**Deployment of TCP congestion control algorithms using Network Simulator (ns-3)**

**MINOR PROJECT 1**

Submitted by:

**Yash Shukla (9917103124)**

**Nikhil Kumar (9917103088)**

**Kulgaurav Singh (9917103074)**

**Tejasv Sharma (9917103072)**

**Table of Contents**

Page No.

**Abstract** i

**List of Figures** iii

**Abbreviations and Nomenclature** iv

**1: Introduction** 1

1.1 General 1

**2: Background Study** 1

2.1 TCP congestion control algorithms 1

2.1.1 New Reno 3

2.1.2 Veno 3

2.1.3 Westwood 4

2.1.4 BIC 4

2.1.5 CUBIC 6

**3: Requirement Analysis** 7

3.1: Software Requirements 7

3.2: Hardware Requirements 7

3.3: Functional Requirements 7

3.4: Non-Functional Requirements 8

**4: Detailed Design** 9

4.1: Topology 9

4.2: Implementation 9

**6: Results** 10

6.1: Congestion window vs Time Plots 10

6.2: Comparative Analysis 13

**7: Conclusion** 15

**8: References** 16

**List of Figures**

**Figure Title Page**

4.1 Topology 7

6.1(a) Congestion window vs time Plot for New Reno 12

6.1(b) Congestion window vs time Plot for Veno 12

6.1(c) Congestion window vs time Plot for Westwood 13

6.1(d) Congestion window vs time Plot for BIC 13

6.1(e) Congestion window vs time Plot for CUBIC 14

6.2(a) Throughput vs Bandwidth 15

6.2(b) Throughput vs Bandwidth (CUBIC vs BIC) 16

**Abbreviations and Nomenclature**

TCP Transmission Control Protocol

UDP User Datagram Protocol

Ns-3 Network Simulator -3

IPv4 IP Addressing version 4

IP Internet Protocol

ACK Acknowledgement

RFC Request For Comments

BIC Binary Increase Congestion control

RTT Round Trip Time

MSS Maximum Segment Size

**1: Introduction**

World of Internet is growing as we speak. Nowadays, 4G networks are becoming famous for which higher bandwidth and lower latency are being used. Applications like video streaming in 4G increase traffic in network very fast. With the modern advancements in networking, comes the emergence of new network environments such as Gigabit Ethernet or satellite links with challenging characteristics: high bit-error rate, long propagation delay, high link capacity, and asymmetric channels. Therefore congestion control is important. Optimization will increase the throughput, efficiency and downloading speed which is required by operators as well as customers.

Network congestion may occur when a sender overflows the network with too many packets. At the time of congestion, the network cannot handle this traffic properly, which results in a degraded quality of service. The typical symptoms of congestion are: excessive packet delay, packet loss and retransmission. Transport Layer’s TCP (Transmission Control Protocol) Congestion Control techniques prevent congestion or help mitigate the congestion after it occurs. Standard TCP has been enhanced for the Future Internet, resulting in many variants such as those studied in our project.

The study of existing congestion control algorithms and the development of any new enhancements gain substantial benefits through the use of open-source network simulators such as ns-3. We implement and compare variants including New Reno, Veno, BIC, Westwood and CUBIC. This project presents our implementation details of these algorithms. Through our evaluation, we highlight the key features of each algorithm.

**2: Background Study**

This section provides the theoretical background of different congestion control algorithms implemented in our project including the standard TCP New Reno, Veno, Westwood and BIC.

**2.1: TCP Congestion Control Algorithms**

TCP congestion control algorithms may be loss based, delay based or hybrid based. Loss based algorithms treat the occurrence of a packet loss as an indication of congestion. Delay-based algorithms infer congestion based on the increasing queueing delay when traffic load exceeds network capacity. Hybrid algorithms take advantage of both loss and delay-based mechanisms. Most TCP congestion control algorithms are derivatives of the standard defined in RFC 5681[1], which is known as the Reno algorithm. The standard specifies four intertwined algorithms that together play a principal role in the stabilization of the Internet and the prevention of congestion: slow start, congestion avoidance, fast retransmit, and fast recovery. The implementation of these algorithms requires the definition and maintenance of three state variables: cwnd, rwnd, and ssthresh. Congestion window (cwnd) determines the amount of data a sender can transmit before it receives an ACK to prevent network overflow. The receiver window (rwnd) indicates the amount of data a receiver is willing to accept. The actual sending window is the minimum of cwnd and rwnd. Slow start threshold (ssthresh) provides the transition point between slow start and congestion avoidance phases.

