Congestion Control in TOR

Ambuj Ojha, Neha Sharma

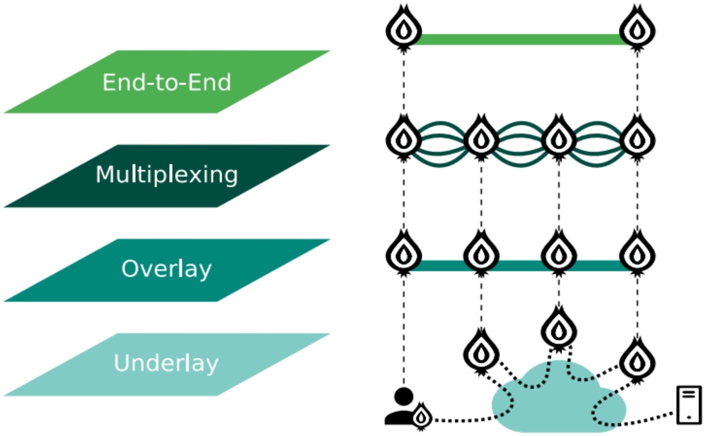
Courant Institute, NYU

apo249@nyu.edu, ns3786@nyu.edu

May 10, 2018

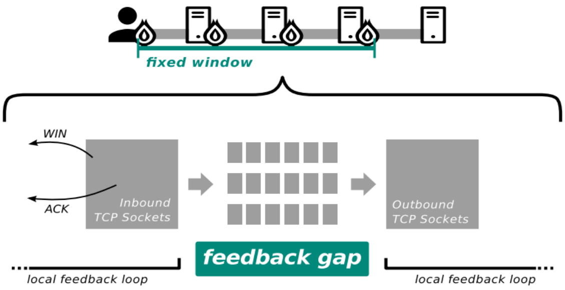
Overview

TOR Design Review



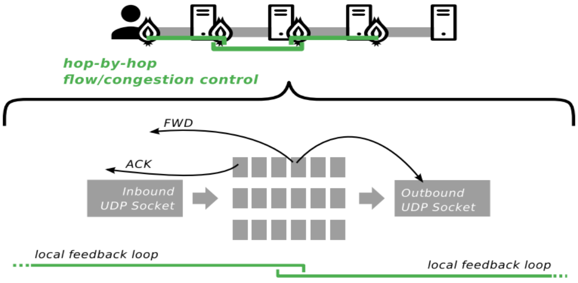
TOR routes application-layer data, packaged into equally-sized cells, along a cryptographically secured virtual circuit through an overlay network.

Problems with TOR Design



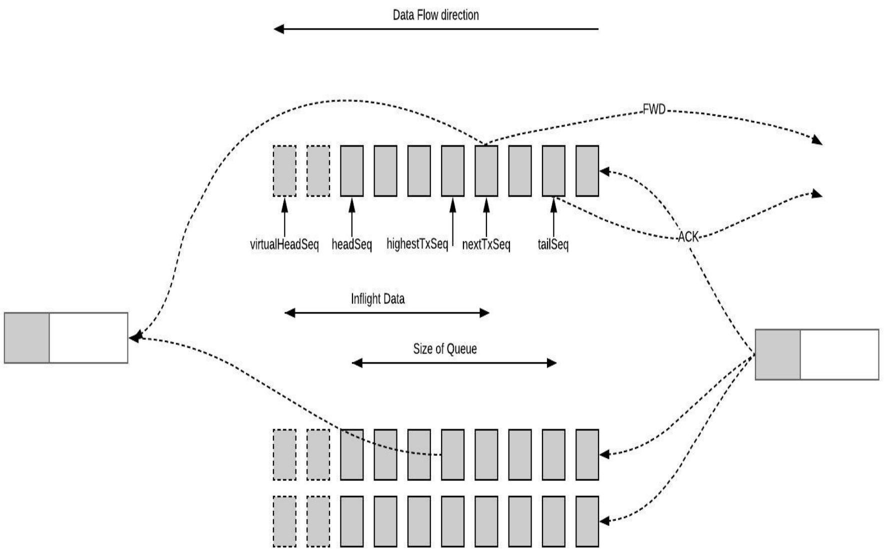
- Tor relays read from incoming TCP connections regardless of fill level of corresponding circuit queues in the relay
- Limited outflow of a circuit does not propagate back to the incoming side of the relay
- The end-to-end sliding window with its non-adaptive constant size and its long feedback loop is the only mechanism that will eventually throttle the source

