future internet

MDPI

Article

Effective IoT Congestion Control Algorithm

Husam H. Hasan * and Zainab T. Alisa

Department of Electrical Engineering, University of Baghdad, Baghdad 10001, Iraq
* Correspondence: husam.hasan1502m@coeng.uobaghdad.edu.iq

Abstract: The Internet of Things (IoT) connects devices via the Internet. Network congestion is one of the key problems that has been identified by researchers in the IoT field. When there is a huge number of IoT devices connected to the internet, this creates network congestion. Transfer control protocol is a transport layer protocol that provides a reliable end-to-end connection between two devices. Many Congestion Control Algorithms have been proposed to solve network congestion. However, there is no perfect solution to this problem. This paper proposes an effective loss-based Congestion Control Algorithm to effectively adapt the congestion window in the IoT environment. It uses simple experiment scenarios to test the algorithm for wired and wireless channels and observes important performance metrics: link utilization, inter-protocol fairness, intra-protocol fairness and throughput. The results are impressive, and the proposed algorithm is shown to outperform other standard algorithms.

Keywords: IoT; congestion control; TCP/IP; wireless TCP; CWDN; RTO

Citation: Hasan, H.H.; Alisa, Z.T. Effective IoT Congestion Control Algorithm. *Future Internet* **2023**, *15*, 136. https://doi.org/10.3390/fi15040136

Academic Editors: Nouman Ashraf and Sachin Sharma

Received: 4 March 2023
Revised: 24 March 2023
Accepted: 28 March 2023
Published: 31 March 2023

1. Introduction

There is significant enthusiasm in both industry and academia about the Internet of Things (IoT). Its goal is to create universally deployable and interoperable "smart" gadgets that can connect wired or wirelessly and function independently. Due to technological advancements and the infrastructure provided by the IoT, individuals can now use a wide variety of gadgets to access the internet and exchange real-time data. Transmission control protocol (TCP) is the most reliable of all the protocols used for data transmission over the Internet. The IoT facilitates the connection and communication of a wide variety of electronic gadgets through the Internet. There is currently a wide range and rapid growth of connected IoT devices. Consequently, there has been an exponential rise in network congestion. There is a clear correlation between the quantity of internet-connected devices and the level of congestion. The exponential growth of internet-connected devices has dramatically increased network congestion. Therefore, modifications to TCP are necessary. Specifically, there is a need for modifications that are well suited to the IoT environment and that can start the connection in terms of capacity and adjust the rate of transmission as congestion rises.

Most IoT devices have limited energy and work well in sensitive areas. Critical data are sent from various sensors that serve different areas, including military surveillance and health care monitoring [1]. Electric smart meters have emerged as a relevant topic in the IoT and are working well as efficient IoT applications [2]. In addition to other application layer protocols, the IoT also provides special application layer protocols that serve its uses and requirements, including Message Queuing Telemetry Transport (MQTT), Extensible Messaging and Presence Protocol (XMPP) and Representational State Transfer (RESTful) HTTP. These algorithms do not satisfy the requirements of IoT.

A Congestion Control Algorithm (CCA) is required to balance their varying bandwidth, latency, and delay requirements. Since the 1980s, researchers have proposed

numerous TCP CCA, and developed several standards and modifications for these algorithms. The first TCP specification was proposed in 1981 [3].

In late 1986, the first congestion collapse occurred. At the time, there was no clear reason for this collapse, and it was assumed that it was due to the malfunctioning of a TCP protocol. In 1988, a new policy was suggested by Jackobson: the TCP Tahoe [4]. It included Slow Start (SS), Retransmission Timeout (RTO) and Fast Retransmit and Congestion Avoidance Algorithm (CAA). To solve the congestion problem, the policy aimed to deal with different sending phases. Mathis et al. [5] created TCP SACK (Selective Acknowledgment) to address packet losses. TCP SACK enables the receiver to identify the segments that have been delivered. This allows the TCP sender to promptly identify and rebroadcast the missed segment. SACK further allows the TCP receiver to selectively acknowledge out-of-order segments. It transmits segments to the source and allows it to only transmit missing parts. SACK's main drawback is the receiver's lack of appropriate acknowledgements. Selective acknowledgement is difficult and requires transmitter and receiver acknowledgement protocol changes. Following this, numerous techniques and algorithms were proposed to adjust sending rate over TCP and avoid congestion. The degradation in throughput in TCP Tahoe occurred because the congestion window (CWND) decreased to one when any packet loss occurred. This issue was identified by Floyd and Henderson [6], who suggested TCP NewReno.

Most of these CCAs use the concept of "Additive Increase Multiplicative Decrease" (AIMD); this means the CWND increases in an exponential fashion to reach maximum bandwidth. The CWND decreases when network bandwidth is reached and congestion in the network is detected because of segment losses. Congestion in the network is encountered when retransmission timeout (RTO) expires, or three duplicate acknowledgements (3DupAck) are reached at the sender [3]. TCP NewReno allows a modification of the Fast Recovery Algorithm (FRA) where partial acknowledgement is permitted.

This paper suggests an effective congestion control algorithm for the IoT environment. A stable TCP algorithm ensures that the adjustment of new windows is taken precisely to reduce packet loss and increase throughput.

Section 2 of this paper provides the related work. Section 3 presents the proposed algorithm and explains the working principles and their implementation. Section 4 outlines the important findings in implementing the proposed algorithm and discusses the results. Section 5 offers conclusions related to this work.

2. Related Work

Congestion control policy is generally concerned with how congestion is dealt with. It can be categorized into the following groups:

Loss-based CCA is a type of CCA where packet loss is used as a congestion indication signal to trigger further action. TCP NewReno does not perform well in a high bandwidth-delay product (BDP) because there is sufficient bandwidth to occupy; this is not what happens in these protocols. In 2004, Lisong et al. [7] proposed TCP Binary Increase Congestion (BIC) control, which can deal with fast long-distance networks. This algorithm uses a binary search algorithm to identify the optimal CWND. The aggressive nature of this algorithm causes losses to increase in the network; it is therefore not intended for low-speed networks.

Delay-based CCA means that any increase in delay is counted as an increase in network congestion and vice versa. Numerous protocols under this policy have been encountered. TCP Vegas [8] suggests a better Round-Trip Time (RTT) estimation than Reno estimation. As a result, it increases the general throughput; however, TCP Vegas is not suitable for high-speed networks.

Hybrid CCA means that the adjustment of CWND is controlled by both Delay and Packet Loss. One of the most well-known algorithms that uses both factors to adjust the congestion window is a standard algorithm: TCP Compound [9]. It is used by Windows 7 and later. Using the same approach, TCP-FIT [10] hybrid strategies are used to control

CWND. AIMD has been used to alter the CW regardless of whether there have been losses or acknowledgements. Increasing throughput is accomplished by parallel protocol operation.

