Hindawi
Journal of Computer Networks and Communications
Volume 2019, Article ID 2187543, 12 pages
https://doi.org/10.1155/2019/2187543

Research Article

Early Window Tailoring: A New Approach to Increase the Number of TCP Connections Served

Marcos Talau, Mauro Fonseca, and Emilio C. G. Wille

Graduate Program, Electrical and Computer Engineering (CPGEI), Federal University of Technology—Paraná (UTFPR), 80230-901 Curitiba, Paraná, Brazil

Correspondence should be addressed to Marcos Talau; mtalau@utfpr.edu.br

Received 23 April 2019; Revised 7 September 2019; Accepted 16 September 2019; Published 10 October 2019

Academic Editor: Roberto Nardone

In the absence of losses, TCP constantly increases the amount of data sent per instant of time. This behavior leads to problems that affect its performance, especially when multiple devices share the same gateway. Several studies have been done to mitigate such problems, but many of them require TCP side changes or a meticulous configuration. Some studies have shown promise, such as the use of gateway techniques to change the receiver's advertised window of ACK segments based on the amount of memory in the gateway; in this work, we use the term "network-return" to refer to these techniques. In this paper, we present a new network-return technique called early window tailoring (EWT). For its use, it does not require any modification in the TCP implementations at the sides and does not require that all routers in the path use the same congestion control mechanism, and the use in the gateway is sufficient. With the use of the simulator ns-3 and following the recommendations of RFC 7928, the new approach was tested in multiple scenarios. The EWT was compared to drop-tail, RED, ARED, and the two network-return techniques—explicit window adaptation (EWA) and active window management (AWM). In the results, it was observed that EWT was shown to be efficient in congestion control. Its use avoided losses of segments, bringing expressive gains in the transfer latency and goodput and maintaining fairness between the flows. However, unlike other approaches, the most prominent feature of EWT is its ability to maintain a very high number of active flows at a given level of segment loss rate. The EWT allowed the existence of a number of flows, which is on average 49.3% better than its best competitor and 75.8% better when no AQM scheme was used.

1. Introduction

Through inference, the TCP protocol makes use of network information in its congestion control [1]. Segments are sent with an increasing and exponential rate with the use of the algorithm *slow-start*. When the transmission window exceeds a predefined level (*ssthresh*), TCP enters the congestion-avoidance mode and the number of sent segments is then increased linearly. This process is done until losses occur. In this case, TCP reduces the size of the data burst and thus sends fewer segments. Losses are detected with the use of timeouts and can also be predicted by receiving a number of duplicate ACKs. In the first case, the size of the burst is reduced to one segment. When a sequence (usually three) of duplicate ACKs is received and the timestamp of the corresponding segment has not been expired, the TCP protocol uses the fast-retransmit/fast-recovery mechanisms, thus halving the burst size. If out-of-order segments arrive, TCP also sends duplicate ACKs, which can lead to a size reduction of burst. In addition, TCP only confirms the last segment received. If multiple segment losses occur (in the burst), it will enter the slow-start phase, thus reducing its transmission. In short, TCP seeks to cause losses to infer the capacity of the network.

This behavior leads to problems that affect the TCP performance, especially when multiple devices share a same gateway. Problems occur because of the constant dispute over gateway resources, which leads to degradation of throughput, fairness, and delays. In this context, to improve TCP performance, several proposals have been developed that can be grouped into active queue management (AQM) [2–4], TCP side changes [5–7], AQM + TCP [8–11], and network-return techniques [12–16].

The use of active queue management is intended to avoid the occurrence of congestion through drop/mark (explicit congestion notification (ECN)) of segments before filling the gateway queue. Its use may be inefficient if TCP sides do not use markup properly and also if they are not perfectly configured for the desired scenario. However, the side changes necessitate a change of TCP in the sides, which makes their use impracticable. AQM + TCP makes use of AQM with TCP side changes, so it goes through the same problems. There are approaches performed at gateways that change the TCP receiver's advertised window of ACK segments based on the amount of memory in the gateway; in this paper, we use the term "network-return" to refer to these techniques. Using network-return techniques is expected to make the TCP adjust its transmission based on information about the usage of the network.

Another problem, not specific to TCP, is bufferbloat [17]. Bufferbloat is an unwanted latency that can occur when network equipment (typically a router or switch) has a high memory capacity. When using large memory, segments are lost less often, but high memory usage generates latency, which affects system performance. Network-return techniques, while being used in the context of AQM, are not intended to address bufferbloat but rather to control the transmission rate of TCP flows based on available gateway memory; but if adapted, they can easily be used. Also in this sense, TCP congestion control on the Internet operates largely at endpoints, and congestion control algorithms are used on clients and servers. On Windows systems, the use of compound TCP [18, 19] predominates, and on Linux systems, CUBIC TCP predominates [20, 21]. Network-return techniques are independent of the congestion control algorithm used (this is discussed at the end of Section 4), as they act directly on the receiver's advertised window, which is respected by any TCP implementation.

Several authors [2, 13, 22–25] reported that the most effective detection of congestion can occur in the gateway. It is in this context that the network-return techniques act. The use of these techniques has the following advantages: it does not require any modification in the TCP implementation at the sides and does not have to be present in routers in the end-to-end path, and its installation in the gateway is sufficient. The use of network-return techniques has been shown to be promising in the control of congestion, as verified in several papers [12–16, 26–28]. However, a large part of the published studies was limited to a few experiments. Another negative point is the complexity of the methods, where many of them need to know the number of active flows. In order to fill these gaps, early window tailoring (EWT) was developed.

The EWT is based on early congestion control (ECC) [27] and acts on gateways by changing the value contained in the receiver's advertised window (W_r) of the TCP ACK segments. The change is made in proportion to the memory space available in the gateway. Like other network-return techniques, the EWT does not require any modification in the TCP implementations at the sides, and for its operation, only the gateway installation is sufficient.

The contributions of this paper are as follows:

(i) A new network-return approach, called EWT, is proposed and evaluated

(ii) Simulations are done, with the use of ns-3 [29], by applying the EWT in multiple scenarios, following the recommendations of RFC 7928 [30]

(iii) The EWT is compared with the main network-return techniques, such as the explicit window adaptation (EWA) [12] and active window management (AWM) [13]

The remainder of this paper is organized as follows: In Section 2, some TCP problems are described and illustrated in an internetwork environment. Section 3 presents the main network-return techniques in the literature. The EWT is presented and detailed in Section 4. In Section 5, the EWT is evaluated by computer simulations. Finally, conclusions and future work are drawn in Section 6.

2. Motivation

Disregarding the problem of misleading reduction, where losses in the physical layer cause TCP to reduce its transmission rate improperly, there are problems that occur in an internetwork environment.

When TCP is used in an internetwork environment, it is necessary to have memories in the routers to enable the routing of multiple segments. The memory of these is used in the form of a queue, traditionally managed by a technique called drop-tail or tail drop. In order to perform the input and output of segments, drop-tail follows the first-in-first-out (FIFO) model, and segments are discarded when the queue reaches its maximum size [31, 32].

The drop-tail method presents the following problems: (1) When the queue is full, multiple losses can occur in one flow or losses in different flows at the same time. In the first case, TCP will enter the slow-start phase and its transmission rate will be reduced to one segment. In the second case, a loss synchronization can occur, where multiple connections reduce their transmission rate. (2) One or more connections may monopolize the queue space [32].

In order to illustrate some TCP problems in an internetwork environment, a dumbbell topology with tree nodes on each side was used, where sources S [1–3] transmit data simultaneously to receivers D [1–3] in a way to use the memory of router $R1$ to its limit.

