An Evaluation of Node-Based Delays in Tor

2014 Annual IEEE CQR International Workshop

May 13th -15th 2014

Westward Look Wyndham Grand Resort & Spa
245 E. Ina Rd Tucson, AZ 85704, USA

Timothy Girry KALE, Satoshi OHZAHATA, Wu CELIMUGE and Toshihiko KATO
Graduate School of Information Systems,
The University of Electro-Communications
1-5-1 Chofugaoka, Chofu-shi, Tokyo 182-8585, Japan

Contents

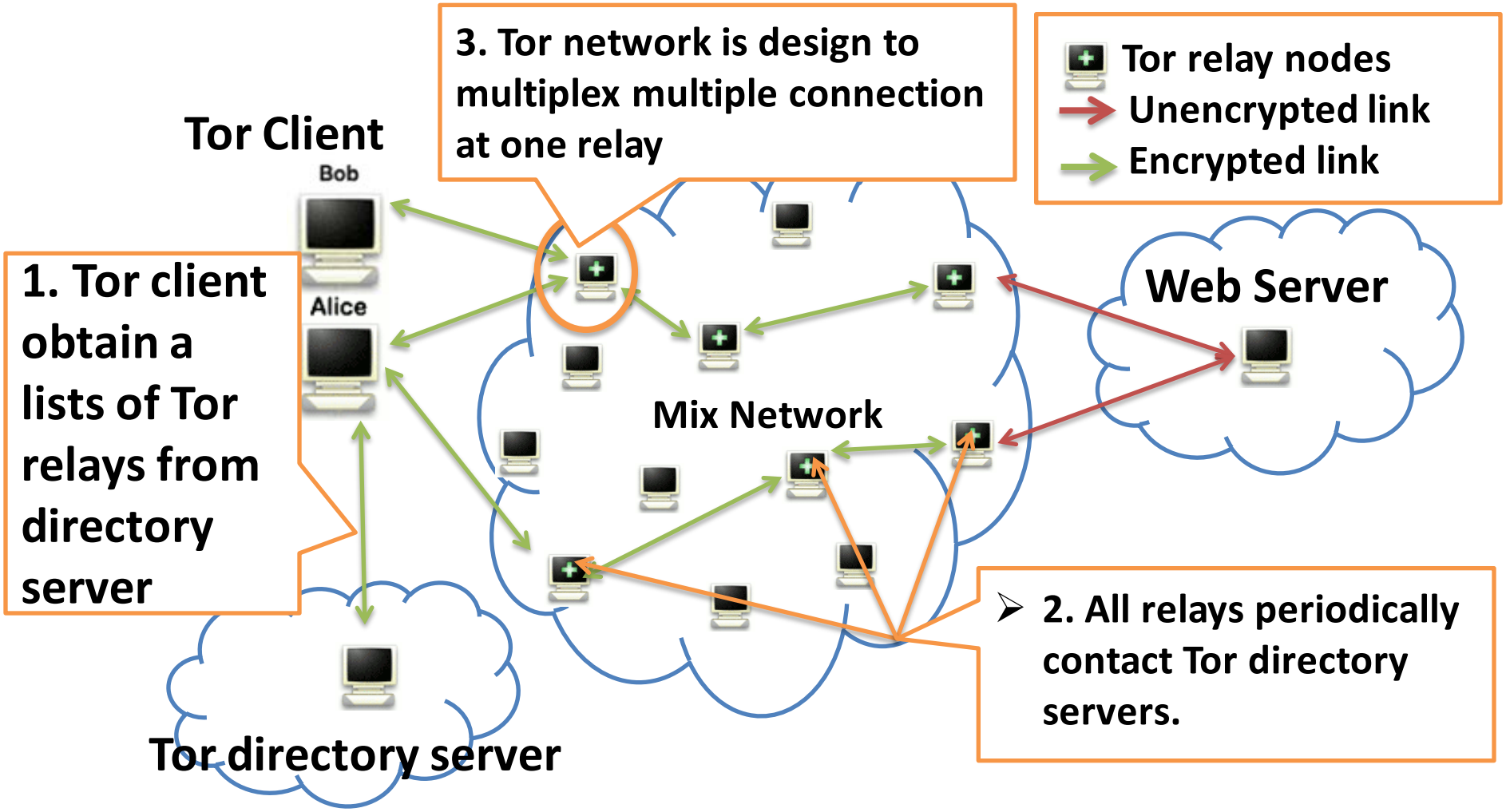
Introduction of the Tor Network

- **What is Tor ?**
- **Tor – The onion routing (2nd Generation)**
  - Focused on low-latency application such as web browsers
  - System and an open network that defend against network surveillance causes threaten to personal freedom and privacy
  - **Tor has been deployed to public since 2003**
  - **13th November 2012, there are 3,193 Routers**
  - **16th September 2013, there are 4,183 Routers**
  - **21st January 2014, there are 5,045 Routers**

Tor Technical Background

- Tor builds anonymous connections within 3 onion routers (relay nodes) to relay encrypted circuit.
  - Accepts fixed-length messages (512 bytes of cells) from different sources.
  - Onion Proxy (OP) presents a SOCKS proxy interface to local applications
  - The client (OP) picks a OR1, and makes a TCP connection as well as the transport layer security (TLS) on that connection.
  - TCP hop-by-hop congestion control, reliability and in-order delivery of data.
  - The TLS conceals data and encrypt the segments of the circuit connections at the application layer for the TCP transport layer.
- Tor Network consists of Volunteer-run Relays
  - Volunteer have all the rights over their routers

How Tor Network Works



- Tor client choose same or different routes to the destination web server
    - Relay traffic through mix network (Onion routers)

Problems of Tor

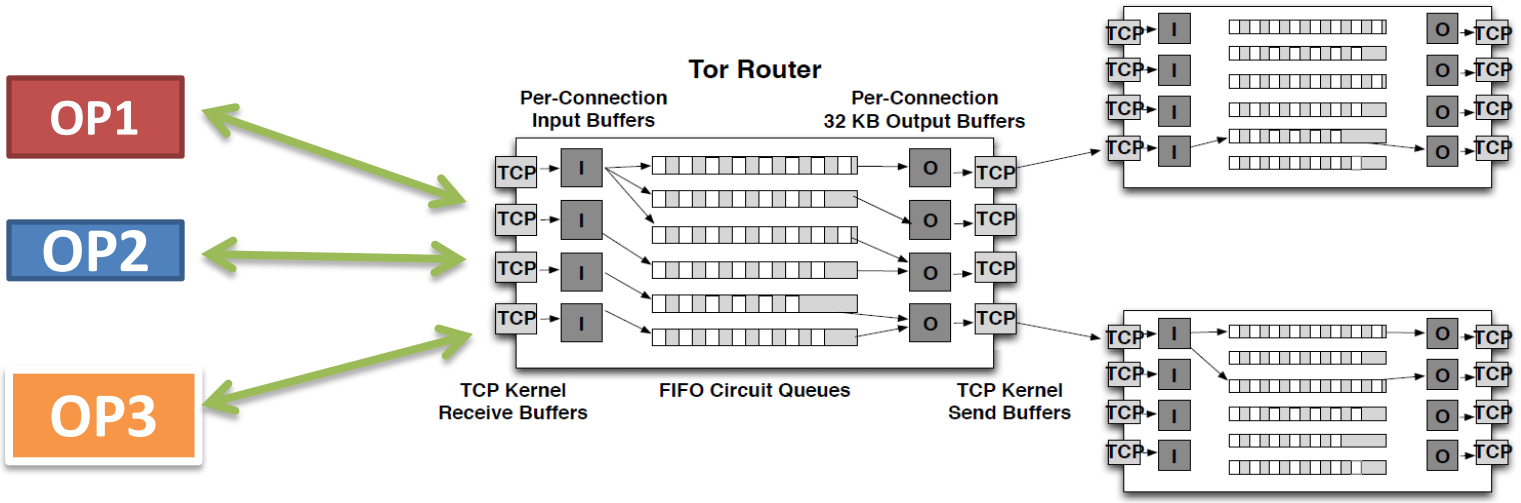
- Tor doesn't work very well when relay nodes have unequal bandwidth
  - Because Tor has separate service link rates between each hop
  - Multiple data streams competing to send data over a TCP stream that gives priority to circuits that send more data.
    - Delays are varies for all hops

- When outgoing bytes are all dropped, the TCP push-back mechanisms don't really transmit this information back to the incoming streams.