TCP Illinois [11] is a hybrid protocol developed for networks with high bandwidth. It alters the standard AIMD algorithms so that the modification of window size is dependent on loss and the direction is determined by delay, or more specifically RTT. It employs the default settings for the FR and the retransmission features of regular TCP NewReno.

The key advantages of TCP Illinois are that it is designed to handle different types of network environments, such as wireless networks, satellite links and high latency networks. It has a range of parameters that can be adjusted to optimize performance under different network conditions, making it a flexible and adaptable algorithm.

While TCP Illinois offers many advantages, there are also some potential disadvantages to consider; it may be more aggressive than other CCAs, which can lead to increased packet loss. This may occur in high-speed networks, or in networks with high packet loss rates. Also, it may not be compatible with all types of networks and may not work well with other CCAs. This can lead to interoperability issues and may require additional configuration to ensure proper functioning. TCP Illinois may not perform well in certain network conditions, such as networks with high latency or jitter; this can affect its ability to accurately measure network congestion and adjust its sending rate [12].

TCP YeAH [13] is another high-speed Hybrid CCA TCP variant. This algorithm has two modes of operation, one of which is identical to TCP Reno. The other makes use of STCP's [14] aggressive behavior. It is a CCA that is designed to improve TCP performance in high-speed and high-latency networks. While TCP YeAH is a standalone algorithm, it does incorporate elements of the NewReno and STCP congestion control mechanisms. It incorporates the fast recovery mechanism from NewReno, allowing it to quickly recover from packet loss and maintain high throughput. Specifically, when TCP YeAH detects that a packet has been lost, it enters a fast recovery phase. During this phase, it reduces the congestion window (CWND) size by half and retransmits the lost packet. This helps to prevent congestion collapse and maintain efficient network performance.

It also incorporates the adaptive window sizing algorithm from STCP, which allows it to dynamically adjust the CWND size based on network conditions. Specifically, during periods of low congestion, TCP YeAH uses a concave control mechanism to gradually increase the CWND size at a rate proportional to the square root of the current CWND value. This helps to prevent sudden spikes in network traffic that may lead to congestion. During congestion, TCP YeAH switches to an additive increase/multiplicative decrease (AIMD) approach to congestion control, reducing the CWND size by a certain percentage to slow down the sending rate and prevent further congestion.

While TCP YeAH offers significant advantages in terms of improving TCP performance under challenging network conditions, its complexity, potential fairness issues, and compatibility limitations must be carefully weighed against its benefits when deciding whether to implement it in a particular network environment, especially in an IoT environment.

Bottleneck Bandwidth and Round-trip propagation time (BBR) [15] is a CCA developed by Google in 2016. It is designed to improve network performance by more efficiently utilizing available network bandwidth and reducing network latency. The TCP BBR algorithm works by continuously measuring the available bandwidth and the RTT of data packets as they traverse the network. Based on these measurements, the algorithm determines the ideal sending rate for the connection. TCP BBR operates by dynamically adjusting the sending rate based on the network's available bandwidth and RTT. It uses a model of the network to estimate the maximum amount of data that can be transmitted without causing congestion, and it calculates the ideal sending rate based on this estimate.

TCP BBR aims to provide fair sharing of network resources; however, there is some concern that it may prioritize traffic from larger RTT over smaller RTT nodes. This could

potentially create an uneven playing field and limit access to network resources for some nodes [16].

Verma et al. [17] developed a CCA intended for use with lightweight IoT application protocols such as Message Queuing Telemetry Transport (MQTT) and Extensible Messaging and Presence Protocol (XMPP). It defines a new window initialization method and modifies the SS and CA phases by defining new parameters in order to work in two modes: reactive and proactive modes. There are two parameters in this context: rho and beta. The basic rule is that TCP works in a reactive mode when beta is less than rho and in a proactive mode when beta is greater than rho. The author compares the suggested algorithm with other standard protocols. The author claims that the algorithm increases the throughput and maintain fairness in TCP Cubic.

Chappala et al. [18] proposed an Adaptive Congestion Window (ACW) for IoT devices. The design of the ACW depends on three parameters: sending rate, receiving rate and the available bandwidth of the path. The authors assume that there is a communication between the nodes to share connection information. The connection information is used to adjust the sending window when some nodes release/share the link; however, this information may burden the node and increase the packet overhead, which in turn reduces the link utilization and consumes the node resources and energy.

Gupta et al. [19] proposes a novel STCP approach to control congestion in the IoT environment. In this approach, a new window initialization technique is used based on the current available bandwidth of the path in order to reach the available bandwidth as fast as possible. However, this method is not recommended according to [3]; SS phase is preferable. This process develops a new approach for detecting congestion prior to collapse and sets the CWND limits accordingly. This approach is called Early Congestion Detection; this factor is computed depending on the queue size of the most congested link on the path, available bandwidth and RTT. Due to a prior set of window limits, it may underutilize the available bandwidth.

A small segment size window is available in microcontrollers with limited capabilities. On such devices, a special TCP/IP stack called micro-IP (uIP) sets the sending window to one segment size by default. Congestion control in such a scenario relies heavily on careful management of the RTO. In [20], the author proposes an enhanced scheme for RTO to manage the problem of large RTT variation caused by certain systems. By using the idea of weak RTT estimate from CoCoA [21], this technique changes the RTO so that it is variable, rather than the fixed RTO set by the original TCP uIP. This kind of congestion control is limited to very small buffers, less than or equal to one maximum segment size.

In [22], the authors propose a new technique to reduce the delay in multipath TCP (MPTCP) by reducing the number of transmissions using an Opportunistic Routing (OR) technique. The OR routing model is implemented to increase the throughput and reliability of wireless networks via the use of the broadcasting method. The authors compare the proposed scheme with other MPTCP algorithms. Though this scheme may be applicable for large memory devices such as smartphones, smartwatches and tablets due to their heterogeneous interfaces, it may not be applicable for devices with limited buffers and single connections deployed for IoT, such as esp32 [23] and esp8266 [24]. The author in [25] analyzed and designed QoS-aware personalized privacy for MPTCP for use in Industrial IoT (IIoT) to optimize the tradeoff between efficiency and privacy protection. IIoT data may be vulnerable to attack and requires more attention when MPTCP is the use case.

A second group of researchers have taken a novel approach and employed state-of-the-art controllers, such as fuzzy controllers, to fine-tune and optimize the congestion control technique. Fuzzy controllers were used by Zaineb et al. [26] to improve the TCP Protocol in a mobile network. Other networks, such as Software Defined Networks (SDN), also impact congestion. Researchers have used the Neural Network approach to solve congestion problems in 5G communication [27]. However, examining these forms of controllers and networks is beyond the scope of our study.

