**Fundamentals of Computer Networks**
**CSE 534**
**Project Report**

# Title - An Improved TCP Congestion Control Algorithm of Based on Bandwidth Estimation in Heterogeneous Networks

**By - Rakshit Gautam (111168264), Naveen Kumar Rai (111207633)**

**Abstract -**
We are implementing an improved TCP variant (called as TCP NewBR) for heterogeneous networks, as described in the paper [1]. The paper proposes an improved bandwidth estimation algorithm that uses bottleneck link utilization, and a more precise time interval of ACK to improve the accuracy of estimation. The paper also proposes a faster recovery and faster retransmission algorithm based on the queue length of the bottleneck link. We have compared the bandwidth estimation of TCPNewBR with the available bandwidth, compared the throughput with that of TCP Westwood, and compared its fairness with other TCP Variants.

**Introduction -**
The Bandwidth Estimation Algorithm of TCP NewBR builds upon the Bandwidth Estimation and Rate Estimation algorithms of TCP Westwood. TCP Westwood is a sender-side modification of TCP congestion window algorithm that performs better than TCP Reno in wired and wireless networks.

TCPW Bandwidth estimate uses TCP acknowledgement to compute the estimate.
TCPW BE = size of packets confirmed by the ACK / the arrival time interval of confirmation
TCPW BE is an instantaneous available bandwidth estimation of the bottleneck link. Since TCP traffic has a burst characteristic, BE is an overestimate. BE algorithm is more effective in networks with random error and less effective in networks with congestion losses.

TCPW Rate Estimate uses last RTT as the sample interval to estimate bandwidth.
TCPW RE = size of packets confirmed by the ACK during last RTT interval / RTT
This is the average sending rate

TCPW Rate Estimation provides a good estimate of bandwidth under the condition of packet loss due to congestion, but it underestimates the bandwidth in case of random loss, resulting in low throughput. TCPNewBR computes a utilization factor based on the queuing delay and maximum queuing delay, and uses this factor to merge BE and RE estimates of TCPW.

TCPNewBR uses an improved faster retransmission and faster recovery algorithm. The algorithm first estimates the queue length of the bottleneck. If the queue length is less than a threshold, it indicates that the packet loss is due to random error, and accordingly reduces the congestion window to avoid waste of bandwidth. Prediction of bottleneck queue length

maintains the end to end principle, and is done by combining bandwidth estimate with the round-trip delay.

**TCPNewBR Bandwidth Estimation Algorithm -**

It has been found that the BE of TCPW is a better estimate in networks with random losses while RE is a better estimate in networks with congestion losses. Therefore, these estimation techniques have limitations on different applications.
TCP NewBR uses the above 2 estimates and uses bottleneck link utilization as a factor to weigh the 2 bandwidth estimates. TCP NewBR can distinguish random loss from congestion loss and correctly predict available bandwidth in case of no packet loss.
TCP NewBR uses the timestamps of ACK's, reflecting the received time of ACK sent to acknowledge the corresponding packet received at the receiver. This makes it easier to remove the effects of asymmetric links and reverse flow on bandwidth computation. The bandwidth estimate is independent of the return path of the ACK.

The maximum value of RTT (current round trip delay) is calculated as -

$$rtt_{max} = \beta * rtt_{(n-1)max} + ((1-\beta)/2) * (rtt_{(n-1)} + rtt_n)$$

$rtt_{max}$ represents the maximum value of rtt using an exponential averaging of RTT measurements.
β , $rtt_{(n-1)max}$ , $rtt_n$ , and $rtt_{(n-1)}$ are the exponential smoothing factor.
β = 19/21

Using the above formula, $rtt_{max}$ is calculated and is further used to compute Link Surplus Index (u) as follows -

$$d = rtt - rtt_{min}$$
$$d_{max} = rtt_{max} - rtt_{min}$$
$$u = exp(-\delta * (d/d_{max}))$$

Here, $d$ represents the current queuing delay, $d_{max}$ represents the maximum queuing delay, u is the Link Surplus Index.
We use the u computed above to estimate the bandwidth as -

Eligible Rate Estimation (ERE) = $u * BE + (1-u) * RE$

The assumption made here is that when $rtt$ is equal to $rtt_{min}$ , the bottleneck link is underutilized, and the ERE is set to BE. This means that u is set to 1.
As $rtt$ increases, ERE is exponentially reduced to appear close to RE. Thus, we reduce u exponentially as $rtt$ increases.
$\delta$ is a constant chosen to make sure that u approaches to 0 as $rtt$ approaches $rtt_{max}$ . $\delta$ is generally taken as 10.