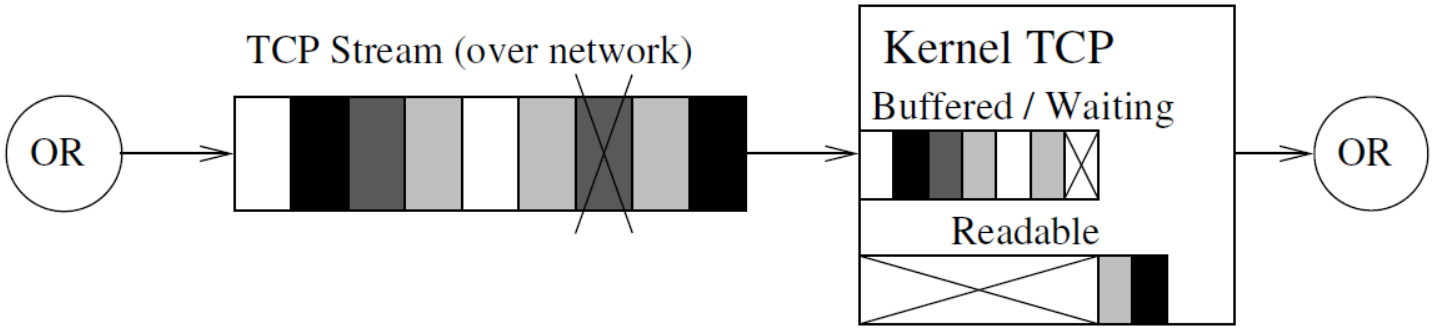
- Data holding up in the TCP output and input buffer for too long due to packet drops in the Tor application layer.

TCP multiplexes/demultiplexes problem

- **Unfair distribution of circuit queue**



R. Pries, W. Yu, S. Graham and X. Fu, "On Performance Bottleneck of Anonymous Communication Networks," IEEE Transactions on Networking Security, pp. 1-11, 2008.



- **Allocate TCP streams into the queue where further delay occurs (queuing delay).**

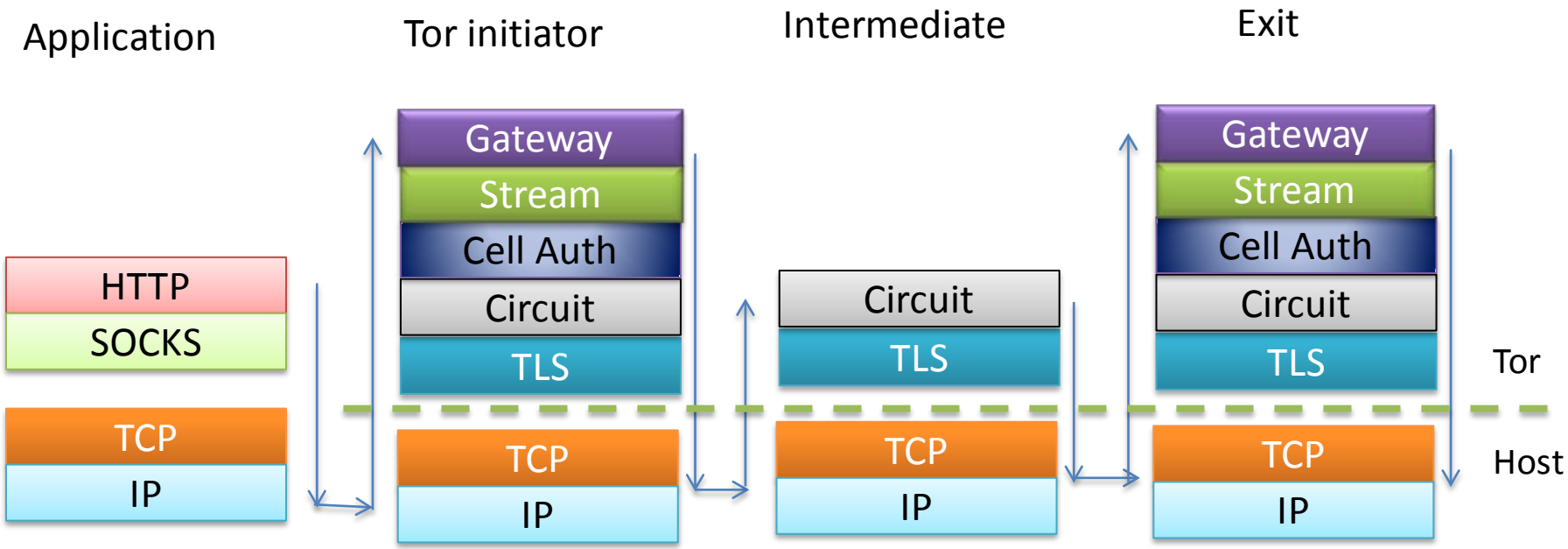
Joel Reardon and Ian Goldberg. Improving Tor using a TCP-over-DTLS Tunnel. In Proceedings of the 18th USENIX Security Symposium, pp. 119–133, 2009.

Motivation and Goals

- Our motivation comes from the most significant high and variable delays which occur in the relay nodes. (Node based delays)
  - Understanding drawbacks and sources of delay contributions in the node host TCP stack, as a prerequisite for addressing the Tor poor congestion control and increasing end-to-end latency.

- Our goals
  - To evaluate the relative contributions of the Tor node delays
  - Analyze the overall end-to-end latency experience along the circuit (RTTs).

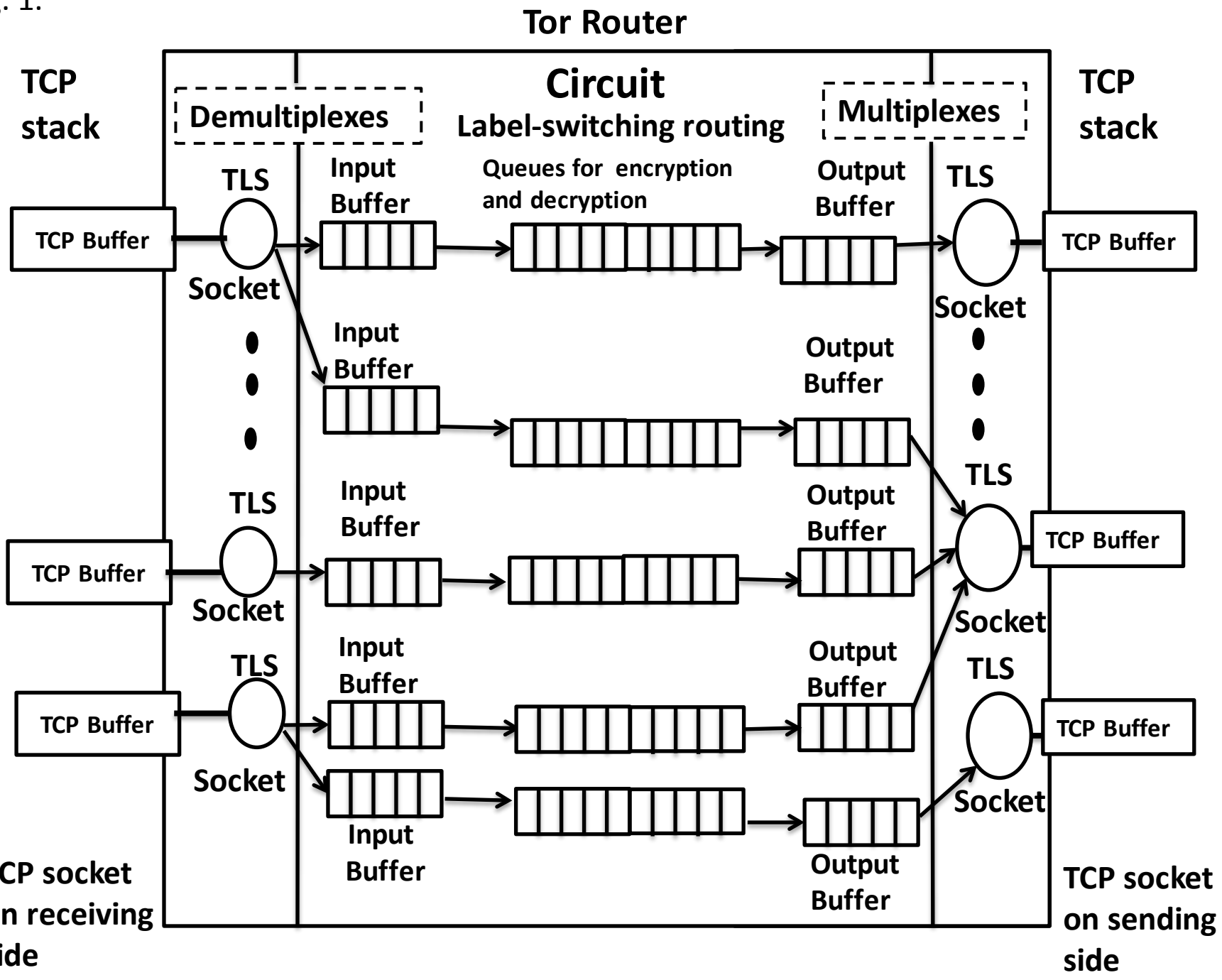
- Each packet of data in the Tor network has fixed cell size of 512 bytes with the header and a payload.