3. The Proposed Algorithm

This paper proposes an effective CCA that modifies how the congestion avoidance phase is dealt with by changing the parameter updating process. TCP constantly evaluates the capability of a network in order to forward traffic to the destination. This process starts with the initial CWND, which is set to a default maximum segment size of 10 [28].

This algorithm utilizes two variables: alpha and beta. Alpha is responsible for increment in the congestion window at the sending node, while beta represent the decrement factor. It uses delay as a back-off factor (beta) to decrease the sharpness of alpha increment when there is an increase in delay. This limits the increment rate of the congestion window.

To explain the working principle of the algorithm, two cases are shown: the first case, when the algorithm running under normal conditions i.e., the link is not congested. In the SS phase, the algorithm behaves as TCP NewReno; when the CWND become equal or greater than the SSThresh value in this case, the algorithm enters into congestion avoidance phase and the behavior of the algorithm is as follows:

First Case flow:
Step 1: Set initial parameters alpha = 1, beta = 0.67, deltal = 1//initial value at the connection establishment.
Step 2: For each packet acknowledgement delta = time.now () − time of last congestion;
Step 3: calculate throughput = bytes sent * 8/time.now − last send time;
Step 4: If delta > deltal then calculate difference = delta − deltal;
Step 5: Update alpha = 1 + 10 * difference + 0.3 * difference * difference;
Step 6: update min_RTT, max_RTT, RTT of the last acknowledged segments
Step 6: Refine the increase variable alpha = 2 * (1 − beta) * alpha;
Step 7: Calculate the increament inc = ((segment size * segment size) + CWND * alpha)/CWND;
Step 8: update CWND = CWND + inc;

The second case is when the link is congested and in its congestion avoidance phase. There are two ways to detect TCP congestion. First, there may be TCP congestion when there is no acknowledgement from the receiver to the sender; this means that the sender timeout variable is expired. The second way to detect congestion is when there are three duplicate acknowledgements between the receiver and the sender. When there is a congestion due to RTO expiration, the congestion window returns to one segment size and returns to Slow Start in order to follow TCP NewReno behavior. On the other hand, when the congestion is due to three duplicate acknowledgments, the algorithm follows the behavior described below:

SSThresh is calculated depending on the current inflight bytes. It is multiplied by beta to leave some space in the pipe and reduce congestion. This factor will increase its friendliness to other CCAs. Beta is only calculated when the increment of throughput is below one-third of the previous throughput; this means that there is no increase in the congestion window by higher values while the link is congested.

At each acknowledgement, RTT can be calculated as follows; Equation (1) shows the difference in time between packet send time and its acknowledgement:

$$RTT\_i = Ack\_rt(i) - Packet\_st(i) \quad (1)$$

where RTT_i is Round Trip Time of the ith packet, Ack_rt (i) is the time of receiving acknowledgement of ith packet and Packet_st (i) is the ith packet delivery time.

If the connection is normal and no packet drops due to congestion, throughput can be calculated as follows:

$$\text{throughput} = (\text{data sent in bytes}*8)/(\text{current time-last send time}) \quad (2)$$

where throughput is the current throughput, data sent in bytes is the number of delivered bytes, current time is the current time registered by the sender and last send time is the registered time of the last successful data sent. The increment factor, alpha, is flattened by the beta factor after being calculated (steps 5 and 6) as stated in the algorithm. With Equations (1) and (2), it would be possible to adjust sending rate to improve performance.

Second case: The following illustrate the algorithm at congestion event:
Step 1: Set CWND = SSThresh;
Step 2: update alpha = 1;
Step 3: calculate diff = throughput − last throghput;
Step 4: if diff/last throughput <= 0.33 go to Step 5: else go to Step 6:
Step 5: Calculate beta = 1.1 * minRTT/maxRTT;
Step 6: SegWin = 25 * Segmentsize;
Step 7: SSthresh = max (SegWin, bytesInFlight * beta * 1.25);

4. Results and Discussion

According to [29,30], Network Simulator 3 (NS3) [31] has been used due to its various advantages, such as its open source, parallel processing and faster computational time. Returning to the topology of simulation, it is essential to use dumbbell topology as a testbed for the validation process. Figure 1 outlines the dumbbell topology; it contains sending nodes, bottlenecks and receiving nodes. On the left side, there are N sending nodes numbered (S1, S2, …., Sn). They are connected to the left of router 1 (R1); router 2 (R2) forms the bottleneck. The destination nodes (D1, D2, …, Dn) are connected to R2. The parameter setting for the network is depicted in Table 1.

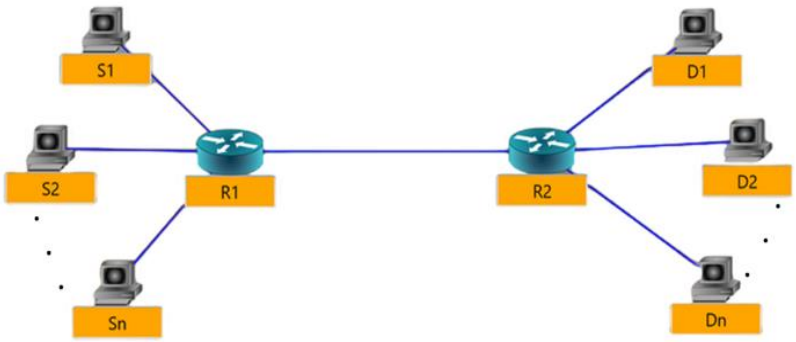


Figure 1. Testbed (Dumbbell Topology).

In the IoT, many devices with different Transport Layer Protocols and different system capabilities are connected to the network. Within this context, algorithm performance may be investigated in wired and wireless access link channels and at different link-speeds and different background traffic levels. During the simulation, the time was fixed to 30 s, which was sufficient to explore the behavior of the algorithm.

Table 1. General parameter setting.

Parameter	Setting
Simulation time	30 s
Access Link	10 Mbps, 10 ms
Bottleneck	1 Mbps and 30 ms, 2 Mbps and 30 ms
Error Rate	10^−6
TCP Variant	TCPNewReno, TCPBbr, TCPIllinois, TCPYeah
Channel	Wire, Wireless

Different experiments were conducted and the results of these were compared with four standard internet stack protocols: TCP BBR [26], TCP NewReno, TCP Illinois and TCP Yeah.

