Congestion Control Enhancement Over Wireless Networks

By

Taha Saedi

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Supervisor: Hosam El-Ocla

Department of Computer Science

Lakehead University

May 2019

Copyright © Taha Saedi

Abstract

In this thesis, we analyze the performance of wireless LAN networks subject to random loss. In this regard, we use a congestion control technique that has been introduced in a previous study, titled TCP Congestion Control Enhancement for Random Loss (TCP CERL). TCP CERL is a sender-side modification of TCP Reno protocol. TCP CERL is an end-to-end technique that achieves high performance over wireless links and does not decrease the congestion window and slow start threshold if the random loss is detected. TCP CERL assumes a static threshold (A) equal to 0.55 which does not perform well when considering a heavy traffic load compared to new protocols. In this thesis, we propose a modified version of TCP CERL, called TCP CERL PLUS (TCP CERL+, in short). TCP CERL+ works similarly as TCP CERL, but its main idea is to use a dynamic threshold (A) in terms of Round-Trip Time (RTT) rather than static threshold. By doing so, we employ the average RTT and the minimum RTT measurements made over the connection to evaluate the queue length of the bottleneck link. In this thesis, we compare TCP CERL and TCP CERL+ with TCP New Jersey+, TCP mVeno, TCP Westwood+, TCP Cubic, TCP YeAH, and TCP NewReno by using Network Simulation NS-2. The results will show that CERL+ outperforms CERL when there are many users and Two-way transmission.

Publications

[1] T. Saedi and H. El-Ocla, "Performance analysis of TCP CERL in wireless networks with random loss," in *Proc*. 32nd Canadian Conference on Electrical and Computer Engineering, Edmonton, AB, Canada, 5-8 May 2019.

Acknowledgments

My sincere appreciation is extended to my supervisor, Dr. Hosam El-Ocla for his direct supervision, guidance and leadership in this interesting research area throughout the years of master's and the preparation of this thesis. I would also like to extend my appreciation to the faculty members of the Computer Science department for all their help and support.

Above all, I give the utmost gratefulness and thanks to the Almighty Allah (God) for the patience, endurance, and determination. He has granted me for the completion of this academic research.

Dedication

I dedicate my dissertation work to my father and mother (Mr. Faraj and Mrs. Amina) who have planted the seed in me to pursue this work, and encouraged me all these years

I dedicate my dissertation work to my brothers (Siraj and Hakim) and lovely sisters (Nissren,Yasmin ,Mariam), who have always been by my side with their unconditional support.

Table of Contents

List of figures

Chapter 1

1 Introduction

Over the past few decades, considerable progress has been made in the field of mobile computing in terms of increased access to the Internet thanks to continuous technology advancement. There have been considerable developments in wireless local area networks (WLANs), as well as in cellular networks.

However, there are many problems related to wireless networks that affect the performance of data communication such as multimedia traffic, medium access control, and random loss.

In the transmission of multimedia data over a wireless network, there are two main issues that affect the performance of the network:	1) Energy-efficiency: Devices like laptops and mobile phones have some constraints, such as device size, energy consumption, and communication bandwidth. Therefore, wireless devices must be capable of handling several different classes of data traffic (e.g., voice and video) over a limited bandwidth. When the energy of a mobile device is low, the performance of the data transmission will decrease between this device and the access point [1]. 2) Quality of Service: "Provides the basis for modern high-bandwidth and real-time multimedia applications like teleteaching and video conferencing. The notion of quality of service originally stems from communication, but because of its potential in the allocation of all scarce resources, it has found its way into other domains, e.g., operating systems" [1].

Medium access control on the wireless network has many challenges that affect the transmission between nodes:	1) The problem of node mobility on an ad hoc network is that it affects the performance of transmission between them since they are not usually static (i.e., they move around most of the time). Therefore, data exchanged could possibly get disconnected if the mobility of nodes is high [2]. 2) Location constraint is a problem that occurs when there are several nodes in a geographical region, which will result in excess load on the wireless channel, thus causing much contention between nodes [3]. 3)Bandwidth efficiency is one of the most important resources in wireless networking. It "must be designed in such a way that the limited bandwidth should be utilized in an efficient manner. This approach keeps the involved control overhead to the lowest level possible and protects the network from overloaded " [4].

Another problem that affects the performance of data transmission through a wireless network is a random loss, whereby one or more packets fail to reach their destination, resulting in issues on network recourses. In this thesis, we focus on random loss problems of data packets in wireless networks.

1.1 Random Loss Problem

Random loss occurs when there are problems in wireless links or intermittent faults in hardwires [6]. Transmission errors are usually caused by random loss, and packets may be corrupted due to errors. Wireless media are more prone to transmission errors than the wired type because of noise and fading [7], high bit error rate, and hidden or exposed terminal problems [8].

Noise is one biggest problem affecting transmission between sender and receiver. Noise is unwanted random energy that travels with the signal across the elements of communication systems, causing distortion of transmitted information, resulting in the data not being correctly transmitted to the receiver.

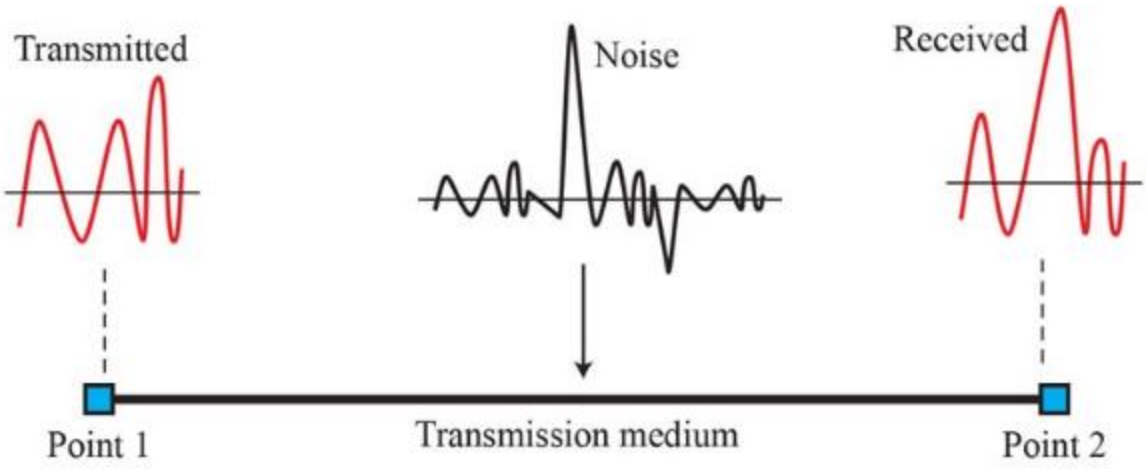

Figure 1.1: Noise [10]

In wireless communication (e.g., mobile cellular telephone), there are several varieties of noise "that could degrade the quality of communication, such as acoustic background noise, thermal noise, electromagnetic radio-frequency noise, co-channel interference, radio-channel distortion, echo, and processing noise" [9].

Hidden terminal causes in wireless communication as a result, the receiver station is not able to properly receive a segment from the sender station because of interference from other stations. This can occur when the sender is within receiver range but is not within the range of the station. For example, station A is within B station range, and C is also within B range, but C is not within a range. Therefore, when A sends a packet to B, and C sends a packet to B at the same time, a collision occurs because A and C are not within the same range [10]. The Hidden terminal problem usually happens in ad hoc network, since it does not require additional resources.

In wireless communication, fading happens result the transmission signal between the sender and receiver is distributed on multiple paths, therefore the signal receives by received will have some changes. "Multipath propagation can lead to fluctuations in the amplitude, phase, and angle of the signal received at a receiver" [11]. Figure 1.2 shows a multipath propagation.

