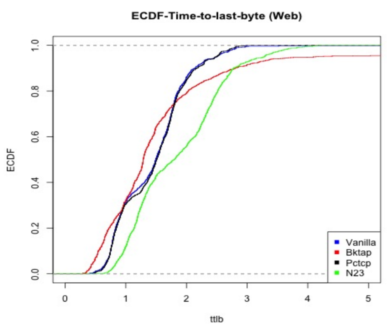
Congestion window is updated using $CumulativeCongestion$ only at the TOR end-nodes and not at the middle nodes

B. *Anonymity Properties*

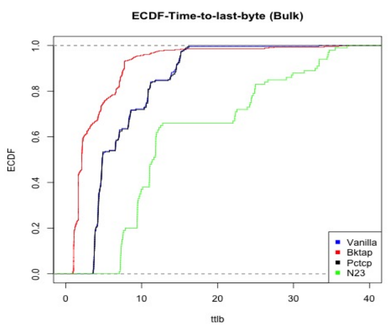
As in the BackTap algorithm, the FWD cell preamble containing the '*c_diff*', is encrypted using DTLS so that network sniffing attacks are mitigated. Having several data cells, ACK and FWD cells encapsulated in one UDP packet also makes attacks difficult. In case of a malicious TOR Node, we do leak some extra information but the problem is alleviated because of several factors. Firstly, we only pass along the max of '*diff*' upstream for any circuit, which means that a particular value could potentially belong to any hop on the circuit. Hence under Tors attacker model, i.e., a local adversary with a partial view of the network, would not find it easy to break anonymity by identifying all the nodes for a



(a) Bulk and Web Circuits

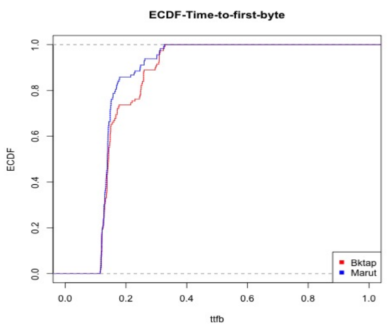


(b) Web Circuits

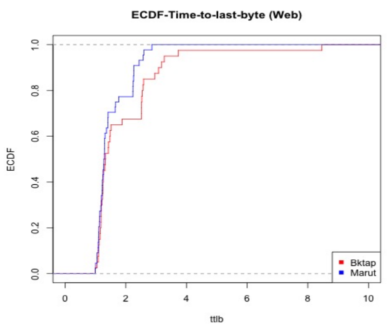


(c) Bulk Circuits

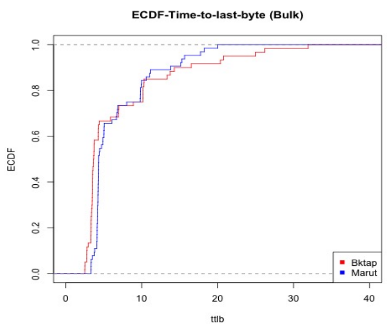
Fig. 7. Time to download files over Tor for a 100 circuit, 100 relay in Dumbbell topology and a 3 node circuit length



(a) Bulk and Web Circuits

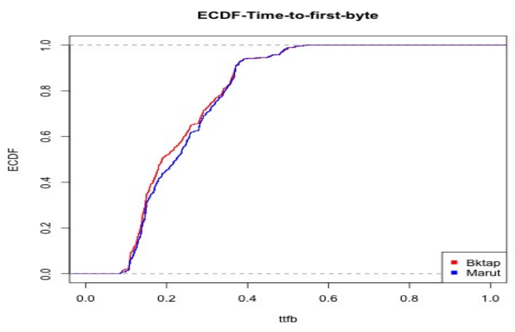


(b) Web Circuits

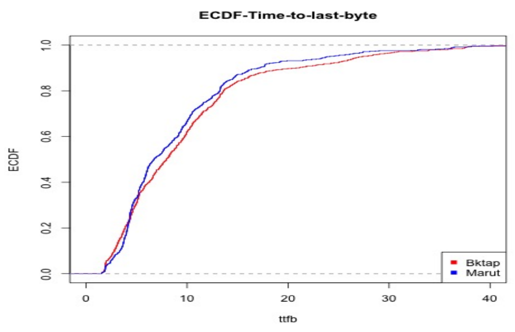


(c) Bulk Circuits

Fig. 8. Time to download files over Tor for a 20 circuit Tree topology



(a) Bulk Circuits only



(b) Bulk Circuits only

Fig. 9. Time to download files over Tor when all the circuits are bulk download

circuit. Also in an actual network the presence of many flows makes the value contained in a '*diff*' difficult to exploit. Hence, Marut does not significantly impair the anonymity provided by TOR over and above BackTap.