4.1. First Experement: Proposed Approach (TCP Modified)

Four stationary sending nodes (S1, …, S4) were connected to a router (R1) via an access link, and four stationary receiving nodes (D1, …, D4) were connected to a router (R2) via an access link. A bottleneck link was connected to the routers (R1 and R2), as shown in Figure 2. To ensure the proper working of the proposed algorithm, its CWND and shape were investigated. All sending nodes were set to have the same internet stack protocol (TCP modified, i.e., the proposed algorithm). The parameter setting for this simulation is shown in Table 2.

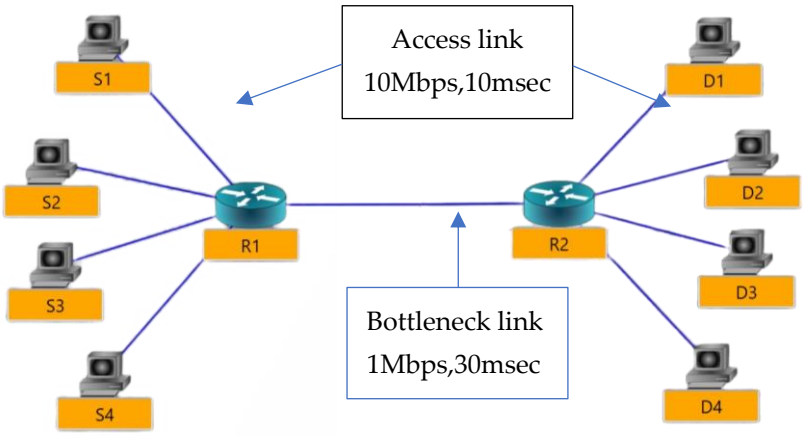


Figure 2. Testbed (Dumbbell Topology) for TCP Modified.

Table 2. First experiment parameters setting.

Parameter	Setting
Simulation time	30 s
Access Link	10 Mbps, 10 ms
Bottleneck	1 Mbps and 30 ms
Error Rate	10^−6
TCP Variant	TCPModified
Channel	Wire
Segment size	536 bytes

Figure 3 shows the congestion window of the four flows and demonstrates how it behaves for different phases. These include the Slow Start phase, network congestion (the congestion detection phase), and the congestion avoidance phase. The different phases of the CWND are discussed at each point and further explained. All the sending nodes start with the SS phase; in this phase, the algorithm acts as the TCP NewReno SS phase. When congestion is detected due to RTO expiration or three duplicate acknowledgments, the algorithm modifies the CWND to be equal to one segment size; this occurs because the default SSThresh is equal to (65535) bytes. When congestion is detected, the algorithm recalculates the SSThresh. If the new value of SSThresh after the detection of the congestion would be lower than the previous one, and hence after the second Slow Start, it is clear that the congestion avoidance algorithm is activated, as indicated by the arrow in Figure 3, which points to the CA phase of flow S1. Figure 3 shows that the CCA is functioning properly.

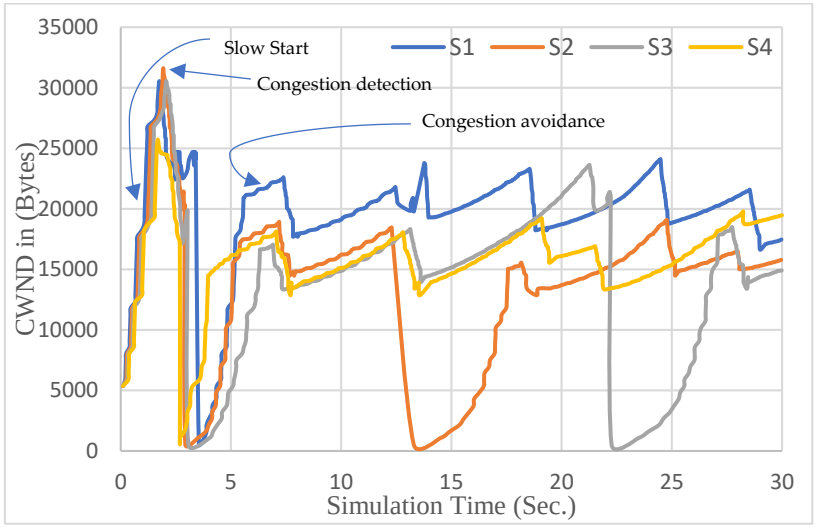
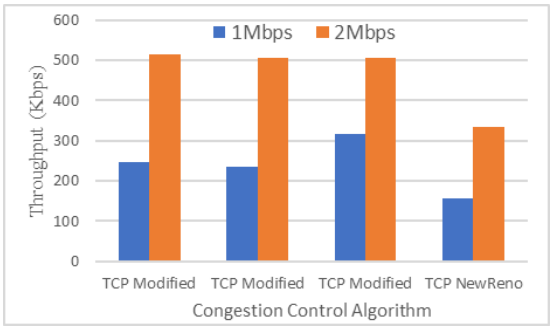


Figure 3. CWND of the proposed algorithm versus time.

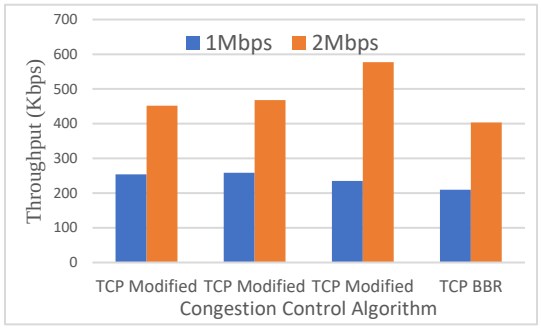
4.2. Second Experement: Proposed Approach with Onother Protocols I

The topology used in this experiment is shown in Figure 2. The intra-protocol test was applied using three sending nodes. The installed internet stack protocol had the same protocol as the suggested algorithm. The fourth node was one of the algorithms outlined in Table 1 with a parameter setting as highlighted in Table 2. Each Si-Di represented the dedicated flow: S1-D1 represents the first flow, S2-D2 represents the second flow, and so on.

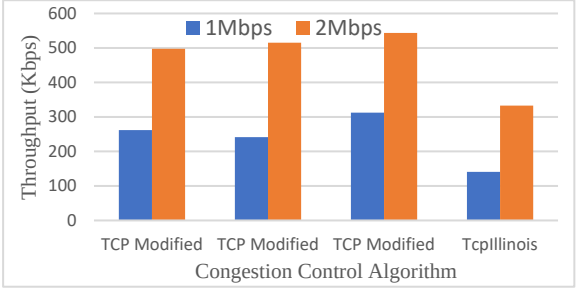
The bar charts in Figure 4 show the average throughput when the bottleneck link speed was set to 1 Mbps and 2 Mbps. Figure 4a shows that TCP NewReno flows for a 1 Mbps bottleneck link occupied only 16.5% of the link. Furthermore, the 2 Mbps bottleneck link only occupied 17.9% of the link. Jain's Fairness Index will be discussed below. Figure 4b–d show the throughput of the TCP variant for TCP BBR, TCP Illinois and TCP Yeah, respectively. The throughput of these flows was compared to TCP Modified throughput. For all the tests, the throughput of the suggested protocol was higher when acceptable fairness was maintained.