BkTap Design: delay-based per-circuit congestion control loops



- Hop by hop reliability (in Application layer)
- Hop by hop congestion control (in Application layer)
- Emit flow control feedback (FWD cell) only when a cell has been forwarded out of the local relay $\implies$ tight feedback coupling between consecutive hops
- Congestion window estimated based on the outflow in the downstream node and the conditions of network path between consecutive relays

Bktap Queue Management



BackTap Congestion Control

- FWDs are used by upstream node to determine *actualRTT* and *baseRTT*
    - actualRtt: Smallest RTT sample during the last RTT
    - baseRtt: Minimum over all RTT samples of all circuits directed to the same relay
- Congestion Parameter, 'diff'

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

- sending window (cwnd). At most cwnd cells are kept in the transmission pipe.

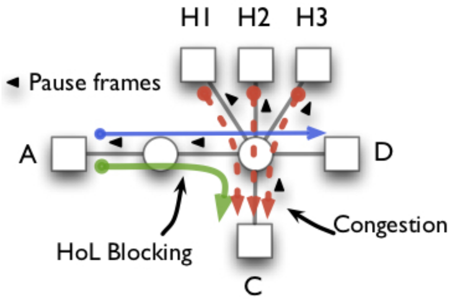
$$cwnd = \begin{cases} cwnd + 1 & \text{if } diff < \alpha \\ cwnd - 1 & \text{if } diff > \beta \end{cases}$$

- cwnd changes by at most one. Follows Additive Increase, Additive Decrease (AIAD) policy

Problems with Bktap Design

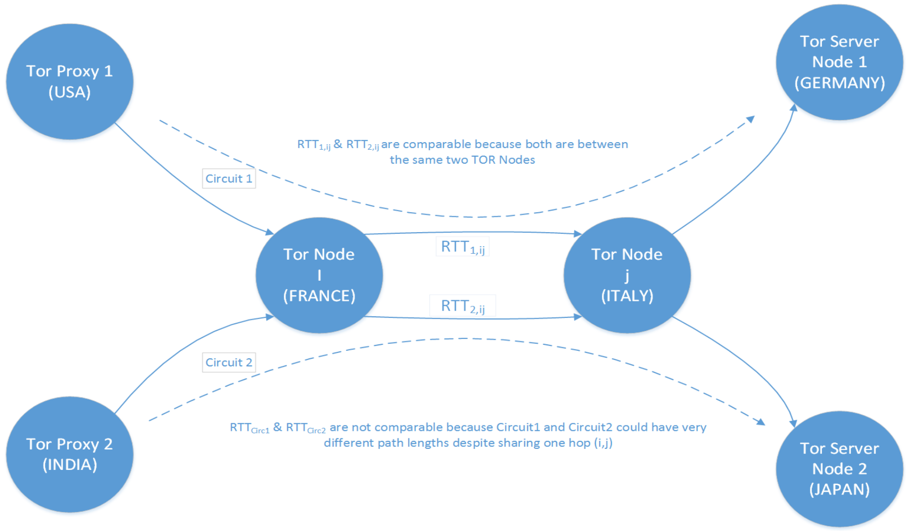
- **Slow Congestion Update**: At each relay, the queues need to fill up in order for upstream node to detect congestion and decrease sending window. Effectively, queues at all the hops end up filling before the sender becomes aware of congestion.

