International Journal of Innovative
Computing, Information and Control
Volume **17**, Number **6**, December **2021**

 ISSN 1349-4198
pp. **1907–1920**

ENHANCED SLOW-START (ESS): IMPROVING BANDWIDTH UTILIZATION BY ESTIMATING BANDWIDTH DELAY PRODUCT

Zhiyuan Wang[1,2], Hong Ni[1,2], Xiaoyong Zhu[1,2,*] and Xu Wang[1,2]

[1]National Network New Media Engineering Research Center
Institute of Acoustics, Chinese Academy of Sciences
No. 21, North 4th Ring Road, Haidian District, Beijing 100190, P. R. China
{ wangzy; nih; wangx }@dsp.ac.cn; *Corresponding author: zhuxy@dsp.ac.cn

[2]School of Electronic, Electrical and Communication Engineering
University of Chinese Academy of Sciences
No. 19(A), Yuquan Road, Shijingshan District, Beijing 100049, P. R. China

Received June 2021; revised September 2021

Abstract. *In large bandwidth delay product (BDP) networks, standard slow-start algorithm has exponentially increasing congestion window, but is still increasingly difficult to meet the demand for fast utilization of bandwidth. The hindsight of standard slow-start algorithms on network congestion often results in significant packet loss as well. This phenomenon has become increasingly serious with the upgrading of network devices and the widespread deployment of the Internet. In this paper, we proposed the enhanced slow-start (ESS). ESS rapidly increases the congestion window (CWND) at a higher rate than the standard slow-start algorithm and stops the window growth in time when the window grows to a suitable value. Our simulation evaluation in NS3 shows that ESS has better performance in terms of bandwidth utilization compared to several widely used slow-start algorithms.*

Keywords: Slow-start, Congestion control, Congestion window, NS3

1. **Introduction.** Transmission control protocol (TCP) is a reliable end-to-end sender-driven transport protocol that carries the majority of traffic in the Internet. It is the de facto standard transport protocol [1]. However, the emergence of various new applications also brings new requirements to TCP, which makes TCP face great challenges in areas such as latency and throughput [2]. The performance of slow-start, the congestion control algorithm for TCP to detect link throughput quickly, directly affects TCP's latency and throughput in the early stages of startup.

On the one hand, in current common high bandwidth delay product (BDP) network, the exponential growth strategy of the standard slow-start algorithm seriously affects bandwidth utilization in early stage of TCP startup. Based on the exponential growth pattern which can be calculated that in networks with BDP greater than 800 segments, the slow-start algorithm requires at least 6 round-trip times to bring the TCP congestion window to a suitable size [3], and this delay will not be negligible when the round-trip time (RTT) is large. On the other hand, the standard slow-start algorithm exits only after a congestion event occurs or when the ssthresh is exceeded. Too small ssthresh will cause TCP to exit the slow-start process prematurely, so that TCP cannot make full use of the network bandwidth, while too large ssthresh will cause TCP to send too many packets to the network. In the initial stage of TCP startup, the ssthresh will be set to the maximum value, so it can only wait for a congestion event to occur before exiting the

DOI: 10.24507/ijicic.17.06.1907

slow-start process. This means that packets can fill up or even overflow the bottleneck buffer in the end of slow-start. With the decrease of the memory cost, the memory of the current switching devices is getting larger and larger [4], and TCP can only withdraw from the slow-start process to start congestion avoidance. It will take up much network resources and will take huge packet retransmission [5].

Some strategies have been proposed to optimize the above problem [5-16], while AFStart [6] improves the growth rate of the congestion window by estimating BDP through ImTCP [17] and can exit when the window grows to a reasonable value. Bottleneck bandwidth and round-trip propagation time (BBR) [7,8] is a new congestion control algorithm proposed by Google, which continuously increases the number of packets in the startup phase, and when the transmission rate no longer increases it will exit the startup phase. In addition, the work represented by HyStart [5], CapStart [9], and StopEG [10] determined the exit time from the slow-start to the congestion avoidance process to reduce the probability of massive packet loss in the end of slow-start. Stateful-TCP [11] increases the transmission rate of subsequent flows on that path through the bandwidth and RTT measured by the previous flows.

In order to make full use of bandwidth faster and avoid serious packet loss that may occur in the end of slow-start process, in this paper, we proposed a novel efficient slow-start algorithm, called enhanced slow-start (ESS). It can calculate a reasonable congestion window based on the information collected in an RTT, and continuously adjust the current congestion window according to the calculated value, so as to achieve a rapid growth of the congestion window. When the congestion window is not less than the estimated congestion window value, a large increase in delay or other congestion events occur, the ESS will exit the slow-start and enter the congestion avoidance process.

The rest of this paper is organized as follows. Section 2 introduces the existing slow-start optimization work. Section 3 introduces the principles and problems of the slow-start algorithm. Section 3 also analyzes the law of data transmission in slow-start, derives the calculation formula of the currently available congestion window, and explains the algorithm model. In Section 4, we use NS3 to build a network simulation environment and compare the enhanced slow-start algorithm with several widely used slow-start algorithms. Section 5 summarizes the work of this paper and explains the future research directions.

2. **Related Work.**

The goal of the slow-start algorithm is to quickly adjust the congestion window to a value that can fully utilize the network bandwidth and exit [18]. Studies have shown that the congestion window value is not the bigger the better, excessive congestion window value will only cause additional queuing delay [4]. The best congestion window point is shown in Figure 1. This point is also known as the Kleinrock's optimal operating point [19]. By using this point as the congestion window, it can make full use of the bandwidth and have the smallest round-trip time [7].

Jacobson's slow-start algorithm [20] is designed to solve the problem of Internet congestion collapse by detecting the largest congestion window in Figure 1, while any value in the middle of the optimal congestion window point and the maximum congestion window point is better than the maximum congestion window. In order to enable the congestion window to grow quickly to the optimal congestion window and exit in time, many researchers have tried to improve the slow-start algorithm.

Some researchers have developed new algorithms to solve the problem that the congestion window is not growing fast enough. Cavendish et al. [9] proposed CapStart, which divided the network bottleneck into two categories according to whether the network interface card (NIC) bandwidth is greater than the estimated path bandwidth, only activate limited slow-start if the NIC bandwidth is higher than the estimated path bandwidth.