**TCPNewBR Faster Recovery and Faster Retransmission -**

Since we increase the congestion window from slow-start to congestion avoidance phase, we can say that a bottleneck congested router would have been the cause of a packet loss due to congestion. If the loss would have occurred due to wireless lossy channel, it would mean that the bottleneck router must still have a queue length less than some threshold.
The TCPNewBR estimates the bottleneck queue length (q) as -

$$q = B * ( rtt - rtt_{min})$$

Where B is the bandwidth estimate and ( $rtt - rtt_{min}$ ) is the current queuing delay.

The faster recovery and retransmission algorithms have been modified as below -

```
If ( 3 duplicate acks ) {
        If ( bottleneck queue length < epsilon ) {
                ssthresh = avg_rtt * bandwidth / seqment_size
                cwnd = ssthresh
        }
        else {
                ssthresh = min_rtt * bandwidth / seqment_size
                cwnd = ssthresh
        }
}
else if (timeout) {
        If ( bottleneck queue length < epsilon ) {
                ssthresh = avg_rtt * bandwidth / seqment_size
                cwnd = 1
        }
        else {
                ssthresh = min_rtt * bandwidth / seqment_size
                cwnd = 1
        }
}
```

where bandwidth is the estimated bandwidth, segment_size is the size of the packet, avg_rtt is the average value of rtt during the connection, min_rtt is the minimum value of rtt during the connection. A number of tests have shown that networks performs best when epsilon = 3

**NS2 Implementation and Test Topology -**

NS2 (Network simulator) is a simulation tool to perform various network simulation using C++ and TCL programming languages. It is an open-source software that runs on Linux. It provides the support for a number of protocols like UDP, TCP and it's variants; multiple protocols and IP protocols; wired and wireless networks.

For this project, we have used version - Network simulator 2.35. It can be downloaded from NS2 home page and can be installed using the scripts provided with the package.

- We could get the implementation of TCPW BE. Based on that, we implemented TCPW RE and TCPW NewBR algorithm using C++.

- We integrated the obtained code and the implemented part with NS2 .tcl scripts.

To create a network topology, a .tcl script is required which must contain all the required information in order to build the virtual network and simulate it properly. There is also an option for GUI based simulation which can show the network in a window with options (like buttons) to start and end the simulation to make it more user friendly and helps in visualisation. The GUI window (also called as network animator (NAM) window) also shows a visual transfer of packets along the flow defined in the .tcl script. Below is one example of the simulation window -

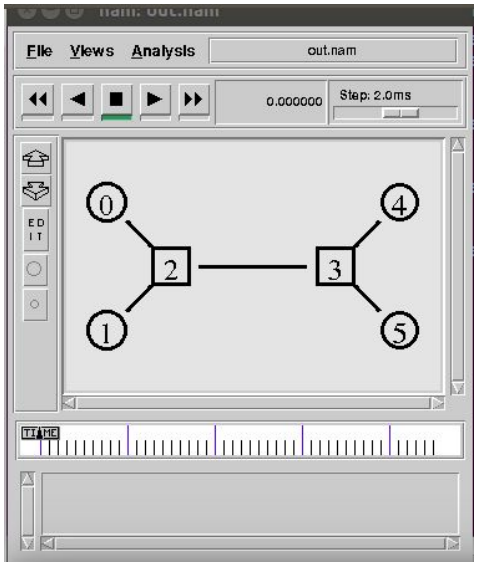


Fig.1.