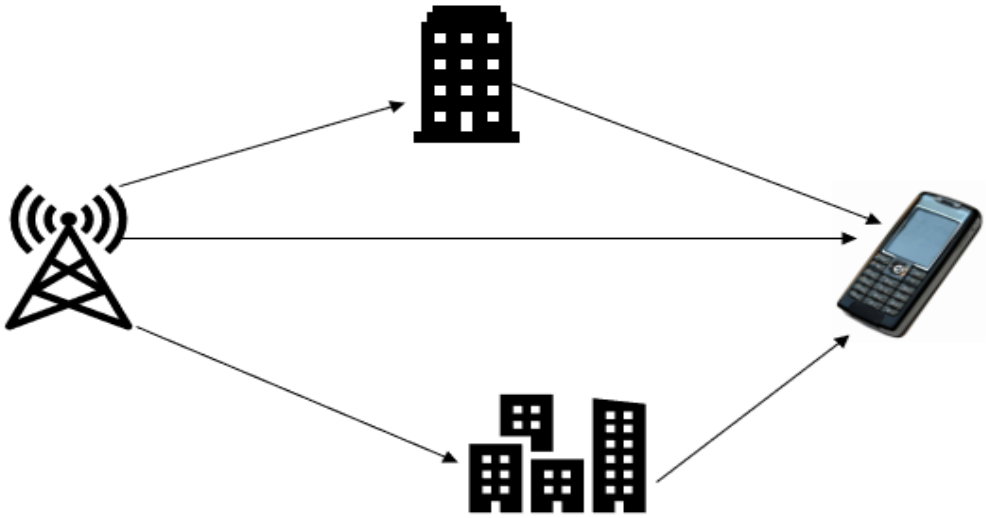

Figure 1.2: Multipath propagation

High bit error rate occurs in both wireless and wired communication whenever there is a difference between the percentage of transmitted data and the received data. It occurs when there are problems in the medium between sender and receiver, such as fiber links, ADSL, and cellular communication.

1.2 Congestion control theory

Nowadays, many the Internet users employ reliable transport protocols such as Transmission Control Protocol (TCP). TCP provides reliable and connection-oriented transport between applications [12]. It controls data flow on networks through both a sender-side congestion window and a receiver-side advertised window, which means that TCP does not send segments larger than the size of the congestion window of the sender, nor does it receive more segments that are advertised on the window of the receiver [13, 14]. Congestion window size relies on traffic on the network. When there is a high load on a router or link, the size of the congestion window becomes small; the size increases gradually depending on the network [15]. Packets of TCP are progressively acknowledged, which means that they reach their destination in a sequence. A duplicate packet acknowledgment signifies that there is an issue on the network. The sender detects a lost packet by receiving three duplicates [16]. In addition, TCP senders reveal a lost packet through timeouts when they do not receive an acknowledgment [17]. Retransmission times are consistently changed, depending on Round Trip Time (RTT) measurements.

Traffic overload on the routes results in congestion in the router queues, which increases delays and causes the packets to become lost. TCP increases progressively, and transmission rates rely on networks capability. If there are any packets lost, then TCP resends the lost packets and decreases the transmission rate [18].

Congestion control has two initial phases: slow start and congestion avoidance. Slow start begins the congestion window size with one segment; the congestion window size will increase by one segment with each acknowledgment received, and the segment size will double [19]. The transmission rate will be raised by the slow start until a loss is detected, if the receiver's advertised window is full, or until it reaches the slow start threshold [20]. If packet loss occurs due to congestion, the sender reduces the transmission rate in order to keep the network balanced. After reaching the slow start threshold, TCP progresses to the congestion avoidance stage, and the congestion window size will increase by one segment for each round trip_unless there is packet loss, or unless it times out [21]. Figure 1.3 outlines the slow start and congestion avoidance design, and Figure 1.4 depicts the congestion window threshold.

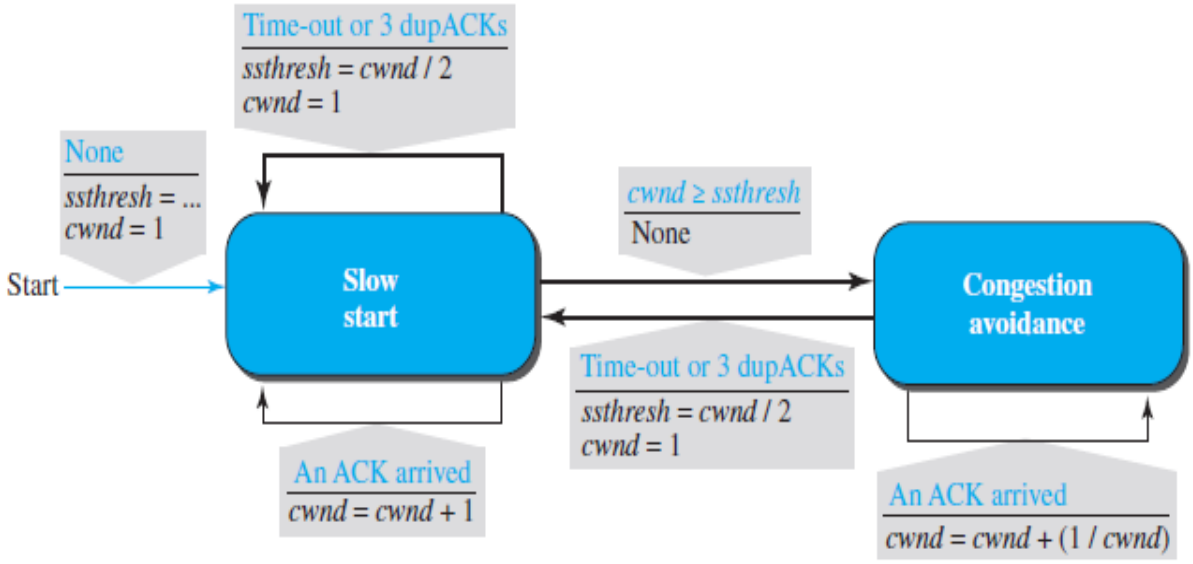

Figure 1.3: Start and congestion avoidance design [37]

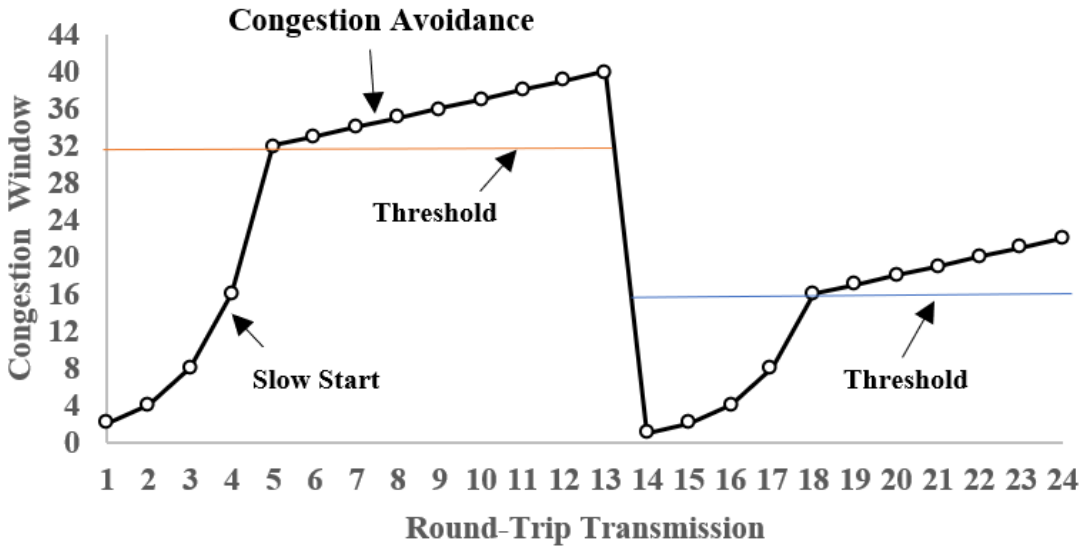

Figure 1.4: Congestion window threshold

TCP is considered to be full duplex protocol, which means that data flow could be in both directions [21]. This indicates that data may be sent from the sender or receiver, which is two-way communication, as shown in figure 1.5.

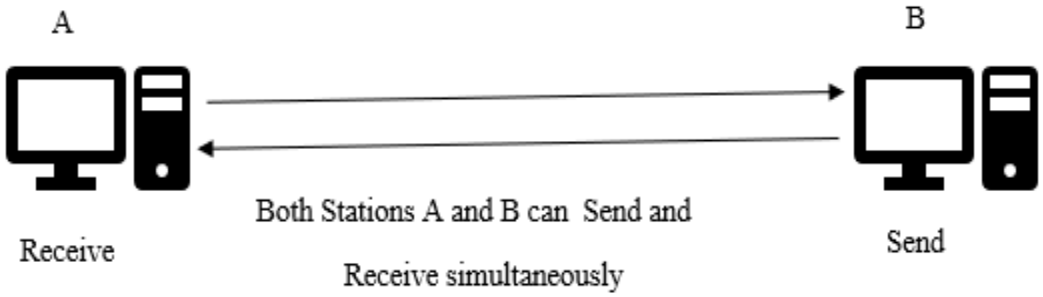

Figure 1.5: Full duplex protocol

TCP Tahoe [23] is the first version of TCP that utilized slow start and congestion avoidance. Another popular version of TCP is TCP Reno, which was developed from Tahoe and includes a fast recovery feature [24].