The slow start allows TCP to regulate the case where too much data is sent to a network and the network is incapable of processing that amount of data, thus resulting in congestion. During slow start, cwnd is incremented by 1 for every new ACK received, resulting in an exponential increase of the sending rate until a loss happens as shown in Equation 1.

cwnd = cwnd + 1 (1)

Once the ssthresh is reached and cwnd > ssthresh, congestion avoidance algorithm is started and the cwnd is incremented by 1 for every RTT (round trip time), resulting in a linear increase over time until the experience of a loss. This is equivalent to the cwnd modification per Equation 2 upon a new ACK receipt.

cwnd = cwnd + 1/cwnd (2)

When the segment acknowledgements are not received, ssthresh is set to half of the current cwnd size.

The fast retransmit algorithm is responsible for promptly detecting and recovering lost data by observing the number of received duplicate ACKs, with the arrival of three duplicate ACKs signifying the loss of a segment. At this moment TCP makes cwnd = 1 and ssthresh = cwnd/2.

The fast recovery governs data transmission after fast retransmit until a new ACK arrives informing the recovery of the loss. The occurrence of a loss requires Reno to halve its slow-start threshold and sending rate according to Equations 3 and 4, respectively.

ssthresh = cwnd/2 (3) cwnd = ssthresh + 3 (4)

**2.1.1: New Reno**

New Reno (RFC 6582) modifies the Reno fast recovery algorithm explained above by introducing a mechanism for responding to partial acknowledgments to enhance TCP’s ability by recovering more efficiently from multiple losses occurring in a single sending window. New Reno defines an additional state variable named recover to keep track of the highest sequence number transmitted before the sender enters fast retransmit, and it only leaves its fast recovery state upon the receipt of a full ACK, which is an ACK that acknowledges all sent data up to and including recover. In case a partial ACK arrives with acknowledgment number less than recover, the algorithm remains in fast recovery trying to retransmit the next in-sequence packet while sending a new segment if cwnd and rwnd allow. The New Reno algorithm is an alternate solution for multiple data loss recovery in the absence of TCP selective acknowledgment.

**2.1.2: Veno**

TCP Veno enhances Reno algorithm to deal with random loss in wireless access networks by employing the Vegas algorithm for estimating the current network condition to identify the actual cause of a loss. Specifically, Veno does not use the estimated number of packets enqueued (backlog N calculated in Equation 5) at the bottleneck to proactively detect congestion, but to distinguish between a corruption-based loss and a congestion-based loss. When Veno learns that a loss is non-congestive, instead of halving ssthresh (Eq. 3), it reduces ssthresh by a smaller amount using Equation 6. Veno also refines the Reno congestion avoidance algorithm by increasing the sending rate by 1 every 2 RTTs if the backlog exceeds its predefined threshold β, allowing it to operate longer in the stable state during which network capacity is fully utilized.

N = actual × (RTT − BaseRTT) = diff × BaseRTT (5)

ssthresh = cwnd × 4/ 5 (6)

**2.1.3: Westwood**

TCP Westwood is a sender-side-only modification to New Reno. In Westwood, an "Eligible Rate" is estimated and used by the sender to update ssthresh and cwnd upon loss indication, or during its "Agile Probing" phase, a proposed modification to the well-known Slow Start phase. In addition, a scheme called Persistent Non Congestion Detection has been devised to detect persistent lack of congestion.

**2.1.4: BIC**

BIC TCP (Binary Increase Congestion control) is optimized for high speed networks with high latency: so-called "long fat networks". For these networks, BIC has significant advantage over previous congestion control schemes in correcting for severely underutilized bandwidth.

BIC implements a unique congestion window (cwnd) algorithm. This algorithm tries to find the maximum cwnd by searching in three parts: binary search increase, additive increase, and slow start. When a network failure occurs, the BIC uses multiplicative decrease in correcting the cwnd.