This example network topology is a wired network with 4 wired nodes (0, 1, 4, 5) and 2 routers (2, 3). For our project, we created two networks (with both wired and wireless nodes). Below are the two networks we used in the project -

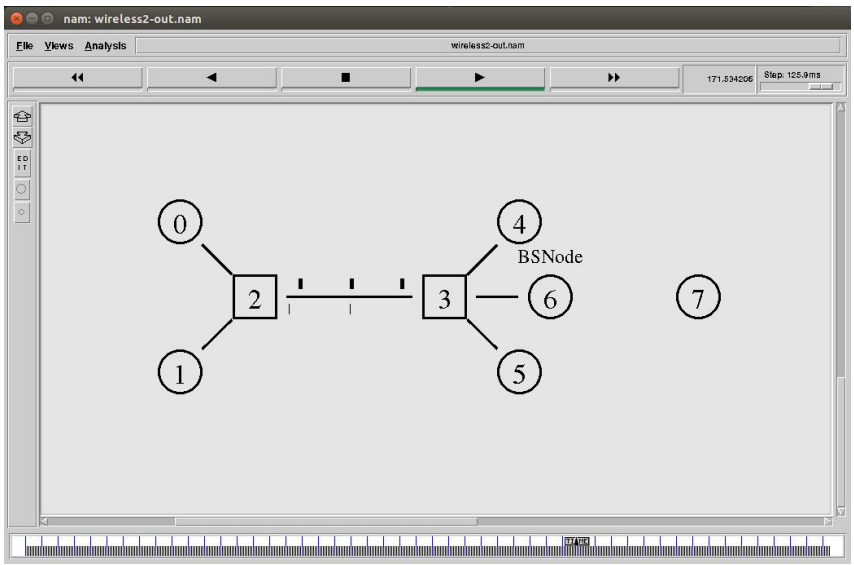

Fig. 2. - Here wired nodes - 0, 1, 2, 3, 4, 5. Base station node - 6 and wireless node - 7.

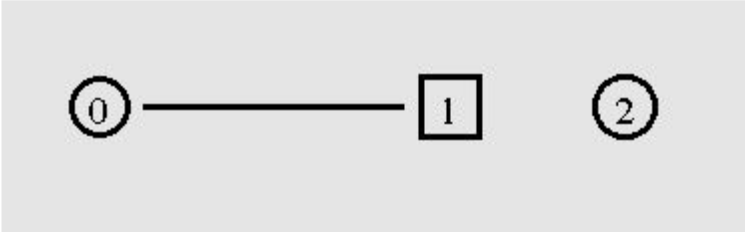


Fig. 3. - One hop network topology where 0 - wired node, 1 - Base station node, 2 - mobile node.

Fig.2. topology is a heterogeneous topology created to test the TCPNewBR.

1. It has three parts - 1) wired network, 2) Base station, 3) mobile node.
2. Wired network has 4 nodes, 2 routers and 1 Base station node connected (nodes(0-6)).
3. Node labelled as 7 in the below figure is the mobile node.
4. Nodes are distributed in domain and clusters.
    a. Wireless node is in domain 1 and wired nodes are domain 0.
    b. We could've used more clusters to group nodes based on geographical sense but this is also fine. We might try this at a later stage.
    c. Grouping in clusters helps to assign IP addresses easily.
5. Node 7 which is the mobile node is set to move in a defined boundary (starting from an initial position with a constant speed).
6. Properties like rate, packetsize, queue length all have been set according to the paper.
7. We have checked the trace file generated by this network animator and the packets from different source are able to reach the mobile node via base station node.

Here, we will take a brief look at some important lines in the .tcl script based on the network topology shown in Fig. 2. -
First, we define the set of properties required for wireless node for e.g. channel type, MAC type, set data rate for the wireless node, queueType, routing protocol etc.

```
set opt(chan)           Channel/WirelessChannel    ;# channel type
.
set opt(netif)          Phy/WirelessPhy            ;# network interface type
set opt(mac)            Mac/802_11                 ;# MAC type
$opt(mac) set dataRate_ 2Mb;
$opt(mac) set delay_ 10us;
set opt(ifq)            Queue/DropTail/PriQueue    ;# interface queue type
.
.
set opt(nn)             1                          ;# number of mobilenodes
set opt(adhocRouting)   DSDV                       ;# routing protocol
```