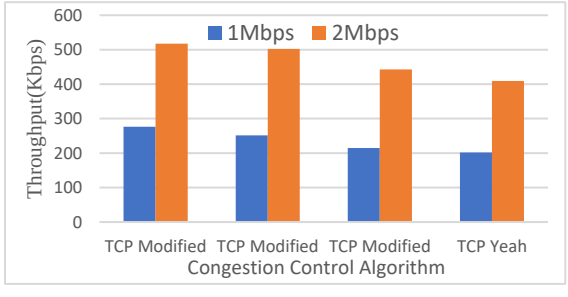
(**a**)



(**b**)



(**c**)



(**d**)

Figure 4. (**a**) Throughput versus TCP Variant (3 TCP Modified and 1 TCP NewReno); (**b**) Throughput versus TCP Variant (3 TCP Modified and 1 TCP BBR); (**c**) Throughput versus TCP Variant (3 TCP Modified and 1 TCP Illinois); (**d**) Throughput versus TCP Variant (3 TCP Modified and 1 TCP Yeah).

The other evaluation metric is the fairness index. Jain's Fairness method is the optimal method to calculate this index. Equation (3) shows this equation. Figure 5 shows the fairness index when there are three flows of the suggested algorithm with a single flow of the standard algorithm for both bottleneck link speeds.

$$\text{Jain's Fairness Index} = (\sum \text{throughput}(i))\ ^2/(N \times \sum (\text{throughput}(i))\ ^2) \quad (3)$$

where N represents the link number and throughput(i) is the ith node throughput.

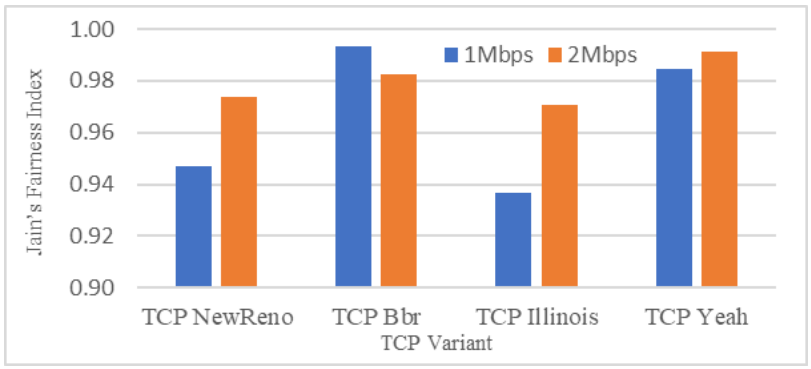
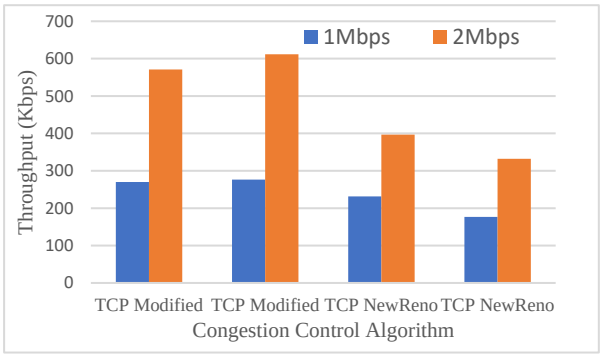


Figure 5. Jain's Fairness Index for the proposed algorithm for the other flows.

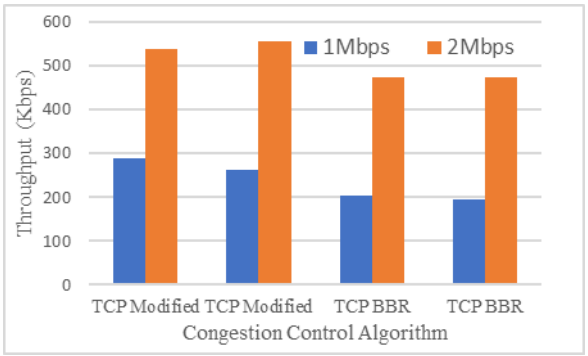
4.3. Third Experement: Proposed Approach with Another Protocols II

The topology used in this experiment is shown in Figure 2. This time, an intra-protocol test was applied with two sending nodes. The installed internet stack protocol had the same protocol as the suggested algorithm. The installed internet stack protocol for the remaining two nodes is highlighted in Table 1. The parameters outlined in Table 2 were applied in this simulation.

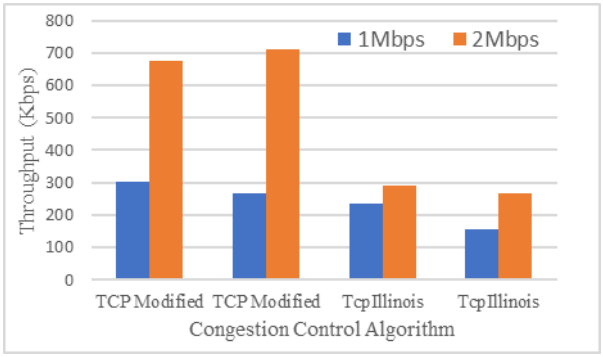
Figure 6 shows that the suggested algorithm outperformed the other algorithms, even though in the sharing traffic for this experiment, 50% of nodes were from other types of congestion control mechanisms. This result also indicates that the suggested algorithm performs better than others. Figure 7 shows the fairness index for this experiment.



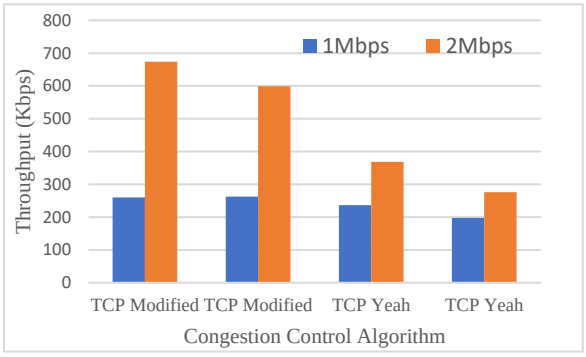
(**a**)



(**b**)



(**c**)



(**d**)

Figure 6. (**a**) Throughput versus TCP Variant (2 TCP Modified and 2 TCP NewReno); (**b**) Throughput versus TCP Variant (2 TCP Modified and 2 TCP BBR); (**c**) Throughput versus TCP Variant (2 TCP Modified and 2 TCP Illinois); (**d**) Throughput versus TCP Variant (2 TCP Modified and 2 TCP Yeah).