- **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)

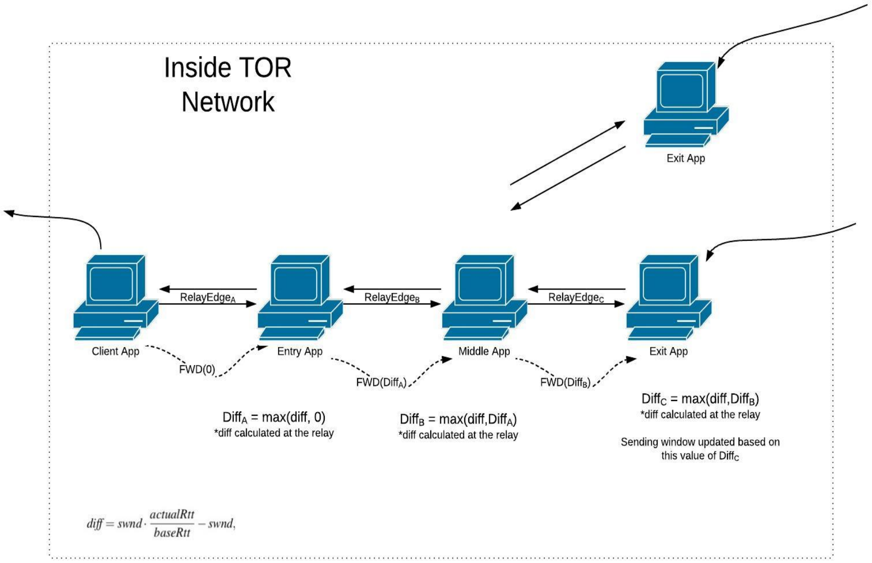
Marut: End to End Window Update with BkTap not possible


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

Marut: Design Specifics

- We propose **Marut**,
  - end-to-end congestion
  - hop-to-hop reliability
  - uses UDP channels between TOR nodes same as BkTap
- Marut allows congestion updates to reach the end-host within 1 RTT for the circuit
- For any circuit
  - Each TOR node still calculates the congestion parameter,'diff', which defines congestion till the next hop
  - FWD cell now contains a new field, 'c_diff', which encapsulates information on congestion in the downstream portion of the circuit
  - $CumulativeCongestion = max(diff, c\_diff)$
  - 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

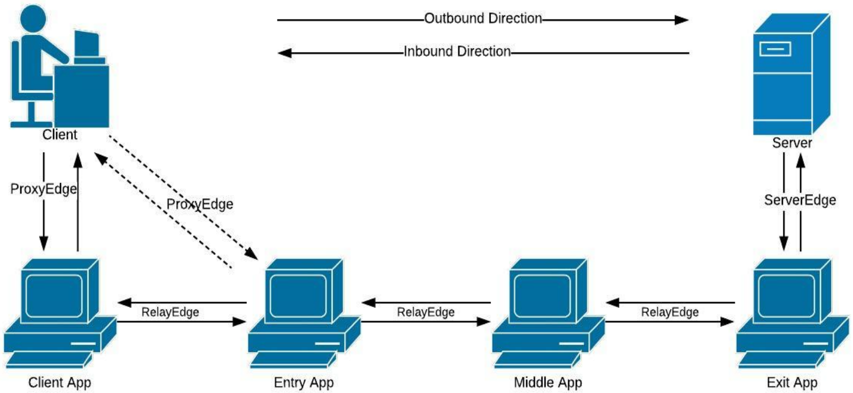
Marut: Proactive Congestion Control



Inside TOR Network
Exit App
Client App
Entry App
Middle App
Exit App
RelayEdge$_A$
RelayEdge$_B$
RelayEdge$_C$
FWD(0)
FWD(Diff$_A$)
FWD(Diff$_B$)
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$_C$

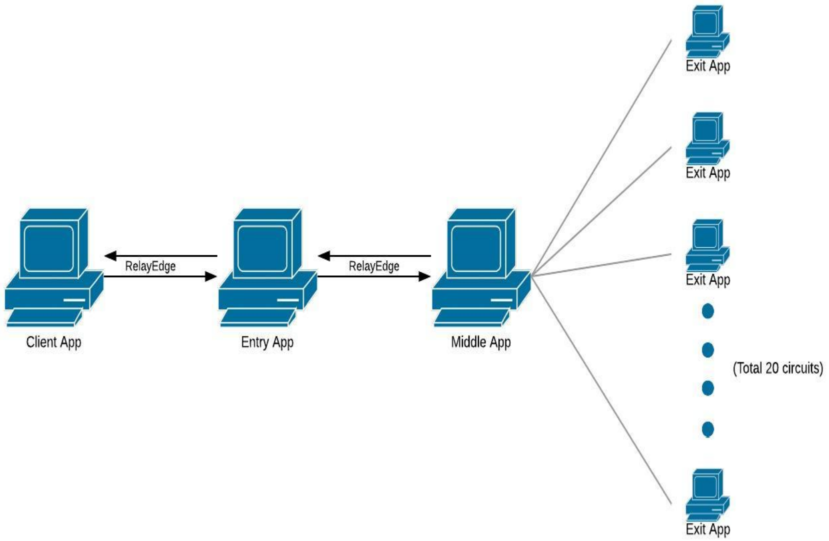
$$diff = swnd \cdot \frac{actualRtt}{baseRtt} - swnd,$$