VI. Evaluation

For our evaluation we have used nstor which is the open source Tor module for ns-3 as developed by Tschorch et al [10]. This module has been modeled along the lines of the original Tor software focusing on the network aspects. It allows direct and reproducible comparisons of protocol design alternatives. In addition, with Network Simulation Cradle (NSC), 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 key point in setting up an environment for a valid evaluation of Tor is to model the overlay appropriately. In order to recreate the results mentioned in the BackTap paper [1], we follow guidelines developed in [7]. In our simulations, we have used a dumbbell topology for larger-scale, more realistic experiments. Since approximately 93% of all Tor relays are currently hosted in North America or Europe [11], the dumbbell topology can be thought to approximate the geographical clustering. We use nstor modules for creating circuits with network characteristics similar to real world. The module nstor adjusts the circuit delay according to the iPlane [8] RTT measurements and the client access rates according to

Akamais state of the Internet report [9] by inverse transform sampling.

In accordance to the model proposed in [7], 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. The ratio of simulated web and bulk circuits in relation to the number of relays requires calibration to produce network characteristic that approximate Tor. We base our experiments on the calibration performed by [1] which concludes that in a larger setting, a scenario with 100 relays and 375 circuits with 10% bulk circuits approximates Tors performance reasonably well. We have run all our simulations for 90 seconds.

We evaluate our simulations on two important metrics, time-to-first-byte (*ttfb*) and time-to-last-byte (*ttlb*). *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. *ttlb* is an important measure to assess the download times which also has a significant impact on the bulk download requests. We first show the time-to-first-byte and time-to-last-byte results of Backtap as compared with Vanilla Tor as well as the two algorithms we described before- N23 and PCTCP. It can be seen from Fig. 7 that Backtap performs better than all these algorithms for both *ttfb* and *ttlb* metrics.

In our later experiments, we have compared the performance of Marut only with Backtap based on our conclusion from the earlier experiment result that Backtap performs better than Vanilla Tor, N23 and PCTCP. We first simulate a relatively simple scenario of a tree topology with 50% bulk circuits as shown in Fig. 10 to closely understand the behavior of Marut & BackTap protocols in high congestion conditions on just one hop. We analyzed the log files to get an idea of how each protocol modified the sender window size and the queue lengths at each TOR node at different time intervals. We then plotted the empirical cumulative distribution of *ttfb* and *ttlb* for both Backtap and Marut which showed that Marut performs marginally better as also visible in Fig. 8.

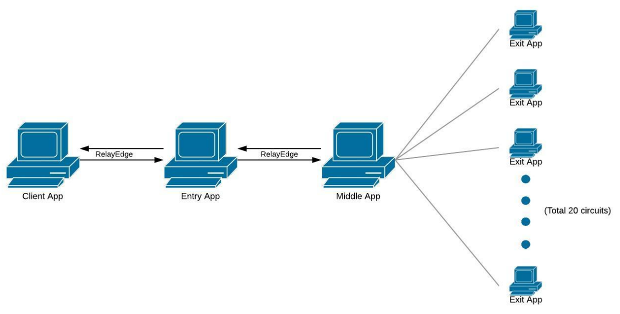


Fig. 10. Toy scenario of tree topology with 20 circuits

We then increased our scale of analysis and evaluated our protocol in a scenario of Dumbbell topology with 100 circuit and 100 relays. Based on the results of this simulation, we plotted the empirical CDF for *ttfb* and *ttlb* for both Marut and Backtap as shown in Fig. 11. This also shows only marginal improvement in performance of Marut over Backtap. This is fine because we expect the difference between Backtap and Marut to become visibly significant in two cases: either if the congestion in the network increases or if the circuit length is significantly higher. In both these cases, we expect an end-to-end congestion control to be faster than a back pressure based control which depends on queues being full at all hops before the sender becomes aware of congestion. The topology of 100 circuits with 100 relays doesn't necessarily guarantee a congested network.