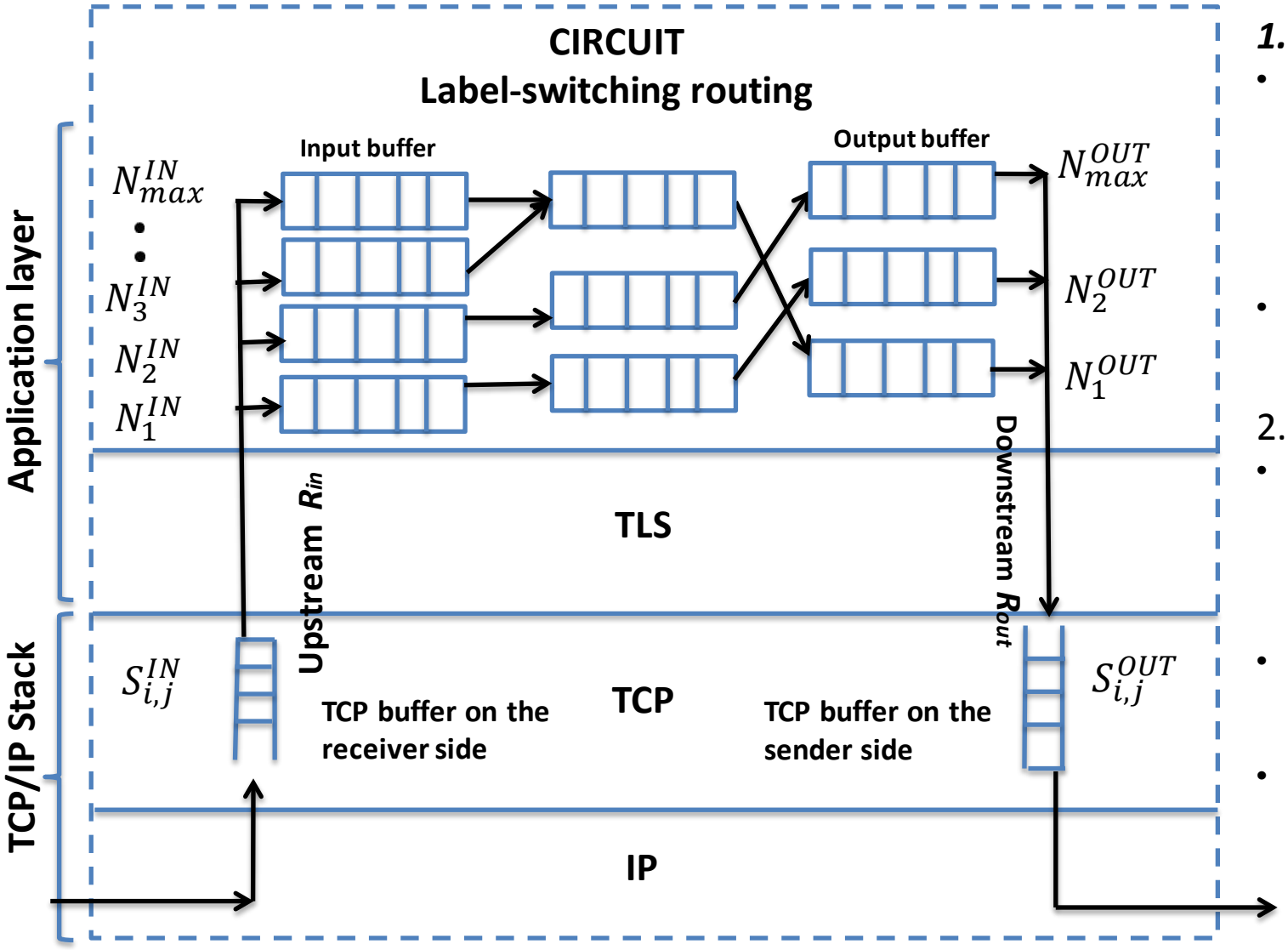


- TCP/IP stack are responsible for transporting the SOCKS stream to the Tor initiator

Joel Reardon and Ian Goldberg. Improving Tor using a TCP-over-DTLS Tunnel. In Proceedings of the 18th USENIX Security Symposium, pp. 119–133, 2009.

Fig. 1.


Tor Router
TCP stack
Demultiplexes
Circuit
Label-switching routing
Multiplexes
TCP stack
TLS
Input Buffer
Queues for encryption and decryption
Output Buffer
TLS
TCP Buffer
TCP Buffer
Socket
Socket
Input Buffer
Output Buffer
TLS
TLS
Input Buffer
Output Buffer
TCP Buffer
TCP Buffer
Socket
Socket
TLS
Input Buffer
Output Buffer
TLS
TCP Buffer
TCP Buffer
Socket
Socket
Input Buffer
Output Buffer
TCP socket on receiving side
TCP socket on sending side



1. $R_{in} < R_{out}$,

- Data comes in at the Tor router at a rate slower than what can be forwarded on the outgoing hop.
- R_{out} is limited by the data rate of R_{in}.

2. $R_{in} > R_{out}$,

- Data comes in at the Tor router at a rate faster than the outgoing connection
- The Tor router has to buffer the excess data.
- Once the buffers are full, the incoming connection has to throttle back

- The incoming and outgoing rates are generally unequal because of data queue at the Transport TCP level and Application level

Research focuses

- **Node based delays**
  - We analyze the problems contributing to increase the delays in the host TCP stack buffers, which affects the total end-to-end delay in Tor.
  - Data is read from a TCP receiving connections and placed on TCP input socket buffer before transfer to the application layer.
  - Data is read from an application layer and placed on a TCP output socket buffer before transfer to other relay node.
  - Our enquiry is focused on the transfer time from the TCP receiving side to sending side of a relay node.
    - The delays includes the TCP level and the Application level

- We discussed the TCP kernels receive and send buffers can increase the node based delays, when packets are holding up for too long and not quickly transferred.

- We showed that this problem has a direct impact to TCP window sizes, which the receiving nodes cannot accept more newly incoming cells.

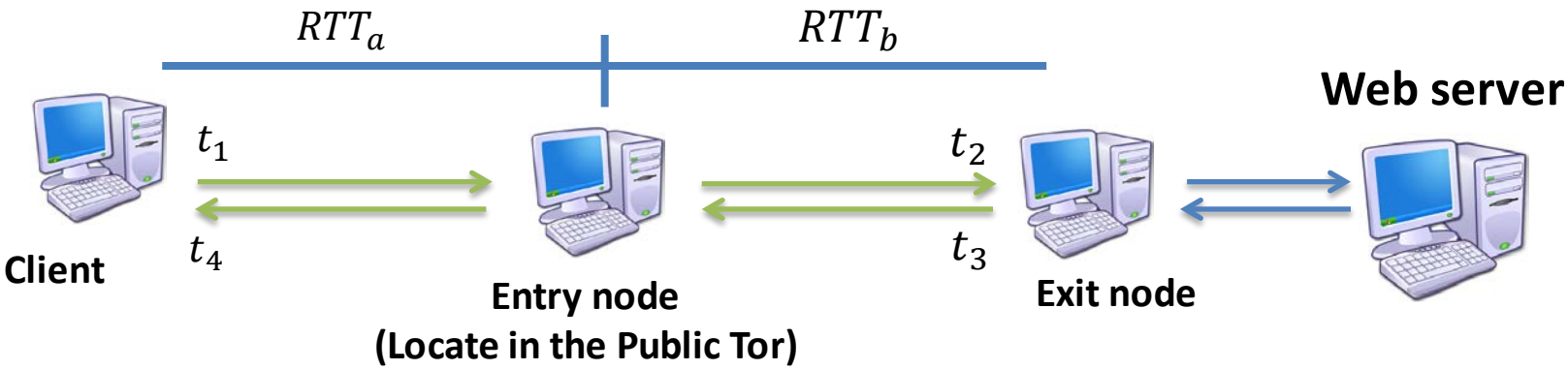
Proposed Analysis

- **We are evaluating**
  - Identify the node based delays (ND)constitute the major performance bottleneck on TCP connections.
  - The overloaded links (Total end-to-end delay RTTs) and (RTT between routers)
  - Total Propagation delays - time spent by packets on a link between neighboring Tor routers.