Simulation Code Architecture-Circuit Details



- We have
  - web circuits (request 320 KiB files with a random think time of 1s to 20s between requests)
  - bulk circuits (continuously transfer 5 MiB files)

Tree Topology, 20 Circuits

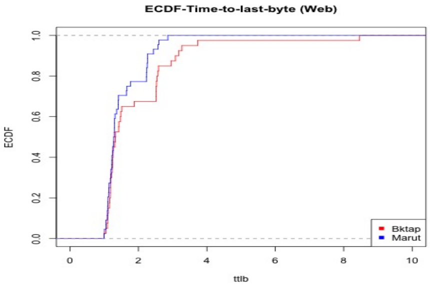


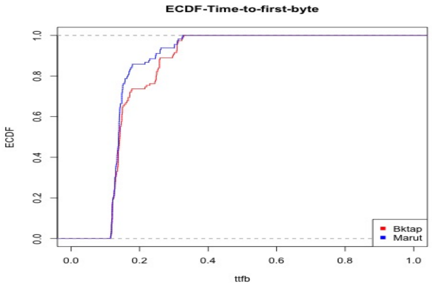
Tree Topology, 20 Circuits: ttfb and ttlb

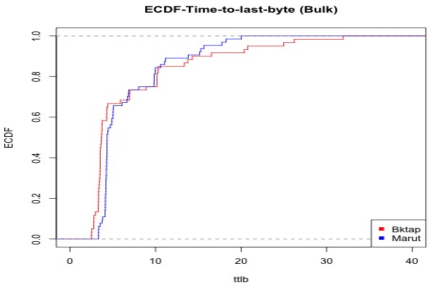
ECDF Graphs

Time of simulation: 90s

Percentage of bulk circuits: 50







Tree Topology, 20 Circuits: completed downloads and mean rate

Protocol	Bulk		Web	
	#dwnlds	avg. rate	#dwnlds	avg. rate
BkTap	60	719.4	40	167.54
Marut	64	749.18	44	211.9

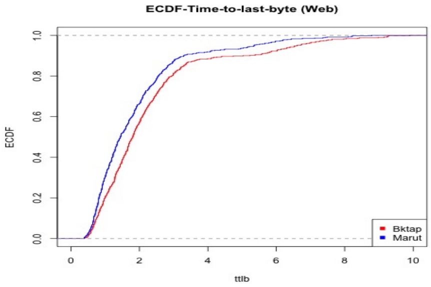
Dumbbell Topology, 100 Circuits: ttfb and ttlb