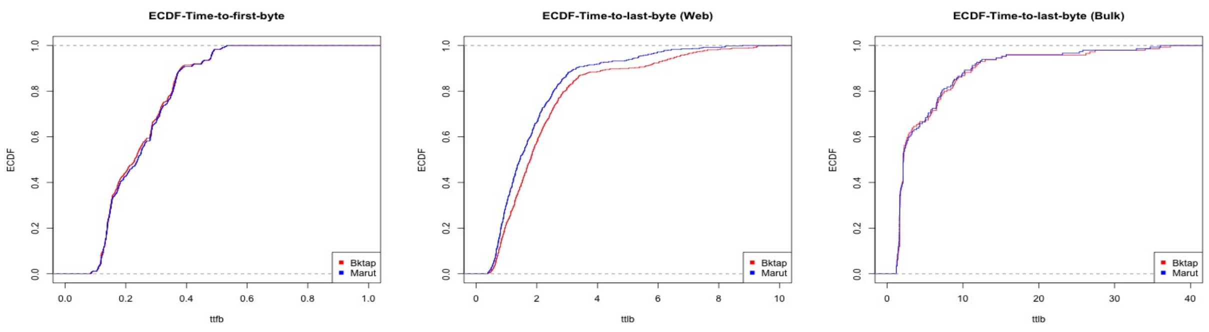
We then performed two more simulations by first increasing the congestion in the network and next by increasing the circuit length in the network. In the first simulation, we increased the number of circuits in our previous scenario from 100 to 375 and evaluated our protocol. The graphs as shown in Fig. 12. demonstrate a clear improvement in the performance of Marut as compared to Backtap which goes well with our theoretical reasoning.

In the second simulation, we increased the circuit length in the network from 3 to 4 keeping the number of circuits as 100. The Fig. 13. shows the plots of empirical CDF of *ttfb* and *ttlb* for both Backtap and Marut which shows that Marut, does in fact perform better and confirms our theoretical understanding again.

We then performed another simulation in which we increased both the number of circuits from 100 to 375 as well as the the length of circuit from 3 to 4 and plotted the empirical CDF of *ttfb* and *ttlb* for both the protocols. This is presented in Fig. 14. and shows clearly that the gap between the performances of Marut and BackTap has increased, which is how we expect it to be.

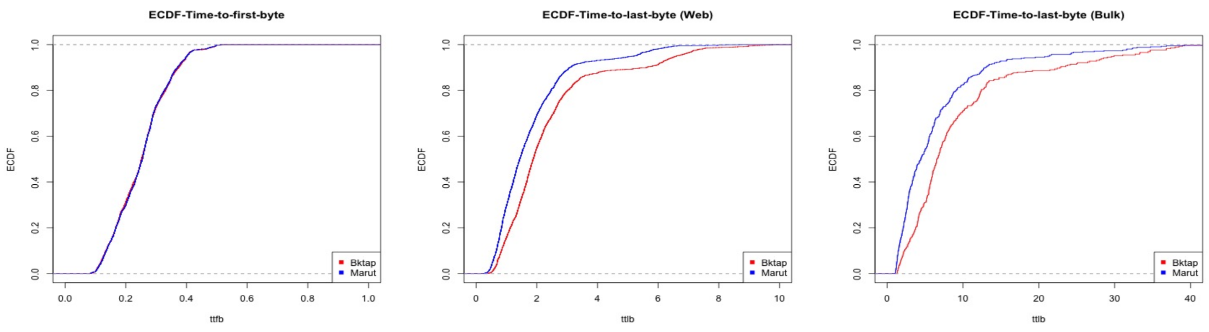
There was another hypothesis that we wanted to check. We knew that web flows, which are more interactive than bulk flows, are performing better but we wanted to confirm if they are doing so on expense of bulk flows. So we ran a simulation with 100% bulk flow in a dumbbell topology of 100 circuit, 100 relay and plotted the empirical CDF of TTFB and TTLB again as shown in Fig. 9. In the figure, we can see that Marut is actually doing marginally better for all bulk flows which lets us to confirm that there is no loss of performance in Marut for bulk flows.