Figure 1 shows the congestion windows (W_c) of the three connections. It is observed that all three have a similar initial behavior up to the moment when the memory of router $R1$ becomes full, causing segment losses in the three connections; with a loss sequence, the connections can enter the slow-start phase and set the congestion window to the size of one segment. Then, another behavior is observed: the S3D3 connection monopolizes the use of the router's memory since its value increases continuously, while the other two

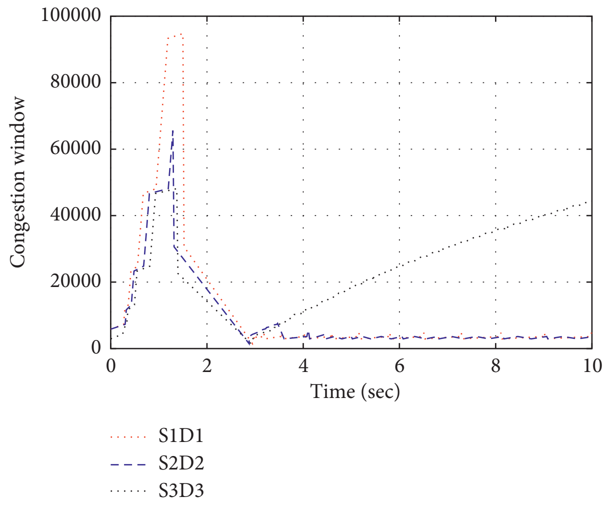

Figure 1: TCP congestion window obtained in simulation of motivation.

connections are synchronized between the sending of few segments and the occurrence of losses.

As illustrated, resource contention causes TCP to present problems in the internetwork environment. Techniques of network-return have already been proved effective in mitigating such problems. In Section 3, the main network-return techniques present in the literature are reviewed.

3. Related Works

The network-return techniques are characterized by inserting/changing values in the header of the IP/TCP segments in order to make TCP clients adjust their transmission based on network information. To make this paper as self-contained as possible, we now present succinct survey works that have addressed network-return techniques.

3.1. Explicit Window Adaptation (EWA). EWA [12] is designed to work on ATM networks. The method acts on gateways by monitoring the usage of their memory. In the presence of segments in memory, the EWA reduces the value contained in the receiver's advertised window (W_r) of the TCP ACK segment header. The change of the value of W_r is made using the following equations:

$$\begin{aligned} B_e(t) &= B - Q(t), \\ W_r'(t) &= \min\left(W_r(t), \max\left(f\left(B_e(t)\right), \mathrm{MSS}\right)\right), \end{aligned} \tag{1}$$

where B is the total amount of memory, $Q(t)$ is the occupation of memory at time t, $B_e(t)$ indicates the amount of memory available at time t, MSS is the maximum size of a segment, and $f(B_e(t))$ is a function defined by

$$f\left(B_e(t)\right) = \alpha \log_2 B_e(t). \tag{2}$$

The parameter α is dynamically updated based on the average memory size ($\overline{Q}_t$):

$$\overline{Q}_t = (1-g)\overline{Q}(t-1) + gQ(t), \tag{3}$$

where $\overline{Q}(t-1)$ is the last average memory size and g is set to 1/128. Two marks (*threshold*) are defined around $\overline{Q}(t)$. If the average memory size is less than the lower mark (T_l), then α is incremented by an amount W_{up} in every Tseconds; if the average occupation is greater than the upper mark (T_h), then α is reduced by W_{down}:

$$\alpha = \begin{cases} \alpha + W_{\mathrm{up}}, & \text{if } \overline{Q}(t) < T_l, \\ \alpha - W_{\mathrm{down}}, & \text{if } \overline{Q}(t) > T_h, \end{cases} \tag{4}$$

where W_{up}, W_{down}, and T are user-set parameters.

Results in [12] indicated that EWA was effective in controlling the use of the gateway memory obtaining an improvement over random early detection (RED) [2] in reducing losses, fairness, and throughput.

3.2. Smart Access Point with Limited Advertised Window (SAP-LAW). This method is designed to work with UDP and TCP traffic [14]. Just like others, it changes the value contained in the receiver's advertised window of ACK segments. The change is made according to the following equation:

$$\max \mathrm{TCPrate}(t) = \frac{(C - \mathrm{UDPtraffic}(t))}{\#\mathrm{TCPflows}(t)}, \tag{5}$$

where C is the capacity of the bottleneck link, UDPtraffic(t) represents the total UDP traffic at time t, and #TCPflows(t) indicates the number of active TCP flows at time t. After calculating max TCPrate(t), its value is entered in W_r of the header of the ACK segments. Although the equation is simple, the method is costly to implement since it requires every moment (t) to compute the total UDP traffic as well as the number of TCP flows. Simulations were done in [14] where the performance of SAP-LAW and TCP Vegas was compared on a wireless network. The results indicated that the methods had similar results, where both improved network performance. Recently, in [26], SAP-LAW was compared to RED and ECN. In the results, obtained in simulations with constant TCP traffic in the presence of low UDP traffic, SAP-LAW brought an improvement in throughput.

3.3. Active Window Management (AWM). AWM [13] also acts on the gateway and changes the value of the receiver's advertised window of the ACKs. The change is made only if the window value is larger than swnd (suggested window), when it is, the value is set to swndswnd.

The swnd variable is updated when a segment enters or leaves the gateway memory. The update is done using the following equation:

$$\mathrm{swnd}_t = \max\left(\mathrm{swnd}_{t-1} + \mathrm{DQ}_t + \mathrm{DT}_t, \mathrm{MSS}\right), \tag{6}$$

where t is the actual time instant and MSS is the maximum segment size. The term DQ_t is defined by

$$\mathrm{DQ}_t = \frac{1}{N}\left(q_{t-1} - q_t\right), \tag{7}$$

where N is the estimated number of flows and q_t is the amount of memory in use. Finally, DT_t is defined by

$$\mathrm{DT}_t = \alpha(\text{target} - q_t), \quad (8)$$

where α and target are user-defined parameters.

Results in [13], comparing AWM with drop-tail and RED, indicated constant memory usage with few variations and a reduction in the number of losses.

3.4. Proactive INjection into acK (PINK). PINK [15, 16] is another method that acts on gateways by changing the value of W_r of ACK segments. To operate, the method requires the number of active flows, the RTT of each one, and the transmission channel bandwidth. Because of this, it is the network-return technique that most needs information to operate, and some of them are expensive to maintain (like the number of active flows and the RTT of each). For this reason, the method was not considered in the simulations performed in this work.

4. Early Window Tailoring (EWT)

The EWT brings to TCP an additional control in its transmission by forwarding gateway information that will be used in the TCP transmission equation. This information is indirectly used by TCP because it is in your window field which is read by the TCP transmitter, and the EWT installation at the gateway is sufficient to provide the additional control. This control is done by updating the value contained in the receiver's advertised window (W_r) of TCP ACK segments based on the number of bytes available in the gateway's memory.

The receiver's window will be updated by EWT if its new value is larger than the maximum segment size (MSS); otherwise, the value will be adjusted to one MSS. During the processing of the ACKs, it is not considered which flow they belong to; in this way, all are treated in the same way and receive the same update (proportional to the size of its window field). By simply changing the value of the receiver's advertised window, the EWT is compatible with the default TCP implementation and does not require protocol changes because the receiver's window is naturally used by the transmitter to limit its transmission. Figure 2 illustrates the main parameters of the EWT.

