ГОДИШНИК НА СОФИЙСКИЯ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ“
ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА
Том 108

ANNUAL OF SOFIA UNIVERSITY “ST. KLIMENT OHRIDSKI”
FACULTY OF MATHEMATICS AND INFORMATICS
Volume 108

INVESTIGATION OF ALGORITHMS FOR PREVENTION OF CONGESTION IN COMPUTER NETWORKS AT LEVEL OF TCP PROTOCOL USING MININET NETWORK EMULATOR

PLAMEN D. ROZINOV and STEFAN S. DIMITROV

This article examines the performance and values of network parameters when applying different algorithms to control network congestion. The Mininet network emulator is used, which creates software-defined networks that give us the opportunity to conduct various experiments with the network and manipulate its parameters. The conclusion is made for the best algorithm to prevent network congestion. An estimate of the relative value of the error was made, which shows that in Mininet deviations in the values can be observed with a larger number of hosts in the network.

Keywords: software-defined networks, network congestion prevention algorithms, Mininet network emulator

CCS Concepts:
• Networks~Network performance evaluation~Network performance analysis;
• Networks~Network performance evaluation~Network measurement;
• Networks~Network performance evaluation~Network experimentation;
• Networks~Network performance evaluation~Network simulations

1. Research

1.1. Introduction

This article examines the performance and values of network parameters when applying different algorithms to control network congestion. The Mininet network emulator was used, which creates software-defined networks that give us the opportunity to conduct various experiments with the network and to manipulate its parameters [16].

The network congestion prevention algorithms used in the TCP protocol are used by many Internet-based applications. Their main purpose is to prevent more

DOI: 10.60063/GSU.FMI.108.77-95

data from being sent than the network can transmit. This prevents network congestion. Algorithms react differently to network load, but what they have in common is that they all have the principle of preventing network congestion [28].

The most commonly used algorithms to prevent network congestion are BBR (Bottleneck Bandwidth and Round-Trip Propagation Time), BIC (Binary Increase Congestion Control), CUBIC - successor to the BIC algorithm, DCTCP (TCP Congestion Control for Data Centers), HighSpeed TCP, HTCP (Hamilton TCP), Hybla, Illinois, LP (TCP Low Priority), Reno, Scalable, Vegas, Veno – Combination of Reno and Vegas, Westood, Yeah (Yet Another High-speed TCP) [2–6, 9, 11, 12, 15, 17–24, 28].

The study takes into account the values of network parameters influencing the behavior of the TCP protocol such as sent data over time, Initcwnd (Initial Congestion Window), Cwnd (Congestion Window), MSS (Maximum Segment Size), MTU (Maximum Transmission Unit), repeated Retransmits, RTT (Round-Trip Time), Jitter – big difference in delay (Delay Variance), RTT Variance – RTT variability in time, Throughput – time interval bandwidth [1, 7, 8, 10, 13, 14, 20, 25, 27].

1.2. Realization

1.2.1. Creating a network

A software-defined network topology with minimum parameters is created – two hosts, one switch and one controller using the sudo mn command.

1.2.2. Setting bandwidth

In order to recreate the operation of Ethernet cable category 6a, which is widely used in practice, network bandwidth is limited by Linux Traffic Control. A maximum value of 10 Gbit/s is set for both hosts and the switch with the command:

```
sudo tc qdisc add dev h1-eth0 root handle 1:
tbf rate 10gbit burst 5000000 limit 15000000
```

1.2.3. Checking algorithm for network overload control

In order to check the currently set algorithm for controlling network congestion, we need to use the `sysctl net.ipv4.tcp_congestion_control` command.

1.2.4. Setting an algorithm

The command helps us to set an algorithm for controlling congestion in the network

```
sysctl -w net.ipv4.tcp_congestion_control=congestion_control_algorithm
```

1.2.5. Network performance and network parameters

To measure network performance and measure the values of individual network parameters, we use the network tool `iperf3`. In the experiments, host h2 acts as a server and host h1 as a client. Hosts and the switch are set to the same network congestion control algorithm. With the command: `iperf3 -s` the host h2 is started in the role of server and with the command: `iperf3 -c 10.0.0.2` a client of host h1 is started to `iperf3` server – host h2.

1.2.6. Preparation of results

From each measurement with `iperf3` a json file is created with the command:

```
iperf3 -c 10.0.0.2 -J> congestion_control_algorithm.json
```

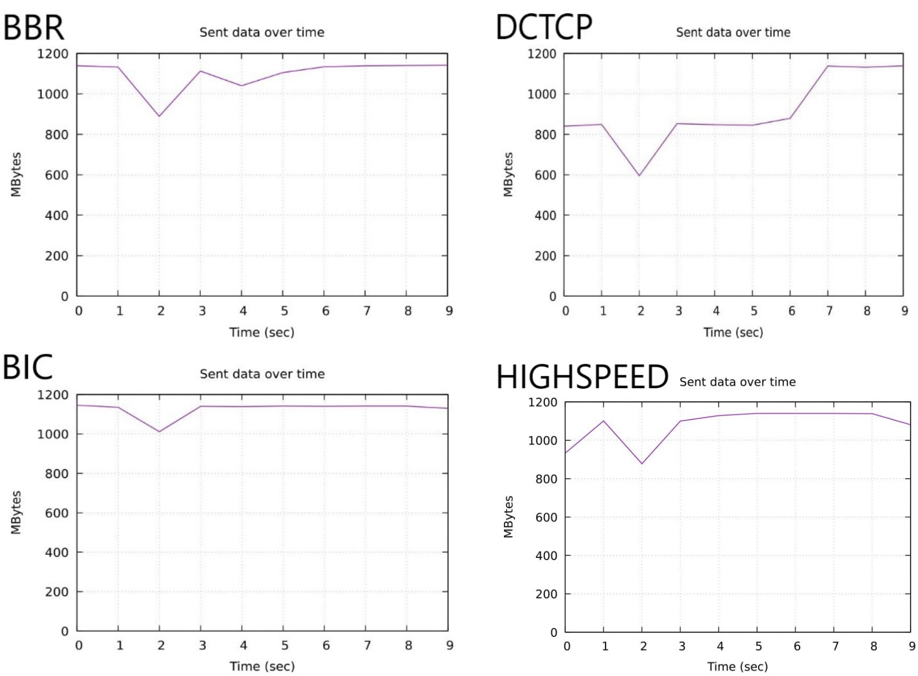
Then, with the help of the `plot_iperf.sh` script and the

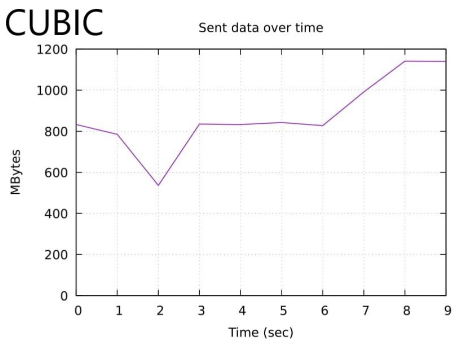
```
plot_iperf.sh congestion_algorithm.json
```

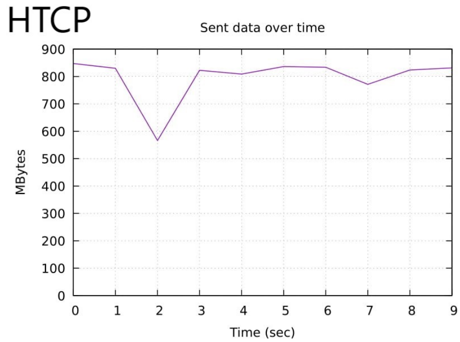
command, graphs of the individual network parameters are prepared, and for each network parameter there is a separate graph with a separate file in PDF format.