Here, we define all the links between the wired nodes and the base station -

```
$ns_ duplex-link $W(0) $W(2) 100Mb 10ms DropTail
$ns_ duplex-link $W(1) $W(2) 100Mb 10ms DropTail
$ns_ duplex-link $W(2) $W(3) 10Mb 250ms DropTail
$ns_ duplex-link $W(3) $W(4) 100Mb 10ms DropTail
$ns_ duplex-link $W(3) $W(5) 100Mb 10ms DropTail
$ns_ duplex-link $W(3) $BS(0) 10Mb 10ms DropTail
```

This is how we defined a single flow in the .tcl script. It shows the TCP agent name, the source and sink node, the application type and the time at which this flow should start.

```
# setup TCP connections
set tcp1 [new Agent/TCP/WestwoodNR]
$tcp1 set class_ 2
set sink1 [new Agent/TCPSink]
$ns_ attach-agent $W(0) $tcp1
$ns_ attach-agent $node_(0) $sink1
$ns_ connect $tcp1 $sink1

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ns_ at $opt(ftp1-start) "$ftp1 start"
```

We can also set the stop time for each flow with the following lines -

```
# Tell all nodes when the simulation ends
for {set i } {$i < $opt(nn) } {incr i} {
    $ns_ at $opt(stop).0 "$node_($i) reset";
}
$ns_ at $opt(stop).0 "$BS(0) reset";

$ns_ at $opt(stop).0002 "puts \"NS EXITING...\" ; $ns_ halt"
$ns_ at $opt(stop).0001 "stop"
```

Since this is a heterogeneous network, we can set the motion of the mobile nodes to either zero, random or explicitly define it's path. Below lines shows - the starting position of the mobile node, it's speed of movement and the destination to reach during the simulation.

```
$ns_ at 2.000000000000 "$node_(0) setdest 100.0 120.00 0.0"
.
$node_(0) set Z_ 0.000000000000
$node_(0) set Y_ 0.0
$node_(0) set X_ 60.0
```

**Results -**

We have evaluated the proposed algorithm and compared it to BE and RE algorithms. The comparisons are presented with 1) available bandwidth estimation, 2) algorithm throughput,

3) fairness. As we have mentioned earlier, we used the network topology shown in Fig.2. for all the evaluations. The links, rate and delay values were exactly same as mentioned in the paper. We would like to mention that TCPWBE - TCP Westwood protocol with BE algorithm TCPWRE - TCP Westwood protocol with RE algorithm, and TCPW NewBR - TCP Westwood protocol with the proposed algorithm.

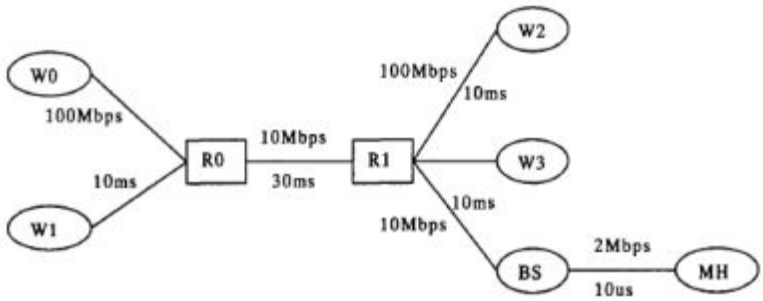


Fig. 4. Heterogenous network with links, rate and delay values.

Following the paper, we created three type of scenarios on the same network topology -

1) Competitive flow (Fig. 5) - Multiple flows from different wired nodes (basically the source nodes) to mobile node (destination node) were set to run simultaneously. From the graph, it is clear that BE is always overestimating and RE is underestimating at a lower level. TCPW NewBR uses the utilization factor to combine both these estimations and predict the available bandwidth more accurately.