The return provided by the EWT is based on the number of available bytes (B_a) in the gateway memory. B_a is defined by

$$B_\mathrm{a} = B - U, \quad (9)$$

where B is the total amount of memory and U indicates the current occupancy level of the memory. If flows meet this amount, there are hardly any losses.

New flows tend to have their congestion window low, so the return of the EWT would not have an immediate effect; to mitigate this and to save resources of the gateway when the memory usage is low, the parameter S_t is used to define the start-up point of the method. With this, EWT will only work when the memory usage is greater than S_t. When this

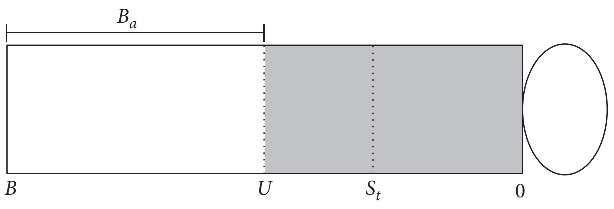

Figure 2: Key parameters used by the EWT shown in a memory size of B.

occurs, the EWT changes the window value of the segments proportionally to B_a. Considering W_ewt as the return value of the EWT, we can write

$$W_\mathrm{ewt} = \frac{B_\mathrm{a}}{B} \cdot W_\mathrm{r}, \qquad W'_\mathrm{r} = \max(W_\mathrm{ewt}, \mathrm{MSS}), \quad (10)$$

where W_r is the value of the receiver's advertised window and W'_r is the new value of W_r calculated by EWT.

Using these equations, the EWT will reduce the W_r proportionally to the available space in the gateway memory; that is, the value of windows decreases as memory usage increases. We design the EWT in this way to control the rate of all TCP flows based on the memory usage of the gateway; with this, we try to avoid losses due to memory full while maintaining fairness between the flows.

The most appropriate place to use EWT is at the gateway (but it can also be used at the routers), where the contention for resources is greater. In order to use the additional control provided by the EWT, no changes are necessary for the TCP protocol, so it can be used with any TCP like congestion control algorithm. In TCP, the equation $\min(W_\mathrm{c}, W_\mathrm{r})$ is respected, so the EWT becomes effective when the congestion window (W_c) is greater than W_r. Related to this, three main cases can occur: (1) *no congestion*: W_c is high and the memory occupancy reaches a certain level. The EWT will reduce W_r, and with this, the flow will reduce its rate of transmission because W_c will probably be larger than W_r; (2) *congestion*: the TCP transmits data using W_c, but if the congestion is high, the W_r calculated by EWT may be smaller and the data burst will be reduced; and (3) *congestion recovery*: W_c is used in the transmission, but W_r can also be used if the memory has a considerable occupancy level. With the use of the EWT, it is expected to avoid the occurrence of congestion in the gateway; for this reason, cases 2 and 3 tend to occur only by traffic bursts or losses in the physical layer.

4.1. Pseudocode. The EWT operates on gateways bringing a return of memory state to TCP. Gateways make use of AQM methods for memory management. Such methods are modular; that is, a router can support a number of methods, but the router administrator chooses which will be used. Considering this, the EWT was implemented in the form of AQM.

The implementation of the EWT as AQM has the basis of drop-tail operation, inserting and removing segments of a queue using the FIFO policy, and rejecting them when the queue becomes full.

Algorithm 1 shows the EWT implementation as AQM.

The EWT goes into operation after the queue segment exits. In line 2, we obtain the segment that is in the first position of the queue, after which it is passed as a parameter to the function ewt. In the function, it is checked whether the segment is of TCP ACK type; otherwise, the function will terminate. It is then checked if the memory usage has exceeded the start point of the EWT; if the level S_t has not been reached, the function is terminated. After this, the value of the receiver window (W_r) and the size of the gateway memory and its use are obtained. Next, the amount of available memory (B_a) is calculated based on B and the EWT window (W_{ewt}) is calculated. Finally, the segment window is changed to the maximum value between W_{ewt} and MSS.

4.2. Example of Operation. For a better understanding of the operation of the EWT, we present some numerical examples. Considering the values $B = 100\,\text{kB}$, $S_t = 30\,\text{kB}$, and $W_r = 60\,\text{kB}$, we show in Table 1 the values of W_{ewt} in function of the occupation level U.

The first row shows the case where 20% of the queue is being used; in this case, $U < S_t$, and the windows will not change. It may be considered that, in this case, $W_{ewt} = W_r = 60\,\text{kB}$.

The second row considers average queue usage. As $U > S_t$, EWT will operate, producing $W_{ewt} = 30\,\text{kB}$. The last example case corresponds to when there is a high use of the queue, so the EWT forces a bigger reduction in the windows because $W_{ewt} = 12\,\text{kB}$.

5. Evaluation Setup

This section presents the evaluation of EWT using the ns-3 software package. Simulation results in multiple scenarios are analyzed.

5.1. ns-3 Simulation Environment. To test and evaluate EWT, ns-3 [29] was used. For the EWT operation, its implementation was done together with the module *Traffic-control* [33]. In this module, a new class was created by inheritance of class *QueueDisc*. In the new class, the structure of the drop-tail technique was followed, and in the output of the segment of the queue, it was passed to the EWT.

5.2. EWT and Queue Management. First, we present a simulation to show the ability of EWT to control the use of the queue in a gateway. In Figure 3, the use of a gateway memory of a bottleneck with four TCP flows is shown. Without AQM (using drop-tail), a significant variation of values is observed. When the EWT adjustment policy is used, the flows transmit data with a smaller variation.

It can be observed that when drop-tail is used, there is a high variation in the memory usage of the gateway, and that with the application of the EWT, the memory usage becomes constant and unchanged.

5.3. Testbed Environment. To test the method, a topology widely used in the literature was adopted: the dumbbell. This topology has already been recommended in [30, 34, 35]. It consists of $n \times n$ *hosts* and two routers ($R1$ and $R2$). In this case, n hosts are directly connected to the router $R1$, and the other n hosts are connected to the router $R2$. Finally, $R1$ is connected to $R2$, forming a bottleneck. Figure 4 illustrates this topology. The characteristics of the links depend on the simulation scenarios.

In the executed simulations, the leftmost *hosts* of $R1$ have been configured to be TCP sources, having as receptors the *hosts* to the right of $R2$. At the beginning of the simulations, each source, in random time (from one to eight seconds), establishes a connection with the respective receiver and transfers a 5 Mb file. Simulations end when all sources complete the transmission of their files.

During the simulations, the number of sources/receivers is equal to the number of flows; that is, each source/receiver creates a single flow. To define the ideal number of flows to be used in the experiments, the RFC 7928 standard [30] was used, which defines three levels of congestion: mild, medium, and heavy. The mild level is characterized by a loss (of segments) of about 0.1%, medium by 0.5%, and heavy by 1%. To find these levels, preliminary simulations were made for each scenario used. In this case, the drop-tail method was used with i (≥ 1) flows, and it was verified at the end of the simulation if the percentage of losses fell into any of the three categories. Until not finding an i number, for each category, i was incremented by one unit and the process was repeated. At the end of this process, there were three values for i, one for each level of congestion. For our simulations, the number of flows for each scenario is presented in Table 2.

For performance evaluation, the following metrics were calculated, some of which are recommended by RFC 7928:

(i) TCP efficiency: this metric was extracted from the study in [36] and represents the percentage of data (in bytes) that were not retransmitted. It is defined by

$$\frac{\text{transmitted bytes} - \text{retransmitted bytes}}{\text{transmitted bytes}} \times 100. \quad (11)$$

In the presentation of the results, the average value of the calculated metric from the efficiency presented by each flow was displayed.