- **These metrics dominates the node based delays:**

  - TCP stack input/output queuing delays
  - TCP kernel memory usage and
  - TCP window size limitations.

Fig. 2. Node based delay measurements



- TCP Ping SYN message - RTTs are measured by the first bytes sends from client t_1, passing through middle router (processing/queuing delays) before reaching the exit router, which $t_2$is recorded. $t_3$is recorded as soon as bytes writes on the socket of exit relays, and packets passes through middle router and arrives at the client, then t_4 is recorded.

- Total RTT delay *(TD)*= $(t_4 - t_1) - (t_3 - t_1)$
- $RTT_{a+b} = RTT_a + RTT_b$ (Delays exit between OP <-> EN and EN <-> EN)
- Node delay = $TD - RTT_{a+b}$ (Delay between the TCP input and the TCP output)

- This procedure is repeated for 2,567 routers in the entry position, chosen one-by-one at random from the current list of running routers in Tor.
- Each node is measured 10 repeated times after 5 minutes
- Note: Since entry nodes are in the Tor network, it can accommodate other traffics

- This measurement approach is good because it can give the realistic network situation and the propagation distance.

- Direct approach measurement and periodically probe the isolated rates on each hop by performing a data transfer.

- **Tor auto-circuit and nodemonitor tools**
  - To control circuit length, speed, geolocation, and other parameters.
  - To capture the incoming and outgoing transfer rates at the TCP level

- Little's theorem helps us to identify the queuing delays at the TCP input and the output buffers.
  - Average node based delay is calculated by the little theorem

$$T_t = \frac{\sum_{i=0}^{\alpha(t)} T_i}{\alpha(t)}$$

Table I. Average Delays measured between nodes and in the nodes, with 95% confidence interval.

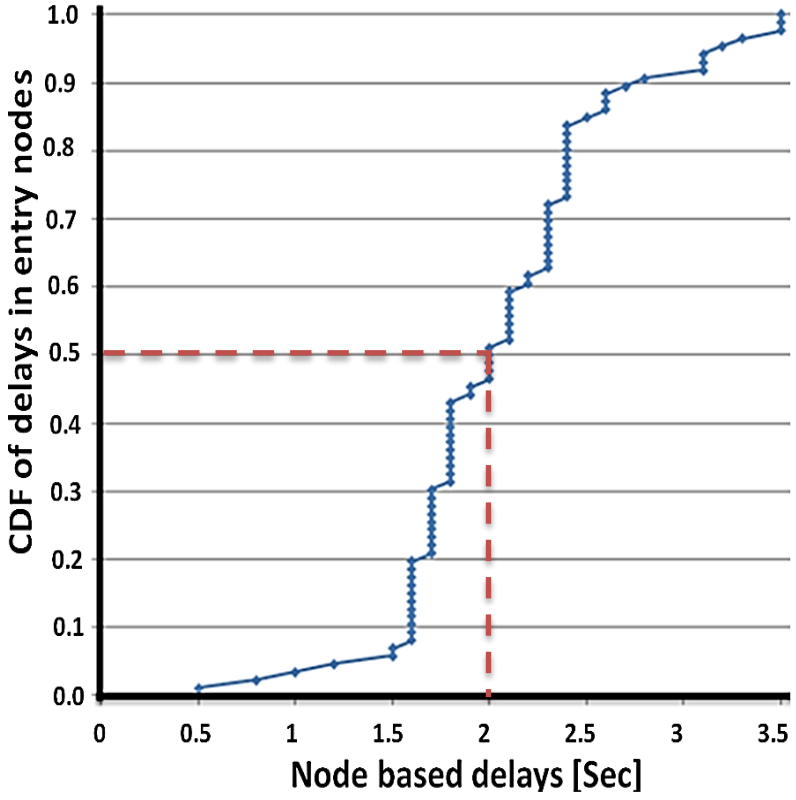
Prop delay OP -> Entry (Sec)	Prop delay Entry -> Exit (Sec)	Tot prop delay OP -> Exit (Sec)	*RTTa* (Sec)	*RTTb* (Sec)	Node delay *(ND)* (Sec)	Total delay *(TD)* (Sec)
0.20 ± 1.51	0.18 ± 1.64	0.38 ± 3.36	0.39 ± 1.54	0.42 ± 1.14	2.13 ± 1.93	2.94 ± 1.77

- The total propagation delays from OP to exit node and the RTT_{a+b} are relatively lower compare to node delays *(ND)*.
- This is because of the increasing queuing and processing delays of cells of overall node based delay, which includes queuing delay at the TCP stack buffers and buffers in the Tor router.
- Packets queuing in multiple buffers increases the node delay *(ND)* and the overall *TD*.
- The increase of *TD* along the circuit is heavily influenced by the delays in the node *(ND)*.

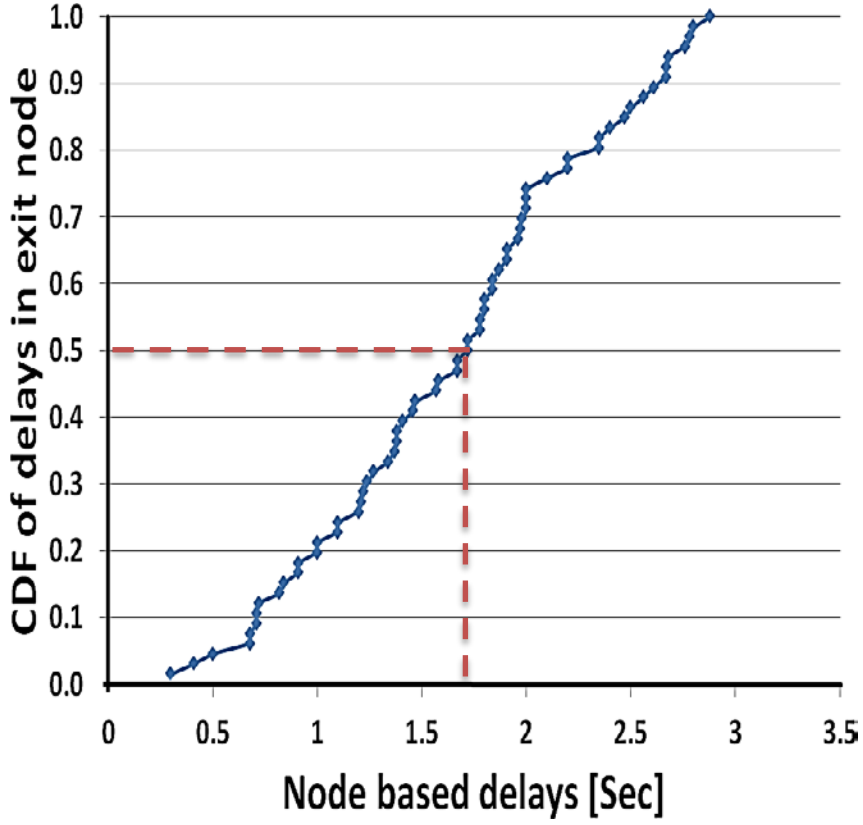
Fig. 4. CDF of node based delays between the TCP input and output connections for (a) entry nodes and (b) exit node.

- Data path Latency for 2 hop circuit = Total RTT delay *(TD)*= $(t_4 - t_1) - (t_3 - t_1)$
  - Average Total latency experience by the packets in full round trip is 2.73 ± 0.08 sec

- Node delays (TD $-RTT_{a+b}$)
  - 50% of all sets of public entry nodes 1,567 have delays less than 2 seconds in average.
  - 50% of delays experienced for different circuits have delays less than 1.72 seconds.



