Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 191 (2021) 437–444

Procedia Computer Science

www.elsevier.com/locate/procedia

The 8th International Symposium on Emerging Inter-networks, Communication and Mobility (EICM)
August 9-12, 2021, Leuven, Belgium

ABCSS: A novel approach for increasing the TCP congestion window in a network

N. M. Kasoro[a,b], S. K. Kasereka[a,b,*], G. K. Alpha[c], K. Kyamakya[d]

[a] *University of Kinshasa, Faculty of Sciences, Mathematics and Computer Science Department, Kinshasa, Democratic Republic of the Congo*
[b] *Artificial intelligence, BIg data and modeLing simulation research center (ABIL), Kinshasa, Democratic Republic of the Congo*
[c] *Université Libre des Pays des Grands Lacs, Faculty of Sciences and Technologies, Goma, Democratic Republic of the Congo*
[d] *Alpen-Adria-Universitaet Klagenfurt, Institute of Smart Systems Technologies, Department of Mathematical Sciences, Klagenfurt, Austria*

Abstract

Congestion control is a useful task to be taken into account in order to maintain a good traffic in a network. In the case of a TCP/IP (Transmission Control Protocol/Internet Protocol) network, congestion creates queues in routers and even packet rejection. It is the reason why research have been conducted to improve congestion control mechanisms. Nowadays, several TCP congestion algorithms have been implemented. However, with the technology evolution and the need in terms of traffic capacity, various mechanisms implemented by these algorithms have shown their limitations and still need to be improved. The aim of this paper is to develop a novel TCP congestion control approach based on two famous algorithms: Appropriate Byte Counting (ABC) and Slow Start (SS). The results obtained show that ABC increases the congestion window appropriately compared to Slow Start. However, the congestion window remains invariant during the first round trips, which causes the TCP burst phenomenon. The TCP splitting can cause buffer overflows and long queues. For theses shortcomings, we propose ABCSS as a novel approach that appropriately increase the TCP congestion window and minimize the TCP splitting problem. The simulations of ABCSS algorithm provide interesting results compared to some known algorithms.

Keywords: Congestion; TCP; ABC; Slow Start; Congestion window; Congestion control; ABCSS.

1. Introduction

Nowadays, the world is more and more connected via the Internet, the underlying network has to work more and more efficiently to obtain a stable and efficient connection all the time. Congestion, in a computer network, occurs

* Corresponding author. Tel.: +243-821-828-964.
E-mail address: selain.kasereka@unikin.ac.cd

10.1016/j.procs.2021.07.075

in the router when it has a queue full of packets. This situation can cause packet rejections and long delays in their transfers. In order to solve this problem, several algorithms have been implemented to control the increase in the TCP congestion window [13], i.e. the number of packets that can be in transit in the network. And for each packet received by the receiver, an acknowledgement packet (ACK) must be sent, which represents a large amount of traffic in the network and requires good congestion avoidance mechanisms. Among the most used algorithms, we can mention SS (Slow Start) [6] and ABC (Appropriate Byte Counting)[12]. The implementation of different algorithms to increase the TCP congestion window leads to irresolution in choosing which one to use. However, it is important to choose an algorithm that can better address performance issues. While some algorithms, such as Slow Start, increase *cwnd* (congestion window) by sending acknowledgements of receipt (ACKs), other algorithms over the Internet can pass through some links with different speeds (bandwidth, latency) and router with different processing speeds. It should be noted that when a router buffer is full, it throws the packets away without notifying the sender. In this case, we say that there is congestion. Congestion occurs when a machine in the packet-switched network receives more packets than it can temporarily store before re-transmitting them. Indeed, the packets are first stored in buffers before retransmission at the output. When packets arrive faster than they are retransmitted, they stick together in the buffers causing a congestion and then, the excess packets are deleted. Based on the information presented below, it is important to maximize the use of the network connecting sender and receiver, i.e. send packets at the maximum speed without causing congestion.