(ii) File transfer latency: this indicates how many seconds are spent for a file to be transmitted successfully. The results show the average of this time calculated from the times presented by the flow.

(iii) Goodput: this is defined as the amount of data received by the application, over a period. The results show the average value of the goodput (per second) of the flows in Mbits/sec. This metric does not count incoming segments already received (duplicates), unlike *throughput*.

(iv) Goodput fairness: Jain et al.'s index [37] was applied to the goodput (gp) obtained by the n flows of the simulation, in order to verify the fairness between the flows. The equation used for this is

```
procedure DEQUEUE
    seg ⟵ queue.get_first()
    ewt(seg)
    return seg
end procedure
procedure EWT(seg)
    if segment_type != TCP_ACK then
        return
    end if
    U ⟵ queue.use
    if $S_t < U$ then
        return
    end if
    $W_r$ ⟵ seg.window
    B ⟵ queue.size
    $B_a$ ⟵ B − U
    $W_{ewt}$ ⟵ $B_a$/B ∗ $W_r$
    seg.window ⟵ max ($W_{ewt}$, MSS)
end procedure
```

Algorithm 1: EWT implementation as AQM.

Table 1: W_{ewt} in function of the occupation level U.

U (kB)	B_a (kB)	W_{ewt} (kB)
20	80	60
50	50	30
80	20	12

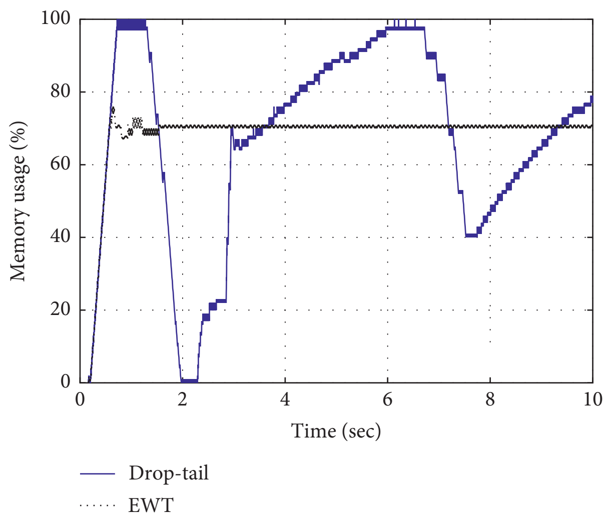

Figure 3: Memory usage of the gateway with drop-tail and EWT.

$$\frac{\left(\sum_{n=1}^{n} \text{gp}\right)^2}{n \cdot \sum_{n=1}^{n} \text{gp}^2}. \tag{12}$$

(v) Mean loss ratio: this value represents the average percentage of segment loss during the simulation. It is calculated using the following equation:

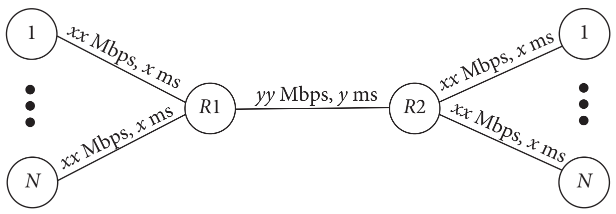

Figure 4: Dumbbell topology.

Table 2: Number of flows used in the simulation for each level of congestion.

	Mild	Medium	Heavy
Scenario 1	8	17	24
Scenario 2	13	24	31

$$\frac{\text{total lost segments}}{\text{total segments received} + \text{total lost segments}} \times 100. \tag{13}$$

In order to obtain the metrics, the *FlowMonitor* [38] method is used, present in the simulator ns-3. In the evaluation of EWT performance, simulations were performed and the results were compared to those of drop-tail, RED, adaptive RED (ARED), EWA, and AWM approaches. For each of the methods, 30 rounds with different seeds of random numbers were executed. A 95% confidence interval was considered. The parameters of RED were adjusted according to the standard found in the Linux *kernel*, and EWA and AWM parameters were adjusted according to the authors' recommendation in [12, 13]. The other simulation configurations considered important are presented in Table 3.

The results presented below make use of the settings described here, varying the RTT and the capacities of the dumbbell topology links (Figure 4).

5.4. Scenario 1. This scenario is used in the works [15, 16]. This has an RTT value in the bottleneck characteristic of distant networks or satellite communications. The settings used in the topology are shown in Table 4.

5.4.1. Results

(1) TCP Efficiency. The results are presented in Figure 5. Drop-tail, RED, and ARED methods have seen a gradual drop in efficiency. The EWA method showed a drop in efficiency from the mild level. The EWT has remained efficient regardless of the level of congestion.

(2) File Transfer Latency. Figure 6 displays the results. At the mild congestion level, the EWT presented a transfer latency lower than drop-tail, RED, ARED, and EWA. With medium congestion, the methods had an increase in transfer latency,

Table 3: Simulation parameters.

Parameter	Value
Memory size	97 kB
min_{th} (RED)	memory_size/12
max_{th} (RED)	memory_size/4
Q_w (RED)	0.002
max_p (RED)	0.02
S_t (EWT)	29 kB
TCP segment size	1458 bytes
TCP	CUBIC

Table 4: Parameters of Scenario 1.

Parameter	Value
Link capacity (sources)	4 Mb
RTT (sources)	5 ms
Link capacity ($R1$-$R2$)	4 Mb
RTT ($R1$-$R2$)	360 ms
Link capacity (receptors)	4 Mb
RTT (receptors)	5 ms

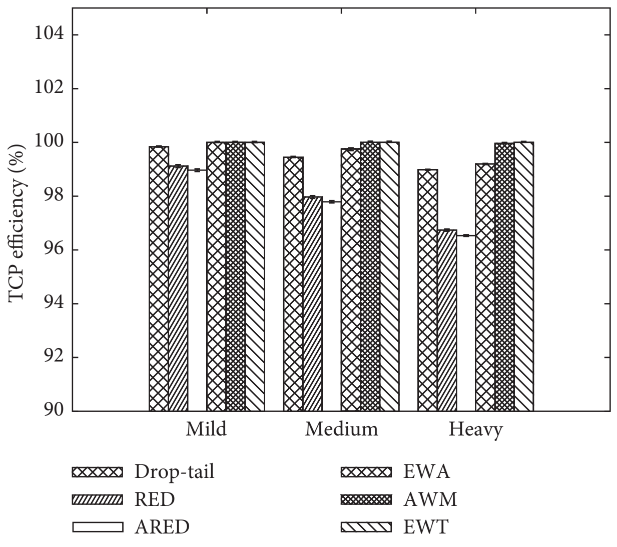

Figure 5: TCP efficiency: Scenario 1.

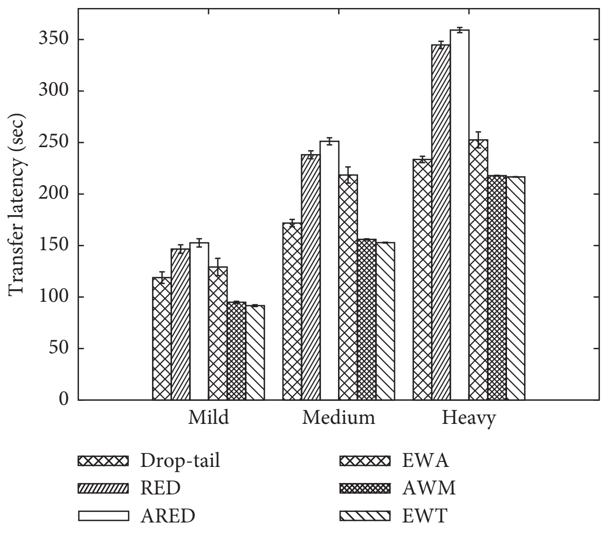

Figure 6: File transfer latency: Scenario 1.