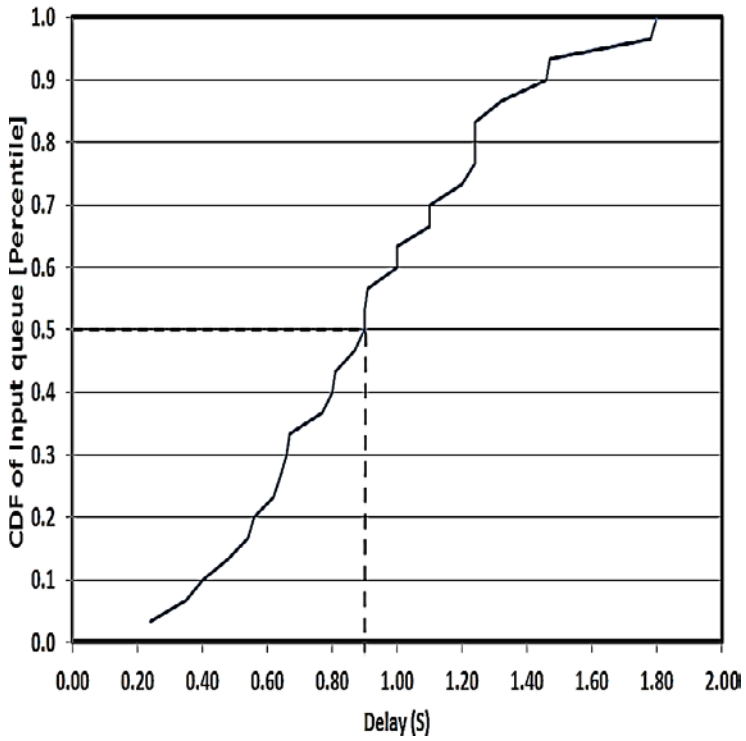
(a) Entry nodes



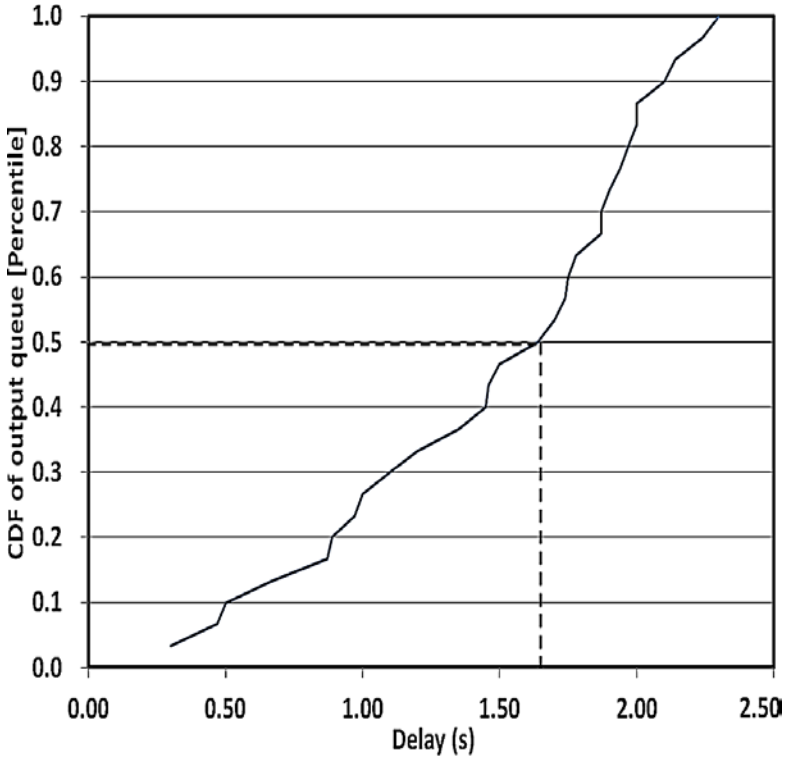
(b) Exit node

- **We observe that increasing delays between selected nodes and the client causes the instability, and variance of RTTs delays along the circuit, due to delays occurs on nodes**

Fig. 5. CDF of TCP stack (a) input and (b) output queue delays. Both results are measured directly at the exit host node when download a 6.3 MB file.



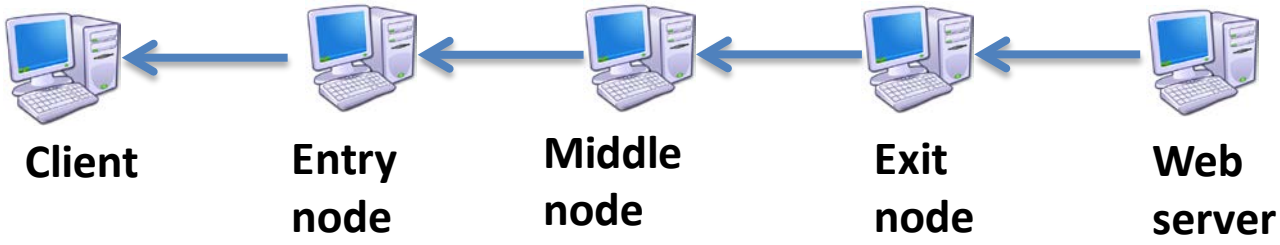
(a) input



(b) output

- The TCP input and output queuing delays are directly measured by capturing the traffic rates passed through the TCP stack.
- 50% of TCP input queues are less than 0.9 seconds and 50% of TCP output queues are less than 1.64 seconds.
- The average CPU usage measured at the exit node when downloaded the file is 2%.
- We stated that the CPU utilization does not affect the delays in the node, since the average CPU usage is very low.

Fig. 3. Second experiment setup. The measurement circuit traffic is from the webserver to client OP during download of 339 MB file.



- TCP Kernel memory usage in nodes
- Affects the TCP window size

- Run our entry and exit relay.
- Both relays runs on Ubuntu 13.10 CPU 2.60 GHz 64 bit with 8 GB of RAM, respectively
- Proper care was taken to observe any significant components that could increase the node based delays
- The experiment was performed 10 times using randomly selected ORs for middle hop
- The allocated bandwidth on both entry and exit relays is greater than 2.5 MB.

Analysis

- We analyzed the sources of node based delays in the TCP stack on two cases;
  - First case is, if the spreading of TCP input and output buffers in the node can exhaust all the TCP kernel buffers.
  - Second case, if the results in the exit node in terms of exhausting all kernel paged memory usage and window sizes, can also affects the neighboring entry node in the same circuit.

TCP kernel buffers

TCP kernel memory usages at the entry and exit relay nodes. The kernel buffer pools monitored is for a single circuit built through entry and exit relay nodes.

Table II.

Kernel Memory				Physical Memory	
Relay nodes	TCP kernel receive buffer size	TCP kernel send buffer size	Non paged	RAM in used	RAM available
Entry	68 KB	28 KB	5.2 MB	1.9 GB	6.1 GB
Exit	63 KB	26 KB	4 MB	1.7 GB	6.3 GB

- The risk to these allocated kernel memory buffers is when multiple circuits are multiplexed over the same TCP connection.

- The sending paged pools accommodating the copied cells from the TCP output buffers can easily running out of memory space, when both entry and exit nodes are accommodating larger circuit streams.

Default buffer sizes allocated by Operating System

TCP receive buffer	TCP send buffer
6291456 Bytes – Max_Th	4194304 **Bytes – Max_Th**
87380 Bytes – Average Buffer size	16384 **Bytes – Average Buffer Size**
4096 Bytes – Min_Th	4096 **Bytes – Min_Th**

First Case:

- When many cells are arriving out of order, the TCP stack can puts more pressure on the kernel buffers to start reducing the memory usage for all pools.
  - Could result in buffer overflowing, memory exhaustion and TCP socket un-writable
  - Unnecessary delays occurs due to larger TCP input and output queues