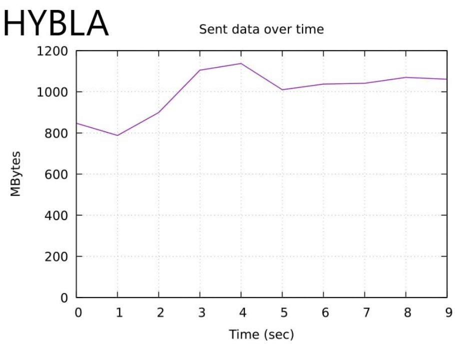
2. Results

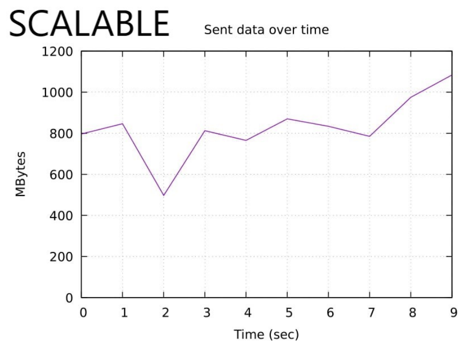
2.1. Sent data over time

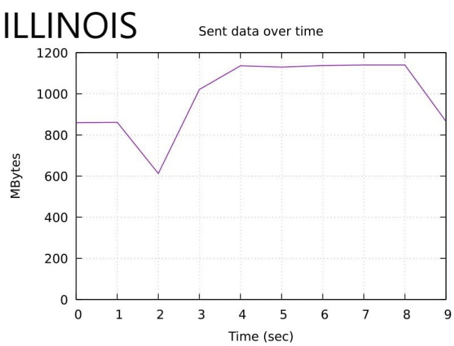


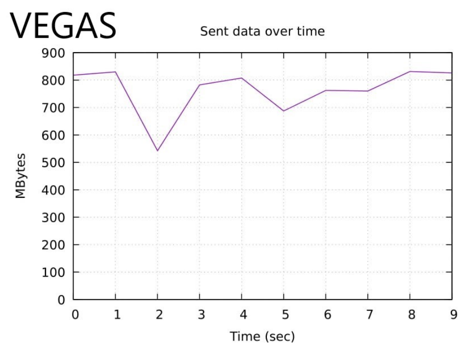

CUBIC
Sent data over time
MBytes
0 200 400 600 800 1000 1200
Time (sec)
0 1 2 3 4 5 6 7 8 9

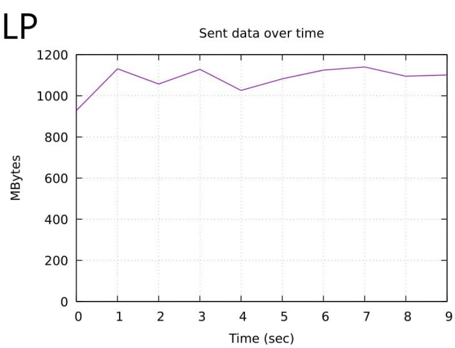

HTCP
Sent data over time
MBytes
0 100 200 300 400 500 600 700 800 900
Time (sec)
0 1 2 3 4 5 6 7 8 9

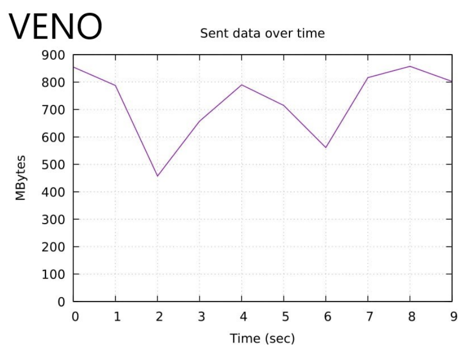

HYBLA
Sent data over time
MBytes
0 200 400 600 800 1000 1200
Time (sec)
0 1 2 3 4 5 6 7 8 9

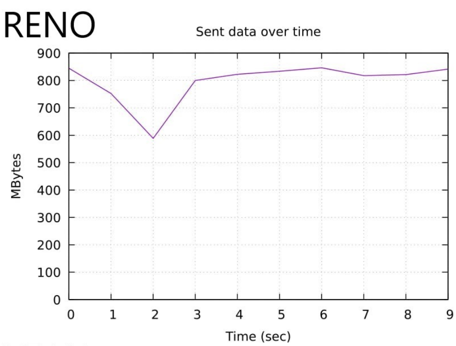

SCALABLE
Sent data over time
MBytes
0 200 400 600 800 1000 1200
Time (sec)
0 1 2 3 4 5 6 7 8 9

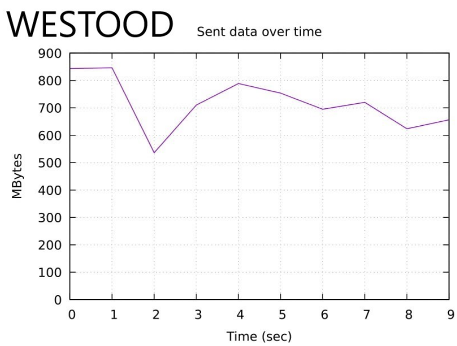

ILLINOIS
Sent data over time
MBytes
0 200 400 600 800 1000 1200
Time (sec)
0 1 2 3 4 5 6 7 8 9

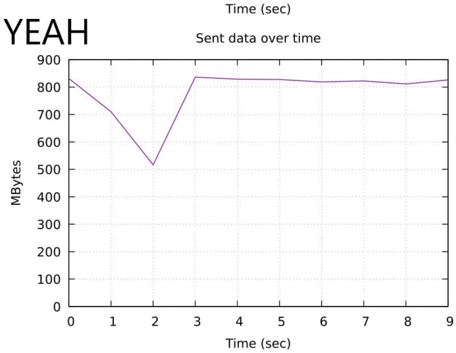

VEGAS
Sent data over time
MBytes
0 100 200 300 400 500 600 700 800 900
Time (sec)
0 1 2 3 4 5 6 7 8 9


LP
Sent data over time
MBytes
0 200 400 600 800 1000 1200
Time (sec)
0 1 2 3 4 5 6 7 8 9

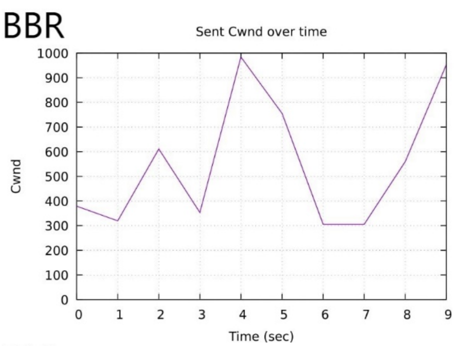

VENO
Sent data over time
MBytes
0 100 200 300 400 500 600 700 800 900
Time (sec)
0 1 2 3 4 5 6 7 8 9