1.3 Random Loss vs congestion loss control Protocols

Several studies have been conducted to distinguish random loss from congestion. Biaz and Vaidya [25] used inter-arrival times at receiver point to identify the between losses. Although they solved the problem, this technique used only single wireless at end-to-end path, and high load on the network.

Mathis *et al.* [26] used selective acknowledgment (SACK) to prevent multiple losses. The receiver sends SACK segments to the sender for data that has been received successfully. The advantage of SACK is that it carries information about some packets seen so far, thereby assisting the sender by retransmitting only the actual packets lost. However, SACK's disadvantage is overhead (excess computation time) when there are a large number of users.

Keshav and Morgan [27] proposed a new technique to reduce overhead, titled Simple Method-to-Aid ReTransmissions (SMART). The idea of SMART is to develop selective acknowledgment. SMART mechanism behaves in a way where every acknowledgment holds the acknowledgment, and the sequence number of packets that started the acknowledgment will inform the sender that the packet has been received correctly. The problem of SMART is that it is still not fully capable of distinguishing between random loss congestion.

Other schemes proposed to distinguish between two losses by congruous. Samaraweera *et al.* [28] developed a technique for wireless network issues called Explicit Loss Notification (ELN). The idea behind ELN is to specify the source of random loss, whereby it allows the sender to retransmit the packet without decreasing the congestion window.

"An identical signal has been proposed to halt the congestion control at the source when a disconnection appears due to handover in cellular networks. The difficulty with this solution is that any packet corrupted at the link level is discarded before reaching TCP" [6].

Other researchers suggested improving the performance of TCP during the congestion loss more than the random loss in order to add more techniques on the network or at the sender to decrease the congestion on the network buffers. By decreasing the size of the queue in the nodes, it avoids retransmission loss, and the performance will be adequate, particularly for certain applications. Explicit Congestion Notification (ECN) [29] and TCP Vegas are new solutions that were developed to solve the problem of end-to-end delay. TCP Vegas uses RTT and congestion window size to calculate the packet in the router's buffer; if the number of packets is more than the threshold, TCP Vegas reduces the congestion window. In ECN, routers utilize an explicit signal to notify the source that there is congestion, rather than dropping the packets. If there is no issue at the senders, receivers or routers, then the congestion losses will reduce slightly, and other losses can be considered as a random loss for both ECN and Vegas. This means that if some queue lengths work well without interruption, they help to detect the congestion. Therefore, any losses are considered to be random for the source, and as a result, the sender retransmits the packet loss without decreasing the window. However, if the congestion loss was not clear in the medial node, the sender will then deal with loss in a serious way. Sometimes, the sender somewhat decreases the congestion window when the real loss occurs.

Floyd and Henderson created TCP NewReno in 1999. TCP NewReno checks to see if more than one segment is lost in the current window when three duplicate ACKs arrive. When TCP receives three duplicate ACKs, it retransmits the lost segment until a new ACK (not duplicate) arrives. If the new ACK defines the end of the window when the congestion was detected, TCP is certain that only one segment was lost.

Baiocchi et al. recommended the use of "Yet Another High Speed" TCP, whereby RTT estimation and loss detection predict network delay[31]. In every RTT, the congestion window decreases by .5 whenever 3 duplicate ACKs are received; the congestion window increases by 1 whenever a loss is identified. The default TCP algorithm is CUBIC TCP. Initially introduced by Rhee and Ha in 2008, it is the modified version of current TCP variants, and is currently used in maximum Linux OS. Rather than using a linear function, CUBIC TCP applies a cubic function to a congestion window increase in order to augment the scalability of a high-BDP network. It also uses BIC algorithm and HTCP's cubic function of the congestion window.

In order to improve fairness and TCP efficiency in wireless networks, Westwood+ TCP [32] was proposed, a revised sender side-only version of NewReno TCP. Westwood+ TCP's primary objective was to conduct an end-to-end estimation of available bandwidth for a TCP connection by accurately counting and correctly filtering the stream of ACK packets. That estimate is then applied to adaptively reduce both the congestion window size and the slow-start threshold size at the completion of the congestion phase. Consequently, Westwood+ TCP the classic multiplicative decrease paradigm is replaced by the adaptive decrease paradigm.

TCP mVeno is a new version of TCP Veno [33]. Its purpose is to make full use of the congestion information of all the subflows belonging to a TCP connection in order to adaptively adjust the transmission rate of each subflow. TCP New Jersey+ differs from TCP New Jersey, in that the main goal is to improve available bandwidth estimation and recovery technique. TC New Jersey+ guarantees high throughput via an increased congestion window when the sender reveals that a packet is lost, or due to retransmission timeout [34].

1.4 Proposed TCP Variant for Random Loss Solution

These variants explained above perform well to distinguish between congestion loss and random loss, however, we need to probe a better variant that can improve the performance of data transmission in wireless networks.

The motivation behind this thesis is to use a congestion control technique that has been introduced in a previous study, titled TCP Congestion Control Enhancement for Random Loss (TCP CERL). TCP CERL is a sender-side modification of TCP Reno protocol. The difference between CERL and other TCP variants is that CERL depends on the maximum sequence number of a segment during the fast recovery algorithm. CERL is an end-to-end technique that achieves high performance over wireless links and doesn't decrease the congestion window and slow start threshold if the random loss is detected. CERL assumes a static threshold (A), whereas it will not perform well when considering piggybacking flow and a heavy traffic load compared to TCP New Jersey+ and other new protocols.

In this thesis, we propose a modified version of TCP CERL, called TCP CERL PLUS (TCP CERL+, in short), which employs the average and minimum of RTT rather than only RTT in TCP CERL. We compare CERL and CERL+ with TCP New Jersey+, TCP mVeno, TCP Westwood+, TCP Cubic, TCP YeAh, and TCP NewReno by using Network Simulation NS-2. The results will show that CERL+ outperforms CERL when there are many users and a piggybacking flow.

1.5 Thesis Organization

The remainder of this thesis is organized as follows: Chapter 2 contains the background, literature review and piggybacking concept. CERL is explained in Chapter 3, and it is compared to other mentioned protocols. In Chapter 4, we examine CERL+ and compare it to other mentioned protocols. Chapter 5 is the conclusion of this thesis.

Chapter 2

Background and Literature Review

This chapter presents background information about certain protocols that are related to this study, as well as the concept of piggybacking.

2.1 General Information about TCP

TCP is a reliable, connection-oriented protocol in transport layers. Before starting the data transmission, TCP prepares three phases: establish the connection, send the data, and terminate the connection. When TCP establishes a connection, TCP uses a three-way handshake technique, as shown in Figure 2.1.

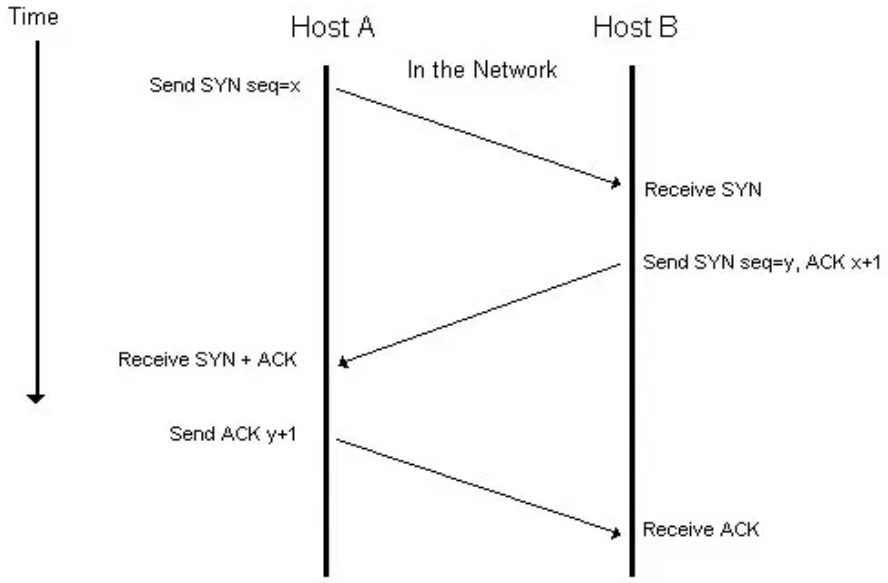

Figure 2.1: Three-way handshake [35]

Host A sets the SYN bit by transmitting a segment with sequence number x.