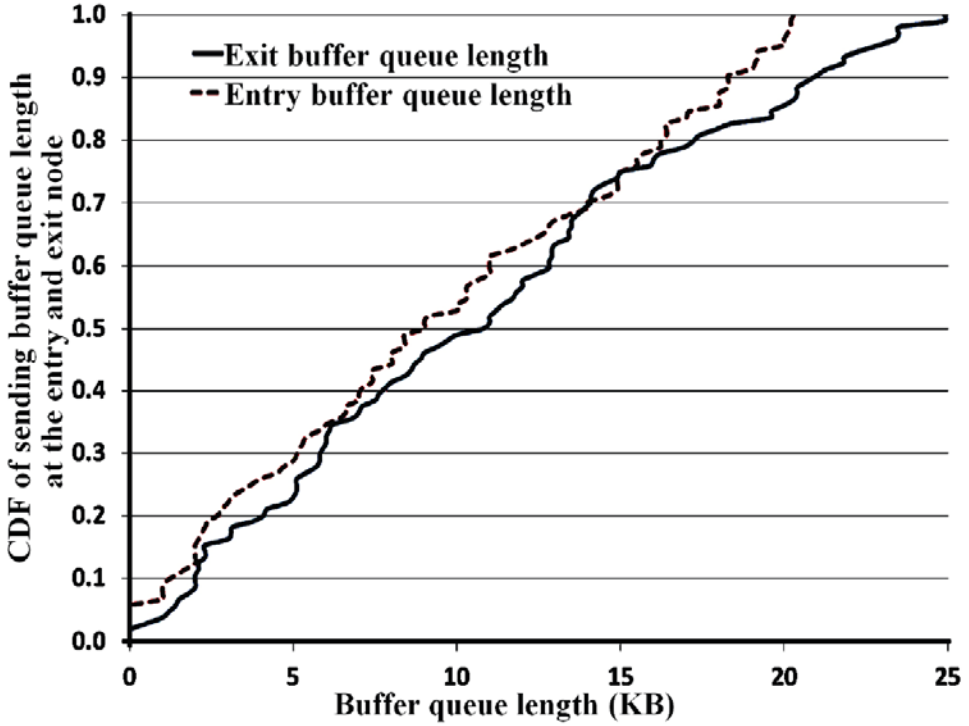
Second Case:

- Increase TCP kernel buffer usage can affects the entry and exit nodes during the download performance, interms of read/write data faster on the corresponding TCP connections.

- Average processing rates [TCP Throughput] on entry and exit node
  - Entry node – TCP input rate is 95 KB/s and the TCP output rate is 65 KB/s.
  - Exit node - TCP input rate is 85 KB/s and the TCP output rate is 55 KB/s.

Unreliable TCP throughput degradation and must buffer up the packet to one full TCP window sizes. This situation leads to increasing TCP buffer length in all nodes.

Fig. 6. CDF of buffer queue length at the entry and exit node when download a 339 MB file.



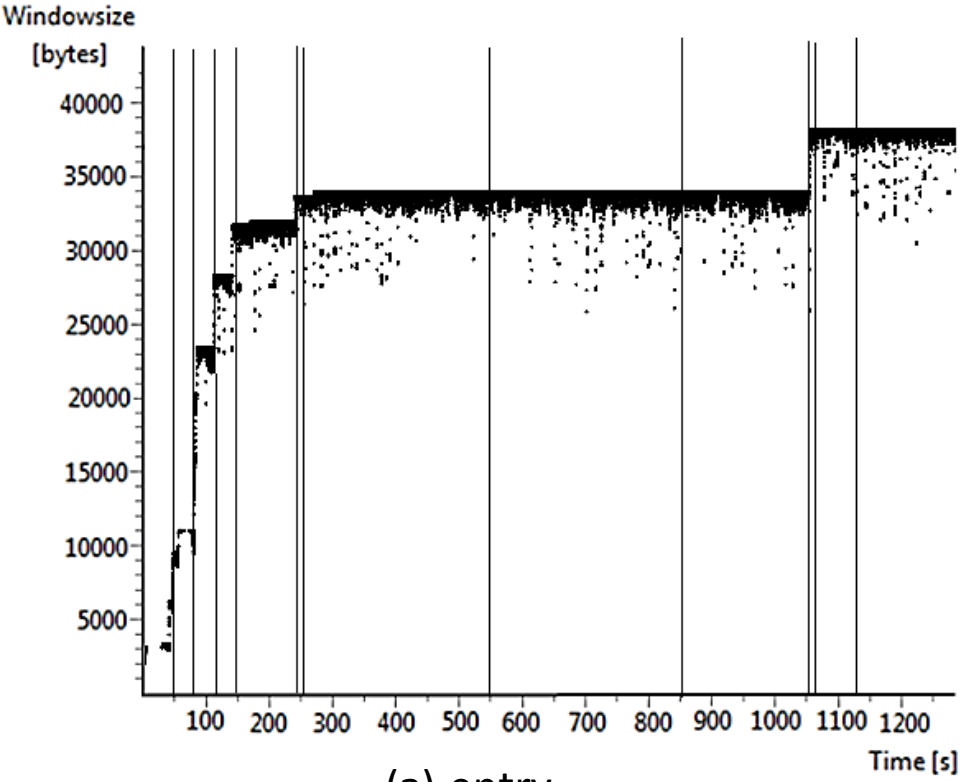
- Effects of TCP kernel send buffers for both entry and exit nodes that increase in buffer length ratio.

- 50% of the circuit we measured at the entry node has 9 KB of TCP sending buffer length and, exit node has 10.8 KB of buffer length.

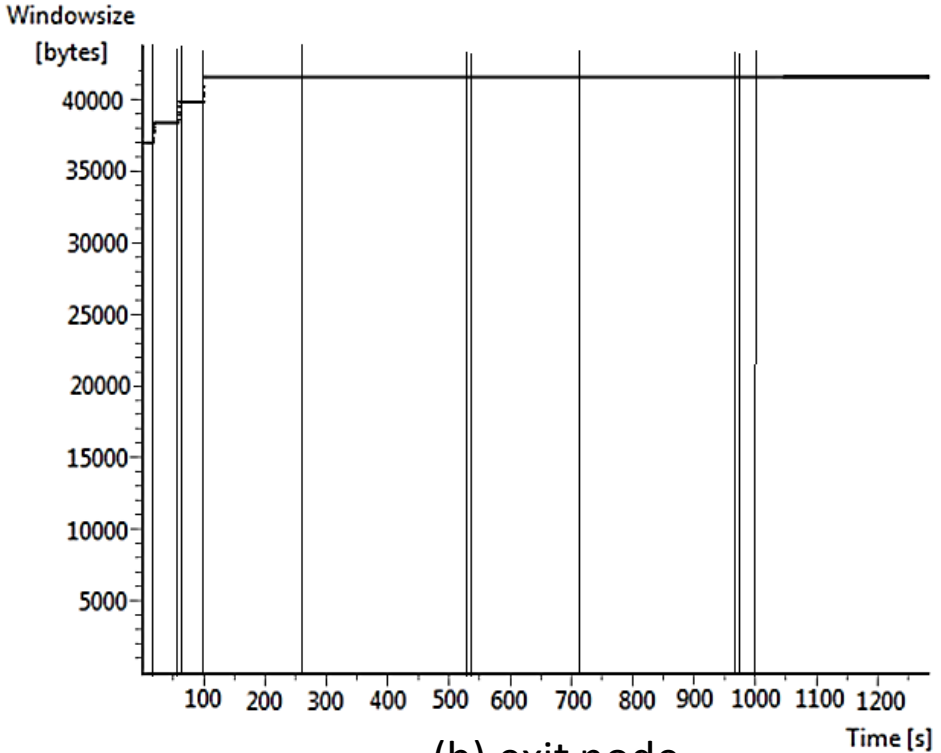
- This scenario depicted a risk condition of TCP kernel buffers at the TCP hosts for both entry and exit that never goes empty, which contributed to increase the node based delay.

TCP window size

Fig. 7. Measurement base on node inability to read more data at the (a) entry node and (b) exit node when download 339 MB file.



(a) entry



(b) exit node

- Increasing TCP queuing buffers can reduce TCP advertised window on Tor nodes for not receiving more data, especially when the allocated TCP receive buffers runs out of space.

- The advertised window at the entry node gradually increases till it stays on the 35 KB, and further increase to 40 KB. At the exit node, the advertised window increases from 37 KB at the time of measurement and stays at 42 KB.

Conclusion

- The variance of TCP window sizes between the entry and the exit node result in the invocation of TCP flow control based on inability to read more data. TCP input/output buffer are full.

- The buffers are swell in the TCP Input buffer
  - TCP buffer is full
  - TCP sending buffer does not write the packets on the circuit
  - Setting of the OS

- Tor's congestion control in application layer does not always keep a steady flow of cells in flight between routers and transport upstream and downstream flow.

- We observed that average CPU utilization in the Tor routers does not affects the node based delays, since the usage is lower

- Variances of the total round trip delays *(TD)* have great influence from the delays in the nodes.

Future Work

- There is need for proper implementing of congestion control in Tor. This would minimizes the longer circuit queuing and improve the TCP advertised window sizes.

- Improve the transmission between the Tor application layer and the TCP buffers would reduce the queuing delays of cells.