Several works have been conducted to increase the window and avoid TCP congestion [1, 2, 4, 8, 10, 11]. However, with the evolution of networks and the increase in the amount of traffic, the different mechanisms implemented by these works have shown their shortcomings and require further improvement [3, 7, 9, 14]. In this paper, we present a hybrid algorithm based on ABC and SS for incresing TCP congestion window. The paper is structured as follows. First, we present an overview on SS and ABC algorithms with a focus on their complexities. Secondly, we propose a novel algorithm termed ABCSS by describing used mathods and matherials. Thirdly, we discuss the results obtained and finaly we conclude this paper.

2. An overview on SS and ABC algorithms

2.1. Slow Start (SS description and complexity)

At the beginning of the connection, it is imperative not to send the entire content of the window. It is the reason why TCP uses the Slow Start algorithm in which the transmission of packets is controlled by the rate of reception of their acknowledgements instead of sending them all at once [12]. When the connection is established or after a period of idle calm, the transmitter sends a first maximum size segment (MSS) and it takes time waiting for acknowledgement. Upon receipt, the transmitter sends two segments, then four segments and continously until it reaches the maximum opening of the window. To calculate the complexity of SS to reach the threshold, we are based on the cost and number of times each operation is performed in the Slow Start pseudocode to reach the *ssthresh*. The Table 1 illustrates the fact.

Table 1. Estimated cost of SS before *ssthresh*

	Cost	Time
$cwnd = IW$	C_1	1
$ssthresh = randomvalue$	C_2	1
$if(cwnd < ssthresh)$	C_3	$(ssthresh - MSS) = N$
$cwnd = cwnd + SMSS$	C_4	N

Based on Table 1, the complexity can be calculated by: $C(N) = C_1 + C_2 + C_3N + C_4N = C_1 + C_2 + (C3 + C4)N$. Then the complexity is given by $O(N)$.

2.2. Appropriate Byte Counting (ABC) description and complexity

Appropriate Bytes Counting is described in the RFC 3465 [2]. This algorithm proposes the increasing of the congestion window based on the number of arriving acknowledgements. The window is increased according to the number of bytes recognized by these arriving ACKs. ABC suggests to store the number of acknowledged bytes in a variable *bytes_acked* in the TCP control block. When *bytes_acked* becomes greater or equal to the *cwnd*, it is then reduced to the value of *cwnd*. And then, the *cwnd* is incremented by a maximum segment size.

Table 2. Estimated cost of ABC algorithm before *ssthresh*

	Cost	Time
$cwnd = IW$	C_1	1
$ssthresh = randomvalue$	C_2	1
$bytes\_acked = 0$	C_3	1
$if(cwnd <= sstresh)$	C_4	N
$if(bytes\_acked < cwnd)$	C_5	$\sum_{i=1}^{N} i$
$if(bytes\_acked == bytes\_acked + nbytes)$	C_6	$\sum_{i=1}^{N} i$
$elseif$		
$bytes\_acked = cwnd$	C_7	0
$cwnd = cwnd + SMSS$	C_8	$\sum_{i=1}^{N} i$

N is a linear value that designates the value $ssthresh - cwnd$, the number of times C_4 should normally be executed. i designates the value $cwnd - bytes\_acked$. It should be noted that operations C_5, C_6, C_7 and C_8 depend on i and N while we calculate the complexity as a function of N. C is an operation on the same *cwnd* variable; it is therefore the same cost as C_7 that does not need to be included in the calculations, and, it is cancelled. Thus, we have:

$$\begin{aligned} C(N) =& C_1 + C_2 + C_3 + C_4N + C_5\sum_{i=1}^{N} i + C_6\sum_{i=1}^{N} i + C_8\sum_{i=1}^{N} i \\ =& C_1 + C_2 + C_3 + C_4N + \sum_{i=1}^{N} C_5 i + \sum_{i=1}^{N} C_6 i + \sum_{i=1}^{N} C_8 i \end{aligned}$$