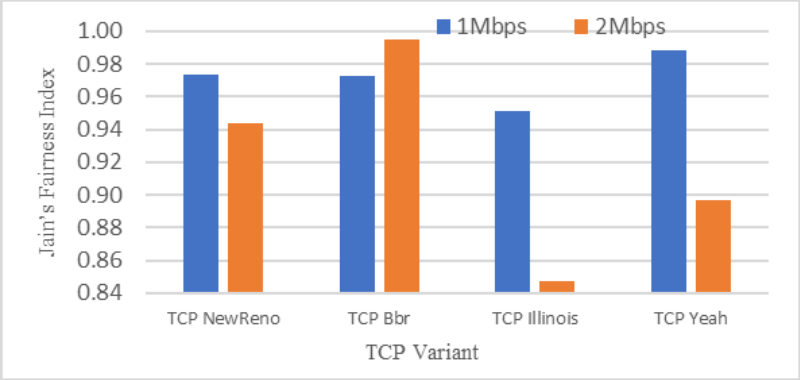


Figure 7. Jain's Fairness Index for the proposed algorithm with respect to other flows for 50% link sharing.

The competition for 50% of nodes of different flow degrades for both TCP Illinois and TCP Yeah were higher than under TCP BBR and TCP NewReno. TCP BBR was developed by Google in 2016 and is currently used by their servers. IoT devices such as esp32 and esp8266 implement lightweight TCP/IP algorithms in their Real Time Operating Systems (RTOS). In turn, RTOS implements most of the TCP NewReno algorithm, such as Slow Start, congestion avoidance, congestion detection and selective acknowledgement. Being friendly to these algorithms will likely have several advantages.

4.4. Forth Experement: Proposed Aproach with Wireless Access Link

In this simulation, it is very important to look at how well the proposed CCA works when the channel is wireless. The sender nodes were WIFI nodes connected to a gateway router (R1) over the WIFI channel. Figure 8 depicts the network produced by this simulation. Figure 9 shows the throughput of each configuration and Figure 10 shows Jain's Fairness Index for each. The results show that the suggested protocol also performs well in a wireless environment. This test was conducted as follows: half of the sending nodes are using the TCP Modified Internet Stack Protocol, while the other half are using one of the standard protocols listed in Table 1. The simulation begins with S1, S2 having TCP Modified and S3, S4 having TCP New Reno; the bottleneck link speed is set to 1 Mbps. After that, the bottleneck link speed is set to 2 Mbps, and the simulation is also conducted. The recorded result of the throughput is depicted in Figure 9a. The same procedure was conducted for Figure 9b–d.

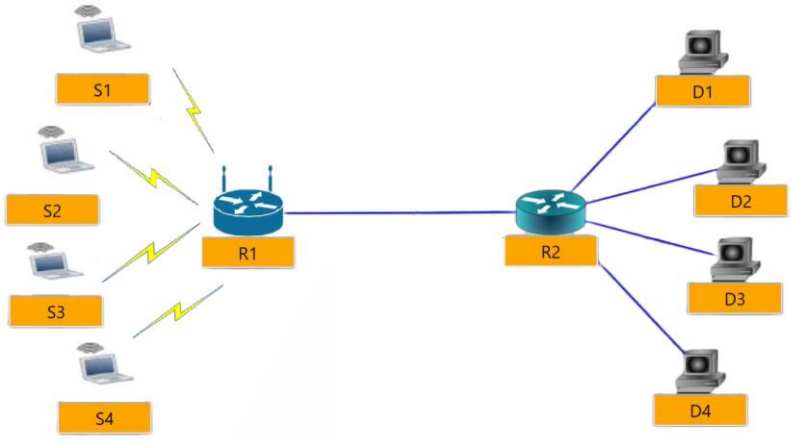
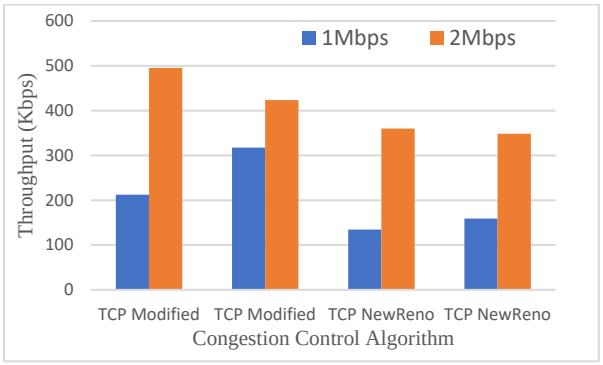
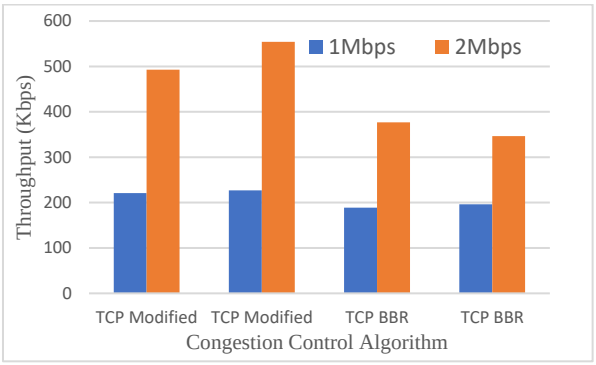


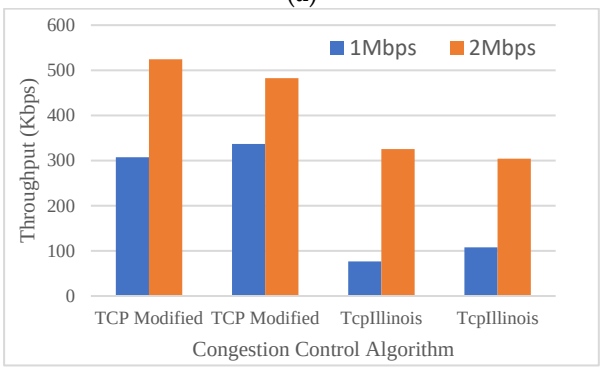
Figure 8. Testbed (dumbbell topology) with wireless sender side nodes for the fourth experiment.