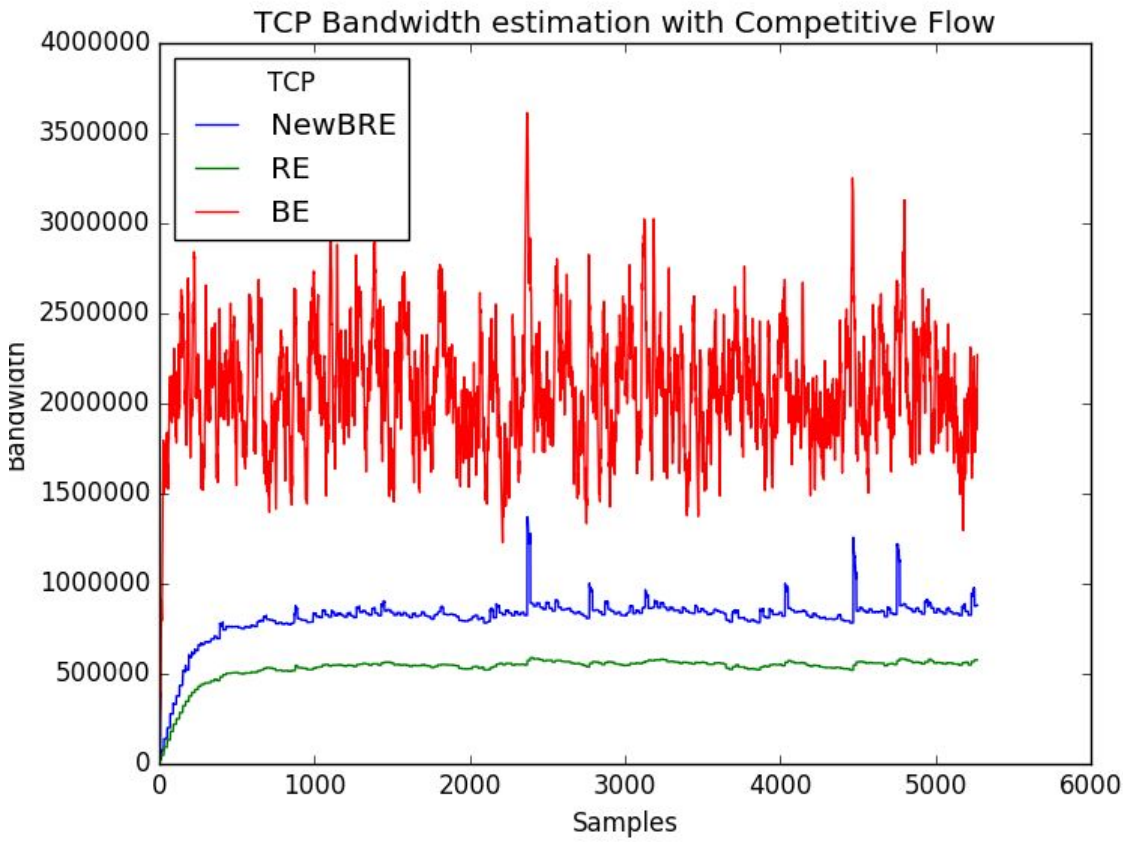

Fig. 5. TCPW with competitive flow

2) Reverse flow (Fig. 6) - this was the next simulation where, two flows were set to run - a) W(0) to Mobile node and Mobile node to W(1) for a short interval of time. The results were similar to the observations presented in the paper, however BE showed a lot of fluctuations. RE and NewBRE clearly predicted better and were unaffected by the reverse flow.