- Restrict the number of circuits based on selected nodes
  - Control by the buffer size usages.

Thank you for listening

Questions & Answers

- Delays in the application layer:
  - Tor router can spends most of its time executing AES operations during cell processing.
  - The additional use of TLS to encrypt outgoing traffic between nodes will increase the overhead.

AES Operations	
Reads	30 microsec
Write	40 microsec
Per TLS Link (En/Decryption)	70 microsec
6 TLS Link (En/Decryption)	420 microsec (a cell to travel up the path and another cell to be returned in reply).
Expected computational latency along a circuit is	540 microsec for a full trip.

Joel Reardon, and Ian Goldberg. Improving Tor using a TCP-over-DTLS Tunnel. USENIX Security Symposium, page 119-134. USENIX Association, (2009)

Related Studies

- **Congestion-aware Path Selection for Tor. Each client maintains a congestion list of all known relays paired with a number of congestion times for each relay.**
  - **Instant Response of relays to switching to another circuit.**

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

- **Torchestra", Reducing Interactive Traffic Delays over Tor. Create two separate connections between each pair of nodes: one for interactive traffic, and one for bulk traffic.**

Gopal, D., Heninger, N.: Torchestra: Reducing Interactive Traffic Delays over Tor. In: Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society. pp. 31–42. WPES '12, ACM, New York, NY, USA (2012)

- **The Exponentially Weighted Moving Average (EWMA) is a statistic used to calculate the moving average while giving more weight to recent data**

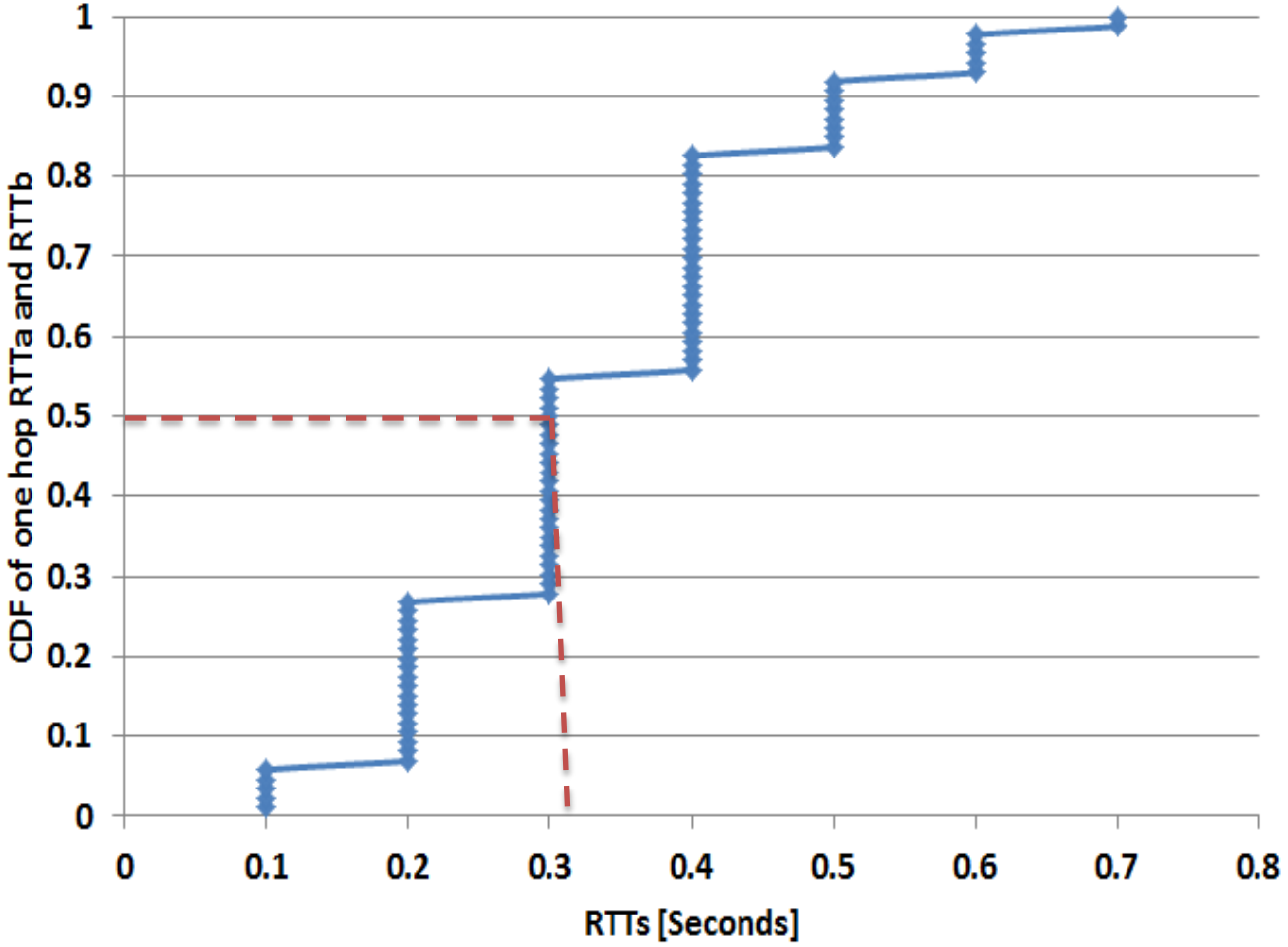
C. Tang and I. Goldberg. An improved algorithm for Tor circuit scheduling. In proceedings of the 17th ACM conference on Computer and communications security , pages 329{339. ACM, 2010

- **Throttling Tor Bandwidth Parasites**

Jansen, R., Syverson, P., Hopper, N.: Throttling Tor Bandwidth Parasites. In: Proceedings of the 21st USENIX Security Symposium (August 2012)

Experiment Results

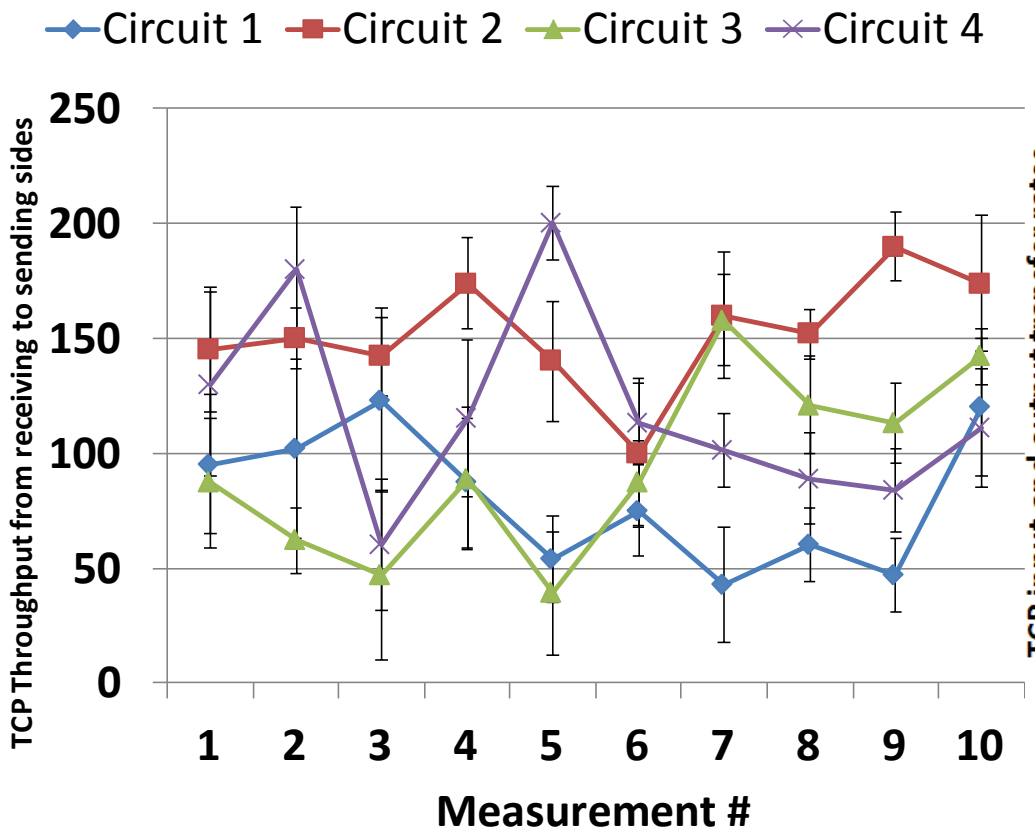
One hop TCP RTT delays from the client to the entry relays, and from exit to the entry relays