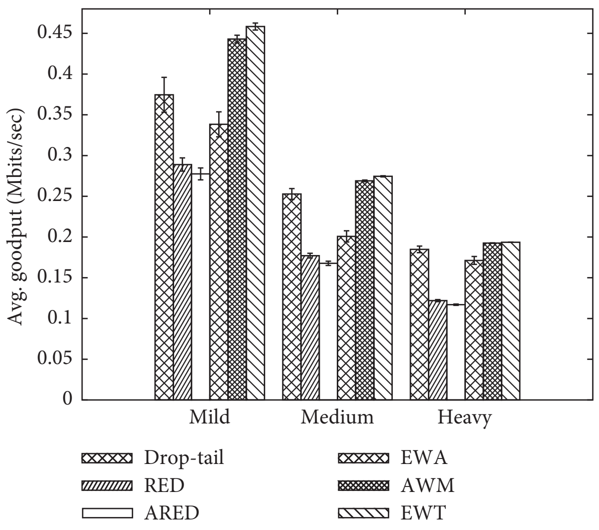

Figure 7: Average goodput: Scenario 1.

and the EWT registered the shortest time. Also with heavy traffic, the EWT presented the shortest time. EWT and AWM presented similar results.

(3) Goodput. The results are shown in Figure 7. In the presence of mild traffic, the EWT had a goodput higher than drop-tail, RED, ARED, and EWA. With medium traffic, with more flows, there was a drop in the average goodput, but EWT maintained its performance over other methods. Finally, with heavy traffic, the EWT continued to register the highest goodput. The AWM followed the EWT in the results.

(4) Goodput Fairness. Figure 8 displays the results. Drop-tail was the most unfair method in goodput. Considering the confidence interval, the RED, ARED, AWM, and EWT methods had similar fairness.

(5) Mean Loss Ratio. The results are presented in Figure 9. The largest percentage of loss occurred with the RED and ARED methods, followed by the drop-tail method. The EWA presented losses at the medium and heavy levels, with the AWM only at the heavy level. In the EWT, the percentage was zero, regardless of the level of congestion.

5.4.2. Result Analysis. This scenario presented a long delay in the bottleneck, which affected the performance of TCP with traditional methods. EWT has been proven to be efficient in this scenario, significantly reducing congestion, bringing significant gains in transfer latency and goodput, and maintaining fairness between flows. No losses were recorded, so the TCP efficiency was maintained. AWM was the method that most approached the results obtained by EWT, but the AWM presented losses.

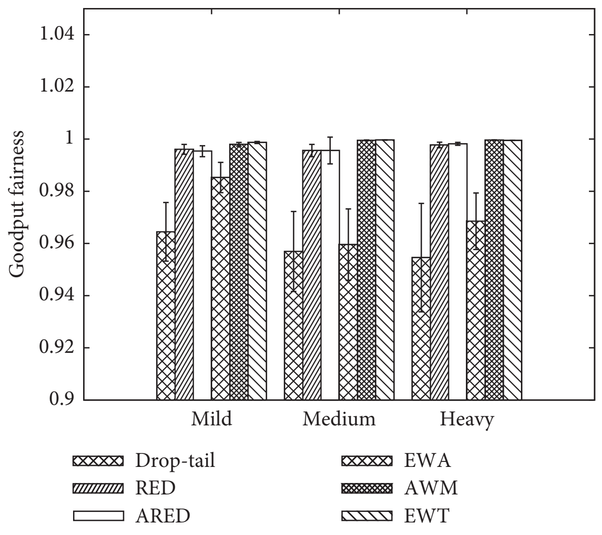

Figure 8: Goodput fairness: Scenario 1.

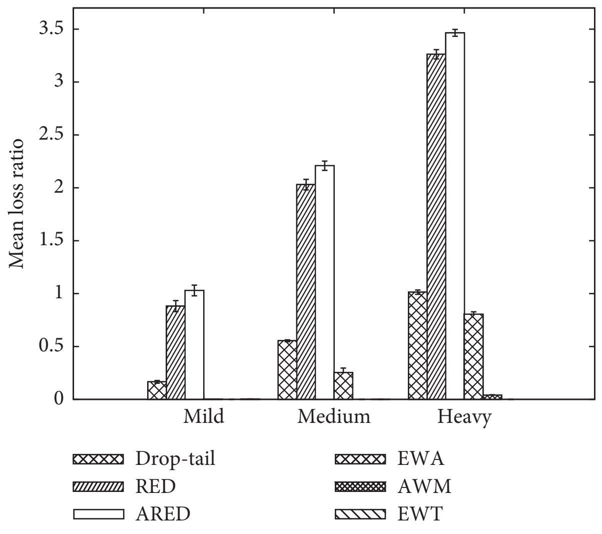

Figure 9: Mean loss ratio: Scenario 1.

Based on this simulation, the good behavior of the EWT was verified in networks with high delay. Its window adjustment mechanism brought gains to TCP. In this scenario, the traditional methods, such as RED, presented a low performance when compared to EWT. The main reason for this is the incorrect adjustment of the congestion window, which causes the flows to send few segments. This can be seen in Figure 10. In order to produce the graph, the effective transmission window ($\min(W_r, W_{\text{cwnd}})$) of one flow was registered. Analyzing the figure, it can be observed that the RED caused the TCP to present a high oscillation in the transmission window, while the EWT kept the window balanced without reaching low values.

5.5. Scenario 2. This scenario was designed to verify the efficiency of the EWT in the presence of UDP traffic. The dumbbell topology (Figure 4) was used with the settings shown in Table 5.

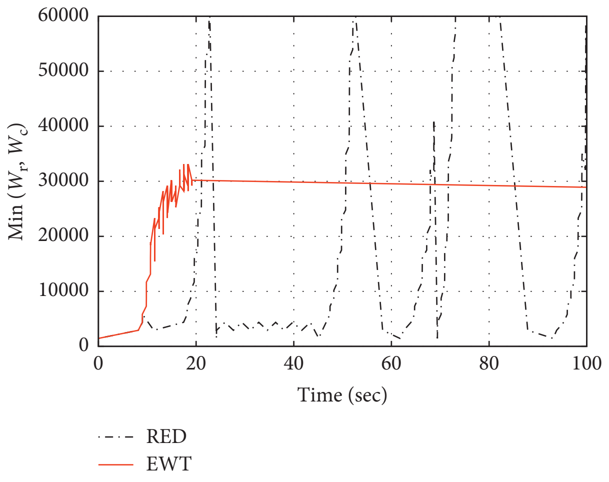

Figure 10: Window used to transmit segments to a flow in Scenario 1.

Simulations were performed by varying the number of TCP flows in the presence of concurrent UDP traffic. UDP traffic was transmitted at the rate of 13.35 Mbps (30% of link capacity) using segments with a size of 1400 bytes. For this, a UDP client was installed on the first node on the left side of $R1$ which transmits data to a UDP receiver located on the first node on the right side of $R2$.

5.5.1. Results

(1) TCP Efficiency. The results are presented in Figure 11. The efficiency of the drop-tail, RED, and ARED methods was similar, with a small difference in medium and heavy congestion. The EWA and AWM had a small loss of efficiency with heavy congestion. The EWT maintained the TCP efficient at the three levels of congestion.