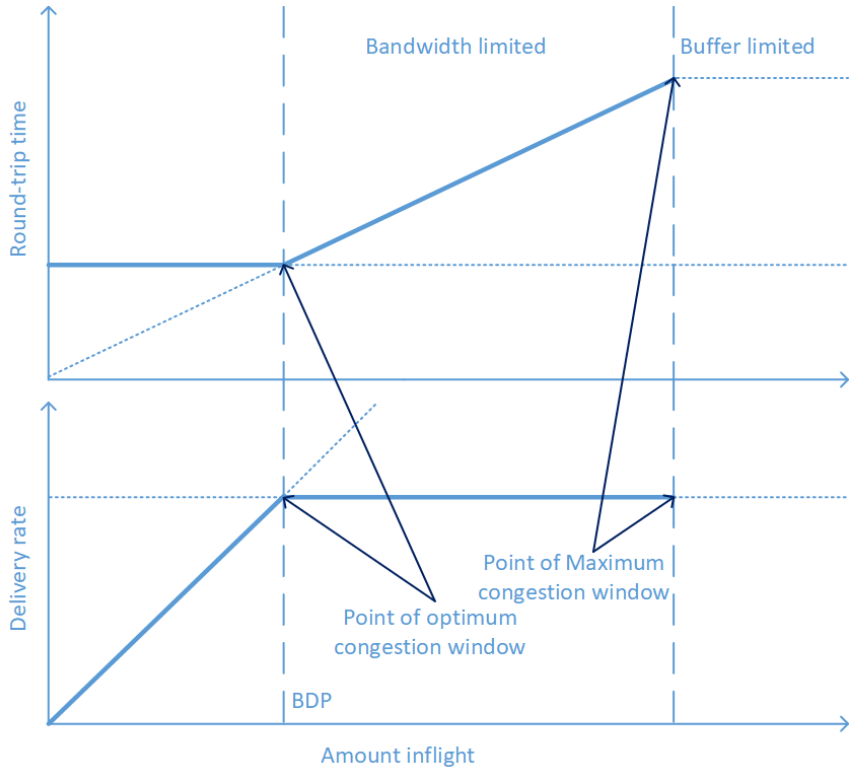

FIGURE 1. The effect of increased inflight data on round-trip time (RTT) and delivery rate based on [7]

Guo and Lee [11] proposed Stateful-TCP, which will record the information of the last connection of the path, and subsequent connections will use the recorded information. Halfback [16] was proposed by Li et al. to improve the transmission speed of short-flow. Liu et al. [14] proposed jump start which allowed TCPs to begin transmission at whatever rate they deem appropriate. Hauger et al. [13] proposed quick-start, where congestion window is adjusted based on explicit feedbacks from network routers. Nie et al. [15] proposed TCP WISE, where the values of initial CWND are proactively learned by the historical experience on the server-side.

In addition to the growth rate of the window, the choice of the exit time of the slow-start algorithm is also an important issue. Arghavani et al. [10] proposed StopEG, which has theoretically calculated that when the bottleneck link is not saturated, the inflight packets in the forward path will not exceed 56.8% of all inflight packets. HyStart [5] was proposed by Ha and Rhee. It defines the ACK of packets sent back-to-back with an arrival interval of no more than a few milliseconds as an ACK train. It has two independent ways of the exiting slow-start. One way is that the ACK train is close to RTT, which indicates that the path capacity has been reused. And the other is that the round-trip time of the first packet of each round increases, which indicates that a new round of packet starts to be sent when there may be the previous round packets still queued in the network. HyStart is already available as part of the latest Cubic [21] algorithm and is used in the Linux kernel. HyStart++ [22] is similar to HyStart, but it adds a limited slow-start process relative to HyStart to solve the problem that the HyStart algorithm may exit the slow-start process early after the window grows slowly and cannot reuse the bandwidth.

AFStart [6] is an algorithm that improves the window growth rate and improves the exit time of slow-start. AFStart measures the available bandwidth through ImTCP. When the first measured value is less than 16 segments, the slow-start process will switch to

the standard slow-start algorithm; otherwise it will continuously estimate the available bandwidth and search for the true bandwidth value based on the measured available bandwidth, and update the ssthresh to the measured bandwidth delay product (BDP).

BBR [7] is a congestion control algorithm based on link bottleneck bandwidth and round-trip propagation time proposed by Google in 2016. BBR will keep increasing the pacing rate and congestion window in the startup state until the transmission rate is no longer increasing for three consecutive measurements to ensure that the current congestion window is not smaller than the optimal congestion window, and then BBR will move to the next state. Since the congestion window of BBR grows rapidly, it is very easy to cause bottleneck buffer overflow. Therefore, packet loss and explicit congestion notification (ECN) [23] are added to BBRv2 for consideration [24].

The above work has optimized the TCP slow-start algorithm to varying degrees; however, new optimization algorithms are necessary to further improve the performance of TCP in the slow-start process. This paper analyzes the slow-start algorithm in high-latency and large-bandwidth networks, the congestion window does not grow quick enough, and a large number of data packets are prone to loss. Then this paper proposes a new slow-start algorithm based on the characteristics of the data packet delivery in the slow-start process, called enhanced slow-start algorithm. The simulation experiment on NS3 proves that the enhanced slow-start algorithm has a faster window growth rate and better slow-start exit time than several widely used slow-start algorithms.

3. **Enhanced Slow-Start.**

In this section, we introduce the traditional slow-start algorithm, and compare the traffic characteristics of the two states at the beginning of the slow-start and after the congestion window reaches the optimum congestion window. By comparing these features, we propose an expression for estimating the optimal congestion window based on the information in an RTT. The enhanced slow-start algorithm uses the RTT of selected packets to divide the slow-start process into many calculation cycles, and within each cycle decides whether to increase the congestion window or exit the slow-start based on the estimation of the optimal congestion window.

3.1. **Slow-start overview.**

The purpose of TCP congestion control is to maximize the use of network bandwidth and improve transmission efficiency. Slow-start is an important congestion control mechanism of TCP. It aims to make full use of network bandwidth in the shortest possible time when the available network resources are unknown, while avoiding serious congestion.

Slow-start, also called exponential growth period, is when a TCP connection has just started and the sender sends data with a default initial congestion window, which in the Linux kernel is initially 10 [25]. After that, the congestion window increases by one segment for each ACK received until TCP detects a congestion event or the congestion window size exceeds ssthresh, at which point TCP will no longer perform the slow-start algorithm and will enter the congestion avoidance process.

When the bandwidth is large, a too small ssthresh will cause TCP to have a long additive increase process after the slow-start ends, and it may even take tens of seconds to detect the maximum congestion window. Therefore, according to the recommendation of RFC5681 [3], sshresh should be set to the maximum value when the connection is initialized to adapt to the high-speed network, and then the sshresh needs to be dynamically modified according to the congestion situation. This means that the first slow-start process after the TCP connection is established can always be stopped after a congestion event occurs, which inevitably requires a large number of data packets to be retransmitted. This is a serious waste of bandwidth.

The initial value of the congestion window is $cwnd_{initial}$, and the round-trip time of the data packet on the path is rtt. For a given target window $cwnd_{target}$, the time for the slow-start algorithm to detect growth to the target window satisfies the following equation:

$$t = \log_2 \left(\frac{cwnd_{target}}{cwnd_{initial}} \right) * rtt \tag{1}$$

From the above equation, it is clear that the time spent in slow-start is proportional to the round-trip time and the target window size. This means that when a link has a large round-trip time or a large $cwnd_{target}$, the slow-start algorithm will waste a lot of time in this process. According to Equation (1), Figure 2 can be drawn. When the congestion window has different initial values, the time required for the congestion window to reach different sizes is shown in Figure 2. The congestion window must be at least the size of the bandwidth-delay product to fully utilize the network bandwidth. When the bandwidth-delay product is greater than 800 segments, even if the initial value of the congestion window is 10, it takes at least 6 RTTs to reach a reasonable value. For short flows, the transmission may even be completed before the congestion window grows to its maximum value.