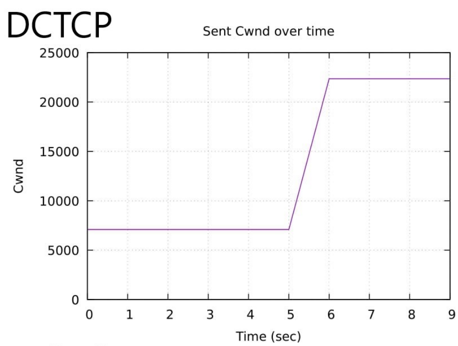


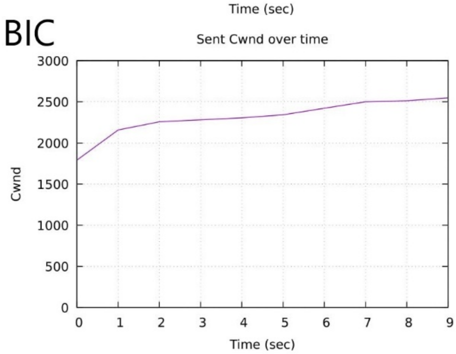


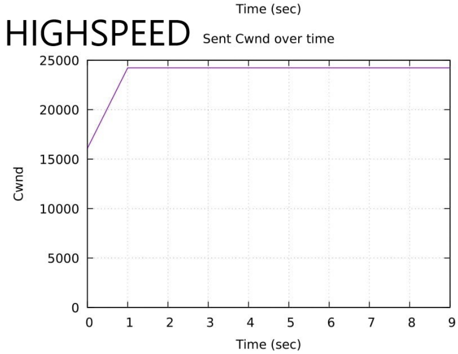


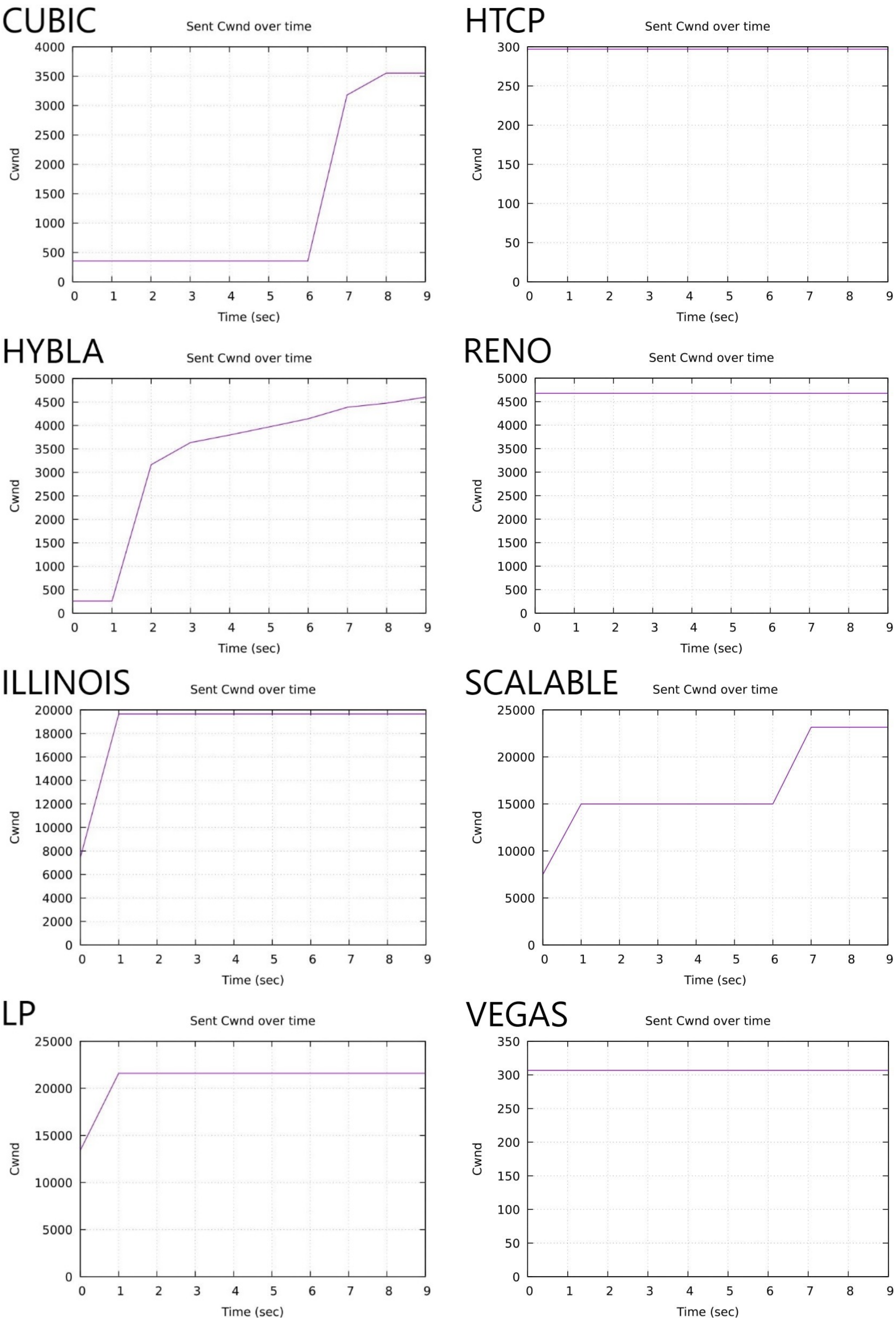
2.2. Congestion window size (CWND)

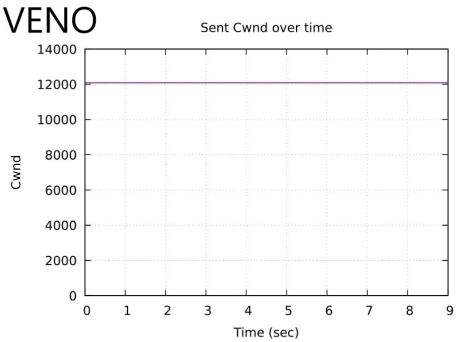


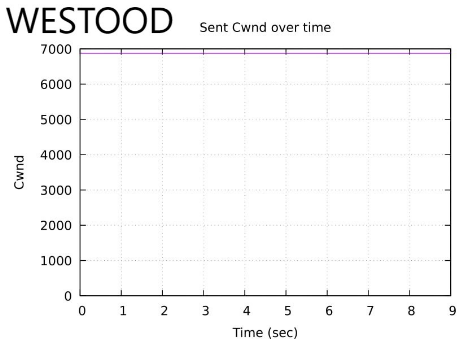


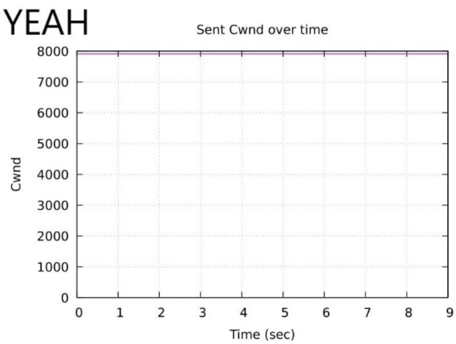





CUBIC
Sent Cwnd over time
Cwnd
Time (sec)
HTCP
Sent Cwnd over time
Cwnd
Time (sec)
HYBLA
Sent Cwnd over time
Cwnd
Time (sec)
RENO
Sent Cwnd over time
Cwnd
Time (sec)
ILLINOIS
Sent Cwnd over time
Cwnd
Time (sec)
SCALABLE
Sent Cwnd over time
Cwnd
Time (sec)
LP
Sent Cwnd over time
Cwnd
Time (sec)
VEGAS
Sent Cwnd over time
Cwnd
Time (sec)







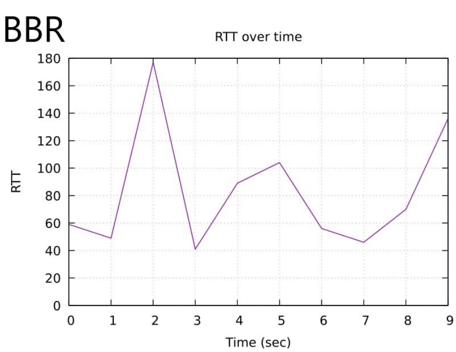
2.3. Maximum transmission unit (MTU)

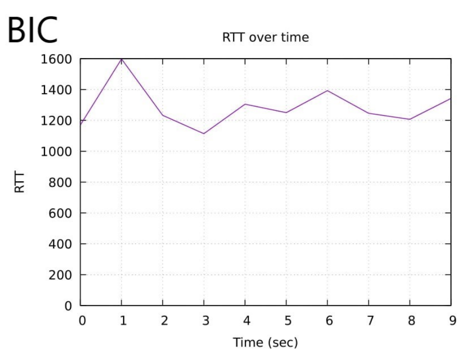
The MTU has a fixed value of 1500 bytes and no change in values, regardless of the applied network congestion control algorithm.

2.4. Retransmits

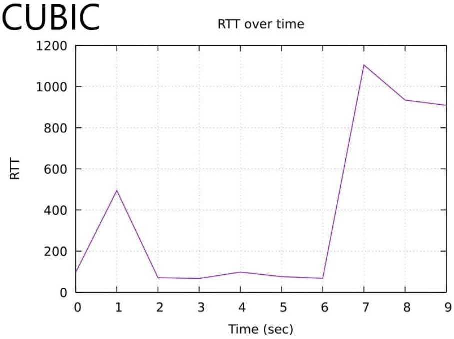
Retransmissions do not occur on the network, regardless of the network congestion control algorithm.