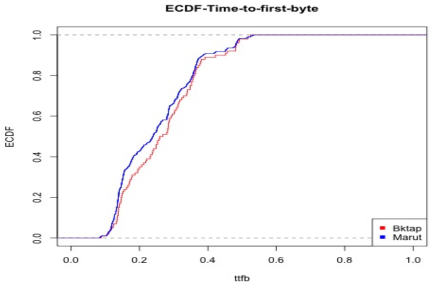
ECDF Graphs

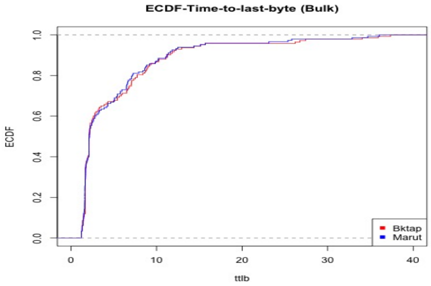
Time of Simulation: 90s

Number of nodes in a circuit: 3

Percentage of bulk circuits: 10



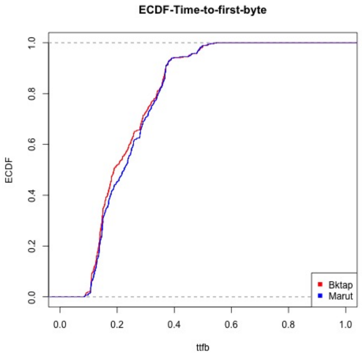


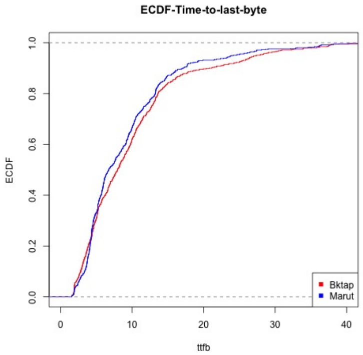


Dumbbell Topology, 100 Circuits: completed downloads and mean rate

Protocol	Bulk		Web	
	#dwnlds	avg. rate	#dwnlds	avg. rate
BkTap	144	964.05	587	141.54
Marut	149	993.18	603	170.61

Dumbbell Topology, 100 Circuits, All Bulk: ttfb and ttlb





Protocol	Bulk	
	#dwnlds	avg. rate
BkTap	711	502.53
Marut	773	546.96

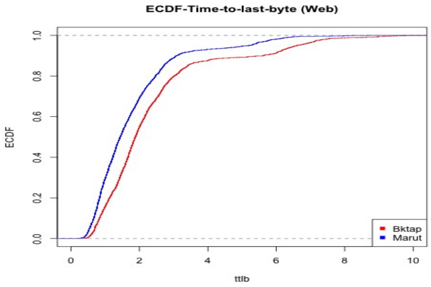
Dumbbell Topology, 375 Circuits: ttfb and ttlb

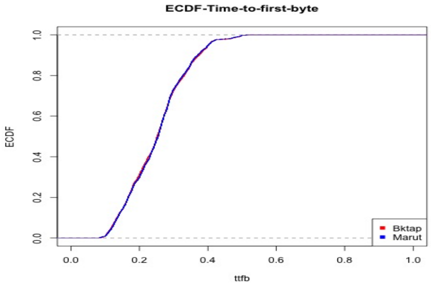
ECDF Graphs

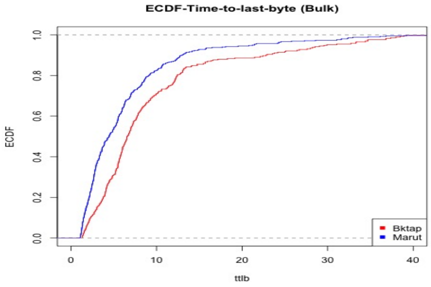
Time of Simulation: 90s

Number of nodes in a circuit: 3

Percentage of bulk circuits: 10%







Dumbbell Topology, 375 Circuits: completed downloads and mean rate

Protocol	Bulk		Web	
	#dwnlds	avg. rate	#dwnlds	avg. rate
BkTap	292	529.46	2126	134.02
Marut	419	767.12	2229	177.97

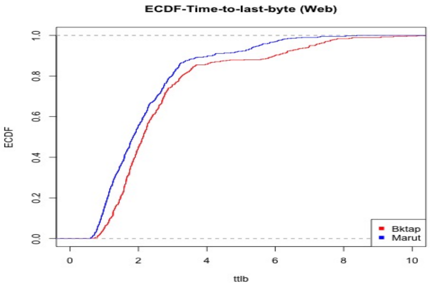
Dumbbell Topology, 100 Circuits, 4 nodes: ttfb and ttlb