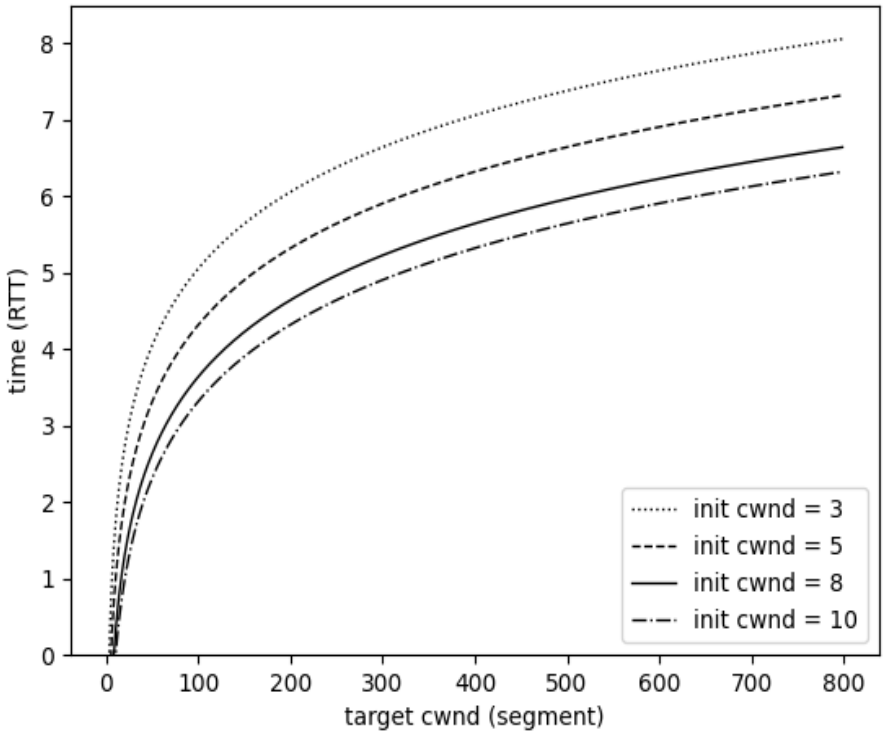

FIGURE 2. Time required to reach the congestion window value

3.2. **Calculation cycle of enhanced slow-start.** We divide the slow-start process into many calculation cycles, where the end of one cycle is the beginning of another. The size of the calculation cycle is not fixed, but depends on the RTT of the selected packets. We take the next packet sent after the first received ACK as the first selected packet. In Figure 3, after receiving the ACK of packet a and sending out packet e, the RTT of packet e is the first calculation cycle. When a calculation cycle ends, the next packet sent is the selected packet. Packets sending and receiving are mainly concentrated at the beginning of each cycle, such as the time from T1 to T2 in Figure 3.

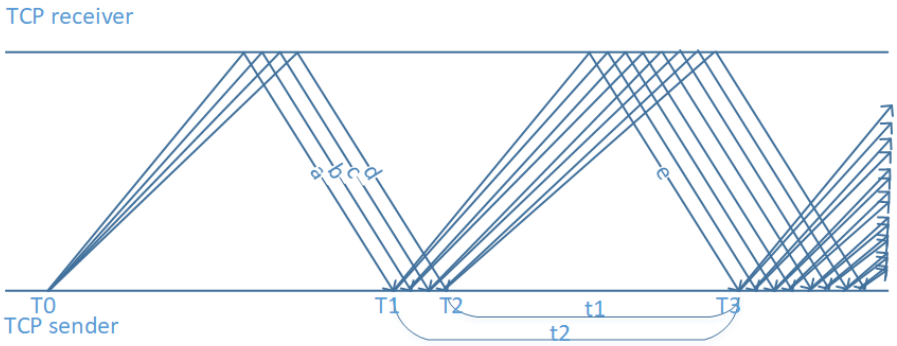

FIGURE 3. Data transfer during slow-start

3.3. **Principle of enhanced slow-start.** When the sending rate is greater than the network bottleneck bandwidth (BtlBw), packets are queued in the bottleneck buffer of the link, and the final transmission rate depends on the size of the BtlBw. The sender sends packets of the initial congestion window size at once. The ACKs of the packets sent at the same time will reflect the size of the BtlBw to some extent. The BtlBw of the link can be roughly calculated based on the time taken from the receipt of the first ACK to the receipt of the ACKs of all packets sent as described above and the amount of data received during this period. Due to the exponential growth mechanism during slow-start, TCP uses a sending rate twice the estimated bottleneck bandwidth during the data sending phase, which makes it easier to queue packets at the bottleneck link buffer, thus bringing the next estimated data transfer rate closer to the BtlBw. As the congestion window increases, the estimated BtlBw will become more accurate.

If the available bandwidth of the network changes dramatically within each RTT, then the network measurement will be useless. Therefore, we assume that the network bandwidth remains stable within one RTT.

In addition, we assume that the BtlBw of the link in Figure 3 as B, the RTT of packet e as t_2, the data size as d_2, the sum of the size of packets b, c, d and e as d_1, and the cwnd of moment T_0 as $cwnd_0$.

For the slow-start process in Figure 3, the BtlBw can be estimated as follows:

$$B = \frac{d_1 - d_2}{t_2 - t_1} \tag{2}$$

When the congestion window is at the value of optimum congestion window, the RTT should have the minimum value, let the congestion window be $cwnd_{kp}$ and the RTT be $rtt_{\min}$. RTT of a sample t_2 should be no less than the minimum RTT, and the ratio of $rtt_{\min}$ to t_2 satisfies $0 < k \leq 1$. As the inflight data increases, the queuing delay in the bottleneck buffer increases, so the value of t_2 will become larger and the value of k will become smaller.

$$rtt_{\min} = k * t_2 \quad (0 < k \leq 1) \tag{3}$$

Then, BtlBw can be calculated based on $rtt_{\min}$ and $cwnd_{kp}$.

$$B = \frac{cwnd_{kp}}{rtt_{\min}} \tag{4}$$

The new expressions can be obtained from the expressions (2), (3) and (4).

$$cwnd_{kp} = \frac{d_1 - d_2}{t_2 - t_1} * rtt_{\min} = \frac{d_1 - d_2}{t_2 - t_1} * k * t_2 \leq \frac{d_1 - d_2}{t_2 - t_1} * t_2 = cwnd_{\max} \tag{5}$$

We use $cwnd_{\max}$ to represent the value on the right-hand side of the inequality, which is a value not smaller than the optimal congestion window. When no packet unorder and loss events occur, the above principle is followed in each cycle, so the value of the optimal congestion window can be calculated in this way in each cycle.