We know that $\sum_{i=1}^{N} ki = \frac{kN(N+1)}{2}$, and then, we obtain:

$$\begin{aligned} C(N) =& C_1 + C_2 + C_3 + C_4N + \frac{C_5N(N+1)}{2} + \frac{C_6N(N+1)}{2} + \frac{C_8N(N+1)}{2} \\ =& C_1 + C_2 + C_3 + C_4N + (C_5 + C_6 + C_8)\frac{(N^2+N)}{2} \\ =& C_1 + C_2 + C_3 + C_4N + \frac{(C_5 + C_6 + C_8)}{2}N^2 + \frac{(C_5 + C_6 + C_8)}{2}N \\ C(N) =& O(N^2) \end{aligned}$$

Then, based on the Table 2, the ABC complexity is given by $C(N) = O(N^2)$.

2.3. ABC vs SS in term of complexity

Based on the results obtened for ABC and SS complexities, it is obvious that Slow Start is faster than ABC. It reaches the *ssthresh* threshold more quickly compared to the ABC algorithm. Indeed, what is important is not to quickly reach the threshold, but to increase the congestion window more appropriately. When the threshold is reached, for Slow Start, as for ABC, TCP switches directly to Avoidance Congestion mode, which increases the congestion window more slowly. However, Slow Start can quickly increase the congestion window while the sender does not have enough packets to send, requiring such a large *cwnd*. It should be noted that the ABC algorithm increases the *cwnd* slowly and does not reach the *ssthresh* threshold very quickly, but it waits until the value of *bytes_acked* is equal to the initial value of *cwnd*. However, even in this case, taking into account the complexity factor, we cannot really determine whether this increase in ABCś *cwnd* is entirely appropriate and whether it is not flawed. Therefore, we cannot rely only on complexity calculation to estimate the performance of these algorithms. A simulation approach of these two algorithms in a physical environment can help to observe their behaviour.

3. Proposed ABCSS algorithm

3.1. Materials

To achieve this research, some materials were needed for the practical realization. We used Network Simulator 2 (NS2) [5] for performing simulations of the proposed novel algorithm termed ABCSS. The network architecture we adopt to implement and test the algorithm is made of two different network interconnected by a router. This architecture is presented by the simple topology shown in Fig. 2.

3.2. Methods

The idea of the new approach is therefore to imitate the start-up phase of the Slow Start algorithm for *cwnd* increase. Thus, instead of incrementing the *cwnd* size of an SMSS when each ACK arrives, we propose to increment this *cwnd* by *N_bytes*, where *N_bytes* is the number of bytes recognized by the last ACK received. Hence, the *bytes_acked* variable will no longer be taken into account in this new approach. Indeed, *bytes_acked*, on which the incrementation of *cwnd* into ABC depends, evolves less quickly for small segments, which causes a constant linearity of *cwnd* at the beginning of a TCP transfer. In this new appoach, if *cwnd* manages to exceed the value of *ssthresh*, *cwnd* is reset to *ssthresh*, before entering the phase of increasing the congestion window, in a more linear way. This is a reset that is done only once before the curve changes to ABCSS. The process allows to improve the phase for achieving *ssthresh*. The pseudocode of this proposed approach is presented in the Algorithm 1.

Algorithm 1 ABCSS

```
Initialization
cwnd = IW                                  ▷ Initial value of the congestion window
ssthresh = random_value                    ▷ High value if necessary
New ACK received
if cwnd < ssthresh then
    cwnd = cwnd + N_bytes
else                                       ▷ Congestion Avoidance
    cwnd = ssthresh
    cwnd = cwnd + N_bytes/cwnd
```

The counting of bytes is indeed performed, but these bytes are no longer stored in the *bytes_acked* variable which, when lower, caused a *cwnd* invariance at the connection establishment. With this new approach, ABC will now increase congestion after each RTT (Round Trip Time), but in an appropriate way, i.e. instead of increasing the congestion window of an SMSS, it will be increased by the number of bytes recognized by the incoming ACK. The algorithm 1 can be presented by the flowchart described in the Fig. 1.