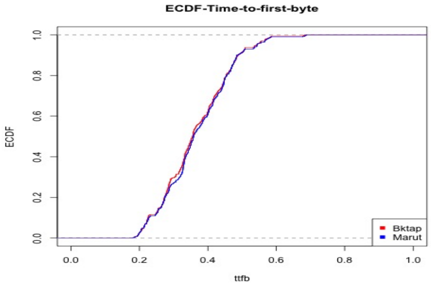
ECDF Graphs

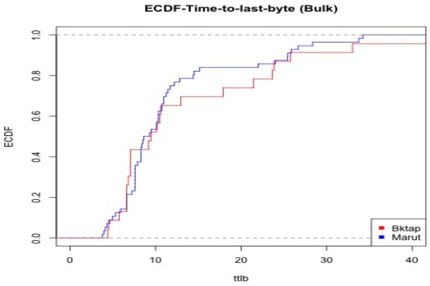
Time of Simulation: 90s

Number of nodes in a circuit: 4

Percentage of bulk circuits: 10%







Dumbbell Topology, 100 Circuits, 4 nodes: completed downloads and mean rate

Protocol	Bulk		Web	
	#dwnlds	avg. rate	#dwnlds	avg. rate
BkTap	23	347.04	561	118.69
Marut	56	428.31	588	144.51

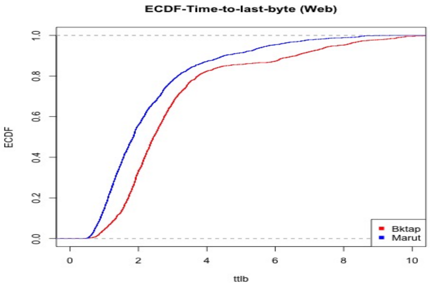
Dumbbell Topology, 375 Circuits, 4 nodes: ttfb and ttlb

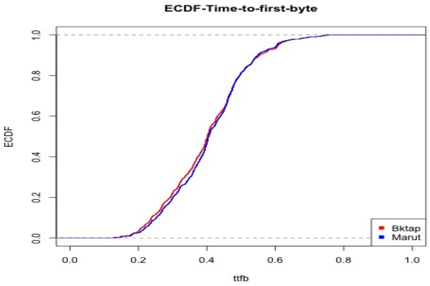
ECDF Graphs

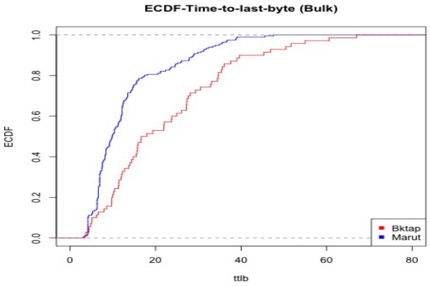
Time of Simulation: 90s

Number of nodes in a circuit: 4

Percentage of bulk circuits: 10%







Dumbbell Topology, 375 Circuits, 4 nodes: completed downloads and mean rate

Protocol	Bulk		Web	
	#dwnlds	avg. rate	#dwnlds	avg. rate
BkTap	96	290.04	2000	104.69
Marut	126	380.31	2136	138.51

Marut: Effects on Anonymity

- The FWD cell preamble contains the 'c_diff', encrypted using DTLS
- Regular cells, ACKs and FWDs, from different circuits, can be encapsulated in one UDP packet between two relays
- Typical MTU can contain up to two regular cells and many messages
- In case of a malicious TOR Node, we do leak some extra information but the problem is alleviated since
  - Since we only pass on the max of 'diff's, a particular value could potentially belong to any hop on the circuit
  - 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

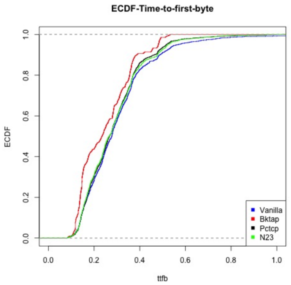
Other Algorithms Explored

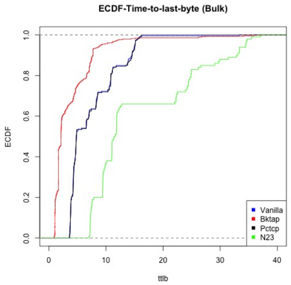
N23

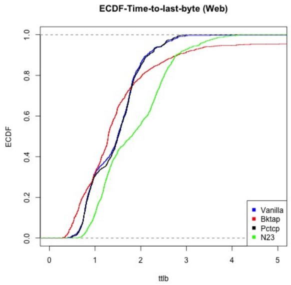
Per circuit queue with credit balance of N2 + N3 cells; Decrements credit balance by one with each forward till zero; Flow control cell to upstream with available credit; Effectively back-pressure based congestion control;