3.4. **Congestion window adjustment and slow-start exit strategy.** In the standard slow-start algorithm, the sender increases the congestion window by 1 for each ACK received. We retain this feature in the enhanced slow-start (ESS) algorithm. The ESS uses the method proposed in Section 3.3 to estimate the available congestion window in each cycle, and adjusts the congestion window to the larger of the exponential growth congestion window and the estimated window $cwnd_{\max}$. The exponential growth approach is retained in ESS for two reasons: First, it causes the sender to send packets with twice the estimated bandwidth of bottleneck link, thus allowing for a more accurate estimation of the BtlBw in the next cycle; Second, it ensures that the ESS does not have a lower congestion window growth rate than the standard slow-start algorithm.

However, it has a huge risk to adjust the congestion window to the larger of the exponential growth congestion window and the estimated value $cwnd_{\max}$. When one sample of the RTT is too large, a congestion window well beyond the optimum congestion window value is calculated, which can easily fill up or even overflow the bottleneck link buffer, resulting in a large number of packet losses. Given that an RTT sample is too large often means that the network is experiencing severe congestion, we decided to limit the size of the calculation cycle. The ESS will exit the slow-start when t_2 is greater than two times $rtt_{\min}$ ($k \geq 0.5$).

The ESS algorithm will also exit the slow-start when the value of the proposed calculation formula in Section 3.3 is not greater than current congestion window. It does not use the ssthresh, but retains the behavior of executing the congestion avoidance algorithm when a congestion event is detected on the sender side.

The ESS algorithm retains the two policies of exponential window growth and exit on occurrence of congestion of the standard slow-start, and accelerates the growth of window and slow-start exit by estimating the value of the congestion window greater than optimum congestion window.

3.5. **Enhanced slow-start algorithm description.** Figure 4 shows the flow chart of the enhanced slow-start. The algorithm takes a selected series of RTT samples as the calculation cycle, the RTT of the next packet sent after the first ACK is received as the first calculation cycle, and the RTT of the next packet sent after the last ACK is received in the previous cycle as the next cycle. At the end of each calculation cycle, the congestion window will be updated to the greater of the current window and $cwnd_{\max}$. The algorithm will exit the slow-start process in the following three cases.

- A congestion event occurs;
- The selected RTT sample is greater than 2 times the minimum RTT;
- The current congestion window is larger than the calculated $cwnd_{\max}$.

4. **Experimental Evaluation and Analysis.** Network Simulator 3 (NS3) supports the emulation of multiple network protocols and is easy to modify. In this section we experimentally validate the proposed algorithm using NS3. We chose additive increase multiplicative decrease (AIMD) [20] as the algorithm of ESS in the congestion avoidance process to meet the needs of TCP's normal operation. The experiments use a dumbbell topology with TCP sender and receiver on both sides and a bottleneck link in the middle, using an MTU of 400 byte and a passive queue management (PQM) algorithm with DropTail policy. In the experiments, we compared ESS with Cubic (HyStart), BBR,

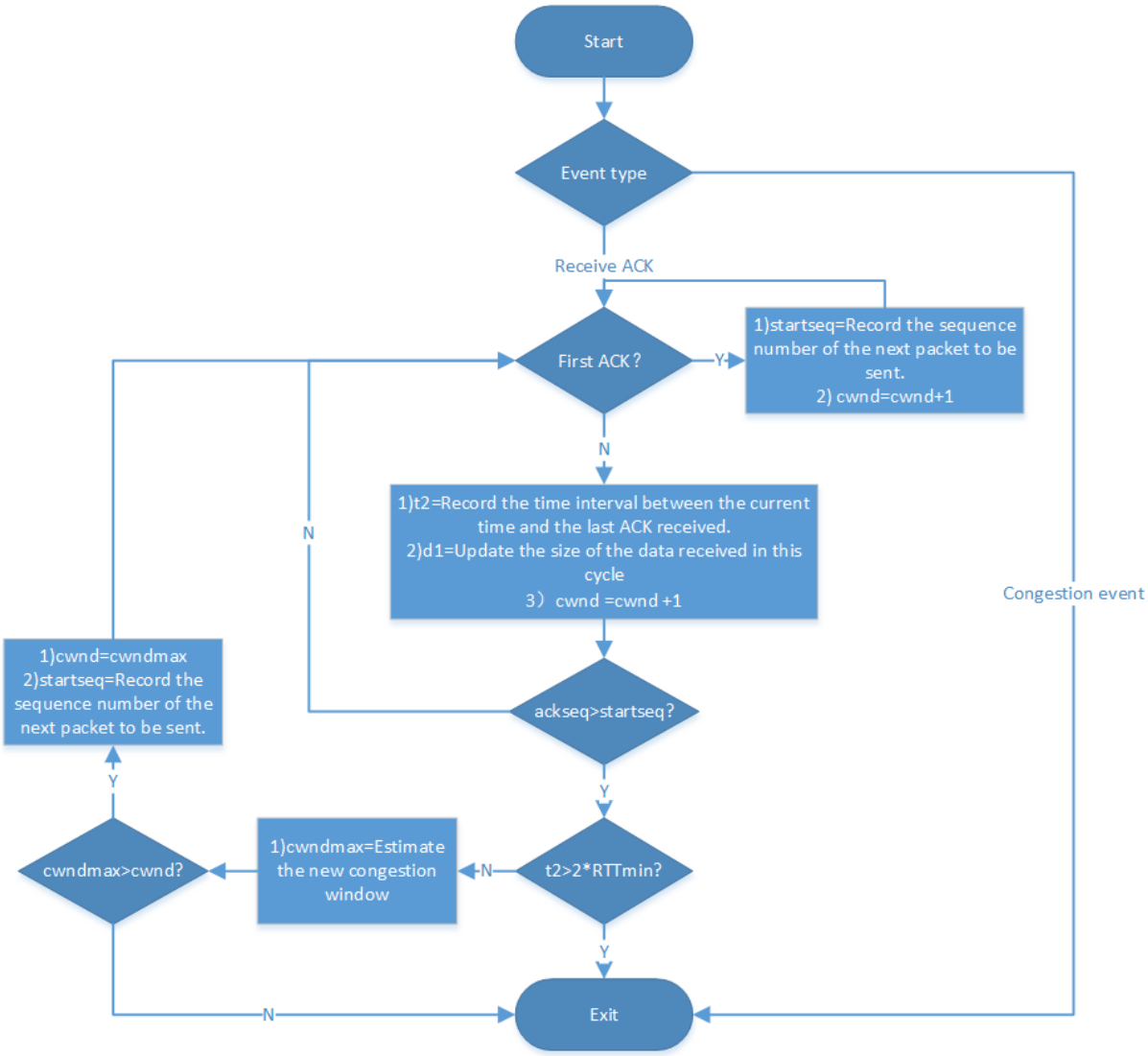

FIGURE 4. Enhanced slow-start algorithm

TABLE 1. The settings of BtlBw and RTprop

Access link delay	Bottleneck link delay	RTprop	BtlBw
1 ms	5 ms	12 ms	10 Mbps
1 ms	10 ms	22 ms	5, 10, 20, 50 Mbps
1 ms	20 ms	42 ms	10, 20 Mbps
1 ms	50 ms	102 ms	10 Mbps

StopEG and NewReno [26] algorithms to evaluate the performance of the five algorithms in the slow-start process.