Host B replies by sending an ACK bit segment, and sets the sequence number x+1 SYN bit; it then sets sequence number y.

Host A sends back a segment with an ACK bit and sets the y+1 sequence number.

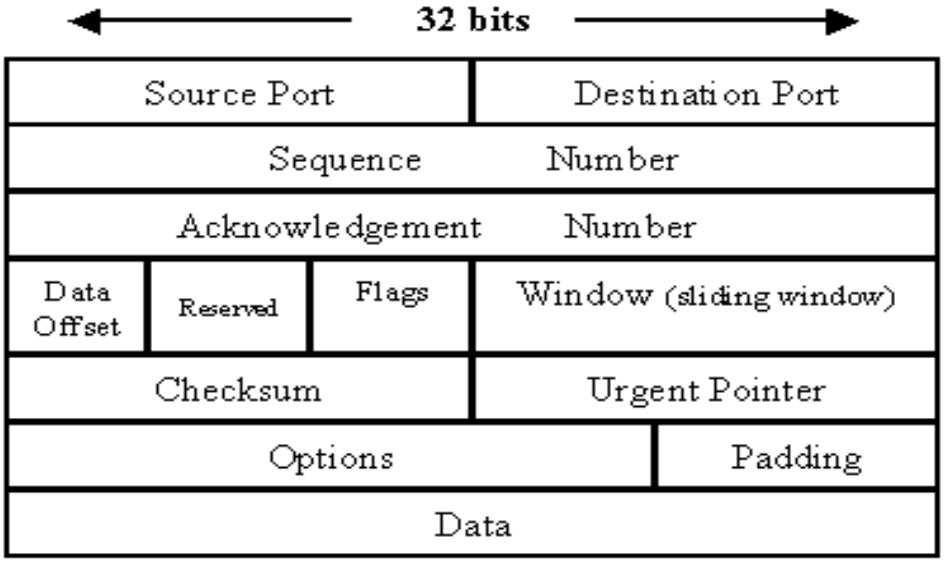

Figure 2.2: TCP header [36]

The TCP segment consists of a segment header and a data section. The header has ten required fields, as well as an optional field. The source port is for sending, and the destination port is for receiving; both are 16-bit fields. Sequence numbers are used by the source to indicate the packet number. The acknowledgment number is utilized to denote the next sequence number sent from the destination to the source. The data offset specifies the initial data. The reserved bit is a 6-bit field, which is set to 0.

The window size is a 16-bit field, which denotes the number of packets that the receiver can carry. Checksum is 16 bits, and it is equipped with an error detection feature. The urgent pointer is utilized when the URG bit is set.

2.2 TCP Protocols Overview

2.2.1 TCP Reno

TCP Reno has three main phases: Slow Start, Congestion Avoidance, and Fast Recovery. The purpose of this protocol is to treat the problems of congestion, timeout, and three duplicate ACKs. When a timeout occurs, TCP Reno sender changes to the slow start phase from either the congestion avoidance phase or the fast recovery phase [37].

In addition, the sender starts a new round if it remains in slow start. However, if the sender receives three duplicate ACKs when in either Slow Start or Congestion Avoidance, the sender changes to the fast-recovery phase and remains there as long as duplicate ACKs continue to arrive. The fast recovery works in the same way as a slow start (increases the congestion window exponentially), however, it begins the congestion window from slow start threshold plus three MSSs (Maximum Segment Size), rather than only one in the slow start phase. There are three cases that occur in fast recovery:

Firstly, if Reno continues to receive duplicate ACKs, it continues in fast recovery and increases the congestion window.

Secondly, when a timeout occurs, it reverts to Slow Start.

Lastly, if a new ACK arrives and is not duplicated, it moves to the congestion avoidance phase. Reno then reduces the congestion window size equal to that of the slow start threshold value. Refer to Figure 2.3 for a diagram of TCP Reno.

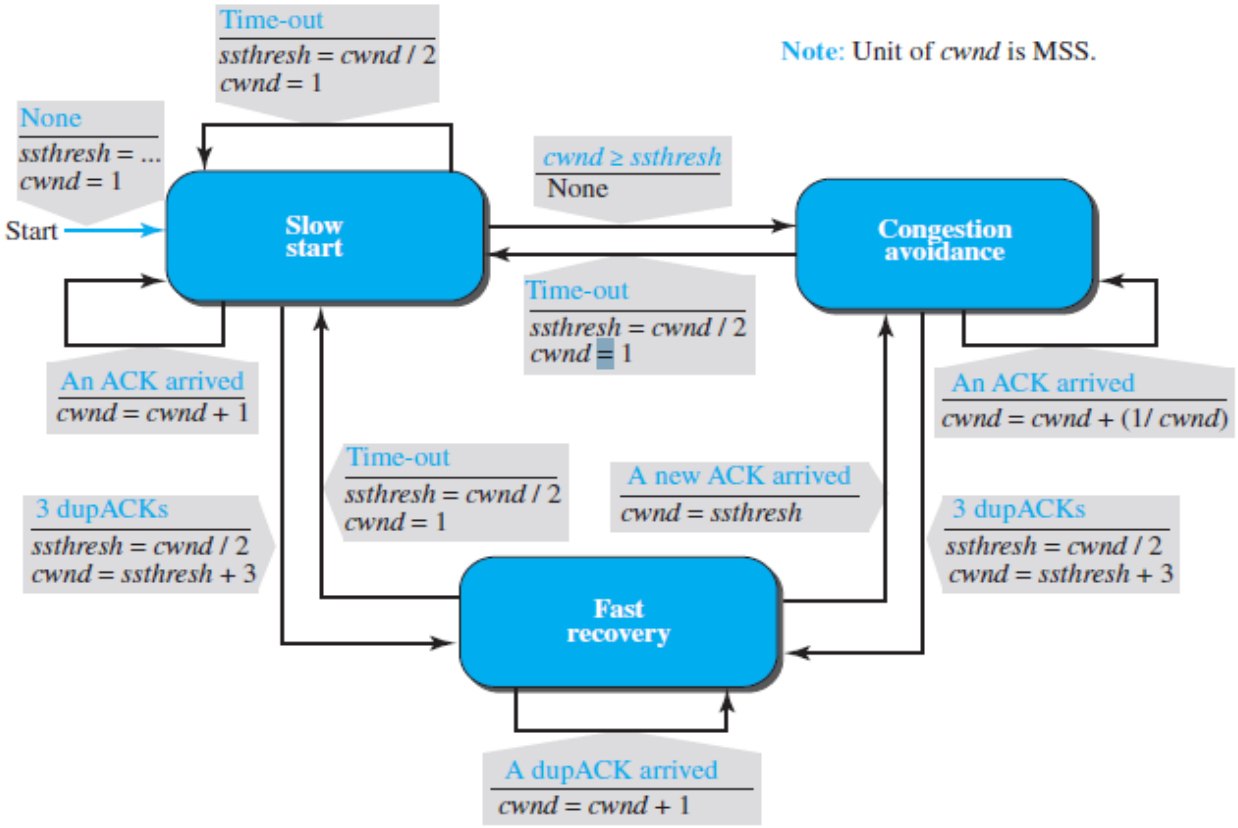

Figure 2.3: TCP Reno diagram [37]

Figure 2.4 shows the congestion window changes in slow start, congestion avoidance, and fast recovery states.

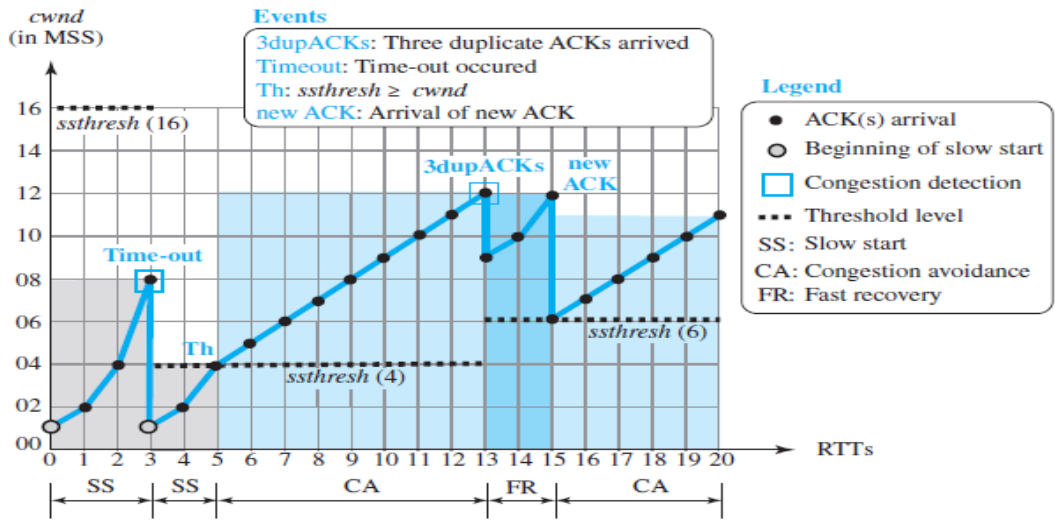

Figure 2.4: Congestion window changes [37]

2.2.2 TCP NewReno

TCP NewReno is a new version of TCP Reno[30]. TCP NewReno has a new mechanism in the fast recovery phase called additive increase multiplicative decrease (AIMD), which means that after slow start phase, the congestion window size increases as tooth pattern, as shown in Figure 2.5.