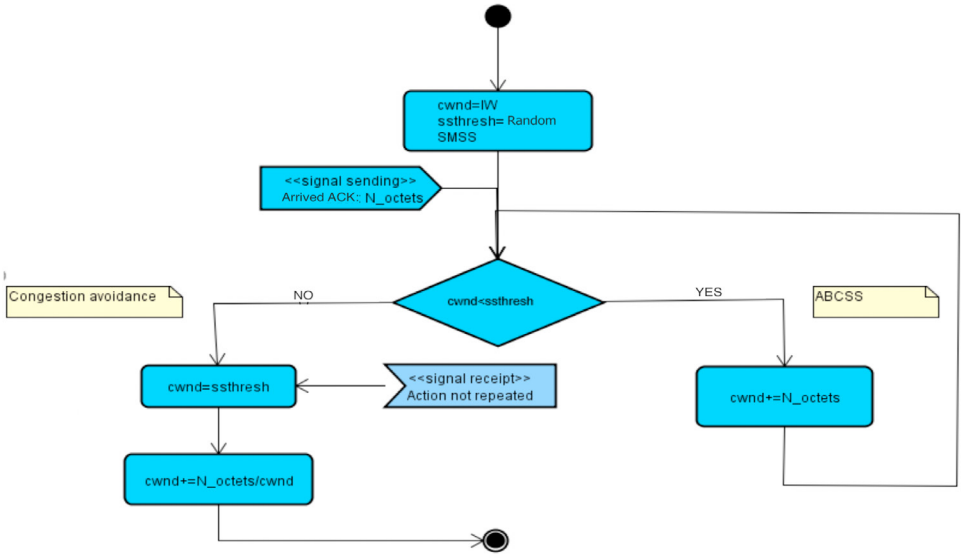


Fig. 1. Flowchart of the ABCSS algorithm combined to the congestion avoidance

3.3. ABCSS algorithm complexity

Table 3. Estimated cost of ABCSS algorithm

	Cost	Time
$cwnd = IW$	C_1	1
$ssthresh = randomvalue$	C_2	1
$if(cwnd < sstresh)$	C_3	$(ssthresh - SMSS) = N$
$cwnd = sstresh$	C_4	1
$cwnd = cwnd + N\_bites$	C_5	N

Based on the table 3 we have:

$$\begin{aligned} C(N) &= C_1 + C_2 + C_3 + C_4N + C_5N \\ &= C_1 + C_2 + C_3 + (C_4 + C_5)N \\ C(N) &= O(N) \end{aligned}$$

4. Simulation of ABC, SS and ABCSS algorithms

4.1. Context and topology of simulation

The following simulation aims at discovering how *cwnd* evolves according to SS and ABC algorithms up to the *ssthresh* threshold. *ssthresh* is the Slow Start threshold that defines whether to move from SS or ABC to the Avoidance Congestion phase. Thus, through these simulations, it is a question of discovering which mechanism increases *cwnd* more quickly or appropriately. Fig. 2 below presents the simple topology used in performed simulations.