In addition, we set up multiple networks with different BtlBw and RTprop to perform experiments on the five algorithms. Table 1 shows the settings of BtlBw and RTprop, and each combination represents one of the experiments.

4.1. **Bandwidth utilization.** To compare the performance of ESS with the other four algorithms at the beginning of TCP, we conducted experiments with each of the five algorithms in networks with different network latencies and bottleneck bandwidths.

According to the bottleneck bandwidth of the link $BtlBw$ and the round-trip propagation delay $RTprop$, the optimal congestion window value $cwnd_{best}$ can be calculated. The slow-start algorithm should stop the exponential growth of the window as soon as possible after CWND exceeds $cwnd_{best}$.

$$cwnd_{best} = BtlBw * RTprop \tag{6}$$

Figure 5 shows the performance of several algorithms with a bottleneck bandwidth of 10 Mbps and different link propagation delays. In Figure 5(a), the propagation delay of the link is 12 ms. The window growth rate of NewReno and Cubic is similar to that of ESS, while the window growth rate of BBR and StopEG is slower. According to Formula (6), the optimal congestion window value can be calculated as $1.2 * 10^4$. It can be seen from the figure that when several algorithms end the slow-start process, the congestion window is larger than the optimal congestion window, and the ESS performance the best.

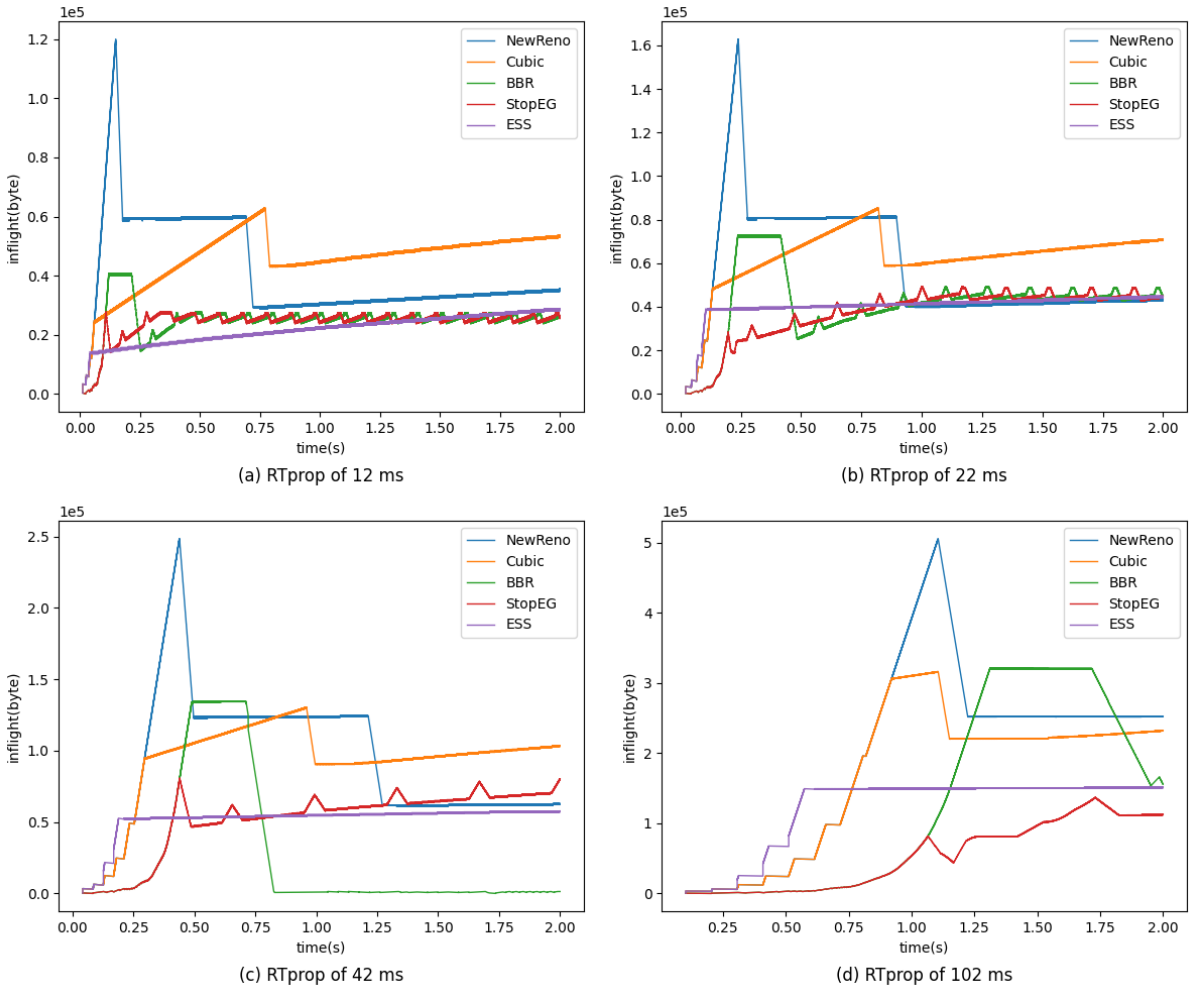

FIGURE 5. (color online) Variation of inflight data with time for BtlBw of 10 Mbps and RTprop of 12, 22, 42 and 102 ms

In Figures 5(b), 5(c) and 5(d), ESS has the highest window growth rate compared to other algorithms, and the greater the round-trip propagation delay, the more obvious the advantage of ESS. According to Formula (6), the optimal congestion windows of Figures 5(b), 5(c), and 5(d) are calculated as $2.75 * 10^4$, $5.25 * 10^4$, $1.275 * 10^5$. It can be seen that ESS is slightly worse than StopEG in Figure 5(b), and is optimal in all other cases.

Figure 6 is similar to the situation in Figure 5. When the bottleneck bandwidth is small, the window growth rate of ESS is similar to that of Cubic and NewReno. When the bottleneck bandwidth is large, it is significantly better than other algorithms. According to Formula (6), the optimal congestion window values of Figures 6(a), 6(b), 6(c) and 6(d) are calculated as $1.375*10^4$, $2.75*10^4$, $5.5*10^4$, $1.375*10^5$. The ESS algorithm is optimal in all cases except that it is slightly worse than StopEG in Figure 6(b).