2.5. Round-trip time (RTT)

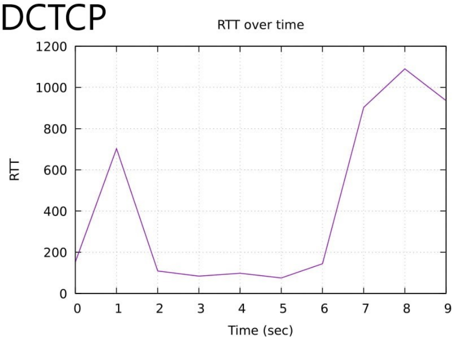




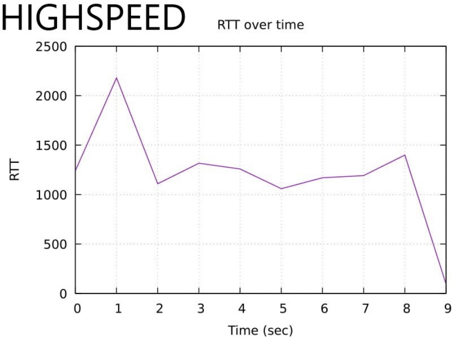
CUBIC



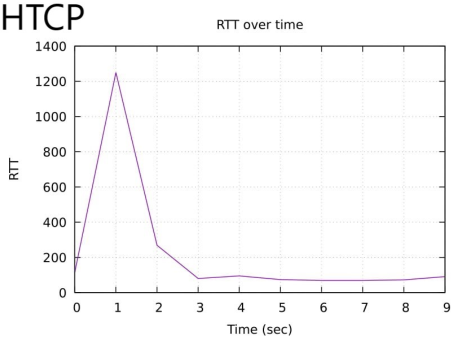
DCTCP



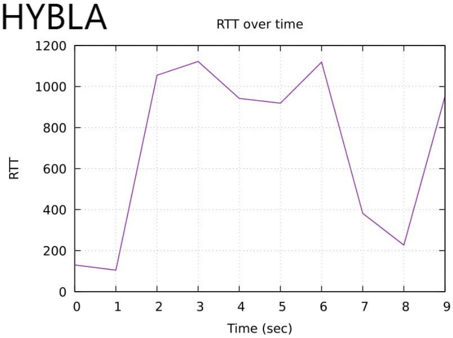
HIGHSPEED



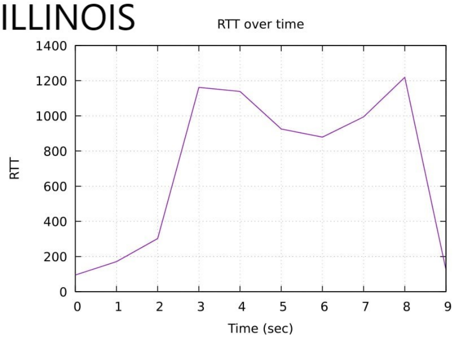
HTCP



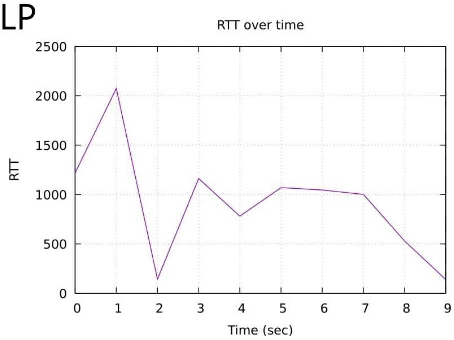
HYBLA



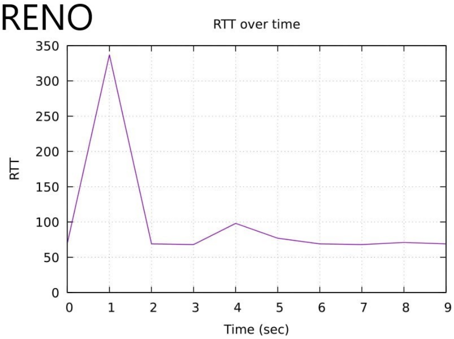
ILLINOIS

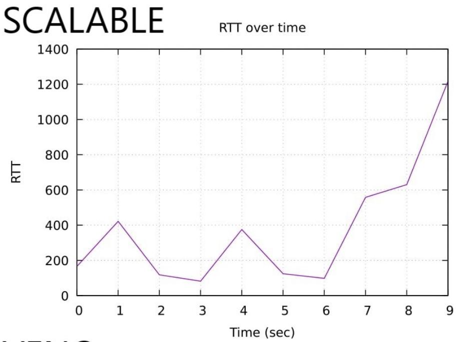


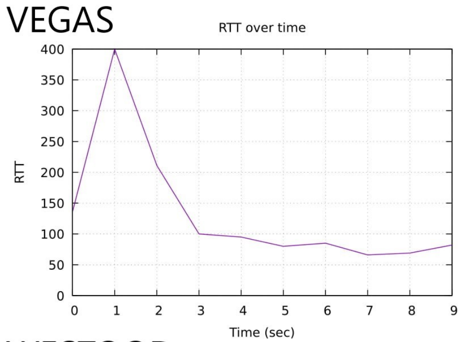
LP

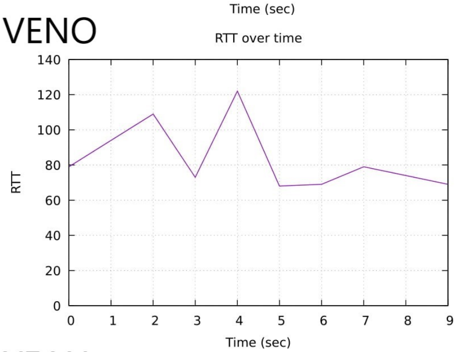


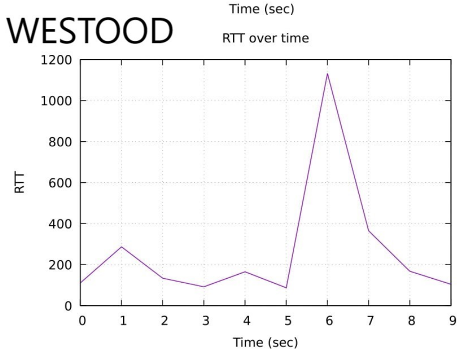
RENO

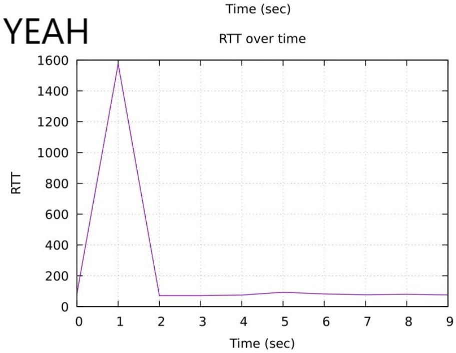




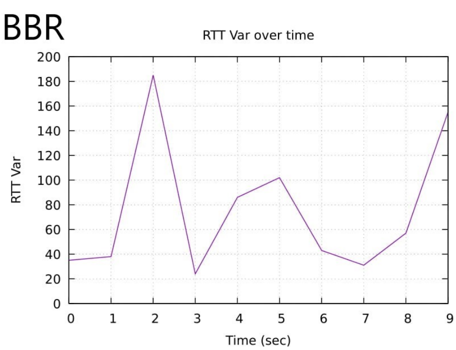


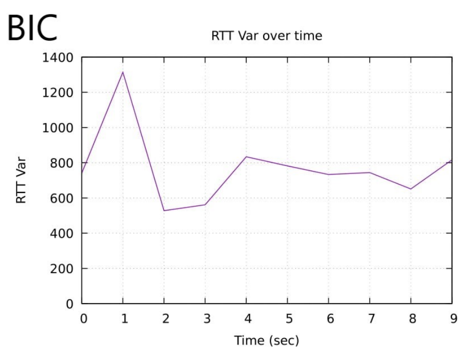


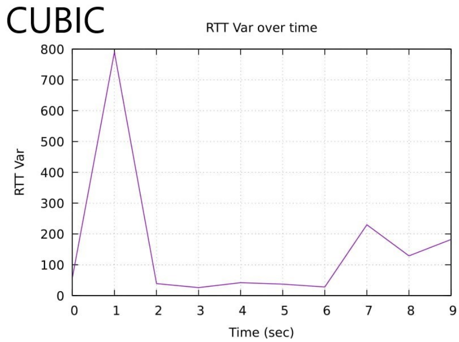


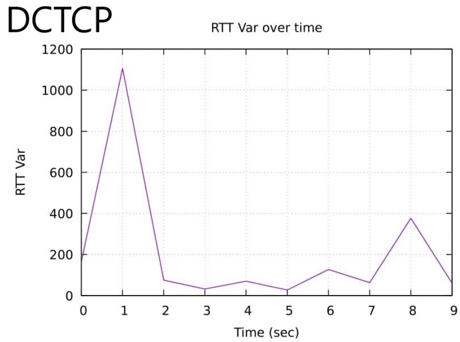


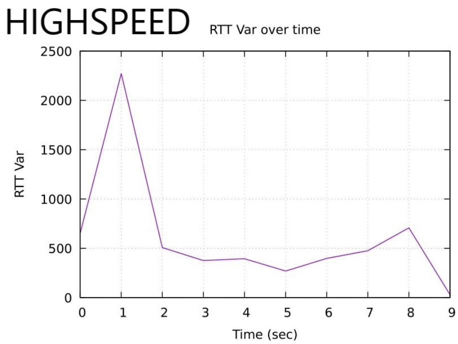
2.6. Round-trip time variance (RTT Variance)