TCP NewReno checks to see if more than one segment is lost in the current window when three duplicate ACKs arrive. When TCP receives three duplicate ACKs, it retransmits the lost segment until a new ACK (not duplicate) arrives. If the new ACK defines the end of the window when the congestion was detected, TCP is certain that only one segment was lost. However, if the ACK number defines a position between the retransmitted segment and the end of the window, it is possible that the segment defined by the ACK is also lost. NewReno TCP retransmits this segment to avoid continually receiving more duplicate ACKs [37].

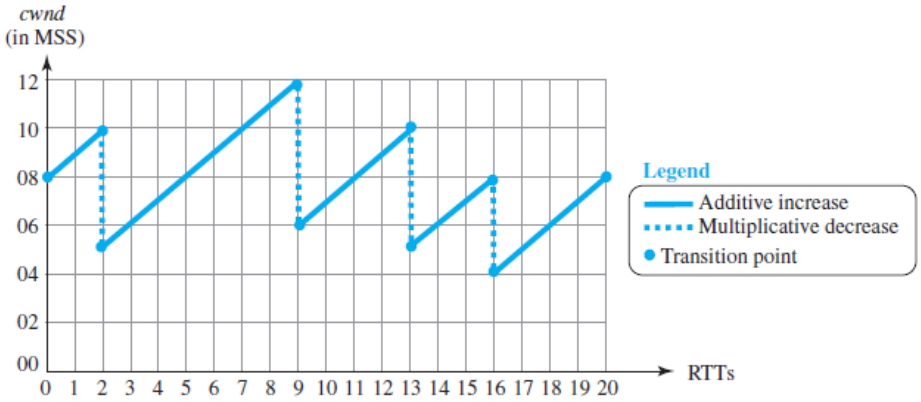

Figure 2.5: NewReno tooth pattern [37]

2.2.3 TCP YeAh

YeAh-TCP (Yet Another High-speed TCP) utilizes a mixed loss/delay approach to calculate congestion windows, which means that RTT estimation and loss detection are used to predict network delay [38].

The target of this protocol is to reach high efficiency and to decrease link loss, which keeps the network load lower. TCP YeAh has two main modes: fast mode and slow mode. In fast mode, if a network connection is still not fully used, the TCP YeAh increases the congestion window (similar to STCP - Scalable TCP) [39]. However, when in slow mode, TCP YeAh works like TCP Reno by in increasing the congestion window.

The phase is determined according to the estimated number of packets that are present in the bottleneck queue.

RTT_{base} is the minimum RTT (which is the predicted propagation delay measured by the sender) and RTT_{min} is the minimum RTT (which is assumed to be in the current data window of the cwnd packets), then the estimated queue delay at that time will be determined as follows:

$$RTT_{queue} = RTT_{min} - RTT_{base} \quad (2,1)$$

From the RTT_{queue}, the number of packets that were placed in the queue by the flow can be estimated to be:

$$Q = RTT_{queue} * G = RTT_{queue} * \left(\frac{cwnd}{RTTmin}\right) \quad (2,2)$$

"Where G is the goodput. We can also evaluate the ratio between the queuing RTT and the propagation delay L = RTTqueue/RTTbase, that indicates the network congestion level. Note that RTTmin is updated once per window of data"[38].

When $Q < Q_{max}$ and $L < 1/\varphi$, this indicates that the algorithm is in fast mode. If not, it is in slow mode. Parameters Q_{max} and φ are tunable. Q_{max} is the highest allowable number of packets that can be stored in the buffers in a single flow.

$1/\varphi$ is the highest degree of buffer congestion in terms of BDP. In slow mode, there is a preventive decongestion algorithm that is initiated: any time $Q > Q_{max}$, there is a reduction in the congestion window by Q, and ssthresh is set to cwnd/2. In addition, because $RTTmin$ is processed only one time for each RTT, one RTT represents the decongestion granularity.

Whenever $Q > Q_{max}$, YeAH-TCP tries to eliminate the packets from the queue, the delay in the queue continues to grow due to the greediness of the Reno flows, whereby the buffer gets overloaded. When that happens, YeAH-TCP will infrequently remain in the fast mode; rather, it spends most of its time in slow mode. On the other hand, in the case of the non-greedy flows that compete against each other (e.g., those that initiate the preventive decongestion algorithm), the YeAH-TCP algorithm changes the phase from fast to slow at any time when the buffer queue increases beyond Qmax and reverts to Fast once preventive decongestion comes into effect [38].

2.2.4 TCP BIC

Binary Increase Congestion (TCP BIC) [40] is a TCP congestion-control that was recommended for high-speed networks with high latency. It consists of two phases: the binary search increase phase, and the additive increase phase.

The binary search phase is utilized by the sender to improve congestion window performance. If the packet is lost, TCP BIC uses factor b to reduce the congestion window size, and BIC sets the value of $cwnd_{max}$ to the value of congestion prior to the loss and sets the value of $cwnd_{min}$ to the value of congestion after the loss to $cwnd_{max} * b$ [40].

If $cwnd_{min}$ and $cwnd_{max}$ are different at a medial point, and $cwnd_{min}$ is lower than the threshold, BIC begins a binary search increase at the mid-point of the congestion window. Otherwise, BIC increases the window size one by one, depending on the number of ACKs received.

When there is no packet loss detected by the sender, the BIC sender sets the value of the actual window to a new minimum window. However, if packet loss is detected, then the BIC sets the value of the actual window to a new maximum window[40].

BIC continues to increase the window by one until it becomes less than the minimum slow start threshold, and the congestion window is equal to the maximum value if the window continues to increase more than the maximum window. BIC switches to a new phase (max probing); In other words, it uses the inverse of the binary search phase first and uses the increase afterward [41].

2.2.5 TCP Cubic

TCP Cubic is descendent of TCP BIC. It is used to solve the problem bandwidth delay product (BDP), utilizing a cubic function rather than a linear congestion window function for congestion control scalability and stability under fast and long-distance networks.

A cubic function is applied by Cubic to determine the time that has elapsed since the last congestion episode. Although the majority of standard TCP algorithms apply a convex increase formula following an episode of loss when the window queue continues to increase, Cubic applies a cubic function to both convex and concave types of increase[42].

Following a loss event, once a window has been decreased, "W_{max}" appears in the window. A multiplicative reduction of the congestion window occurs by a factor of β (where β is the window decrease constant, and the normal fast recover and TCP retransmission [42].

Once it switches from the fast recovery phase to congestion avoidance, the window begins to expand thanks to the cubic function's concave profile. Because the plateau of the cubic function is set at W_{max}, the window size keeps increasing until it reaches the W_{max} level[42]. The cubic function then transforms into a convex profile, which reinitiates the development of the convex window.

The switch from concave to convex window enhances network stability and protocol during high traffic times on the network due to the stable size of the window[42]. A plateau is created around Wmax during times of maximum network use under a steady state. Because the majority of Cubic's samples of window sizes are at (or close to) W_{max}, it results in improved protocol stability

and enhanced function during high network traffic. It appeared that the protocols that had convex growth functions had the most significant increase at the saturation point, which caused substantial packet losses.

The function used by Cubic to determine window growth is as follows:

$$W(t) = C\,(t - K)^3 + W_{max} \quad (2,3)$$

C is a Cubic parameter, t is the elapsed time from the last window reduction, and K is the time elapsed by the above function to increase *W* to W_{max} once there was no new episode[42]. This is calculated as follows:

$$K = \sqrt[3]{\frac{Wmax\ \beta}{C}} \quad (2,4)$$

Once an ACK is sent during the congestion avoidance phase, the rate of increase in the window is calculated by Cubic during the following RTT period. Using Eq.(2,3), it establishes the candidate target value of the congestion window as *W(t + RTT)*[42]. If, for example, the size of the current window size is cwnd, Cubic operates in three modes, depending on the value of cwnd.