PCTCP

Use a kernel-mode Per Circuit TCP Connection instead of multiplexing circuit in one TCP connection







Future Work

- Both BackTap and Marut depend on relative congestion between circuits and not absolute congestion
- End-to-end congestion control algorithms resolve this problem
  - TIMELY: A delay based congestion control for lossless networks; Adjust transmission rates using RTT gradients to keep packet latency low while delivering high bandwidth.
  - DCTCP: Mark the ECN bit on packet if congestion encountered. Then end-nodes limit congestion window based on fraction of marked packets.
- Implement reliability as an end to end feature
  - Allows only those applications to use reliability which need it
  - Makes the network simpler and helps reasoning about congestion easier

References

Roger Dingledine, Nick Mathewson, Paul Syverson (2004)
Tor: The second-generation onion router
https://svn.torproject.org/svn/projects/design-paper/tor-design.pdf

Roger Dingledine and Steven J Murdoch (2009)
Performance improvements on tor or, why tor is slow and what were going to do about it
https://www.torproject.org/press/presskit/2009-03-11-performance.pdf

Florian Tschorsch and Bjrn Scheuermann (2016)
Mind the gap: Towards a backpressure-based transport protocol for the tor network
https://www.usenix.org/system/files/conference/nsdi16/nsdi16-paper-tschorsch.pdf

Florian Tschorsch
NSTOR, open source github repository for TOR
https://github.com/tschorsch/nstor

References

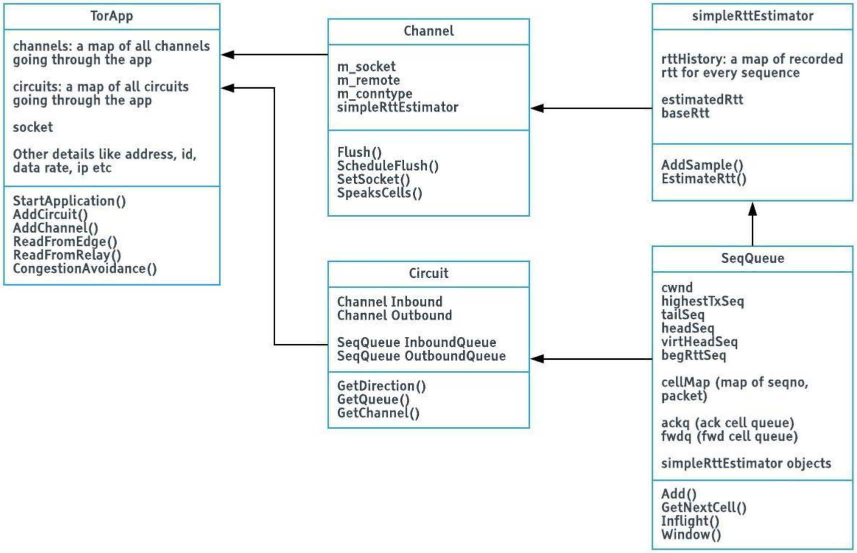
Brent Stephens, Alan L. Cox, Ankit Singla, John Carter, Colin Dixon, Wesley Felter (2014)
Practical DCB for Improved Data Center Networks
https://pdfs.semanticscholar.org/9291/e9688e9cecd7899d673c96b1f961afa2190d.pdf

Amin Vahdat, Yaogong Wang, David Wetherall, David Zats et al (2015)
TIMELY: RTT-based Congestion Control for the Datacenter
https://conferences.sigcomm.org/sigcomm/2015/pdf/papers/p537.pdf

ALSABAH, M., BAUER, K., GOLDBERG, I., GRUNWALD, D., MCCOY, D., SAVAGE, S., AND VOELKER, G. (2011)
DefenestraTor: Throwing out windows in Tor (N23)
https://www.cypherpunks.ca/ iang/pubs/defenestrator.pdf

ALSABAH, M., AND GOLDBERG, I. (2013)
PCTCP: per-circuit tcpover-ipsec transport for anonymous communication overlay networks
https://www.cypherpunks.ca/ iang/pubs/pctcp-ccs.pdf

Appendix - TOR Application Class Diagram



The End