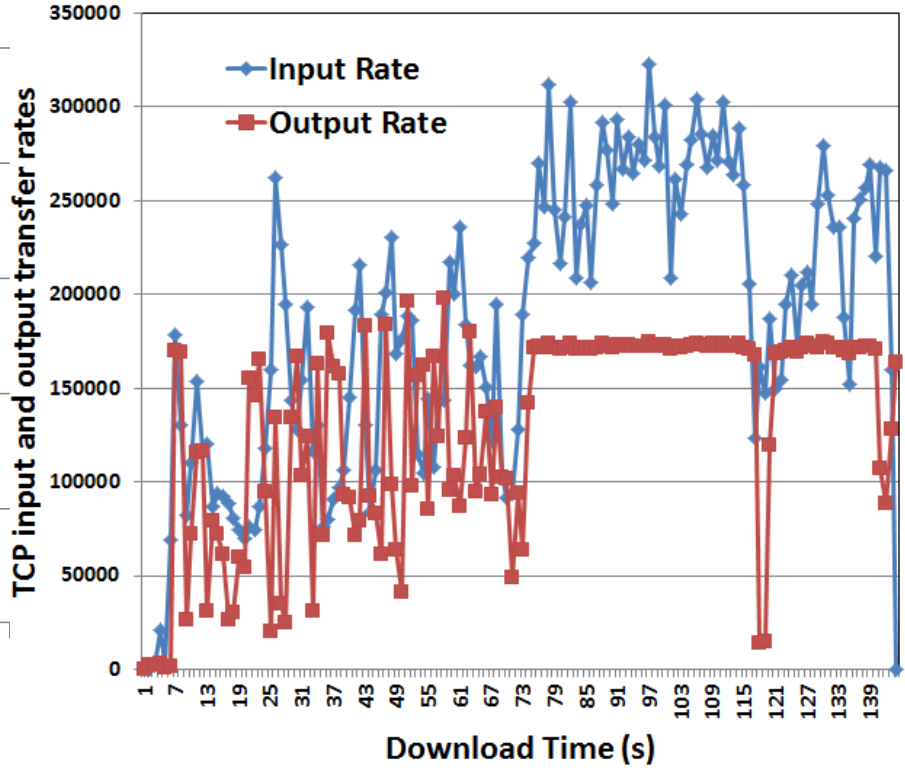


- TCP Ping SYN from the Exit and entry relay to the middle relays are relatively small, due to one hop measurements.
- 50% relays have RTT delays less than 0.3 seconds.
- Delays occurs in full round trip *TD* should be larger due to additional delays in the TLS with the increasing processing delays in the nodes.

One-hop (multiplex circuits)

- TCP Throughput for circuit 1,2,3,4 @ 1 hop - 75.4 KB/s, 145 KB/s, 64 KB/s, 120 KB/s



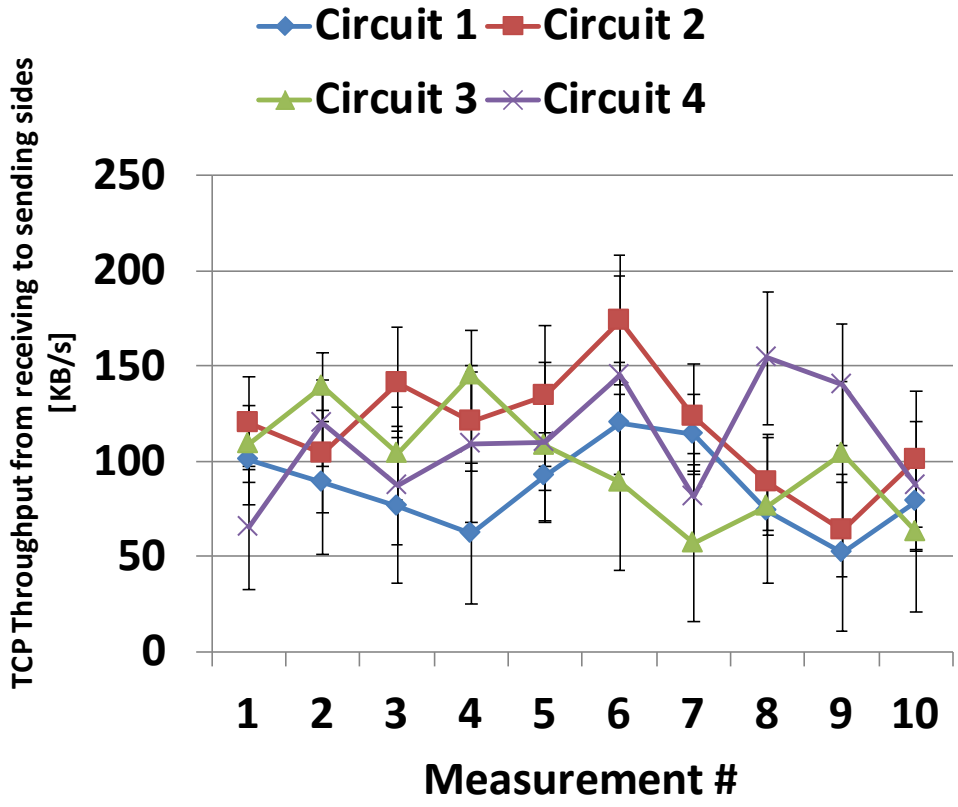


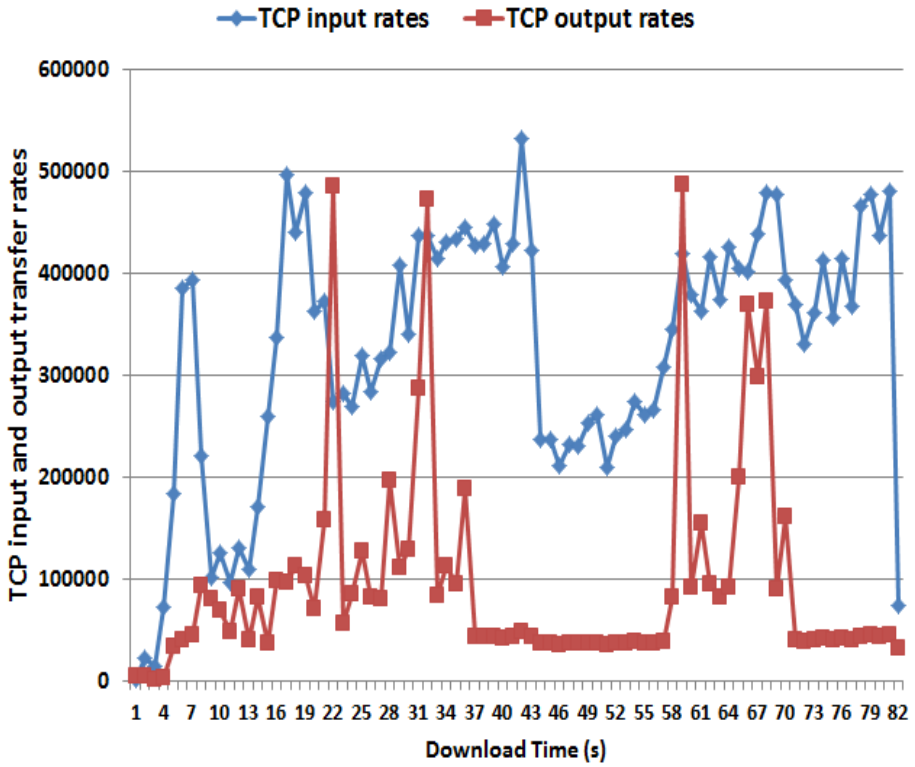
- The duration of time to download is taken from the start time of read time till the last write time. The file download is 23.6 MB

TCP receive buffer	TCP send buffer
6291456 Bytes – Max_Th	4194304 **Bytes – Max_Th**
87380 Bytes	16384 **Bytes**
4096 Bytes – Min_Th	4096 **Bytes – Min_Th**

Two-hops (multiplex circuits)

- TCP throughput for Entry node Circuit 1,2,3,4 – 85.1 KB/s, 100 KB/s, 109 KB/s, 94 KB/s





- Avg. TCP throughput for Exit node C1,2,3,4 – 85.6 KB/s, 65 KB/s, 116 KB/s, 94 KB/s