(2) UDP Efficiency. As this scenario presented UDP traffic, the UDP efficiency (calculated using the same TCP efficiency equation) was also calculated. In Figure 12 the results are shown. With low traffic, the efficiency of the methods was similar. At the midlevel congestion, the drop-tail, RED, and ARED approaches had lost efficiency, and with heavy traffic, there was an even higher drop. In general, the network-return techniques kept the UDP efficient.

(3) File Transfer Latency. Figure 13 displays the results. The EWT obtained a shorter transfer latency for the three levels of congestion. With medium and heavy traffic, considering the confidence interval, the drop-tail, RED, and ARED methods had the same transfer latency. The EWA method presented the worst transfer latency. The AWM presented similar results to the EWT.

(4) Goodput. The results are shown in Figure 14. At the mild congestion level, the EWT registered the highest goodput. When the traffic was moderated considering the confidence interval, the drop-tail, RED, and ARED methods showed the same goodput. With heavy traffic, the same occurred. The

Table 5: Parameters of Scenario 2.

Parameter	Value
Link capacity (sources)	100 Mb
RTT (sources)	2 ms
Link capacity ($R1$-$R2$)	45 Mb
RTT ($R1$-$R2$)	80 ms
Link capacity (receptors)	100 Mb
RTT (receptors)	2 ms

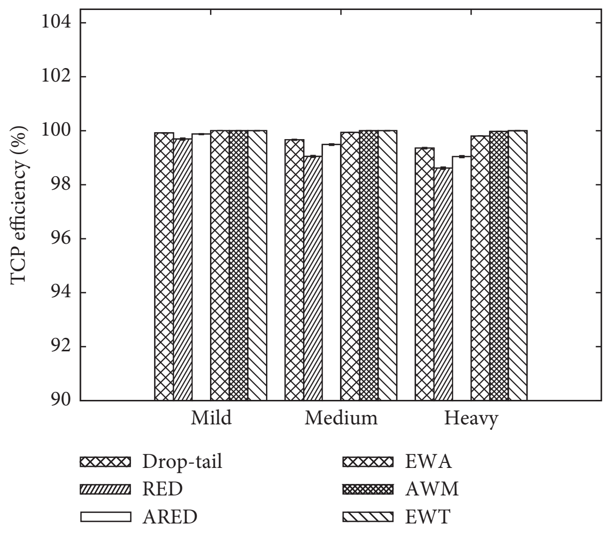

Figure 11: TCP efficiency: Scenario 2.

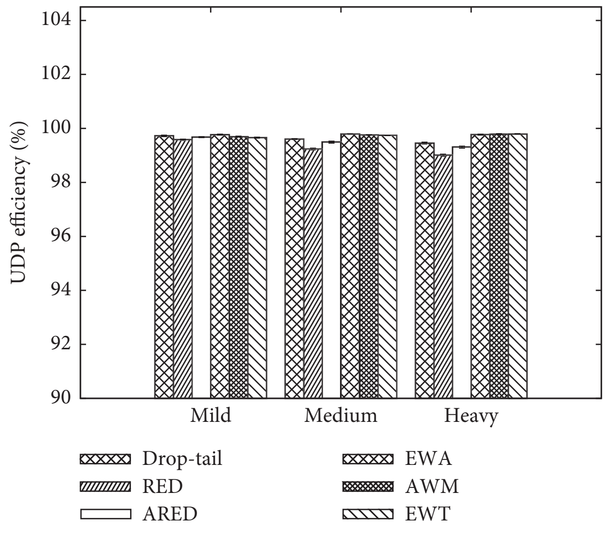

Figure 12: UDP efficiency: Scenario 2.

EWA showed the lowest goodput. The AWM presented results similar to those of the EWT at medium and heavy traffic levels.

(5) Goodput Fairness. Figure 15 displays the results. At different levels of congestion, the drop-tail method was the most unfair method. The EWA remained fair at the medium level; after this, it had a loss of fairness. At the medium and heavy levels, considering the confidence interval, the RED, ARED, AWM, and EWT methods had the same fairness.

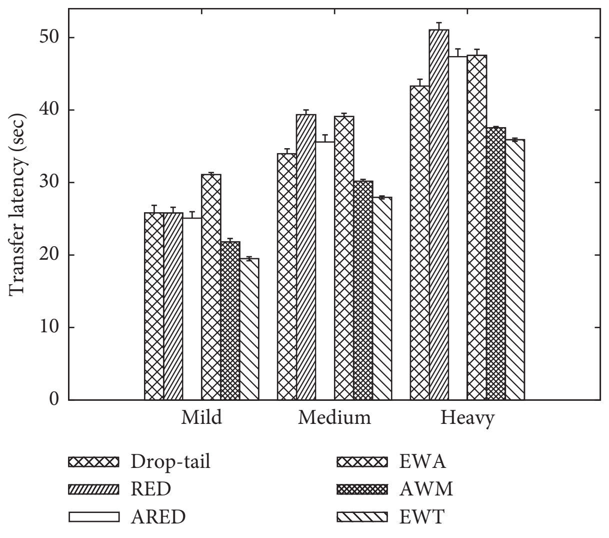

Figure 13: File transfer latency: Scenario 2.

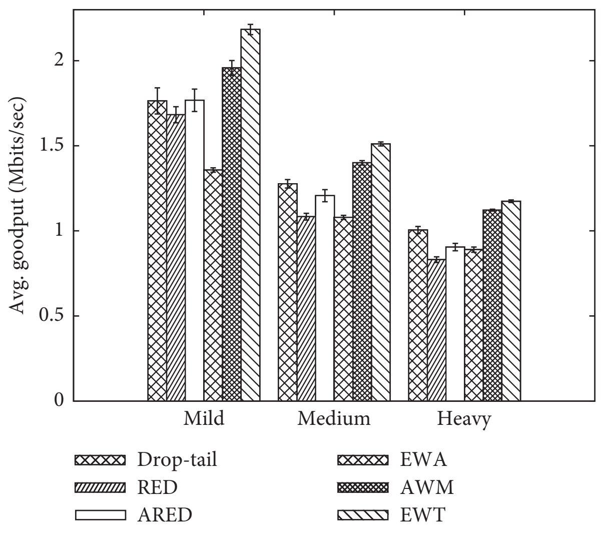

Figure 14: Average goodput: Scenario 2.

(6) Mean Loss Ratio. Since there were TCP traffic and UDP traffic in this scenario, the results are presented in Table 6. The average percentage of loss grew as congestion increased, with the RED registering the highest percentage. The EWA and AWM methods showed losses at the medium and heavy levels. The EWT presented no losses at any level.

5.5.2. Result Analysis. The presence of UDP traffic had no negative influence on the performance of the EWT. In most of the metrics, the EWT obtained better results than other methods, with emphasis on the increase in goodput, decrease of the average transfer latency of files up to 35%, and the absence of losses. The EWT fairness with medium and heavy traffic was preserved, but with low traffic, it was somewhat lower (1.40%) than AWM fairness.