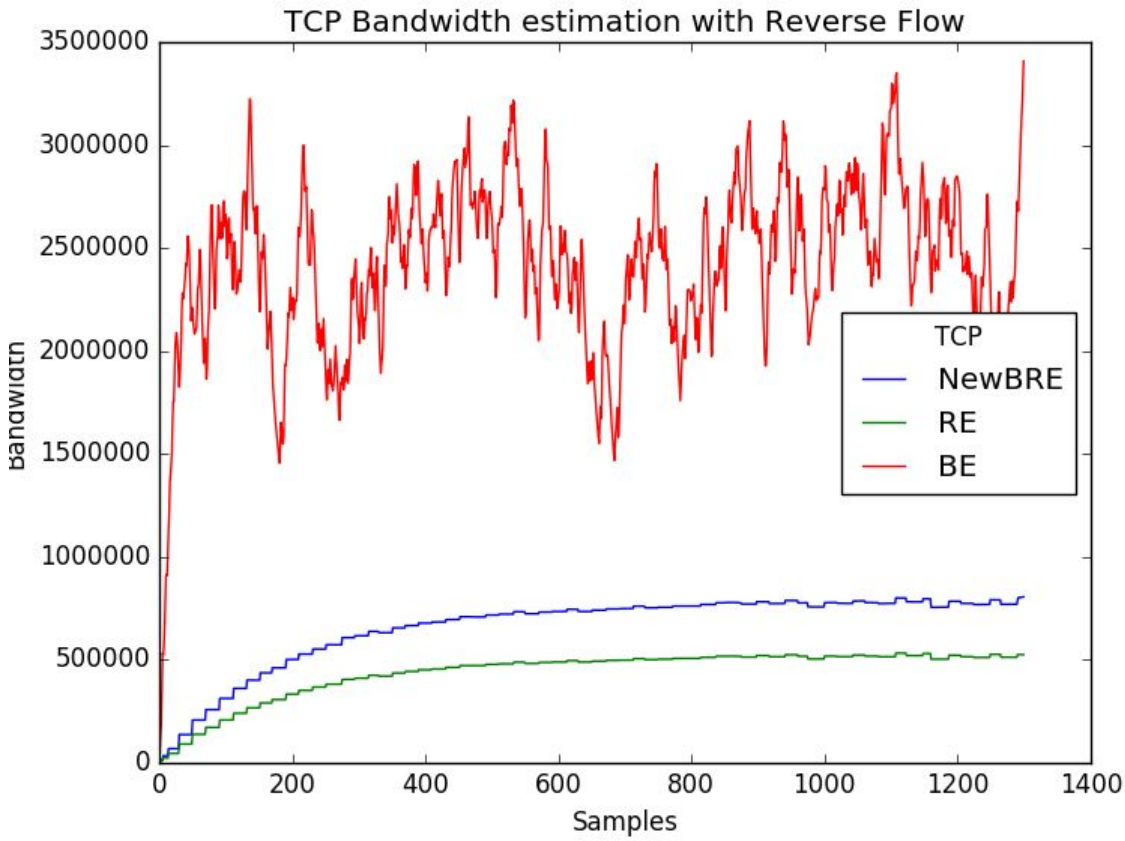


Fig. 6. TCPW with reverse flow.

3) UDP flow (Fig. 7.) - this simulation was done to check whether the algorithm is able to estimate the decrease in the available bandwidth once it is occupied by UDP flow. A UDP flow is created and was set to run for short interval of time twice. The sudden decrease in the graph in those time interval shows the correct estimation as expected.