Algorithm:-

One step of increasing cwnd:

if (cwnd < wmax) // binary search OR additive

bic\_inc = (wmax - cwnd) / 2;

else // slow start OR additive

bic\_inc = cwnd - wmax;

if (bic\_inc > Smax) // additive

bic\_inc = Smax;

else if (bic\_inc < Smin) // binary search OR slow start

bic\_inc = Smin;

cwnd = cwnd + (bic\_inc / cwnd);

One step of decreasing cwnd:

if (cwnd < wmax) // fast convergence

wmax = cwnd \* (2-β) / 2;

else

wmax = cwnd;

cwnd = cwnd \* (1-β);

where,

Smax: the maximum increment

Smin: the minimum increment

wmax: the maximum window size

β: multiplicative window decrease factor

cwnd: congestion window size

bic\_inc: window increment per RTT (round trip time)

**2.1.5: CUBIC**

CUBIC is an enhanced version of BIC. As the name represents, the window growth function of CUBIC is a cubic function. There are two components of window growth, the first is a concave part where the window size quickly increases before the last congestion event. Next is the convex growth where CUBIC probes for more bandwidth, slowly at first then very rapidly. More specifically, the congestion window of CUBIC is determined by the following function:

Wcubic = C(t − K)^3 +Wmax (7)

where C is a scaling factor, t is the elapsed time from the last window reduction, Wmax is the window size just before the last window reduction and K = (Wmax\*β/ C)^1/3, where β is a decrease factor applied for window reduction at the time of packet loss. If cwnd is less than the window size that TCP would reach at time t after the last loss event, then CUBIC is in the TCP. Otherwise, if cwnd is less than Wmax, then CUBIC is in the concave region, and if cwnd is larger than Wmax, CUBIC is in the convex region.

*TCP-friendly region: -* When receiving an ACK in congestion avoidance, we check whether the protocol is in the TCP region or not. We analyze the window size of TCP in terms of the elapsed time t. Using a simple analysis we can find the average window size of additive increase and multiplicative decrease (AIMD) with an additive factor α and a multiplicative factor β to be the following function:

1/ RTT ( (α\*2 – β)/2βp)^1/2 (8)

*Concave region*: - When receiving an ACK in congestion avoidance, if the protocol is not in the TCP mode and cwnd is less than Wmax, then the protocol is in the concave region. In this region, cwnd is incremented by

(W(t+RTT )−cwnd )/cwnd (9)

*Convex region*: - When the window size of CUBIC is larger than Wmax, it passes the plateau of the cubic function after which CUBIC follows the convex profile of the cubic function. Since cwnd is larger than the previous saturation point Wmax, this indicates that the network conditions might have been changed since the last loss event, possibly implying more available bandwidth after some flow departures.

*Multiplicative decrease*: - When a packet loss occurs, CUBIC reduces its window size by a factor of β. We set β to 0.2

*Fast convergence*: - With fast convergence, when a loss event occurs, before a window reduction of the congestion window, the protocol remembers the last value of Wmax before it updates Wmax for the current loss event. At a loss event, if the current value of Wmax is less than the last value of it, this indicates that the saturation point experienced by this flow is getting reduced because of the change in available bandwidth. Then we allow this flow to release more bandwidth by reducing Wmax further.

**3: Requirement Analysis**

Requirement analysis “also called requirements engineering”, is the process of determining user expectations for a new or modified product/software.

**3.1: Software Requirements**

* Language C++
* Network Simulator NS-3 (version 3.30)
* Linux Ubuntu (version 19.04) or any other distribution

**3.2: Hardware Requirements**

* Processor Intel i3 or above
* RAM 4GB or above

**3.3: Functional Requirements**

**3.3.1: Design and Implementation Constraints**

This project is constrained by the number of TCP variants available for implementation on ns3. Implementation of new tcp variants is complex and difficult. Also, ns3 simulation is limited to LINUX kernel only.

**3.3.2: External Interface Requirements**

**User Interface**

* Front-end software – Network simulator (ns3)

**Hardware Interface**

* Linux (Ubuntu distribution version 19.04 or above)
* 4GB RAM or above

**Software Interface**

* This project requires network simulator NS3 version 3.29 or above

**3.4: Non-Functional Requirements**

**3.4.1: Performance Requirements**

Depending on the LINUX distribution and RAM of the system, the simulation will take no more than 10 sec to compile. Once compiled, the program will not take more than 2sec to run and show simulation results.