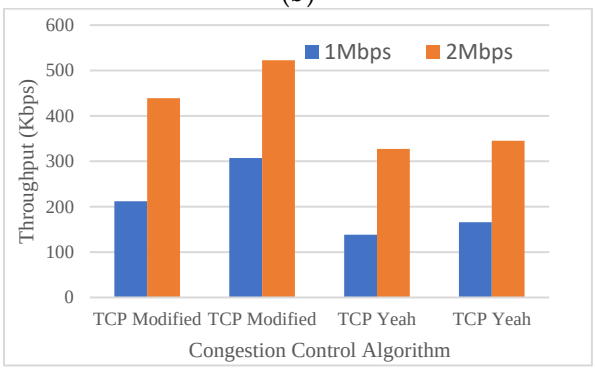
(**a**)



(**b**)



(**c**)



(**d**)

Figure 9. (**a**) Throughput versus TCP Variant for the wireless channel (2 Modified and 2 TCP NewReno); (**b**) Throughput versus TCP Variant for the wireless channel (2 Modified and 2 TCP BBR); (**c**) Throughput versus TCP Variant for the wireless channel (2 Modified and 2 TCP Illinois); (**d**) Throughput versus TCP Variant for the wireless channel (2 Modified and 2 TCP Yeah).

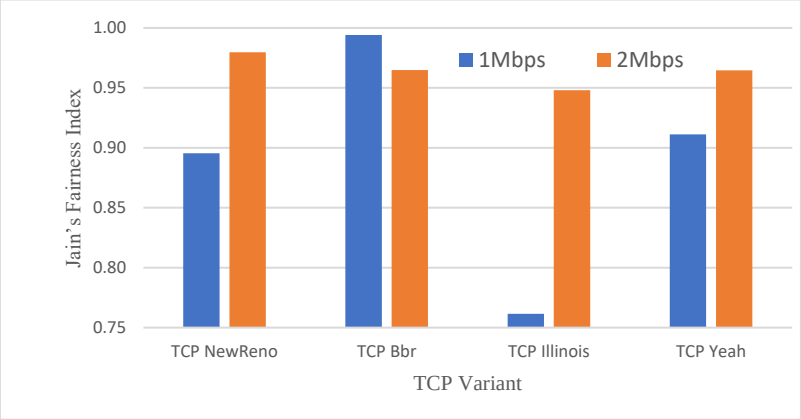


Figure 10. Jain's Fairness Index for the proposed algorithm for other flows at 50% link sharing for wireless channels.

4.5. Fifth Experement: Inter-Protocol Approaches

For this simulation, inter-protocol fairness was tested for both wired and wireless channels. The topologies used were the same as those shown in Figure 2 (for wire channel) and Figure 8 (for wireless channel). In the first step, the topology was set to the wired dumbbell topology, as depicted in Figure 2. In the second step, the parameters were adjusted as outlined in Table 2. In the third step, the TCP variant was set to one of the variants listed in Table 1. The test was then performed and all previous stages for all TCP variations were repeated. The findings were logged, and the previous procedures outlined in Figure 8 were repeated. Figure 11 shows each result's Jain's Fairness Index. The competition here was between TCP BBR and the proposed algorithm. The wired channel gained about 0.6% of link utilization with an expense of 0.3% of fairness. Conversely, when the link is wireless, which is normal for IoT devices, the reduction was 0.4% of link utilization and fairness increased by 0.1%.

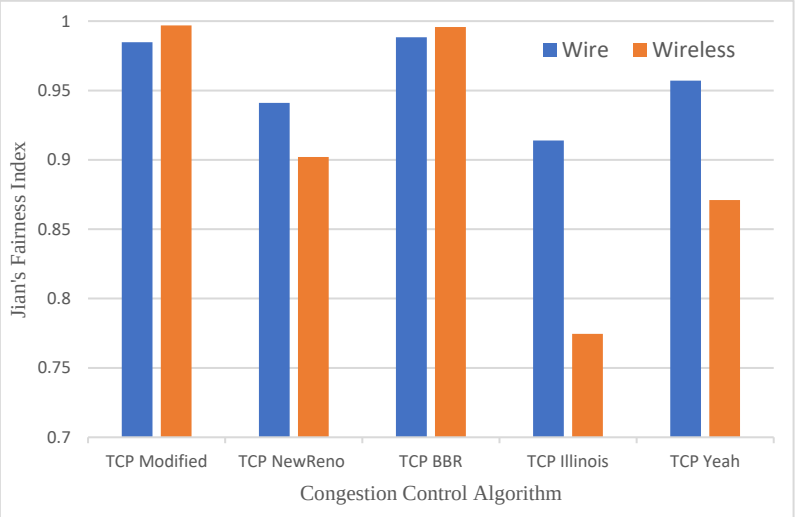


Figure 11. Jain's Fairness Index for the proposed algorithm for inter-protocol fairness for wired and wireless channels.

5. Conclusions and Future Work

This paper proposes an effective TCP CCA that can be installed on IoT devices. The algorithm aims to detect the network status and consider these parameters when setting new congestion windows. This algorithm uses two variables to set the congestion window. These variables do not fully explore the network situation and provide an optimal congestion window as in TCP BBR; however, TCP BBR suffers from fairness with different

RTT. When RTT is high, the proposed algorithm's throughput may suffer. Unlike other loss based CCAs, the effective algorithm looks at the increase in an end-to-end delay to ensure that the CWND does not rise aggressively and affect other intra-protocols. The algorithm was successfully tested, and the results confirmed that it outperforms other standard CCAs in terms of throughput while also maintaining fairness within an acceptable range. While the proposed algorithm performed better in tests than other standard CCAs, it has not been tested for different devices capabilities, such as different RTT for different flows and traffic types. In future work, we plan to extend the work of the algorithm to be able to efficiently deal with single segment size, which is an extension to the algorithm since it needs to deal with RTO.

Author Contributions: Conceptualization, H.H.H. and Z.T.A.; Methodology, H.H.H. and Z.T.A.; Software, H.H.H. and Z.T.A.; Validation, , H.H.H. and Z.T.A.; Formal analysis, H.H.H. and Z.T.A.; Investigation, H.H.H. and Z.T.A.; Resources, H.H.H. and Z.T.A.; Data curation, H.H.H. and Z.T.A.; Writing – original draft, H.H.H. and Z.T.A.; Writing – review & editing, H.H.H. and Z.T.A.; Visualization, H.H.H. and Z.T.A.; Supervision, Z.T.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not Applicable, the study does not report any data.