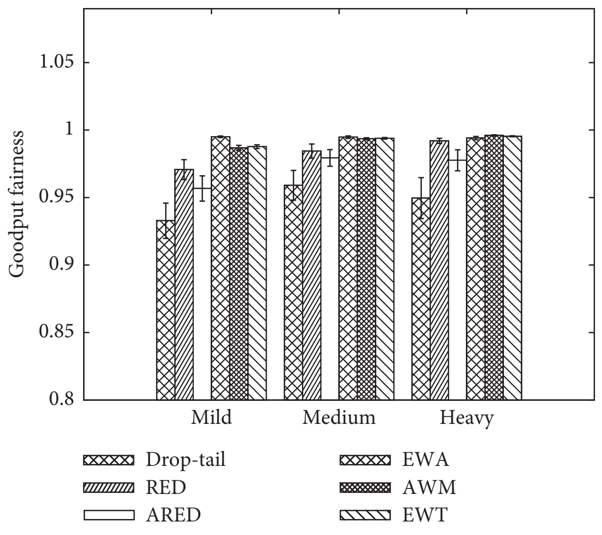

Figure 15: Goodput fairness: Scenario 2.

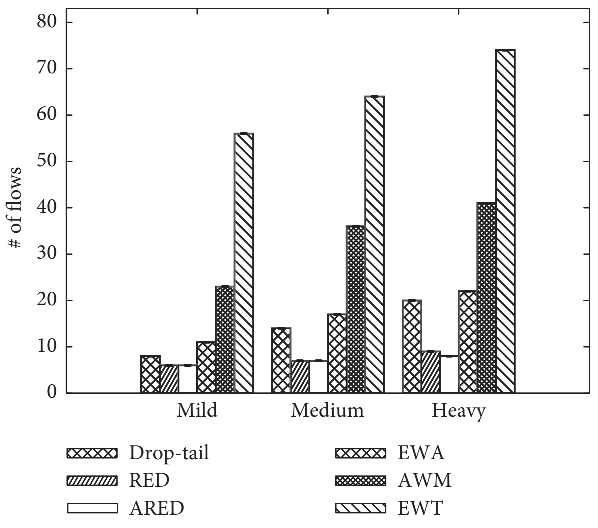

Figure 16: Number of flows needed to meet RFC 7928 in Scenario 1.

Table 6: Mean loss ratio of TCP and UDP: Scenario 2.

	Mild		Medium		Heavy	
	TCP	UDP	TCP	UDP	TCP	UDP
Drop-tail	0.07	0.029	0.31	0.31	0.52	0.43
RED	0.11	0.07	0.54	0.37	0.83	0.62
ARED	0.09	0.05	0.41	0.29	0.62	0.50
EWA	0	0	0.07	0.02	0.19	0.05
AWM	0	0	0	0	0.04	0.02
EWT	0	0	0	0	0	0

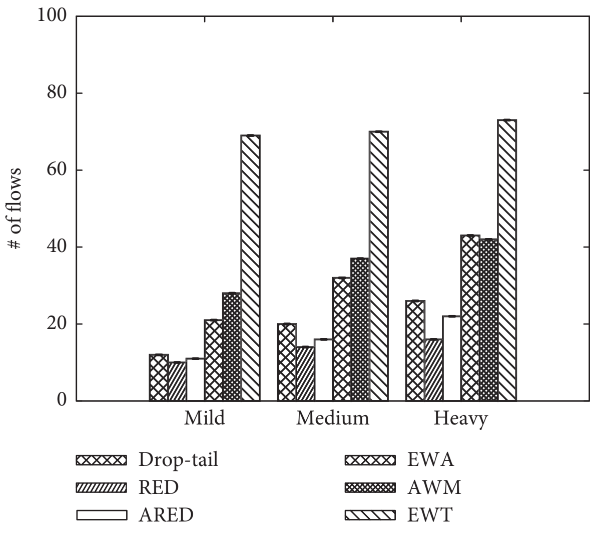

Figure 17: Number of flows needed to meet RFC 7928 in Scenario 2.

Although the EWT is designed to act on TCP congestion control, it operated satisfactorily in the presence of UDP traffic. The EWT maintained the absence of losses; this was due to its mechanism of adjustment of windows that realizes the calculation based on the number of bytes available in the memory of the gateway, also considering the TCP traffic. Of course, if there is excessive UDP traffic, losses will occur. But the EWT performance in conjunction with the congestion control present in TCP will quickly cause the flows to adjust the size of the burst to mitigate the congestion since the return of the EWT is smaller than an RTT because it acts directly on the TCP ACK segments that pass through the gateway. Another interesting point of the EWT application is that the UDP ends up becoming a priority because TCP is controlled and UDP maintains its rate. As UDP flows normally transmit data in real time, this behavior can be desired.

5.6. Increasing the Number of Served TCP Flows. The RFC 7928 defines three levels of congestion based on the segment loss rate. In the simulations performed, the levels were found by increasing the number of TCP flows until the loss rate of each level was reached. As RFC indicates, no AQM scheme was used in this step. In this section, we use the RED, ARED, EWA, AWM, and EWT methods to find the three levels of congestion that the RFC suggests. This was done with the objective of finding the number of flows that each method can handle at each level of congestion.

The results are shown in Figures 16 and 17. The method that reached the greatest number of flows for each level of congestion was the EWT. The method allowed the existence of a number of flows which is on average 49.3% better than its best competitor and 75.8% better when no AQM scheme was used. The EWT algorithm that reduces the W_r in proportion to the level of memory utilization of the gateway was responsible for this good result. With the reduction of W_r of ACK segments, fewer segments were sent when the number of flows increases; because of this, the queue will be filled and losses will occur mainly when $\sum_{f=0}^{nf} f_i \times \text{MSS} > \text{queueSize}$ (where nf is the total number of flows) or when any traffic other than TCP exists.

6. Conclusion and Future Work

In this paper, we presented a new network-return technique called early window tailoring (EWT). The approach was applied in multiple scenarios. Following the recommendations of RFC 7928 and with the use of ns-3, several simulations were performed. The EWT was compared to drop-tail, RED, ARED, and the two network-return techniques—explicit window adaptation (EWA) and active window management (AWM). In the results, it was observed that the EWT was proved to be efficient in congestion control, avoiding losses. In the first scenario, consisting of a delayed bottleneck characteristic of distant networks or satellite communications, the EWT was proved to be efficient, significantly reducing congestion, bringing significant gains in transfer latency and goodput, while maintaining the fairness between flows. The second scenario used UDP traffic to check its influence on the method. The presence of UDP traffic had no influence on EWT behavior. At the end, we used the two scenarios to find the number of flows that each method can handle at each of the three levels of congestion as suggested by RFC 7928. However, unlike other approaches, the most prominent feature of EWT is its ability to maintain a very high number of active flows at a given level of segment loss rate. The EWT allowed the existence of a number of flows which is on average 49.3% better than its best competitor and 75.8% better when no AQM scheme was used. In most of the metrics, the EWT obtained better results than the other methods, highlighting the increase in goodput and a decrease in the average transfer latency up to 35%. Compared to other methods, AWM obtained good results; however, unlike the EWT, it presented losses, and for its use, it is necessary to have knowledge on the number of active flows. In simulations, this is easy to obtain; however, in a real network, this is a problem, and in addition, the AWM requires fine tuning of its two parameters. Finally, because the EWT seeks to avoid losses, the more expensive the retransmission, the greater the advantage of using the method.

Future work includes the analysis of EWT in wireless networks as well as the use of EWT in order to address the bufferbloat problem [17].

Data Availability

The simulation graph data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] M. Allman, D. V. Paxson, and E. Blanton, "TCP congestion control," RFC 5681, IETF, Fremont, CA, USA, 2009.

[2] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993.

[3] S. Floyd, "TCP and explicit congestion notification," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 5, pp. 8–23, 1994.