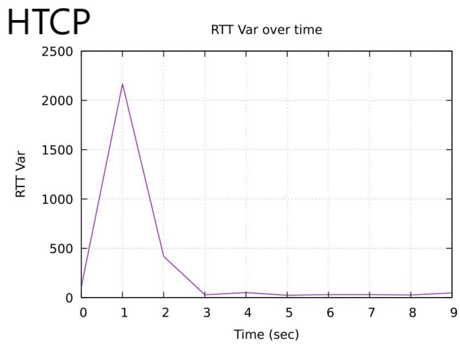


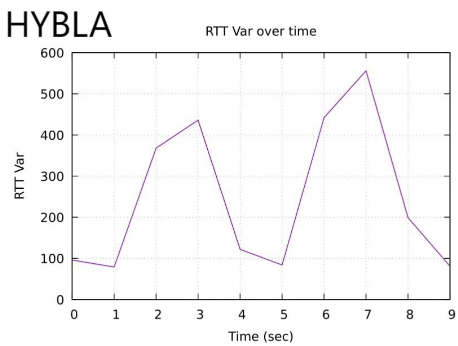


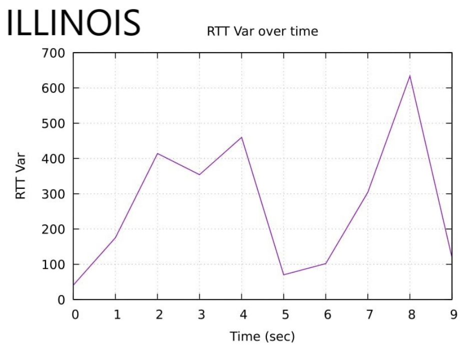

CUBIC
RTT Var over time
RTT Var
Time (sec)

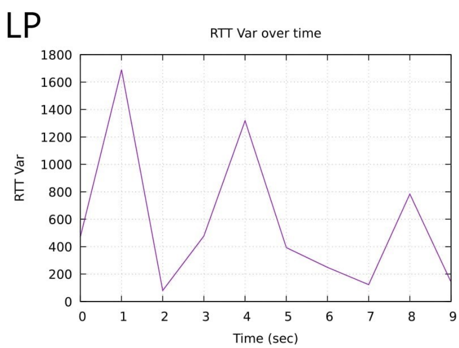

DCTCP
RTT Var over time
RTT Var
Time (sec)

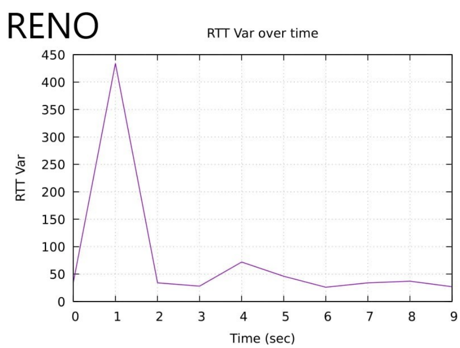

HIGHSPEED
RTT Var over time
RTT Var
Time (sec)

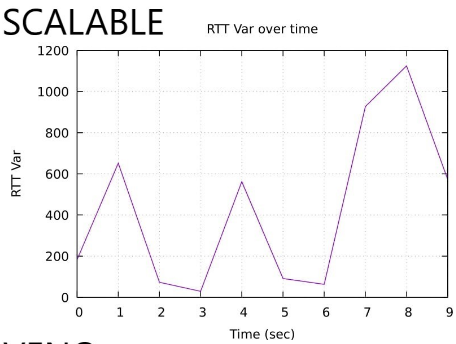

HTCP
RTT Var over time
RTT Var
Time (sec)

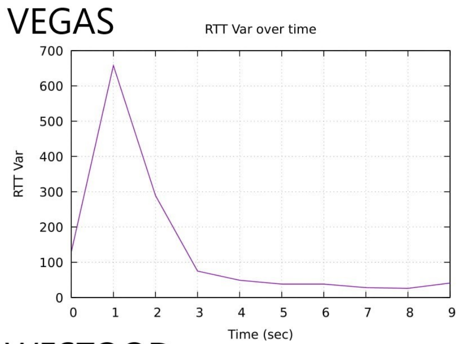

HYBLA
RTT Var over time
RTT Var
Time (sec)

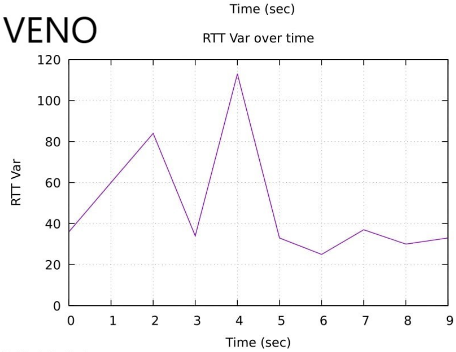

ILLINOIS
RTT Var over time
RTT Var
Time (sec)

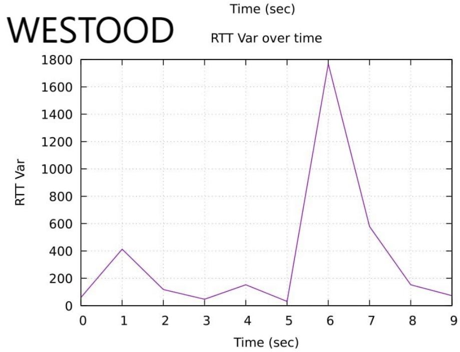

LP
RTT Var over time
RTT Var
Time (sec)

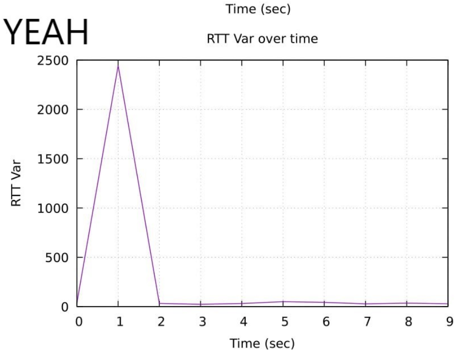

RENO
RTT Var over time
RTT Var
Time (sec)