If *cwnd* is less than the window size that would be reached by TCP (standard) at time t following a loss event, Cubic would be in the TCP mode (see notation below on determining standard TCP window size in terms of time *t*). If *cwnd* is smaller than W_{max}, this signifies that Cubic is in the concave zone, whereas if cwnd is larger than W_{max}, Cubic is therefore in the convex zone[42].

Cubic has certain drawbacks, such as the fact that it does not adequate throughput when there are a large number of packets lost on the network.

2.2.6 TCP Westwood

TCP Westwood is a sender-side modification of the TCP protocol that develops the performance of end-to-end congestion control for both wireless and wired networks. The difference between Reno and Westwood is that Westwood is capable of applying more techniques to identify the type of loss, a feature that is not available with TCP Reno.

TCP Westwood utilizes a bandwidth estimate (BWE) mechanism to control the congestion window and slow start threshold. A Westwood sender checks the average ACKs received to estimate the current bandwidth of TCP connection. "When an ACK is received by the source, it conveys the message that the amount of data corresponding to a specific transmitted packet was delivered to the destination" [43].

With TCP Westwood, all segments have the same size when the source receives three duplicate ACKs, or when a timeout occurs due to congestion. The Westwood source uses the BWE to set the congestion window and slow start threshold.

Westwood implements [43] the pseudo code algorithm for three duplicates, as:

if (n DUPACKs are received)

ssthresh = (BWE * RTTmin)/seg_size;

if (*cwnd* > ssthresh) /* congestion avoid. */

cwnd = ssthresh;

Westwood implements the pseudo code algorithm for a timeout, as:

if (timeout expires)

ssthresh = (BWE * RTTmin)/seg_size;

if (ssthresh < 2)

ssthresh = 2;

end if;

cwnd = 1;

end if

To estimate (BWE), Westwood uses the following equation:

$$b_k = d_k / \Delta k = d_k / t_{k\text{-}tk\text{-}1} \qquad (2,5)$$

Where b_k is the estimated bandwidth, Δk is interarrival ACKs, dk is the transmitted bytes and $t_{k\text{-}tk\text{-}1}$ ACK received at source[43].

2.2.7 TCP Westwood+

TCP Westwood+ is a sender-side-only modification of the TCP Reno protocol that develops the performance of congestion control in wireless networks. TCP Westwood+ is a new enhanced version of TCP Westwood [45]; it is simpler and capable of estimating available bandwidth.

Westwood+ algorithm is based on an end-to-end approximation process of the amount of available bandwidth on the connection path of the TCP [45]. This estimate is acquired by filtering the flow of ACK packets that are being returned, and this is used to adjust the control windows whenever congestion on the network is occurring. Specifically, upon receiving 3 DUPACKS, the congestion window (cwnd), along with the slow start threshold (ssthresh), are set to the same setting as the estimated BWE, multiplied by the minimum round-trip time (RTT_{min}). When a coarse timeout expires, ssthresh is adjusted to its previous setting, whereas cwnd is set to one.

Westwood+'s pseudo code algorithm is as follows[45]:

a) When ACKs are received:

cwnd is increased according to the Reno algorithm; the end-to-end bandwidth estimate BWE is computed;

b) When 3 DUPACKs are received:

$ssthresh = \max(2, (BWE * RTT_{min}) / \text{seg\_size})$; $cwnd = ssthresh$;

c) Upon the expiration of coarse timeout:

$ssthresh = \max(2,(BWE * RTT_{min}) / \text{seg\_size})$; $cwnd = 1$;

According to the pseudo code, Westwood+ also increases the *cwnd* (similar to Reno) upon receiving ACKs. However, when congestion episode occurs, Westwood applies a special function of cwnd and ssthresh. Therefore, we can conclude that Westwood+ adheres to the paradigm of Additive-Increase / Adaptive-Decrease [45].

An interesting fact is that TCP Westwood+ has an adaptive decrease mechanism that enhances the stability of the standard TCP multiplicative decrease algorithm. Furthermore, the shrinking ability of the adaptive window sufficiently decreases the congestion window during heavy traffic, and decreases it only slightly during light traffic, or when losses occur that are unrelated to congestion (e.g., unreliable radio links)[45]. In addition, the control window is equipped with the ability to increase the allocation of available bandwidth to various TCP flows. In other words, one of the features in TCP Westwood+'s window setting is that it can track the estimated bandwidth. Due to an adequate estimation of what is considered to be "fair share", fairness improves. On the other hand, the following could occur:

the setting $cwnd = B * RTT_{min}$ sustains a transmission rate $(cwnd/RTT) = (B * RTT_{min})/RTT$

that is smaller than the bandwidth B, estimated at the time of congestion. Therefore, the flow in TCP Westwood+ empties the backlog after the setting, which allows space for coexisting flows in the buffers[45].

An end-to-end estimation of the "best-effort" available bandwidth is proposed by TCP Westwood+ by properly counting and filtering the flow of ACKs that are being returned[45]. For every RTT, a sample of available bandwidth is calculated $b_k = d_k / \Delta k$, where d_k represents the amount of data acknowledged during the last RTT $= \Delta k$. The amount d_k of is established by applying a proper counting process that considers delayed ACKs and duplicate ACKs[45].

Duplicate ACKs account for one delivered segment, whereas delayed ACKs account for two segments. A cumulative ACK accounts for one segment, or for whichever number of segments that exceed those that were already accounted for in previous duplicate acknowledgments.

2.2.8 TCP New Jersey

TCP New Jersey evolved from the TCP Jersey protocol to estimate available bandwidth. Its main components include Congestion Warning and Timestamp-based Available Bandwidth Estimation. Congestion Warning transfers the capacity of an intermediate router to the sender because it helps the sender to distinguish the reason for the packet losses.

The purpose of Timestamp-based Available Bandwidth Estimation is that any packet that reaches the receiver side obtains a timestamp with ACK that it sent back to the sender, rather than ACK reaching the time in TCP-Jersey[46]. The available bandwidth that is estimated by Timestamp-based Available Bandwidth Estimation is only slightly affected due to the reverse links state, compared to the Available Bandwidth Estimation feature in TCP Jersey.

One of the disadvantages in TCP New jersey is that if the sender reveals a packet loss, the sending rate drops, and might take longer to return than before the packet was lost. Both TCP Jersey or TCP New Jersey estimate the current available bandwidth according to Eq. 2, [46]

$$R_n = \frac{R_{n-1} \times RTT + L_n}{(t_n - t_{n-1}) + RTT} \tag{2,6}$$

R_n is the estimated bandwidth for ACK packet *n* received at time t_n at sender and t_{n-1} is previous ACK received time at the sender. RTT is round-trip time, and *Ln* is the size of payload *n* in TCP Jersey. It is clear that the timestamp feature was used by the receiver for data segment arrival, which is an option in the header.

The optimal congestion window (*ownd*) for TCP New Jersey is calculated as Eq. 2,7:

$$ownd_n = \frac{R_n \times RTT}{segment\_size} \tag{2,7}$$

TCP New Jersey computes the time it takes for the packet to arrive at the receiver to obtain an accurate estimate if there are no issues on the network.

However, New Jersey is not capable of calculating the accuracy of the available bandwidth if there is too much traffic on the network that affects packets sent, which means that TCP Jersey and TCP New Jersey are still not getting good performance if there is a significant amount of traffic in the routers.

2.2.9 TCP New Jersey+

TCP New Jersey+ differs from TCP New Jersey, in that the main goal is to improve available bandwidth estimation and recovery technique. TC New Jersey+ guarantees high throughput via an increased congestion window when the sender reveals that a packet is lost, or due to retransmission timeout.

It is sometimes difficult to calculate the accurate available bandwidth estimation if the network state is deteriorated from background traffic in forward links that transmit data packets; both TCP Jersey and TCP New Jersey suffer from this problem.

The new recovery in TCP New Jersey+ helps to achieve a good throughput compared to other TCP protocols in wireless networks, where packet loss occurs often.

Pseudocode algorithm below describes how to estimate the available bandwidth in TCP New Jersey+.