[4] R. Pan, P. Natarajan, C. Piglione et al., "A lightweight control scheme to address the bufferbloat problem," in *Proceedings of the 2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*, pp. 148–155, Taipei, Taiwan, July 2013.

[5] S. Lee, D. Lee, M. Lee, H. Jung, and B.-S. Lee, "Randomizing TCP payload size for TCP fairness in data center networks," *Computer Networks*, vol. 129, pp. 79–92, 2017.

[6] S. Floyd, J. Mahdavi, M. Mathis, and D. A. Romanow, "TCP selective acknowledgment options," RFC 2018, IETF, Fremont, CA, USA, 1996.

[7] S. Bohacek, J. P. Hespanha, J. Lee, C. Lim, and K. Obraczka, "TCP-PR: TCP for persistent packet reordering," in *Proceedings of the 23rd International Conference on Distributed Computing Systems 2003*, pp. 222–231, Providence, RI, USA, May 2003.

[8] M. Alizadeh, A. Greenberg, D. A. Maltz et al., "Data center TCP (DCTCP)," in *Proceedings of the ACM SIGCOMM 2010 Conference on SIGCOMM'10*, pp. 63–74, ACM, New York, NY, USA, 2010.

[9] A. G. Shewmaker, C. Maltzahn, K. Obraczka, S. Brandt, and J. Bent, "TCP inigo: ambidextrous congestion control," in *Proceedings of the 2016 25th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–10, Waikoloa, HI, USA, August 2016.

[10] M. Gerla, R. L. Cigno, S. Mascolo, and W. Weng, "Generalized window advertising for TCP congestion control," *European Transactions on Telecommunications*, vol. 13, no. 6, pp. 549–562, 2002.

[11] M. Talau and E. C. G. Wille, "Available network bandwidth schema to improve performance in TCP protocols," *International Journal of Computer Networks and Communications*, vol. 5, no. 5, pp. 45–57, 2013.

[12] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan, "Explicit window adaptation: a method to enhance TCP performance," *IEEE/ACM Transactions on Networking*, vol. 10, no. 3, pp. 338–350, 2002.

[13] M. Barbera, A. Lombardo, C. Panarello, and G. Schembra, "Active window management: an efficient gateway mechanism for tcp traffic control," in *Proceedings of the 2007 IEEE International Conference on Communications*, Glasgow, Scotland, UK, June 2007.

[14] C. Palazzi, S. Ferretti, M. Roccetti, G. Pau, and M. Gerla, "What's in that magic box? The home entertainment center's special protocol potion, revealed," *IEEE Transactions on Consumer Electronics*, vol. 52, no. 4, pp. 1280–1288, 2006.

[15] C. A. Grazia, M. Klapez, N. Patriciello, and M. Casoni, "PINK: proactive INjection into acK, a queue manager to impose fair resource allocation among TCP flows," in *Proceedings of the 2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 132–137, Abu Dhabi, UAE, October 2015.

[16] M. Casoni, C. A. Grazia, M. Klapez, and N. Patriciello, "How to avoid tcp congestion without dropping packets: an effective aqm called pink," *Computer Communications*, vol. 103, pp. 49–60, 2017.

[17] J. Gettys and K. Nichols, "Bufferbloat," *Communications of the ACM*, vol. 55, no. 1, pp. 57–65, 2012.

[18] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound tcp approach for high-speed and long distance networks," in *Proceedings of the IEEE INFOCOM 2006. 25th IEEE*

International Conference on Computer Communications, pp. 1–12, Barcelona, Spain, April 2006.

[19] G. Raina, S. Manjunath, S. Prasad, and K. Giridhar, "Stability and performance analysis of compound tcp with rem and drop-tail queue management," *IEEE/ACM Transactions on Networking*, vol. 24, no. 4, pp. 1961–1974, 2016.

[20] S. Ha, I. Rhee, and L. Xu, "CUBIC," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[21] G. Vardoyan, C. V. Hollot, and D. Towsley, "Towards stability analysis of data transport mechanisms: a fluid model and an application," in *Proceedings of the IEEE INFOCOM 2018—IEEE Conference on Computer Communications*, pp. 666–674, Honolulu, HI, USA, April 2018.

[22] M. Allman and V. Paxson, "On estimating end-to-end network path properties," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication SIGCOMM'99*, pp. 263–274, ACM, New York, NY, USA, 1999.

[23] M. Jain and C. Dovrolis, "End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput," *IEEE/ACM Transactions on Networking*, vol. 11, no. 4, pp. 537–549, 2003.

[24] G. Hasegawa and M. Murata, "TCP symbiosis: congestion control mechanisms of TCP based on Lotka-Volterra competition model," in *Procedings of the Interperf'06: Proceedings from the 2006 Workshop on Interdisciplinary Systems Approach in Performance Evaluation and Design of Computer & Communications Sytems*, p. 11, ACM, New York, NY, USA, 2006.

[25] S. Jiang, Q. Zuo, and G. Wei, "Decoupling congestion control from TCP for multi-hop wireless networks: semi-TCP," in *Proceedings of the CHANTS'09: Proceedings of the 4th ACM Workshop on Challenged Networks*, pp. 27–34, ACM, New York, NY, USA, 2009.

[26] A. Bujari, A. Marin, C. E. Palazzi, and S. Rossi, "Analysis of ECN/RED and SAP-LAW with simultaneous TCP and UDP traffic," *Computer Networks*, vol. 108, pp. 160–170, 2016.

[27] M. Talau, M. Fonseca, A. Munaretto, and E. C. G. Wille, "Early congestion control: a new approach to improve the performance of TCP in ad hoc networks," in *Proceedings of the 7th International Conference on the Network of the Future (NOF)*, pp. 1–6, Buzios, Brazil, November 2016.

[28] C. E. Palazzi, S. Ferretti, and M. Roccetti, "Smart access points on the road for online gaming in vehicular networks," *Entertainment Computing*, vol. 1, no. 1, pp. 17–26, 2009.

[29] ns-3(a), Official Website, 2019, http://www.nsnam.org.

[30] N. Kuhn, P. Natarajan, N. Khademi, and D. Ros, "Characterization guidelines for active queue management (AQM)," RFC 7928, IETF, Fremont, CA, USA, 2016.

[31] D. D. D. Clark, G. Minshall, L. Zhang et al., "Recommendations on queue management and congestion avoidance in the internet," RFC 2309, IETF, Fremont, CA, USA, 1998.

[32] F. Baker and G. Fairhurst, "IETF recommendations regarding active queue management," RFC 7567, IETF, Fremont, CA, USA, 2015.

[33] P. Imputato and S. Avallone, "Design and implementation of the traffic control module in ns-3," in *Proceedings of the Workshop on ns-3, WNS3'16*, pp. 1–8, ACM, New York, NY, USA, 2016.

[34] L. Andrew, C. Marcondes, S. Floyd et al., "Towards a common TCP evaluation suite," in *Proceedings of the International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet)*, Manchester, UK, 2008.

[35] D. Hayes, D. Ros, L. L. H. Andrew, and S. Floyd, "Common TCP evaluation suite," 2014.

[36] R. Schrage, G. Forget, R. Geib, and B. Constantine, "Framework for TCP throughput testing," RFC 6349, IETF, Fremont, CA, USA, 2011.

[37] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," DEC Research Report TR-301, Digital Equipment Corporation, Maynard, MA, USA, 1984.

[38] G. Carneiro, P. Fortuna, and M. Ricardo, "Flowmonitor: a network monitoring framework for the network simulator 3 (ns-3)," in *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, pp. 1–10, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, Belgium, Belgium, 2009.