Another perspective on the comparative performance of Backtap and Marut protocol is provided by Table I. In the table, we summarize the number of completed downloads (within the simulation time) and the mean download rate for all the scenarios of our simulation for both the protocols. We note that in all the scenarios, the Marut protocol is able to complete more requests for both web and bulk type of requests. The number of downloads is only marginally better for bulk but it is significantly better for the web requests. This is in accordance with our theoretical insight that there is less queuing in Marut as compared to Backtap. 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. Hence Marut is fairer towards interactive flows than BackTap but not at any cost to bulk traffic. Infact, Marut's



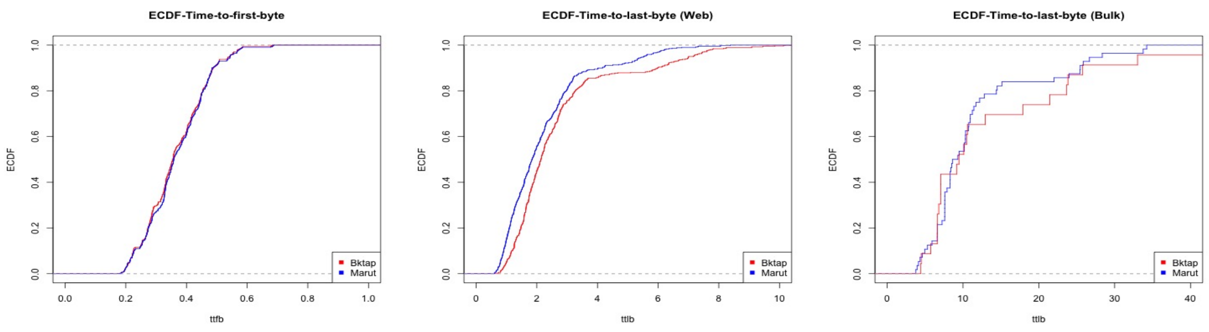
(a) Bulk and Web Circuits (b) Web Circuits (c) Bulk Circuits

Fig. 11. Time to download files over Tor for a 100 circuit, 100 relay in Dumbbell topology and a 3 node circuit length



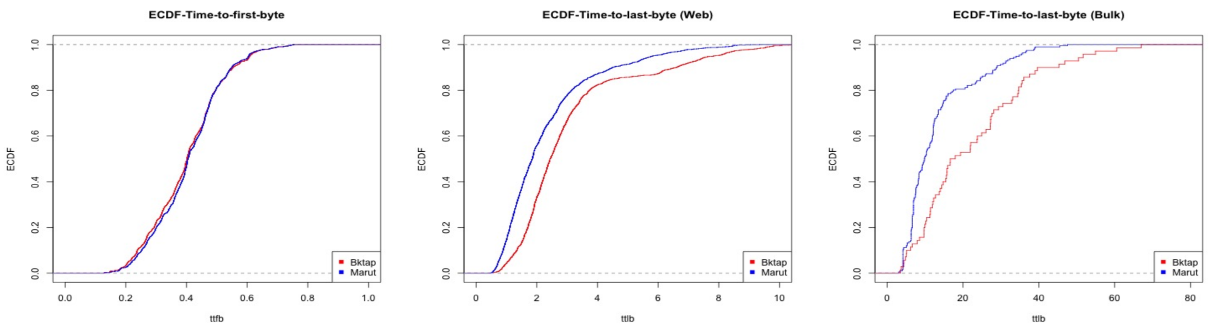
(a) Bulk and Web Circuits (b) Web Circuits (c) Bulk Circuits

Fig. 12. Time to download files over Tor for a 375 circuit, 100 relay in Dumbbell topology and a 3 node circuit length



(a) Bulk and Web Circuits (b) Web Circuits (c) Bulk Circuits

Fig. 13. Time to download files over Tor for a 100 circuit, 100 relay in Dumbbell topology and a 4 node circuit length



(a) Bulk and Web Circuits (b) Web Circuits (c) Bulk Circuits

Fig. 14. Time to download files over Tor for a 375 circuit, 100 relay in Dumbbell topology and a 4 node circuit length

Scenario	Protocol	Bulk		Web	
#circ, #nodes		#dwnlds	avg. rate	#dwnlds	avg. rate
20 C,3 N	BkTap	60	719.4	40	167.54
	Marut	64	749.18	44	211.9
100 C, 3 N	BkTap	144	964.05	587	141.54
	Marut	149	993.18	603	170.61
375 C, 3 N	BkTap	292	529.46	2126	134.02
	Marut	419	767.12	2229	177.97
100 C, 4 N	BkTap	23	347.04	561	118.69
	Marut	56	428.31	588	144.51
375 C, 4 N	BkTap	96	290.04	2000	104.69
	Marut	126	380.31	2136	138.51
100% bulk	BkTap	711	502.53	-	-
	Marut	773	546.96		

TABLE I
Completed downloads (#dwnlds) and mean rate

quicker feedback delivery as compared to BackTap increases performance for both flows but helps interactive flows much more.