Initialization :

$n \leftarrow 1$

$R_{S0}, R_{r0}, t_{S0}, t_{r0} \leftarrow 1$

Procedure:

ACK packet arrived at the sender

if(timestamp)

$R_{Sn} \leftarrow (RTT \times R_{sn-1} + L_n) / ((t_{sn} - t_{sn-1}) + RTT)$

/ ABE based on ACK packet inter arrival time */*

$R_{rn} \leftarrow (RTT \times R_{rn-1} + L_n) / ((t_{rn} - t_{rn-1}) + RTT)$

/ ABE based on data packet inter arrival time */*

$R_n \leftarrow \max(R_{Sn}.R_{rn})$

/ maximum value of two estimations */*

$n \leftarrow n+1$

end if

The R_{sn} is the estimated bandwidth when the ACK packet n arrives at time t_{sn} at the sender, and t_{sn-1} is its previous ACK packet arrival time at the sender. The R_{rn} is the estimated bandwidth, by using the timestamp option, when the data segment n arrives at time t_{sn} at the receiver and t_{rn-1} is the previous data segment arrival time at the receiver. L_n is the size of data packet n, and RTT is the round-trip time [46].

TCP New Jersey+ uses the maximum value of R_{sn} and R_{rn} to specify the transmission rate; therefore, it solves the problem of estimating bandwidth and decreasing traffic on the network. If retransmission timeout (*RTO*) is expired, or due of BER, TCP New Jersey+ follows this pseudocode algorithm [46]:

if (RTO expired)

if (Congestion Warning)

/* if RTO due to congestion*/

cwnd = 1;

ssthresh = $ownd_n$;

else

/* if RTO due to BER */

cwnd = ($ownd_n$ + $ownd_{n-1}$) / 2;

ssthresh = $ownd_n$;

end if

end if

The TCP New Jersey+ sender decreases the *ssthresh* to $ownd_n$ and the *cwnd* to 1 if RTO is expired in order to distinguish the cause of network congestion, and to determine the BER. However, if RTO is caused by BER, then TCP New Jersey+ functions the same as New Jersey by diminishing *ssthresh* to $ownd_n$ and *cwnd* to half total $ownd_n$ plus $cwnd_{n-1}$ [46].

TCP New Jersey+ developed the recovery technique that is available in TCP New Jersey. Whenever New Jersey+ receives three duplicates due to congestion, it behaves like New Jersey. However, if BER causes the packet loss, then New Jersey+ sets *ssthresh* to $ownd_n$, *and* increases *cwnd* by one MSS.

Compared to other TCP protocols, TCP New Jersey+ has high performance when there is a large number of packets losses.

2.2.10 TCP Vegas

With TCP Reno (as well as Tahoe, its older version), the window size is continually increased until packet loss is caused by congestion. At that point, when the window size is throttled due to packet losses, the connection throughput could diminish. This cannot be avoided due to the type of congestion control function featured in TCP Reno. In other words, TCP RENO can identify network congestion only when the congestion has been caused by lost packets. However, throttling of the window size is not adequate when it is the TCP connection itself that caused the congestion because of its oversized window size. If the window size is properly controlled (e.g., the packet loss does not occur in the network), degraded throughput due to the throttled window can be avoided, which is the reason that TCP Vegas was developed[47].

TCP Vegas employs another mechanism, in which it controls its window size by observing RTTs (Round-trip Time) of packets that the connection has sent before. If observed RTTs become large, TCP Vegas developers recognized that when the network starts getting congested, it throttles the window size. However, if RTTs become small, TCP Vegas determines that the network is no longer congested, and subsequently increases the window size again. Afterward, because the window size is in a perfect situation, it reaches the appropriate value. More specifically, in the congestion avoidance phase, the window size is updated as follows[47]:

$$cwnd\,(t + t_A) = \begin{cases} cwnd(t) + 1, & if\ diff < \frac{a}{base_{rtt}} \\ cwnd(t), & if\ \frac{a}{base_{rtt}} \leq diff \leq \frac{\beta}{base_{rtt}} \\ cwnd(t) - 1, & if\ \frac{\beta}{base_{rtt}} < diff \end{cases} \quad (2,8)$$

$$diff = \frac{cwnd\,(t)}{base_{rtt}} - \frac{cwnd\ (t)}{rtt} \quad (2,9)$$

Where RTT represents the observed round-trip time, the smallest value of observed RTTs is shown as $base_{rtt}$, and both α and β are constant values.

TCP Vegas is equipped with a Slow Start mechanism, which is another special feature in its congestion control algorithm. The rate at which it increases its window size in the slow start phase is half of that in TCP Tahoe and TCP Reno (i.e., the increase in the window size is incremental for every other ACK packet received). Eq.2,8 was applied in TCP Vegas to determine that if observed RTTs of the packets are identical, the window size does not change[47]. The conclusion was that TCP Vegas can reach an additional 40% of throughput over that of TCP Reno, as verified in simulation and execution testing[47].

2.2.11 TCP Veno

TCP Veno applies end-to-end congestion control to distinguish between congestive and random losses, similar to TCP Vegas for packet loss. The TCP Veno sender estimates the connection state to specify if the packet loss was due to congestion or random loss. TCP Veno uses *N* measurement for congestive state, which can be defined as equation *Diff *BaseRTT*. If packet loss occurs and N <3, TCP Veno concludes that it is a random loss. But if N>=3 when packet loss occurs, then TCP Veno assumes that the loss is due to congestion. In TCP Vegas, *BaseRTT* constantly changes throughout the TCP connection with the minimum round-trip time. However, in TCP Veno, BaseRTT is reset whenever packet loss is detected, either due to time-out or duplicate acknowledgments (ACKs). *BaseRTT* is then updated as in the original Vegas algorithm until the

next fast recovery or slow start is triggered. This step is taken to take the changing traffic from other connections in consideration, and that the bandwidth that was acquired by a single connection among many other connections may change from time to time, thereby causing the BaseRTT to change [48].

TCP Veno employs two algorithms: Slow Start and Additive Increase. TCP Veno utilizes a Slow Start algorithm if cwnd is less than ssthresh, and it behaves similarly to TCP Reno for increasing the cwnd. Otherwise, TCP Veno uses an Additive Increase algorithm if cwnd is higher than ssthresh, and it modifies TCP Reno as follows pseudocode :

if (*DIFF*BaseRTT* < 3) *// available bandwidth under-utilized*

cwnd=cwnd+1/cwnd when ***every*** *new ACK received*

else if (*DIFF*BaseRTT* ≥ 3) *// available bandwidth fully utilized*

cwnd=cwnd+1/cwnd when every other new ACK received

If packet loss is due to random loss, then TCP Veno reduces the size of the window by 1/5 rather than by1/2 (as in Reno). The main advantage of TCP Veno is that it increases the congestion window slowly, which keeps the TCP at the appropriate transmission rate.

2.2.12 TCP mVeno

TCP mVeno is a new version of TCP Veno [48]. Its purpose is to make full use of the congestion information of all the subflows belonging to a TCP connection in order to adaptively adjust the transmission rate of each subflow.

The multi-path transfer feature in mVeno is based on TCP Veno. It changes the additional increase during the congestion avoidance phase by assigning various weights δ_{sr} for various paths r for effective coupling of the subflows.

TCP Veno applies the concept of the congestion monitoring scheme from TCP Vegas and integrates it into Reno's congestion avoidance phase. Veno calculates the backlog N at the queue and applies it as a gauge to determine whether or not the network is congested. If $N \geq \beta$, the network is in a congestive state.

Veno increases the congestion window $w(t)$ by $1/w(t)$ on every other positive acknowledgment. Conversely, it decreases it by half on each packet loss event. If $N < \beta$, it is in the non-congestive state.

The congestion window $w(t)$ is increased by $1/w(t)$ for every positive acknowledgment and decreased by 1/5 for each packet loss event [49].

The backlog N is given by Eq. 2,10, as follows[49]:

$$N = \left(\frac{cwnd}{base_{RTT}} - \frac{cwnd}{RTT}\right) * \quad baseRTT \tag{2,10}$$

In the above formula, *cwnd* represents the congestion window size, *RTT* is the average RTT in the last round, and *baseRTT* is the least RTT that has been measured up to this point.

Veno's additional algorithms are improved by mVeno via distribution of the weighted parameter (i.e., δ_{sr} for each subflow). mVeno performs the following steps on every subflow.

When $N \geq \beta$, the congestion window $w_{s,r(t)}$ is increased by mVeno flow s on subflow r by $\delta_{sr}/w_{s,r(t)}$ on every second positive acknowledgment and decreases it by (1/2) for every lost packet. However, when N < β, the congestion window $w_{s,r(t)}$ increases by $\delta_{sr}/w_{s,r(t)}$ on each positive acknowledgment and decreases by (1/5) every time a lost packet occurs[49].