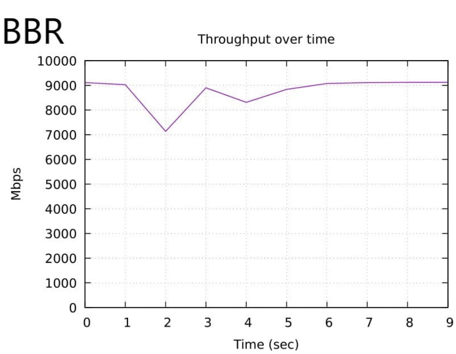


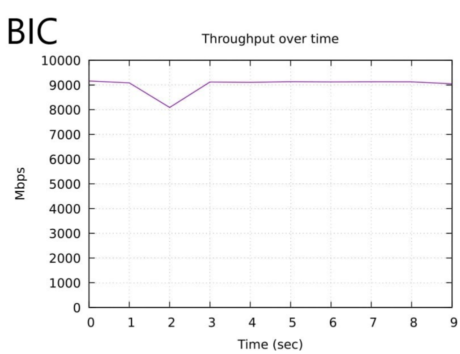




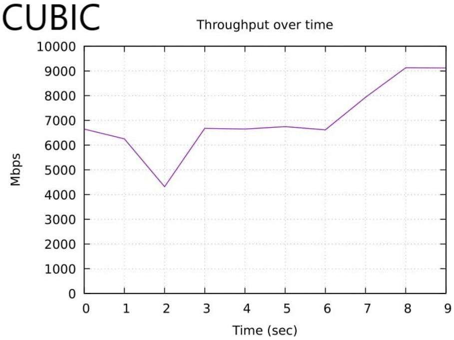


2.7. Throughput

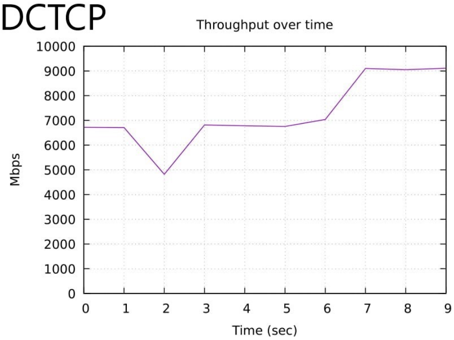




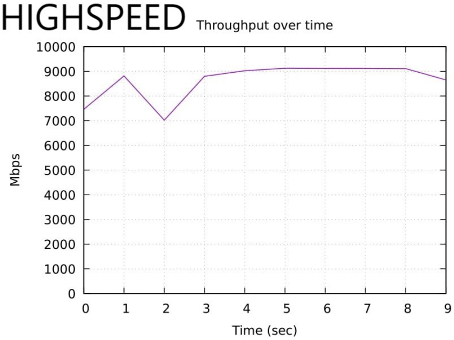
CUBIC



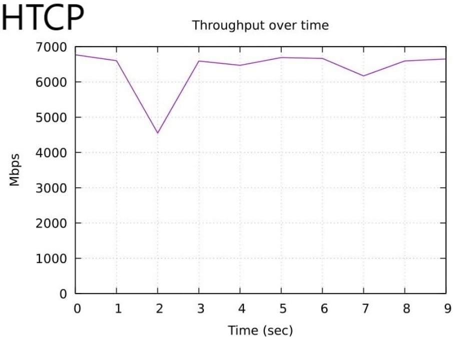
DCTCP



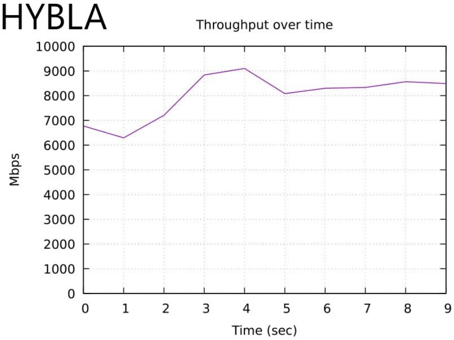
HIGHSPEED



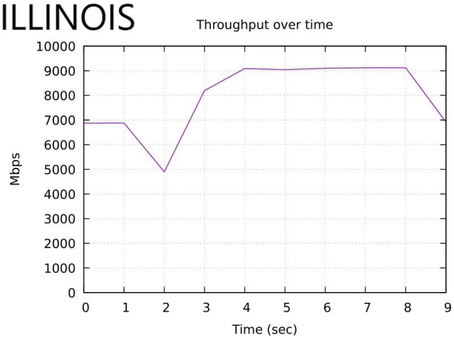
HTCP



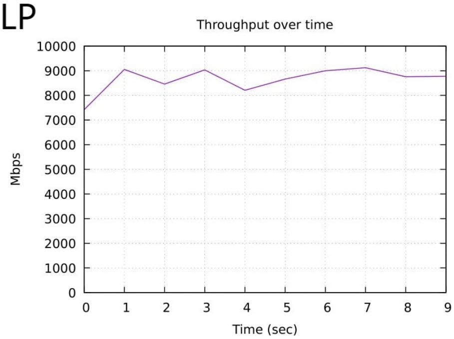
HYBLA



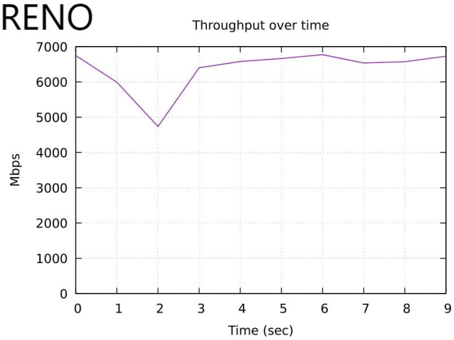
ILLINOIS

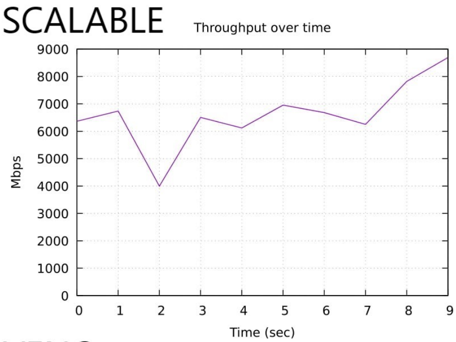


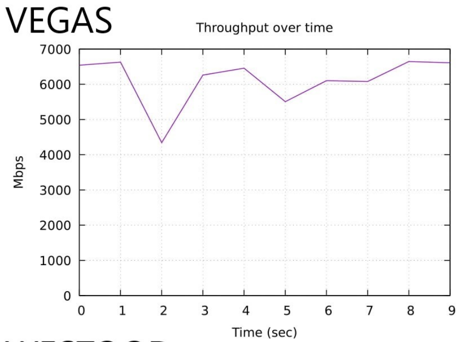
LP

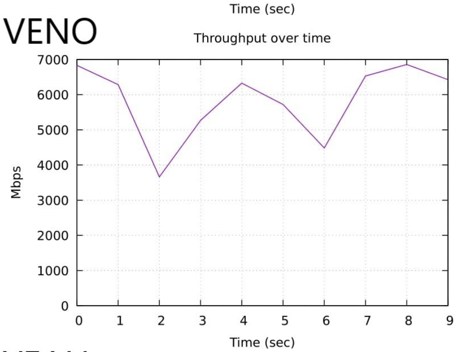


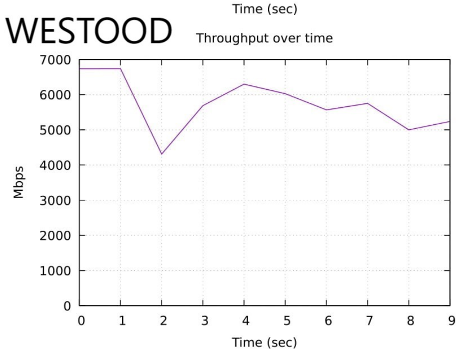
RENO

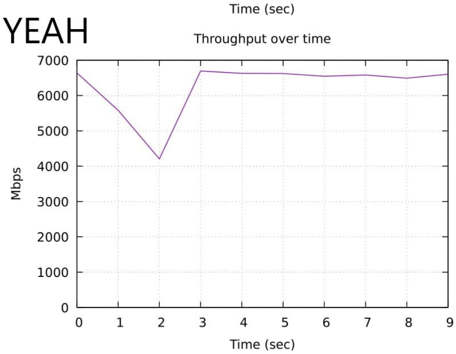












2.8. Summarized results

Summarized results are shown in Table 1 and Diagrams 1–5.

3. Conclusions

3.1. Congestion windows size

The highest number of packets that can be transmitted for 1 MSS was reported at Highspeed, followed by LP and Illinois. They are followed by Veno, Yeah and Westood. The number of packages is the smallest in HTCP, and in Vegas they are a little more, but without much difference. The algorithms BIC, Cubic, DCTCP, Hybla, Reno, Scalable generate a relatively small number of packages.