VII. Future Work & Conclusion

In this report we have presented the salient features of the original TOR design and traced historically impactful ideas which have been proposed to alleviate existing issues in the TOR. We have gone through the most recent of the architectures developed, recreated its results and then identified potential ways to make improvements to the design. Finally we have validated our changes with some simulations over realistic scenarios.

However, there are many directions left unexplored. Both BackTap and Marut depend on relative congestion between circuits and not absolute congestion in any circuit. For instance, in a scenario where there is congestion on all circuits between two TOR nodes, the congestion parameter (difference between expected and actual window sizes) would be 0 and we would keep increasing our sending window whereas we should have decreased it. End-to-end congestion control algorithms which take into account the absolute congestion suffered by a circuit should resolve this problem. Ideally we would want to have a combination of absolute congestion as well as relative congestion parameters because relying solely on absolute parameters can lead to unfairness between different circuits. We have in mind two approaches for end-to-end congestion control:

- TIMELY [12]: A delay based congestion control for lossless networks; Adjust transmission rates using RTT gradients to keep packet latency low while delivering high bandwidth.
- DCTCP [13]: Mark the ECN bit on packet if congestion encountered. Then end-nodes limit congestion window based on fraction of marked packets.

Secondly, in a similar way that we have done for congestion control, we would want to create a design which has reliability implemented as an end to end optional feature which would allow anonymous applications to decide whether they need it. Many applications which do not really need reliable transport get unnecessarily shoe-horned into using it in the current TOR design which makes them suffer attendant problems like head-of-the-line-blocking for their packets. Removing reliability from the TOR network and making it an end-to-end feature would also make the network simpler and help reasoning about its properties easier.

References

[1] Tschorsch, Florian, and Bjrn Scheuermann. Mind the Gap: Towards a Backpressure-Based Transport Protocol for the Tor Network. *NSDI.*, 2016.

[2] Dingledine, Roger, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. *Naval Research Lab Washington DC,*, 2004.

[3] Dingledine, Roger, and Steven J. Murdoch. Performance Improvements on Tor or, Why Tor is slow and what were going to do about it. *Online: http://www. torproject. org/press/presskit/2009-03-11-performance. pdf*, 2009.

[4] AlSabah, Mashael, et al. DefenestraTor: Throwing out windows in Tor. *International Symposium on Privacy Enhancing Technologies Symposium. Springer, Berlin, Heidelberg*, 2011.

[5] AlSabah, Mashael, and Ian Goldberg. PCTCP: per-circuit TCP-over-IPsec transport for anonymous communication overlay networks. *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security.*, ACM, 2013.

[6] Cardwell, Neal and Cheng, Yuchung and Gunn, C. Stephen and Yeganeh, Soheil Hassas and Jacobson, Van. BBR: Congestion-Based Congestion Control *Queue - Network Congestion Volume 14 Issue 5, September-October 2016*, ACM, 2016

[7] Jansen, Rob, et al. Methodically modeling the Tor network *CSET.* 2012.

[8] 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*

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

[10] Tschorsch, Florian. nstor, open source module for Tor *https://github.com/tschorsch/nstor.*, 2016.

[11] THE TOR PROJECT. Tor Metrics Portal. *https://metrics.torproject.org.*

[12] Mittal, Radhika, et al. TIMELY: RTT-based Congestion Control for the Datacenter. *ACM SIGCOMM Computer Communication Review. Vol. 45. No. 4. ACM,* 2015.

[13] Alizadeh, Mohammad, et al. Data center tcp (dctcp). *ACM SIGCOMM computer communication review. Vol. 40. No. 4. ACM,* 2010.

[14] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Hiding routing information. In R. Anderson, editor, *Information Hid- ing, First International Workshop, pages 137150. Springer- Verlag, LNCS 1174,* May 1996.

[15] 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 Net- work and Distributed System Security Symposium* (Feb. 2014).

[16] BRAKMO, L. S., OMALLEY, 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. 2435.