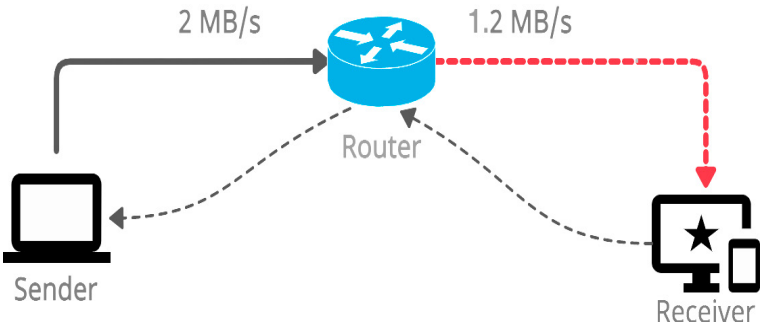


Fig. 2. Topology of simulation

4.2. Used parameters

Some parameters were used in order to perform simulation of all the algorithms concerned by our reserach, we mention ABC, SS and ABCSS. The following Table 4 describes these parameters:

Table 4. Parameters used and their values

Parameter	Value
$SMSS$	2190 bytes
IW	6570 bytes
$sstresh$	34040 bytes
$passband$	$2Mb/s$ and $1.2Mb/s$
RTT	$0.1second$
$router\_queuing$	$FIFO$

4.3. Simulations

The simulations of ABC and SS is presented in Fig. 3. Simulation of ABCSS is presented in Fig. 4. The figure shows an evolution of *cwnd* in ABCSS mode with 1916-byte segments and provides a comparison with the ABC and SS algorithms using the similar parameters.

5. Discussion of the results

As shown in Fig. 3, for the Slow Start mode (blue curve), the congestion window is incremented by one SMSS (2190 bytes) at each RTT. This increase depends only on the arrival of ACKs and not on the size of the segment. Thus, for smaller or larger segments, the increase in the congestion window is clearly the same. For Slow Start, *cwnd* reaches the *ssthresh* threshold after 1.3 seconds. However, in ABC mode, the evolution of the congestion window is clearly slower than in Slow Start. This is a slow but consistent increase in *cwnd*. The congestion window therefore changes in proportion to the amount of data sent over the network. Based on complexity calculations and simulation, the analyses presented above provide us with information on the power of ABC's consistency with Slow Start in increasing the TCP congestion window. It is an algorithm whose counting of bytes is recognized by ACKs. It allows to increase *cwnd* more quickly but according to the traffic observed in the network. However, according to the ABC simulation graph shown in Fig. 3 of RTT 1 to RTT 4, *cwnd* remains constant. ABC can not easily double the sending of segments to each RTT, the worst case, it could cause a burst of TCP. This observation leads us to study this problem in order to minimize this bursting of TCP which would be, in some cases, due to ABC which nevertheless seeks to increase the window of congestion appropriately.

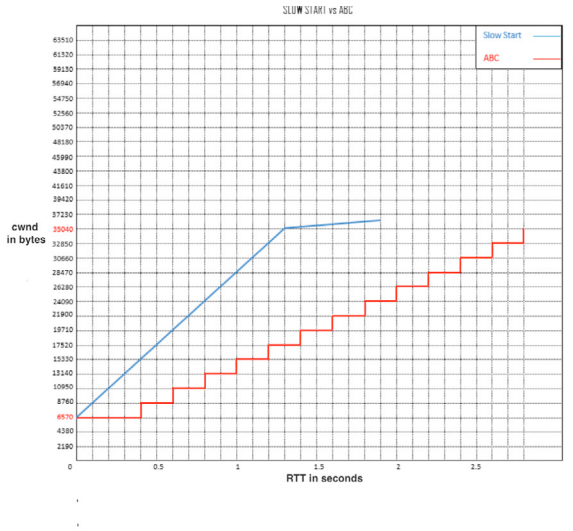
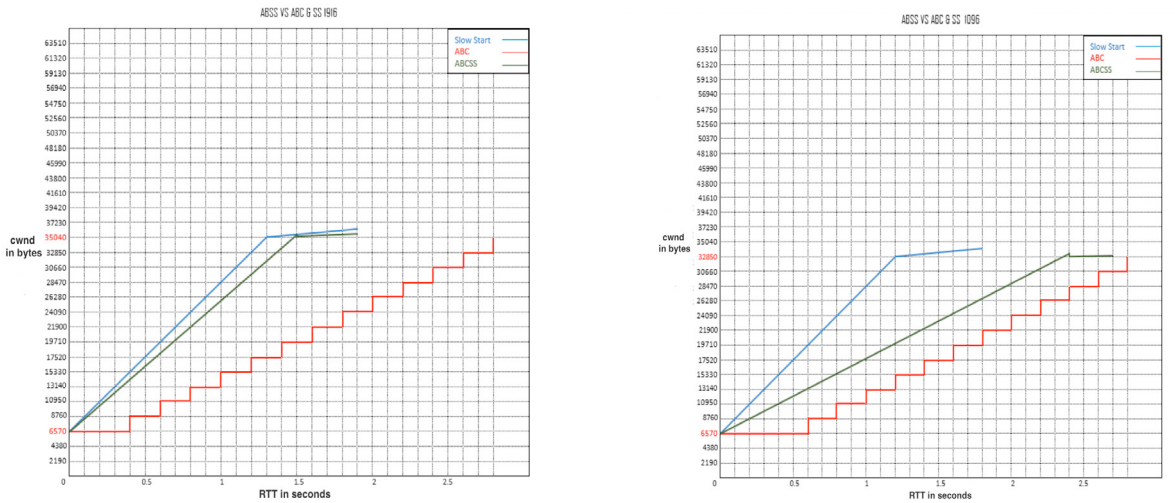