Table 1. Summarized results

	CWND [number of packets]	MTU [bytes]	RTT [ms]	RTT VAR [ms]	THROUGPUT [megabits per second]
BBR	552.62	1500	82.7	75.6	1150.45
BIC	2311.71	1500	1285.8	770.4	1181.17
CUBIC	1276.47	1500	391.9	156.1	918.96
DCTCP	1174.55	1500	1016.5	168.5	1174.55
HIGHSPEED	23415.46	1500	1202.2	608	1130.42
HTCP	296.95	1500	218.4	293.5	835.85
HYBLA	3268.89	1500	695.3	246.3	1048.23
ILLINOIS	18433.01	1500	701	267.3	1038.58
LP	20780.21	1500	916.3	572.9	1134.15
RENO	835.53	1500	99.6	77.3	835.53
SCALABLE	866.65	1500	378.9	427.8	866.65
VEGAS	306.85	1500	132.4	137.2	801.95
VENO	12083.16	1500	765.45	48.5	765.45
WESTOOD	6876.59	1500	752.17	339.5	752.17
YEAH	7906.02	1500	228.5	275.5	820.89

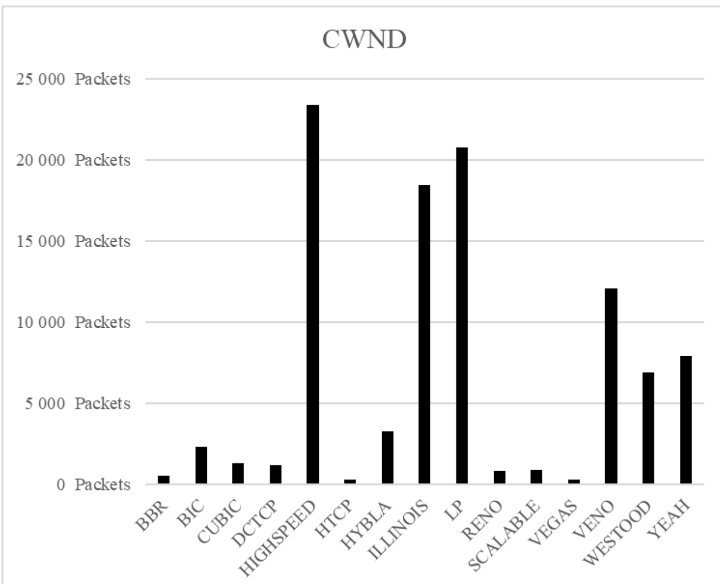


Diagram 1. Measured values of Cwnd [number of packets]

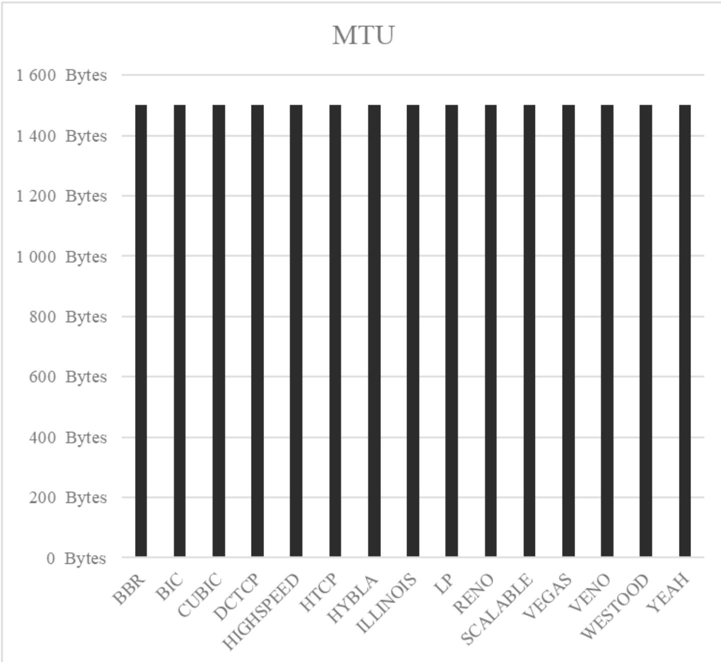


Diagram 2. Measured values of MTU [bytes]

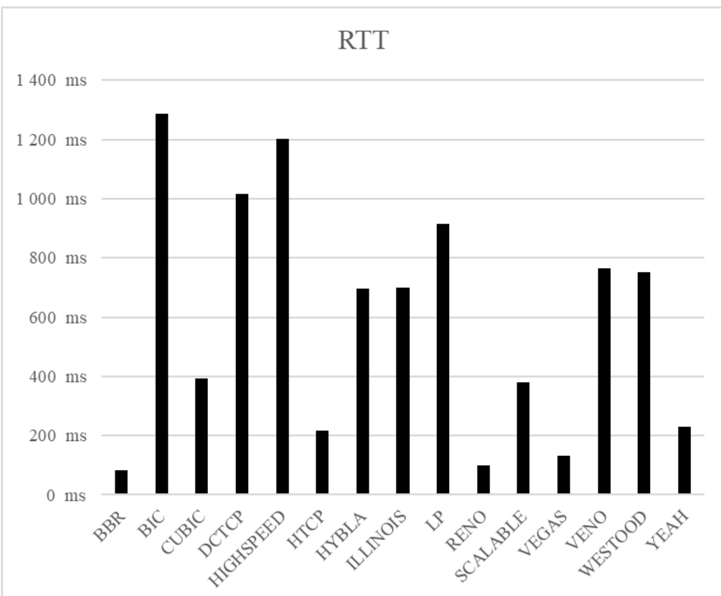


Diagram 3. Measured values of RTT [milliseconds]

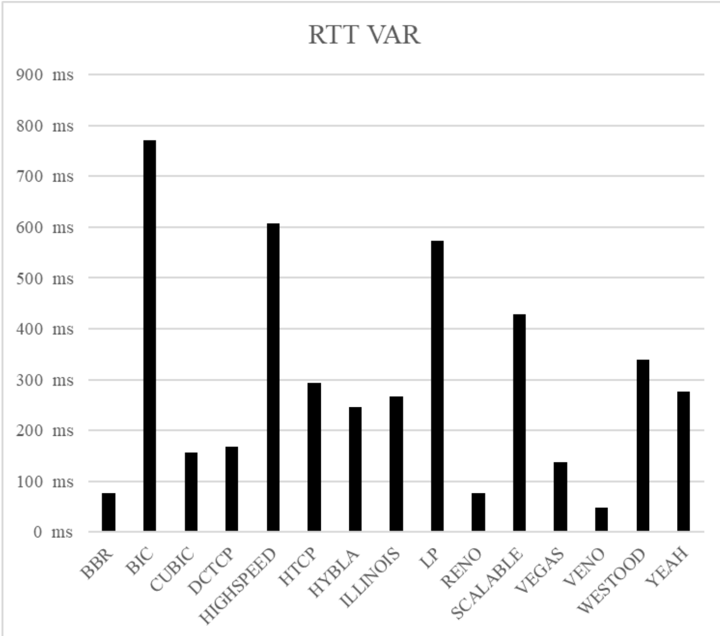


Diagram 4. Measured values of RTT Variance [milliseconds]

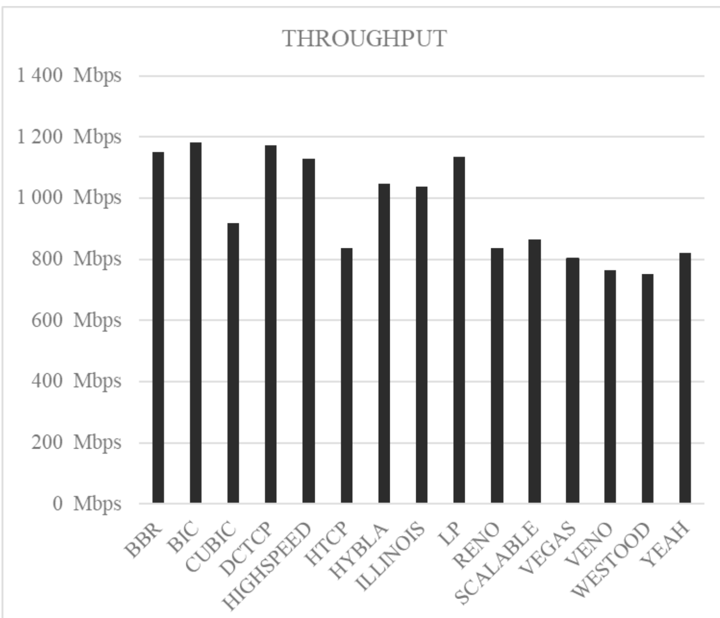


Diagram 5. Measured values of throughput [megabits per second]

3.2. Maximum transmission unit

The MTU value is fixed at 1500 bytes and does not change during the measurement period, regardless of the algorithm used.

3.3. RTT and RTT variance

The highest value is for RTT and RTT Variance using the BIC algorithm, followed by Highspeed, DCTCP and LP. The Hybla, Illinois, Veno, and Westood algorithms generate similar RTT values. The lowest value is on BBR, followed by Reno and Vegas. Yeah and HTCP have similar values, as do Cubic and Scalable.