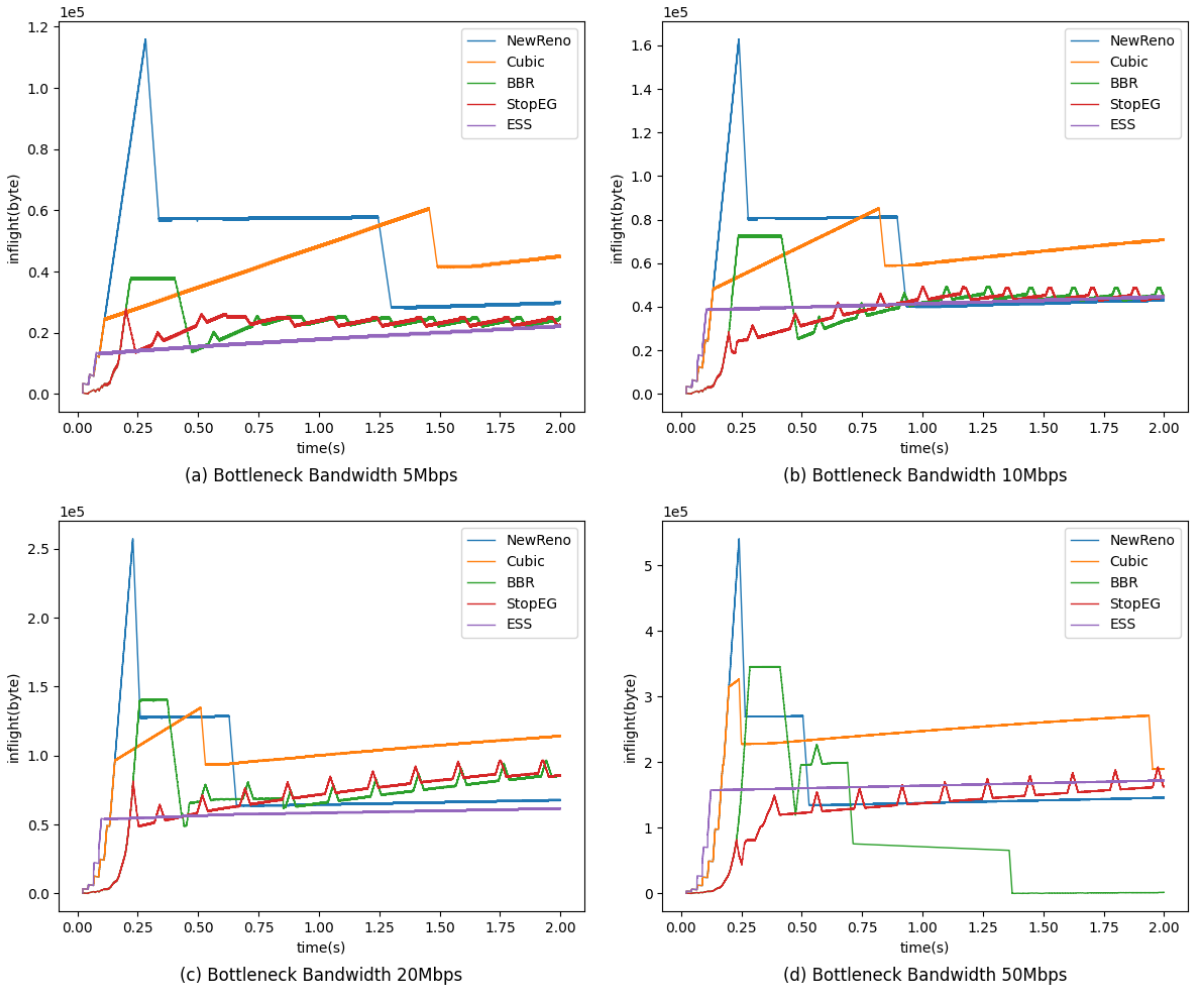

FIGURE 6. (color online) Variation of inflight data with time for BtlBws of 5, 10, 20 and 50 Mbps at RTprop of 22 ms

Figure 5 and Figure 6 show the change of the inflight data of the five algorithms over time in different bottleneck link networks. From the figure we can see that the NewReno algorithm has the highest extreme value, too large extreme value will fill up the bottleneck link buffer or even overflow, which will not only fail to improve the bandwidth utilization, but also reduce the transmission efficiency by losing a large number of packets. The Cubic and BBR algorithms exit the slow-start process earlier than NewReno, and have a relatively small extremum, but still higher than the value of the congestion window when the ESS exits the slow-start. Only StopEG has similar performance to ESS in some network conditions. Both the NewReno and BBR algorithms have a process of window droop after the slow-start. It means that the two congestion control algorithms think that the congestion window has reached an excessively high value at the end of the slow-start, while the StopEG, Cubic and ESS algorithm do not have such a process. In terms of

window growth rate, it can be seen that ESS has a high window growth rate, while the window growth rate of several other algorithms is slower.

Figure 7 shows the variation of successfully transmitted data over time at the beginning of TCP startup, and it can be seen that the ESS algorithm can successfully transmit more data at the same time, while the StopEG, Cubic and NewReno [26] algorithms are slightly worse than ESS, and the BBR algorithm performs the worst, and this gap becomes more and more obvious as the BDP increases.