**3.4.2: Availability**

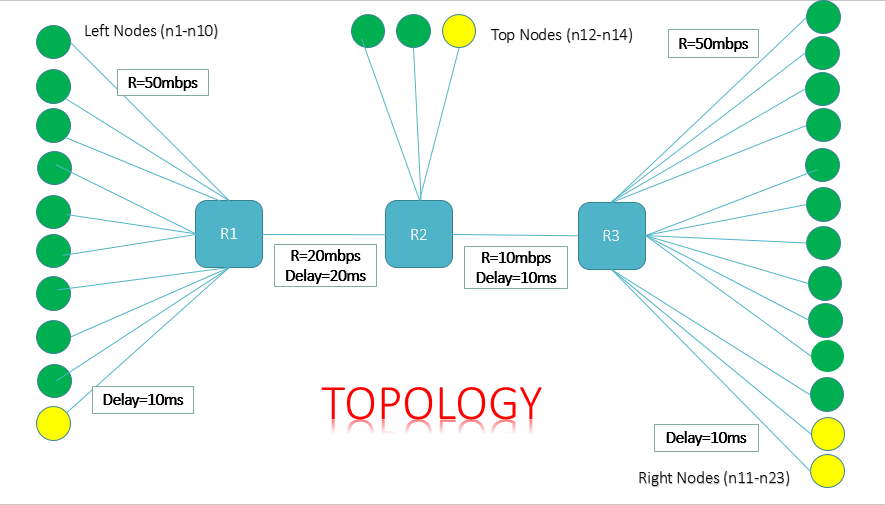
Users can use the program in Linux kernel with ns3 being properly installed.

**3.4.3: Security**

There is no security limitation as there is no important information to keep. No user personal information is required.

**4: Detailed Design**

**4.1: Topology**



**Fig 4.1 -** Topology

Green nodes refer to TCP sources and sinkers while the yellow ones are UDP sources and sinkers.

All the nodes are connected via peer-to-peer architecture. Nodes have a bandwidth of 50Mbps and delay of 10ms. The blue boxes are the three routers R1, R2, R3. R1-R2 have a bandwidth of 20Mbps and delay of 20ms while R2-R3 have a bandwidth of 10Mbps and delay of 10ms.

Number of packets sent = 100000

Size of each packet = 1040 bytes

Simulation Time = 200ms

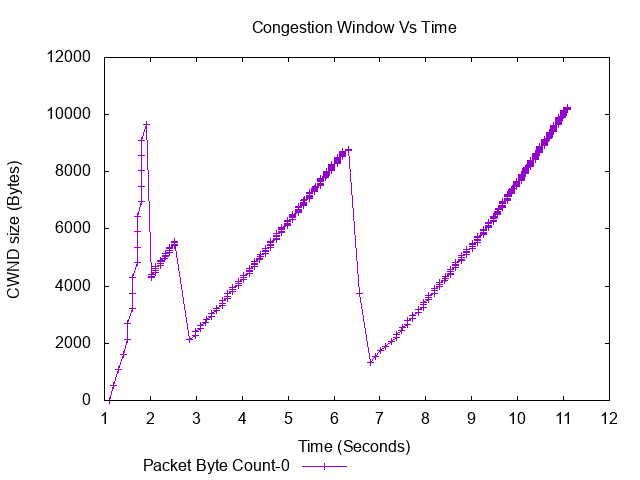
**4.2: Implementation**

10 (9 TCP and 1 UDP) Left nodes start sending data in packets to router R1 at time t=1ms. Then the router R1 push these packets to R2. At router R2, 3 (2 TCP and 1 UDP) Top nodes add data and all these packets are then sent to R3 and finally to destination sinkers (Right Nodes). We have 13 Right nodes with 11 TCP sinkers and 2 UDP sinkers. Sinkers start at time t=0.

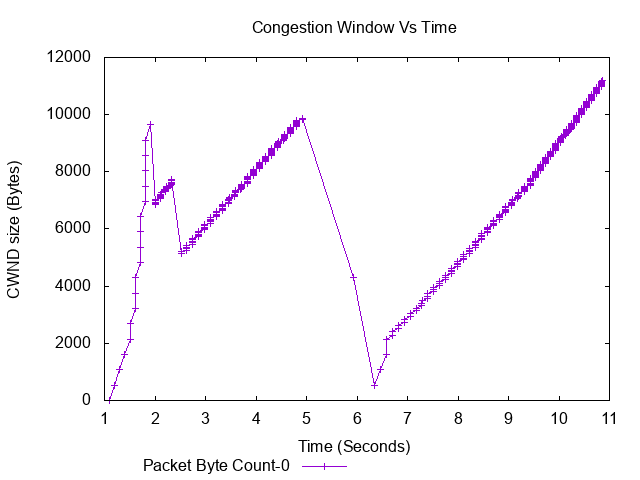
Since bandwidth of R1-R2 and R2-R3 are less than that of the source nodes, congestion takes place. Different algorithms implemented here tackle this congestion in different ways.