Fig. 3. Simulation of ABC and SS algorithms using NS2 with all parameters described in the Table 4



(a) (b)

Fig. 4. Simulation of the ABCSS algorithm (a) Evolution of *cwnd* with 1916-byte segment and (b) evolution of *cwnd* with 1096-byte segments for a 32,850-byte *ssthresh*

Simulations presented in Fig. 3 show that with ABC, the congestion window takes a long time before being, for the first time, increased by an SMSS. According to the graph shown in this figure, we notice that the congestion window remains invariant until RTT 4 (0.4 seconds) before incrementing. With more segments, this constant evolution is at the root of a congestion causing the bursting of the transfer. As shown in Fig. 4(a), the congestion window in ABCSS mode evolves directly to the first RTT as in Slow Start mode, but here it is not incremented by an SMSS (2190 bytes), but *N_bytes* (1916 bytes). We therefore avoid here the invariance observed on the ABC curve which does not evolve up to RTT 4 (0.4 seconds), a linearity that can cause the burst. It should be noted here that the ABCSS and SS curves are similar because of the fact that the 1916 segment tends towards SMSS, with a small difference of 274 bytes. Thus, with smaller segments, or of different sizes, the curve will not look the same as shown in Fig. 4(a). As shown in

the Fig. 4(b), with segments of 1096 bytes, *cwnd* in ABC mode reaches *ssthresh* after 2.4 seconds. This is a slower development than in the previous case. It is still appropriate because it results from byte counting. Considering the invariance of the congestion window, in ABC mode, *cwnd* maintains its initial value for 0.6 seconds while ABCSS is already evolving at each RTT. ABCSS evolves here almost as slowly as ABC, but avoids bursting in the transfer. From these two graphs(Fig. 4(a) and Fig. 4(b)), we can see that ABCSS makes *cwnd* evolve at each RTT and that the bursting problem due to possible congestion at the beginning of the TCP transfer in ABC mode is completely solved.

ABCSS algorithm is powerful for different TCP transmissions regardless of packet size or RTT. However, its impact is directly visible when it comes to small segments. Indeed, it is for these small segments that the invariance of *cwnd* at the beginning of the TCP- transfer becomes very high in ABC mode. ABCSS algorithm thus makes it possible to avoid this fragmentation while increasing, more quickly and appropriately, the number of segments that can pass through the network. It should be noted that, ABCSS algorithm has some weaknesses. Although ABCSS algorithm reaches the *ssthresh* threshold faster than ABC and switches to avoidance congestion mode, during this passage, the congestion window evolves in a very linear way and thus reduces the number of packets that can pass through the network. This is a reduction that occurs faster than in ABC mode.