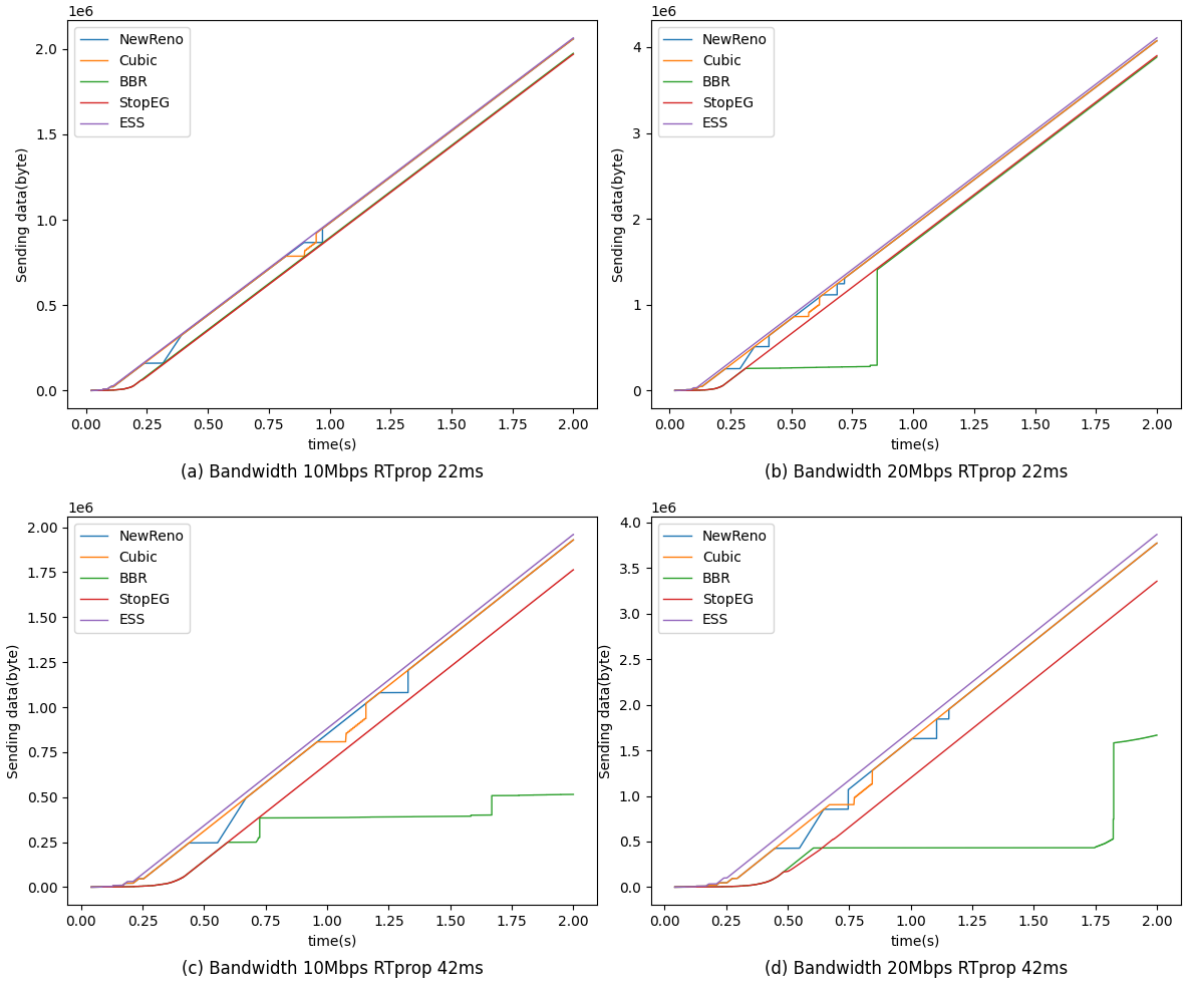

FIGURE 7. (color online) Successfully sent data size over time

According to the above analysis, the ESS algorithm has the best performance compared to the other four algorithms in three aspects: the speed of congestion window growth, slow-start exit time selection and the size of data transferred per unit time, and all the advantages increase with the increase of network bandwidth and RTT.

4.2. **Round-trip time.** During the slow-start process, the TCP sender injects a large number of packets into the network, which are almost inevitably queued in the intermediate switching devices, resulting in a situation where the measured round-trip time is much larger than the RTprop of the link.

Figures 8(a), 8(b), 8(c) and 8(d) have a round-trip propagation delay of 22 ms, 22 ms, 42 ms and 42 ms, respectively. The closer the RTT of the congestion control algorithm is to the path round-trip propagation delay, it means that the congestion control algorithm is more excellent in delay control. Figure 8 shows the variation of round-trip time over

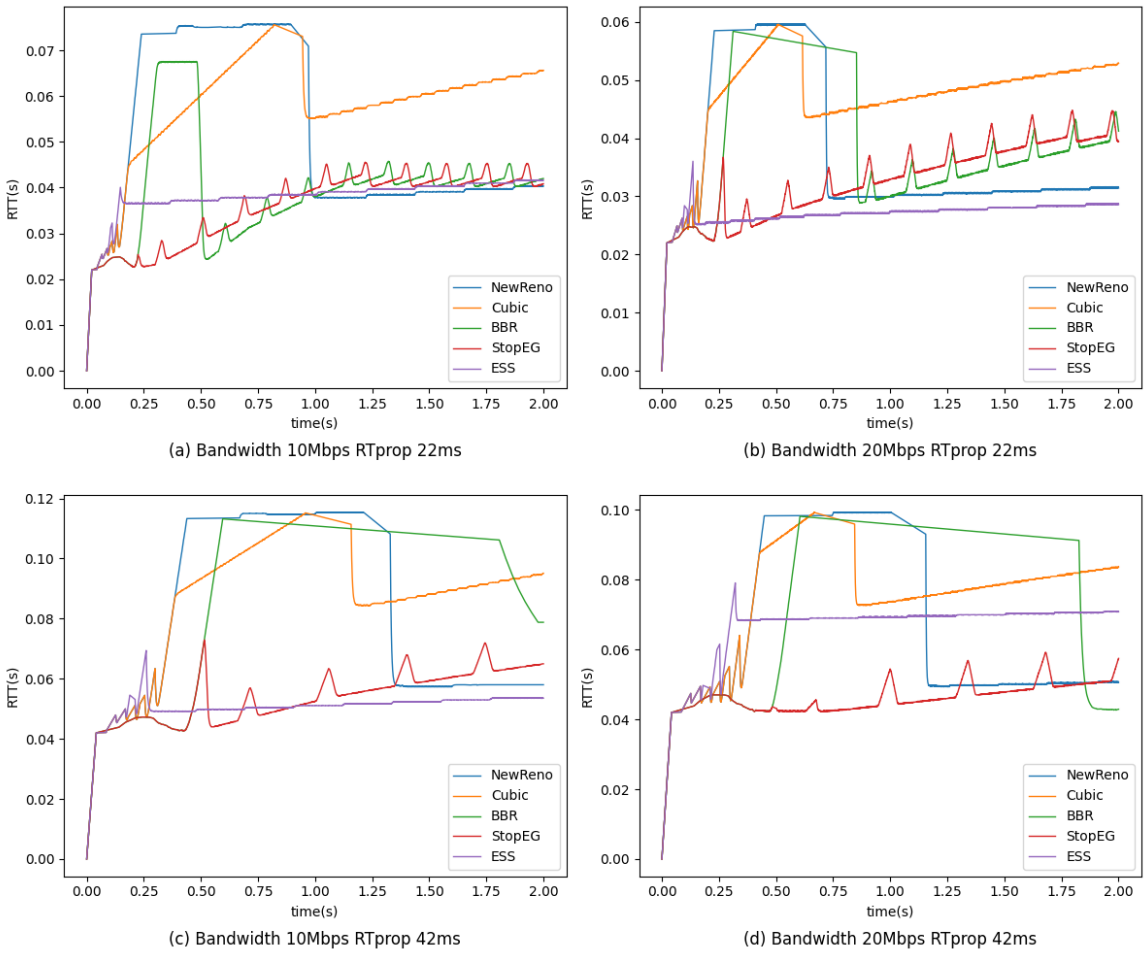

FIGURE 8. (color online) Round-trip time over time

time for the five algorithms in networks with different RTprop and BtlBw. It can be seen that StopEG almost always has a smaller RTT than ESS, and the larger the BDP of the network, the greater the advantage of StopEG. According to Figure 5 and Figure 6, the larger the BDP of the network, the more likely the StopEG algorithm is to exit the slow-start process before reaching a reasonable congestion window. This is the reason why StopEG has better latency performance in high BDP networks instead. The maximum RTT of Cubic, BBR and NewReno is much larger than that of ESS, and their RTTs all last for some time around the maximum RTT. ESS has a smaller maximum RTT, and falls back quickly after reaching the extreme value.

5. **Conclusions.** In this paper, we analyzed the slow growth of the window faced by the classic slow-start algorithm and the large amount of data packet loss that may occur in the end, and then propose the enhanced slow-start algorithm to solve the above problems by studying the law of data transmission in the slow-start process. The enhanced slow-start algorithm accelerates the growth rate of the slow-start window by estimating the optimal congestion window, and by comparing the size of the window with the optimal congestion window and the change in round-trip time, it dynamically determines whether to exit the slow-start process early, rather than simply waiting for a congestion event to occur or the congestion window exceeds the ssthresh.

We use the NS3 platform to compare the enhanced slow-start algorithm with the Cubic (HyStart), BBR, StopEG and the NewReno algorithm for different bottleneck links in

terms of bandwidth and RTprop. Experiments show that the enhanced slow-start algorithm has a good performance in terms of the growth rate of the congestion window and the reduction of packet loss and network delay.