**6: Results**

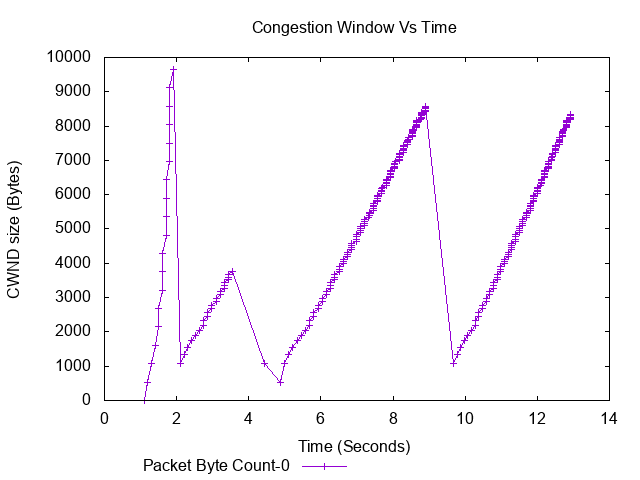
**6.1: Congestion window vs Time Plots**

****

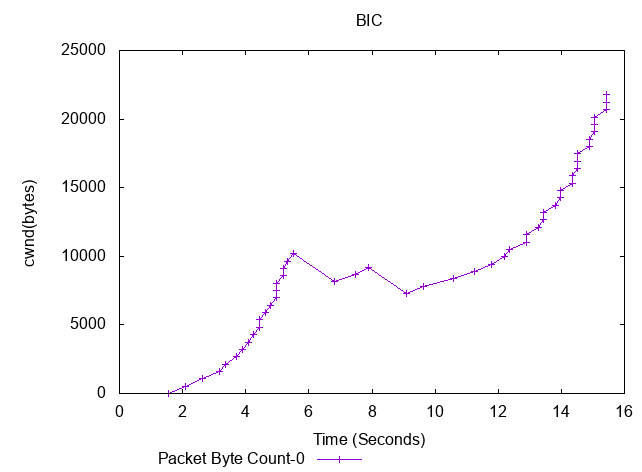
**Fig 6.1(a) –** CWND vs Time Plot for New Reno

****

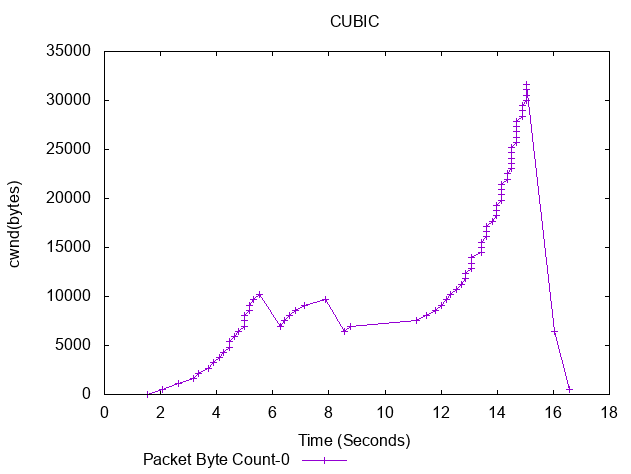
**Fig 6.1(b) –** CWND vs Time Plot for Veno

****

**Fig 6.1(c) –** CWND vs Time Plot for Westwood



**Fig 6.1(d) -** CWND vs Time Plot for BIC



**Fig 6.1(e) -** CWND vs Time Plot for CUBIC

**6.2: Comparative Analysis:-**

Throughput is the aggregate data that is successfully received to all the network terminals. It is an important performance indicator and for maximum throughput, both the connection ends (sender and receiver) should utilize maximum link capacity.