6. Concluding Remarks

In this paper, we analyzed the two typical algorithms for increasing the TCP congestion window, namely Slow Start based on acknowledgement counting and Appropriate Byte Counting based on counting bytes recognized by acknowledgements. Based on a comparative study between these algorithms, we found that ABC, although appropriately increasing the congestion window, would cause a TCP burst problem at the beginning of the transfer due to the invariance of the congestion window to the first Round Trip Times. Then, we proposed a hybrid algorithm (ABCSS) based on these two algorithms. The hybrid nature of this algorithm comes from the fact that it combines ABC and Slow Start to increase *cwnd* consistently while minimizing TCP fragmentation. As a perspective, the simulation of ABCSS algorithm in a complex topology will be performed to test the efficiancy of the algorithm. We attend also to conduct a study of quality of service based on Fast Retransmit, Fast Recovery, RENO, Vegas, TAHOE and others based on the ABCSS algorithm proposed here. This study should therefore take into account the impact of delayed ACKs or loss of ACKs when exchanging data between entities in a TCP/IP network. Thus, ABCSS can be implemented in a physical environment.

References

[1] Abdelmoniem, A.M., Bensaou, B., 2017. Control theory based hysteresis switch for congestion control in data centers .
[2] Allman, M., 2003. TCP congestion control with appropriate byte counting (abc). RFC 3465.
[3] Dukkipati, N., Refice, T., Cheng, Y., Chu, J., Herbert, T., Agarwal, A., Jain, A., Sutin, N., 2010. An argument for increasing TCP's initial congestion window. ACM SIGCOMM Computer Communication Review 40, 26–33.
[4] Geist, M., Jaeger, B., 2019. Overview of TCP congestion control algorithms. Network 11.
[5] Issariyakul, T., Hossain, E., 2009. Introduction to network simulator 2 (NS2), in: Introduction to network simulator NS2. Springer, pp. 1–18.
[6] Jacobson, V., . Congestion avoidance and control/van jacobson, SIGCOMM.
[7] Kanagarathinam, M.R., Singh, S., Sandeep, I., Kim, H., Maheshwari, M.K., Hwang, J., Roy, A., Saxena, N., 2020. Nexgen D-TCP: Next generation dynamic TCP congestion control algorithm. IEEE Access 8, 164482–164496. doi:10.1109/ACCESS.2020.3022284.
[8] Kharat, P., Kulkarni, M., 2021. Modified QUIC protocol with congestion control for improved network performance. IET Communications .
[9] Kim, G.H., Song, Y.J., Mahmud, I., Cho, Y.Z., 2021. Adaptive decrease window for balia (adw-balia): Congestion control algorithm for throughput improvement in nonshared bottlenecks. Electronics 10. URL: https://www.mdpi.com/2079-9292/10/3/294, doi:10.3390/electronics10030294.
[10] Qazi, I.A., Andrew, L.L., Znati, T., 2009. Congestion control using efficient explicit feedback, in: IEEE INFOCOM 2009, IEEE. pp. 10–18.
[11] Savage, S., Cardwell, N., Wetherall, D., Anderson, T., 1999. TCP congestion control with a misbehaving receiver. ACM SIGCOMM Computer Communication Review 29, 71–78.
[12] Stevens, W.R., 1997. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms .
[13] Zhang, H., Bian, Z., 2002. Evaluation of different TCP congestion control algorithms using ns-2. CMPT 885-3: High-Performance Networks .
[14] Zhang, J., Yao, Z., Tu, Y., Chen, Y., 2020. A survey of TCP congestion control algorithm, in: 2020 IEEE 5th International Conference on Signal and Image Processing (ICSIP), pp. 828–832. doi:10.1109/ICSIP49896.2020.9339423.