In the future work, wireless networks are characterized by high jitter, and it is challenging to deploy the enhanced slow-start algorithm on wireless networks. Therefore, we will further study the application of the enhanced slow-start algorithm on wireless networks, and it is also important work to implement the Linux kernel module of the enhanced slow-start algorithm and deploy it to carrier networks for testing.

Acknowledgment. We would like to express our gratitude to Jinlin Wang, Yingjie Duan and Jigang Tang for their meaningful support in this work. This work is funded by the Strategic Leadership Project of Chinese Academy of Sciences: SEANET Technology Standardization Research System Development (Project No. XDC02070100).

REFERENCES

[1] M. Polese, F. Chiariotti, E. Bonetto, F. Rigotto, A. Zanella and M. Zorzi, A survey on recent advances in transport layer protocols, *IEEE Communications Surveys & Tutorials*, vol.21, no.4, pp.3584-3608, 2019.

[2] M. Dischinger, A. Haeberlen, K. P. Gummadi and S. Saroiu, Characterizing residential broadband networks, *Proc. of the 7th ACM SIGCOMM Conference on Internet Measurement*, pp.43-56, 2007.

[3] M. Allman, V. Paxson and W. Stevens, *TCP Congestion Control*, DOI: 10.17487/RFC5681, 1999.

[4] K. Nichols and V. Jacobson, Controlling queue delay, *Communications of the ACM*, vol.55, no.7, pp.42-50, 2012.

[5] S. Ha and I. Rhee, Taming the elephants: New TCP slow start, *Computer Networks*, vol.55, no.9, pp.2092-2110, 2011.

[6] Y. Zhang, N. Ansari, M. Wu and H. Yu, AFStart: An adaptive fast TCP slow start for wide area networks, *2012 IEEE International Conference on Communications (ICC)*, pp.1260-1264, 2012.

[7] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh and V. Jacobson, BBR: Congestion-based congestion control, *Communications of the ACM*, vol.60, no.2, pp.58-66, 2017.

[8] N. Cardwell, Y. Cheng, S. H. Yeganeh and V. Jacobson, *BBR Congestion Control*, https://www.ietf.org/proceedings/97/slides/slides-97-iccrg-bbr-congestion-control-00.pdf, 2017.

[9] D. Cavendish, K. Kumazoe, M. Tsuru, Y. Oie and M. Gerla, CapStart: An adaptive TCP slow start for high speed networks, *The 1st International Conference on Evolving Internet*, pp.15-20, 2019.

[10] M. Arghavani, H. Zhang, D. Eyers and A. Arghavani, StopEG: Detecting when to stop exponential growth in TCP slow-start, *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, pp.77-87, 2020.

[11] L. Guo and J. Y. B. Lee, Stateful-TCP – A new approach to accelerate TCP slow-start, *IEEE Access*, vol.8, pp.195955-195970, 2020.

[12] F. Abdel-Fattah, K. A. Farhan, F. H. Al-Tarawneh and A. M. Al-Naimat, Dynamic intrusion detection technique for dynamic mobile ad hoc network, *ICIC Express Letters, Part B: Applications*, vol.10, no.9, pp.813-821, 2019.

[13] S. Hauger, M. Scharf, J. Kogel and C. Suriyajan, Quick-start and XCP on a network processor: Implementation issues and performance evaluation, *2008 International Conference on High Performance Switching and Routing*, pp.241-246, 2008.

[14] D. Liu, M. Allman, S. Jin and L. Wang, Congestion control without a startup phase, *Proc. of PFLDnet*, 2007.

[15] X. Nie, Y. Zhao, G. Chen, K. Sui, Y. Chen, D. Pei, M. Zhang and J. Zhang, TCP wise: One initial congestion window is not enough, *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*, pp.1-8, 2017.

[16] Q. Li, M. Dong and P. B. Godfrey, Halfback: Running short flows quickly and safely, *Proc. of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, pp.1-13, 2015.

[17] T. Tsugawa, G. Hasegawa and M. Murata, Implementation and evaluation of an inline network measurement algorithm and its application to TCP-based service, *2006 4th IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services*, pp.34-41, 2006.

[18] L. Wang, Analysis of TCP phases and simulation of TCP flavours, *Journal of Network New Media*, 2016.

[19] L. Kleinrock, Power and deterministic rules of thumb for probabilistic problems in computer communications, *Proceedings of the International Conference on Communications*, vol.43, pp.1-43, 1979.
[20] V. Jacobson, Congestion avoidance and control, *ACM SIGCOMM Computer Communication Review*, vol.18, no.4, pp.314-329, 1988.
[21] S. Ha, I. Rhee and L. Xu, Cubic: A new TCP-friendly high-speed TCP variant, *ACM SIGOPS Operating Systems Review*, vol.42, no.5, pp.64-74, 2008.
[22] P. Balasubramanian, Y. Huang and M. Olson, HyStart++: Modified slow start for TCP, *Internet-Draft draft-balasubramanian-tcpmhystartplusplus-03, Internet Engineering Task Force*, 2020.
[23] K. Ramakrishnan, S. Floyd, D. Black et al., *The Addition of Explicit Congestion Notification (ECN) to IP*, http://www.rfc-editor.org/rfc/ rfc3168.txt, 2001.
[24] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, V. Vasiliev, P. Jha, Y. Seung, M. Mathis and V. Jacobson, BBRv2: A model-based congestion control, *Presentation in ICCRG at IETF 104th Meeting*, 2019.
[25] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain and N. Sutin, An argument for increasing TCP's initial congestion window, *ACM SIGCOMM Computer Communication Review*, vol.40, no.3, pp.26-33, 2010.
[26] S. Floyd, T. Henderson and A. Gurtov, *The NewReno Modification to TCP's Fast Recovery Algorithm*, http://www.nic.funet.fi/rfc/rfc6582.txt.pdf, 1999.

Author Biography

Zhiyuan Wang received the BSc degree in measurement and control technology and instrumentation from the China University of Petroleum, Beijing, in 2018.

Dr. Zhiyuan Wang is currently a Ph.D. candidate with the Chinese Academy of Sciences. His current research interest is new network communication and information centric networking (ICN).

Hong Ni received the M.S. degree from the Institute of Acoustics, Chinese Academy of Sciences, in 1989.

Prof. Ni is currently a Researcher, a Doctoral Tutor, and the Deputy Director. His research interests include broadband multimedia communication, network newmedia technology and application, embedded systems, and middleware technology.

Xiaoyong Zhu received the Ph.D. degree in signal and information processing from the University of Chinese Academy of Sciences, in 2009.

Prof. Zhu is currently a Professor with the Chinese Academy of Sciences. His current research interests include embedded systems, virtualization technology, information centric networking (ICN) and new network communication.

Xu Wang received the B.S. degree in automation from Tsinghua University, in 2012, and received the Ph.D. degree in signal and information processing from the University of Chinese Academy of Sciences, in 2018.

Dr. Xu Wang is currently a research assistant with the Chinese Academy of Sciences. His current research interests include information centric networking (ICN), and new network communication.