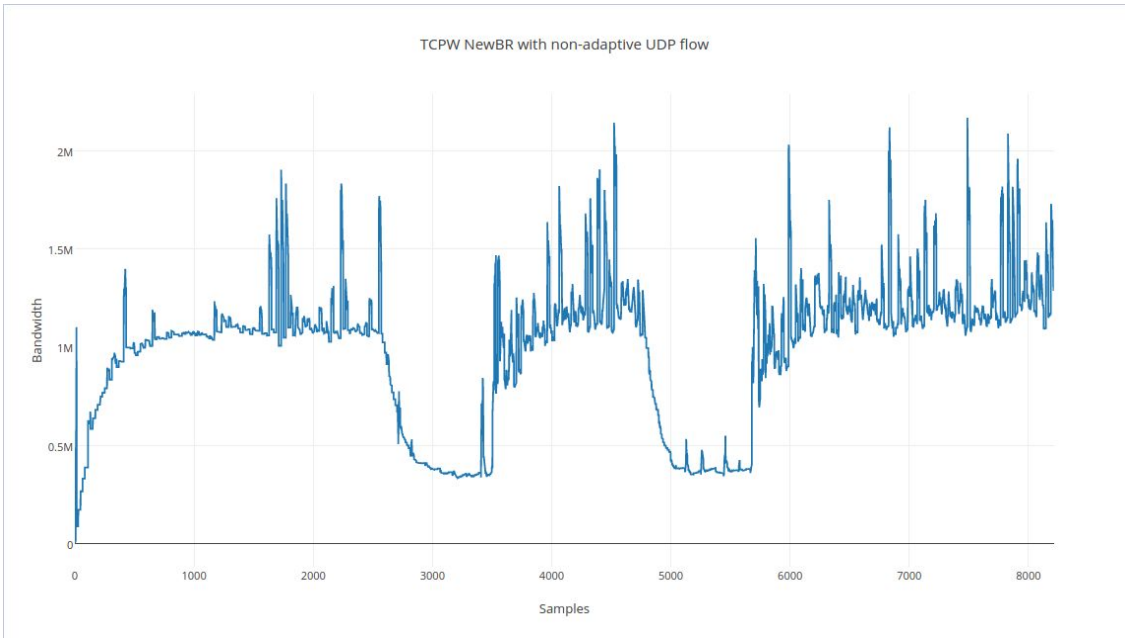


Fig. 7. TCPW NewBR with UDP flow for short period of time.

4) Throughput vs Bottleneck capacity (Fig. 8.) - In this part of the result, we tried to compute the throughput of the network. The expectation was that TCPW NewBR should work better than BE and RE algorithms. As we increased the bottleneck capacity, the throughput increased as expected which also shows that TCP New BR can better utilize link capacity as the bottleneck link transmission rate increases. Therefore, the link utilization is improved.

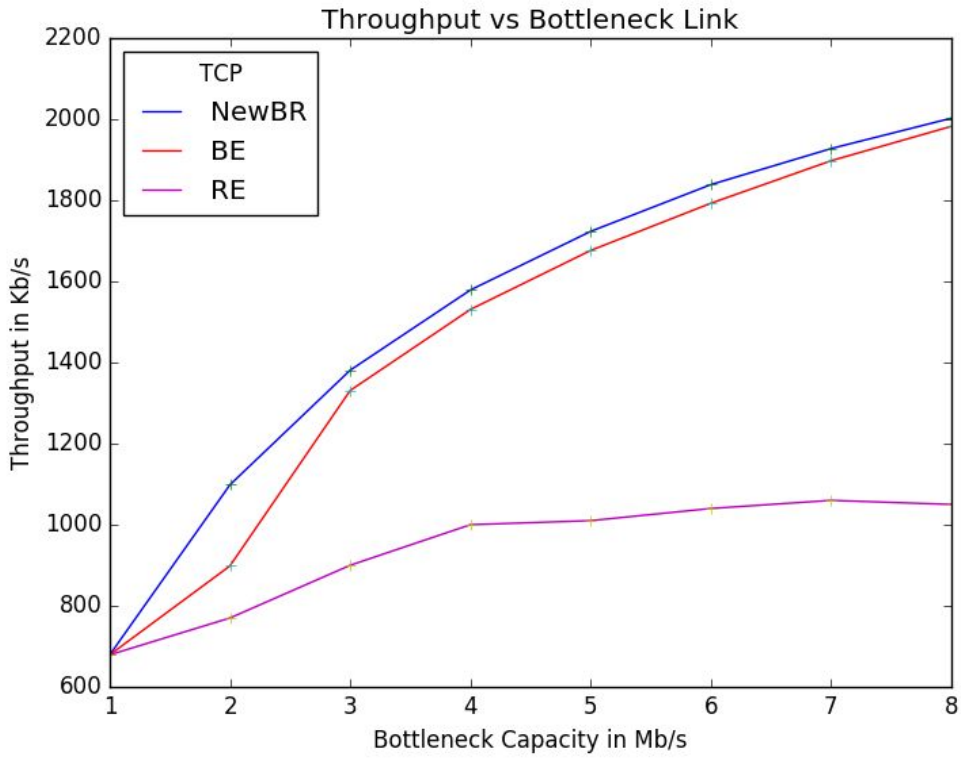


Fig. 8. Throughput vs bottleneck capacity for all TCPW variants.