Acknowledgments: This work received no support.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Alisa, Z.T.; Salman, A.A. Improving the Network Lifetime in Wireless Sensor Network for Internet of Thing Applications. *Al-Khwarizmi Eng. J.* **2019**, *15*, 79–90.
2. Maha, Y.H.; Dheyaa, J.K. Efficient energy management for a proposed integrated internet of things-electric smart meter (2IOT-ESM) system. *J. Eng.* **2022**, *28*, 108–121.
3. RFC Editor. Available online: https://www.rfc-editor.org/rfc/rfc793 (accessed on 2 January 2023).
4. Jacobson, V. Congestion avoidance and control. ACM SIGCOMM *Comput. Commun. Rev.* **1988**, *18*, 314–329.
5. Mathis, M.; Mahdavi, J.; Floyd, S.; Romanow, A. TCP selective acknowledgement options. *RFC2018* **1996**. Available online: https://www.rfc-editor.org/rfc/rfc2018.html (accessed on 3 March 2023).
6. Floyd, S.; Henderson, T. The NewReno modification to TCP's fast recovery algorithm. *RFC2582* **1999**. Available online: https://www.rfc-editor.org/rfc/rfc2582 (accessed on 3 March 2023).
7. Xu, L.; Harfoush, K.; Rhee, I. Binary increase congestion control (BIC) for fast long-distance networks. In Proceedings of the IEEE INFOCOM 2004, Hong Kong, China, 7–11 March 2004; pp. 2514–2524.
8. Brakmo, L.S.; O'Malley, S.W.; Peterson, L.L. TCP Vegas: New techniques for congestion detection and avoidance. In Proceedings of the Conference on Communications Architectures, Protocols and Applications, London, UK, 31 August–2 September, 1994; pp. 24–35.
9. Song, K.T.J.; Zhang, Q.; Sridharan, M. Compound TCP: A scalable and TCP-friendly congestion control for high-speed networks. In Proceedings of the PFLDnet 2006, Available online: https://www.dcs.gla.ac.uk/~lewis/CTCP.pdf (accessed on 3 March 2023)
10. Wang, J.; Wen, J.; Zhang, J.; Han, Y. TCP-FIT: An improved TCP congestion control algorithm and its performance. In Proceedings of the 2011 Proceedings IEEE INFOCOM, Shanghai, China, 10–15 April 2011; pp. 2894–2902.
11. Liu, S.; Başar, T.; Srikant, R. TCP-Illinois: A loss and delay-based congestion control algorithm for high-speed networks. In Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools, Pisa, Italy, 11–13 October 2006; pp. 55–es. https://doi.org/10.1145/1190095.1190166.
12. Wang, Z.; Zeng, X.; Liu, X.; Xu, M.; Wen, Y.; Chen, L. TCP congestion control algorithm for heterogeneous Internet. *J. Netw. Comput. Appl.* **2016**, *68*, 56–64.
13. Baiocchi, A.; Castellani, A.P.; Vacirca, F. YeAH-TCP: Yet another highspeed TCP. In Proceedings of the PFLDnet, February 2007; Volume 7, pp. 37–42. Available online: https://www.researchgate.net/profile/Andrea-Baiocchi/publication/228561173_YeAH-TCP_Yet_another_highspeed_TCP/links/00b7d51ac57e095e39000000/YeAH-TCP-Yet-another-highspeed-TCP.pdf (accessed on 3 March 2023)
14. Kelly, T. Scalable TCP: Improving performance in highspeed wide area networks. ACM SIGCOMM *Comput. Commun. Rev.* **2003**, *33*, 83–91.
15. Cardwell, N.; Cheng, Y.; Gunn, C.S.; Yeganeh, S.H.; Jacobson, V. Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue* **2016**, *14*, 20–53.

16. Zhang, H.; Zhu, H.; Xia, Y.; Zhang, L.; Zhang, Y.; Deng, Y. Performance analysis of BBR congestion control protocol based on NS3. In Proceedings of the 2019 Seventh International Conference on Advanced Cloud and Big Data (CBD), Suzhou, China, 21–22 September 2019; pp. 363–368.
17. Verma, L.P.; Kumar, M. An IoT based congestion control algorithm. *Internet Things* **2020**, *9*, 100–157. https://doi.org/10.1016/j.iot.2019.100157.
18. Chappala, R.; Anuradha, C.; Murthy, P.S.R.C. Adaptive Congestion Window Algorithm for the Internet of Things Enabled Networks. *Int. J. Adv. Comput. Sci. Appl.* **2021**, *12*, 105–111.
19. Gupta, A.K.; Singh, D.; Singh, K.; Verma, L.P. STCP: A Novel Approach for Congestion Control in IoT Environment. *J. Inf. Technol. Manag.* **2022**, *14*, 44–51.
20. Lim, C. Improving congestion control of TCP for constrained IoT networks. *Sensors* **2020**, *20*, 4774.
21. Betzler, A.; Gomez, C.; Demirkol, I.; Paradells, J. Congestion control in reliable CoAP communication. In Proceedings of the 16th ACM International Conference on Modeling, Analysis & Simulation of Wireless and Mobile Systems, Barcelona, Spain, 3–8 November 2013; pp. 365–372.
22. Aljubayri, M.; Peng, T.; Shikh-Bahaei, M. Reduce delay of multipath TCP in IoT networks. *Wirel. Netw.* **2021**, *27*, 4189–4198.
23. Internet of Things with ESP32. Available online: http://esp32.net/ (accessed on 22 March 2023).
24. Espressif. Available online: https://www.espressif.com/en/products/socs/esp8266/ (accessed on 22 March 2023).
25. Pokhrel, S.R.; Qu, Y.; Gao, L. QoS-aware personalized privacy with multipath TCP for industrial IoT: Analysis and design. *IEEE Internet Things J.* **2020**, *7*, 4849–4861.
26. Alisa, Z.T.; Qasim, S.R. A Fuzzy based TCP Congestion Control for Wired Network. *Int. J. Comput. Appl.* **2014**, *975*, 8887.
27. Soud, N.S.; Al-Jamali, N.A.S. Intelligent Congestion Control of 5G Traffic in SDN using Dual-Spike Neural Network. *J. Eng.* **2023**, *29*, 110–127.
28. Chu, J.; Dukkipati, N.; Cheng, Y.; Mathis, M. Increasing TCP's initial window. *RFC6928* **2013**. Available online: https://www.rfc-editor.org/rfc/rfc6928.html (accessed on 3 March 2023).
29. Bilal, S.M.; Othman, M. A performance comparison of network simulators for wireless networks. *arXiv* **2013**, arXiv:1307.4129..
30. Kabir, M.H.; Islam, S.; Hossain, M.J.; Hossain, S. Detail comparison of network simulators. *Int. J. Sci. Eng. Res.* **2014**, *5*, 203–218..
31. NS-3 Network Simulator. Available online: https://www.nsnam.org/ (accessed on 2 January 2023).