3.4. Throughput

The best throughput values were achieved with the BIC algorithm, followed by DCTCP and BBR. In fourth and fifth place respectively are LP and Highspeed, with the difference between them is minimal. Then there are Hybla and Illinois. The lowest values were measured in Westood, followed by Veno. Cubic achieves about 920 Mbps of bandwidth. Vegas, Yeah, Reno, HTCP and Scalable report bandwidth in the 800–870 Mbps range.

3.5. Summary

The algorithms that managed to provide the highest values of network bandwidth are BIC, DCTCP, BBR, LP and Highspeed. Unfortunately, BIC, DCTCP, LP, and Highspeed are among the algorithms that generate the highest values of Round Trip Time and Round Trip Time Variance. Only BBR is an exception, as it achieves the lowest values for Round Trip Time and is in second place for the lowest values for Round Trip Time Variance. This is because BBR purposefully reduces the size of congestion window size, which reduces the likelihood of network congestion. In addition, unlike other algorithms that recognize when a network is congested according to the number of lost packets and low bandwidth values, ie. when congestion is already a fact, BBR is a model-based algorithm and manages to "predict" congestion as it builds its own network model based on bandwidth values and Round-Trip-Time at which the network performs best. BBR is supported on Linux. It is used by default on Google and Youtube as a replacement for Cubic, as Cubic has a longer latency, lower download speeds and data upload. This is due to the fact that Cubic's congestion window size values are independent of Round-Trip-Time. As a result of the present study, BBR is the algorithm that performs best among the most commonly used algorithms to prevent network congestion.

3.6. Possible deviations when using Mininet

Studies have shown that in Mininet, deviations in RTT values can be observed in five, six or seven hosts. This means that the average RTT values extend over a wide range of values and Mininet cannot obtain accurate and consistent results. Therefore, the current study has fewer hosts to avoid this type of deviation. It is also not desirable for the number of hosts to be too large, as packet loss may occur due to failed processor cycles performing network emulation. Because the emulator must run all OpenFlow switches, hosts, and network programs in real time, CPU usage increases significantly as the number of switches on the network increases. This inevitably increases the latency of network packets and a large number of packets remain waiting in the buffer memory, from where they are sent to the recipient. But if the buffer memory is full, the "release" of packets from it begins. To calculate the relative value of the error, we need to divide the number of failed attempts by the total number of attempts made. This attempt failed, in which we have at least 20% of unsuccessfully sent pings [26].

References

[1] T. Amsterdam, Understanding congestion control, 2021. Extracted on 25.11.2021 from `https://granulate.io/understanding-congestion-control/`.

[2] A. Baiocchi, A. Castellani and F. Vacirca, YeAH-TCP: Yet Another Highspeed TCP, INFOCOM Department, University of Roma "Sapienza", 2008.

[3] S. Bensley, D. Thaler, P. Balasubramanian, L. Eggert and G. Judd, Data Center TCP (DCTCP): TCP congestion control for data centers, IETF, RFC 8257, 2017, `https://www.rfc-editor.org/rfc/rfc8257`.

[4] C. Cainin and R. Firrincieli, TCP Hybla: a TCP enhancement for heterogeneous networks, Int. J. Satell. Commun. Network 22 (2004) 547–566, `https://doi.org/10.1002/sat.799`.

[5] CUBIC TCP, Wikipedia, the free encyclopedia, 2016. Extracted on 22.11.2021 from `https://en.wikipedia.org/wiki/CUBIC_TCP`.

[6] P. Dordal, TCP Reno and congestion management, An introduction to computer networks, 2020, `https://intronetworks.cs.luc.edu/current/html/reno.html`.

[7] F5 Clouddocs, TCP::rttvar, 2019. Extracted on 30.11.2021 from `https://clouddocs.f5.com/api/irules/TCP__rttvar.html`.

[8] Fastly: Quic.rtt.variance, 2011. Extracted on 26.11.2021 from `https://developer.fastly.com/reference/vcl/variables/client-connection/quic-rtt-variance/`.

[9] C.-P. Fu and S. C. Liew, TCP Veno: TCP enhancement for transmission over wireless access networks, IEEE Journal on Selected Areas in Communication 21(2) (2003) 216–228.

[10] D. Genkov, Basics of computer networks, Technical University of Gabrovo, 2014.

[11] H-TCP, Wikipedia, the free encyclopedia, 2016. Extracted on 24.11.2021 from `https://en.wikipedia.org/wiki/H-TCP`.

[12] S. Ha, I. Rhee and L. Xu, CUBIC: A new TCP-friendly high-speed TCP variant, 2008, 11 pp, `https://www.cs.princeton.edu/courses/archive/fall16/cos561/papers/Cubic08.pdf`.

[13] IR Media, Network jitter – common causes and best solutions, 2021. Extracted on 28.11.2021 from `https://www.ir.com/guides/what-is-network-jitter`.

[14] V. Jacobson, The congestion control algorithm in TCP, SIGCOMM'88 118(4) (1988).

[15] W. E. Knightly, A. Kuzmanovic and Y. Chen, TCP low priority. Extracted on 25.11.2021 from `https://users.cs.northwestern.edu/~akuzma/rice/TCP-LP/`.

[16] B. Lantz, N. Handigol, B. Heller and V. Jeyakumar, Introduction to Mininet, 2021. Extracted on 22.11.2021 from `https://github.com/mininet/mininet/wiki/Introduction-to-Mininet`.

[17] S. Liu, T. Başar and R. Srikant, TCP-Illinois: A loss and delay-based congestion control algorithm for high-speed networks, ACM DL Digital Library, Association for Computing Machinery, 2006.

[18] D. Mehta, TCP-LP: fair and friendly congestion control approach, IOSR-JCE AETM'16 Special Issue, 2016, 56–61, https://www.researchgate.net/publication/321385338_TCP-LP_Fair_and_Friendly_Congestion_Control_Approach.

[19] K. L. Mills, J. J. Filliben, D. Y. Cho, E. Schwartz and D. Genin, Modeling congestion control algorithms, Study of proposed internet congestion control mechanisms, Special publication 500-282, National Institute of Standards and Technology, 2010, 137–181.

[20] Network Tools and Protocols (NTP) Lab Series, University of South Carolina, 2019.

[21] TCP congestion control, Wikipedia, the free encyclopedia, 2021. Extracted on 22.11.2021 from https://en.wikipedia.org/wiki/TCP_congestion_control.

[22] TCP-Illinois, Wikipedia, the free encyclopedia. Extracted on 24.11.2021 from https://en.wikipedia.org/wiki/TCP-Illinois.

[23] M. Tekala and R. Szabó, Throughput analysis of scalable TCP congestion control, Periodica Polytechnica, Electrical Engineering 51(1–2) (2007) 57–64.

[24] S. Trivedi, S. Jaiswal and R. Kumar, Comparative performance evaluation of TCP Hybla and TCP Cubic for satellite communication under low error conditions, in: Proc. 4th IEEE Int. Conf. on Internet Multimedia Systems Architecture and Application (IMSAA-2010), 2010.

[25] Throughput. Wikipedia, the free encyclopedia. Extracted on 26.11.2021 from https://en.wikipedia.org/wiki/Throughput.

[26] S.-Y. Wang, Comparison of SDN OpenFlow network simulator and emulators: EstiNet vs. Mininet, in: 2014 IEEE Symposium on Computers and Communications (ISCC), Funchal, Portugal, 2014, 1–6, https://doi.org/10.1109/ISCC.2014.6912609.

[27] What is throughput in networking? Bandwidth explained, 2019. Extracted on 26.11.2021 from https://www.dnsstuff.com/network-throughput-bandwidth.

[28] Yet another high speed TCP validation, using DCE (Direct-Code-Execution). Extracted on 23.11.2021 from https://github.com/Tejas111/YeAH-TCP.

Received on April 5, 2022
Accepted on November 9, 2022

Plamen Dimitrov Rozinov and Stefan Stanchev Dimitrov

Faculty of Mathematics and Informatics
Sofia University "St. Kliment Ohridski"
5 James Bourchier Blvd.
1164 Sofia
BULGARIA

E-mails: plamendro@gmail.com
stefansd@fmi.uni-sofia.bg