5) We also tried to estimate the throughput of the network vs the bottleneck link delay. It was expected that as we increase the link delay, it would affect the throughput and detroit the performance. The results are similar to the results observed in the paper.

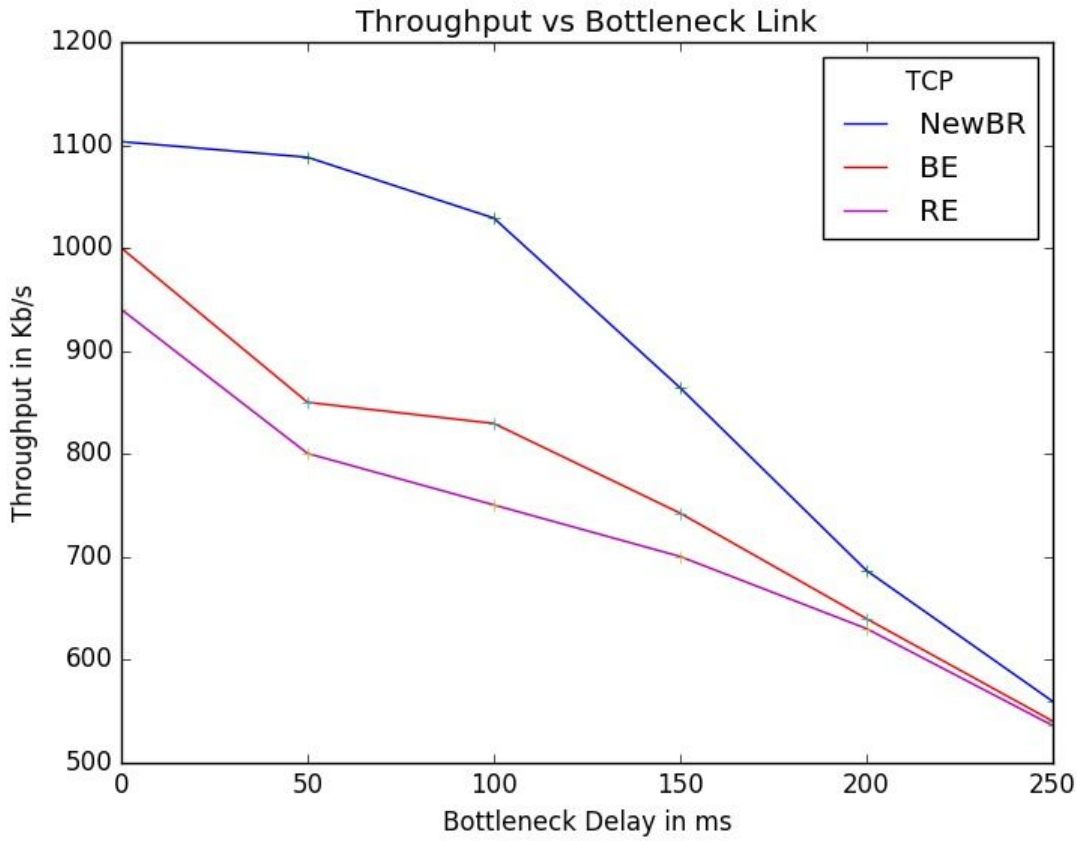


Fig. 9. Throughput vs bottleneck delay for all TCP variants.

**Github Link -** https://github.com/grakshit/FCN_Project

**References -**

**[1]** Rui-Qing Wu, Hong Jie, and Nan Ding, “An Improved TCP Congestion Control Algorithm of Based on Bandwidth Estimation in Heterogeneous Networks”, Journal of Communications Vol. 9, No. 10, October 2014 (Link to the Paper)

[2] NS2 documentation - Link

[3] Claudio Casetti, Mario Gerla, Saverio Mascolo, Ren Wang, “TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks” (Link to the Paper)

[4] Siddharth Gangadhar, Truc Anh N. Nguyen, Greeshma Umapathi, ”TCP Westwood(+) Protocol Implementation in ns-3” (Link to the Paper)