Throughput = Sum of all bits received / connection time

Table 1 shows the value of maximum throughput for various TCP variants based on different values of bottleneck bandwidths and delays. Bandwidth and delay being **50Mbps** and **10ms**.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | R1-R2 | R2-R3 | New Reno  (Mbps) | Westwood  (Mbps) | Veno  (Mbps) | BIC  (Mbps) |
| Rate1(Mbps)  Delay1(ms) | 20  20 | 10  10 | 0.840 | 0.581 | 0.839 | 0.898 |
| Rate2(Mbps)  Delay2(ms) | 5  100 | 2.5  50 | 0.208 | 0.131 | 0.208 | 0.197 |
| Rate3(Mbps)  Delay3(ms) | 2.5  50 | 5  100 | 0.247 | 0.136 | 0.226 | 0.206 |
| Rate4(Mbps)  Delay4(ms) | 2.5  50 | 2.5  50 | 0.237 | 0.285 | 0.237 | 0.224 |
|  |  |  |  |  |  |  |

Table 1

Table 2 shows the value of maximum throughput comparison for CUBIC, BIC and New Vegas based on different values of bottleneck bandwidths and delays. Bandwidth and Delay being **500MBbps** and **10ms.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | R1-R2 | R2-R3 | CUBIC (Mbps) | BIC (Mbps) |
| Rate1 (Mbps)  Delay1 (ms) | 200  20 | 150  10 | 9.505 | 9.503 |
| Rate2 (Mbps)  Delay2 (ms) | 20  20 | 10  10 | 8.329 | 7.174 |
| Rate3 (Mbps)  Delay3 (ms) | 5  100 | 2.5  50 | 9.566 | 9.485 |

Table 2

**7: Conclusion**

We have successfully performed the TCP congestion control algorithms-

* New Reno
* Veno
* Westwood
* BIC

**Case1**- When bt1 (bottleneck1) was 20 Mbps and bt2 was 10 Mbps BIC gave best average throughput of 0.898Mbps followed by New Reno (0.840Mbps), Veno (0.839Mbps) and lastly Westwood (0.581 Mbps).

**Case2** - When bt1 was reduced to 5Mbps and bt2 to 2.5Mbps New Reno and Veno gave best throughput among the four (0.208Mbps) followed by BIC (0.197Mbps) and Westwood (0.131Mbps).

**Case3** – When bt1 was reduced to 2.5Mbps and bt2 increased to 5Mbps, New Reno gave best avg. throughput of 0.247Mbps, Veno (0.226Mbps), BIC (0.206Mbps) and at last Westwood (0.136Mbps).

**Case4** – On keeping bt1 and bt2 2.5Mbps, Westwood gave best performance (0.285Mbps), New Reno and Veno gave 2nd best (0.237Mbps) and BIC at last (0.224Mbps).

We can observe that BIC favors higher bandwidth networks while New Reno and Veno perform better for smaller bandwidths.

Comparing CUBIC and BIC🡪

* CUBIC
* BIC

**Case1**- When bt1 (bottleneck1) was 200 Mbps and bt2 was 150 Mbps CUBIC gave best average throughput of 9.505Mbps followed by BIC (9.503Mbps).

**Case2**- When bt1 (bottleneck1) was reduced to 20 Mbps and bt2 to 10 Mbps CUBIC gave best average throughput of 8.32Mbps followed by BIC (7.174Mbps) .

**Case3**- When bt1 (bottleneck1) was 5 Mbps and bt2 was 2.5 Mbps CUBIC gave best average throughput of 9.56Mbps followed by BIC (9.38Mbps) .

Hence, from our analysis, we can clearly observe that for high bandwidths (**long fat networks**) CUBIC performs better than the other TCP variants, whereas for moderate or small bandwidths, CUBIC and BIC gave almost same throughput.

**8: References**

[1] M.Allman, V.Paxson, ICSI, E.Blanton, Purdue University, September 2009.

[2] La, Richard. “Analysis and Comparison of TCP Reno and Vegas.” INFOCOM'99. Eighteenth … (1999): n. pag. Print.

[3] G. Riley and T. Henderson, “The ns-3 network simulator” in Modeling and Tools for Network Simulation (K. Wehrle, M. Gunes, and J. Gross, eds.), pp. 15--34, Springer Berlin Heidelberg, 2010.

[4] Joel Sing and Ben Soh, “TCP New Vegas: Improving the Performance of TCP Vegas Over High Latency Links” Fourth IEEE International Symposium on Network Computing and Applications, Cambridge, MA, 2005, pp. 73-82.

[5] Joel Sing and Ben Soh, “TCP New Vegas: Improving the Performance of TCP Vegas Over High Latency Links”  Fourth IEEE International Symposium on Network Computing and Applications, Cambridge, MA, 2005, pp. 73-82.