2.3 Concept of Piggybacking

Piggybacking is what some researchers called "two-way communication", and others called "bidirectional flow or full transmission". Generally, piggybacking occurs when the packets communicate in both directions from station A to station B, and the data arrives at B. Instead to sending a control frame directly from B to A, station B waits until the network layer sends a packet to A, and the acknowledgment is attached in the data frame from B to A, using the field of acknowledgment in the data frame header. Therefore, the acknowledgment got a free ride in the data frame [50]. Figure 2.6 shows the ACK frame without data, and Figure 2.7 shows it with data.

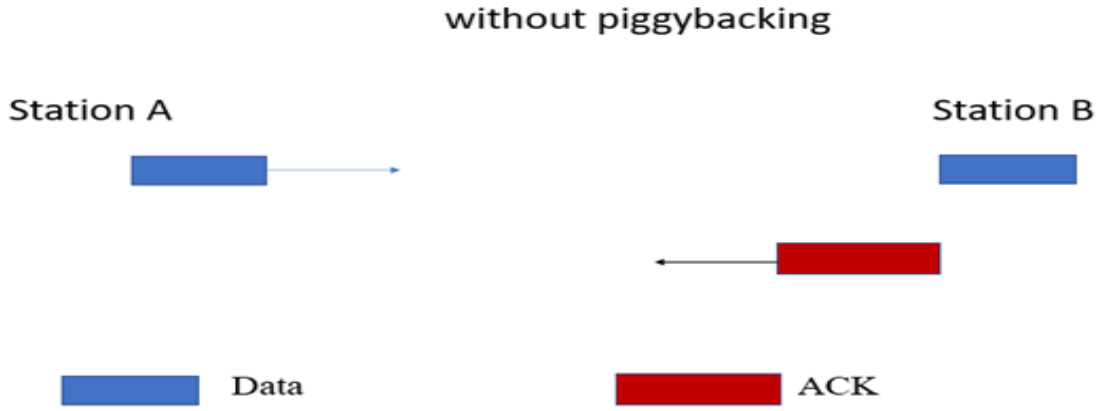

Figure 2.6: ACK without data

In Figure 2.6, station A sends packets to station B and waits for ACK. When station B receives the packet correctly, it returns ACK without adding any data with ACK, therefore the ACK is received quickly by station A.

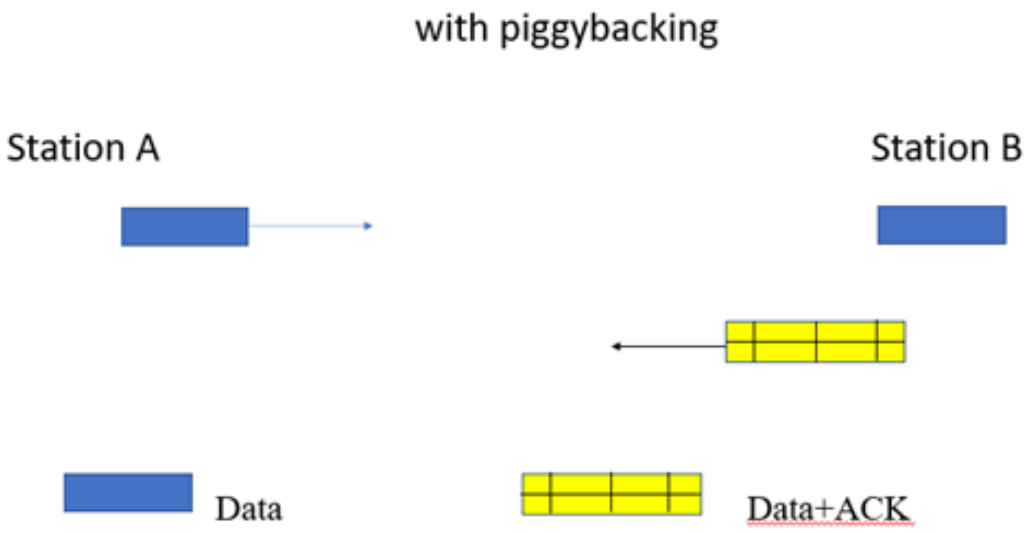

Figure 2.7: ACK with data

In Figure 2.7, station A sends the packets to station B and waits for ACK. ACK packet could contain data added by station B. After station B receives the packet, it waits sometimes before sending ACK because station B might have data to send to A. It is not necessary for station B to always data send to A, according to some researchers.

2.3.1 Piggybacking traffic

There are two cases for data received by a station. In the first case, the data received was corrupted, and in the second case, it was received correctly. It is preferable for the transmission rate to be high in order to use piggybacking because it might be overhead in the network. [51].

When there are several users on the network, the throughput of using piggybacking will be high compared to without piggybacking because ACK packets hold data. Therefore, if they are lost, piggybacking compensates these losses, particularly if the traffic in the network varies, (e.g., the network does not always remain busy, but there could be several delays [52].

Proper utilization of bandwidth can be developed if the bandwidth is properly configured to the channel access parameters, piggybacking policy, and network traffic [53].

2.3.2 Acknowledgment technique

TCP is a reliable protocol, which utilizes an acknowledgment technique to ensure that data transmitted was successfully received by the receiver. The acknowledgment technique in TCP can be either Cumulative or Delayed.

Cumulative acknowledgment is one where the receiver receives the data from the sender, then the receiver sends a feedback byte called the "acknowledgment", which confirms to the sender that all the frames were received correctly. Therefore, in order to save time, the receiver sends one acknowledgment in response to a particular number of frames.

Delayed acknowledgment in TCP occurs when the receiver has the option of sending ACK immediately, or later. The purpose of delaying ACK is to reduce ACK traffic or to enable the

receiver to send data with ACK in one packet rather than separating them. However, delaying might cause timeout and retransmission for the source.

2.4 Thesis Tool

This thesis will apply Network Simulation 2 (NS 2) to compare previously-mentioned protocols in Section 1.3, as well as Gnuplot to plot the results.

2.4.1 Network simulation 2 (NS2)

NS 2 is a free open-source network simulation tool for network and communication research. NS 2 is utilized to simulate routing, multicast protocols, and IP protocols such as TCP and UDP in wired and wireless networks. NS2 runs with many operating systems, such as Linux and SunOS, as well as with Windows versions. NS2 uses two languages: Object-oriented Tool Command Language (OTCL) to control the simulation and schedule discrete the events, and C++ language to NS2 subjects. NS2 supports a large number of built-in C++ classes, and it is appropriate to use these C++ classes to set up a simulation via a Tcl simulation script. NS2 runs traffic and topology before processing the simulation. The trace file in NS2 is used to analyze processing after the processing. "NS2 outputs any text-based simulation results. To interpret these results graphically and interactively, tools such as NAM (Network AniMator) and XGraph are used" [55].

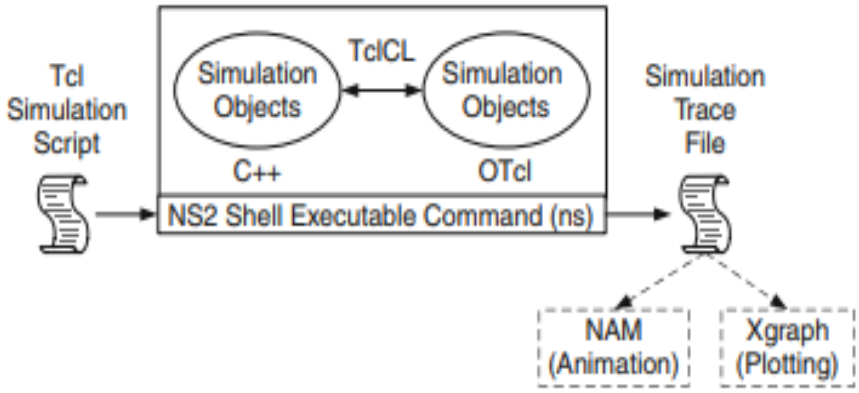

Figure 2.8: NS2 tools [55]

The purpose of using C++ is thatNS2 deals with packets, and when using existing modules. NS2 uses OTCL for configuration, step and runs the simulation with an existing model.

Advantages of NS2:

- It is free and easy to download and install.
- It supports many protocols and models.
- It is flexible in terms of setting up the configuration parameters.

- It enables researchers to design different scenarios.
- It is widely popular, and the recourses can be found on many websites.

Disadvantages of NS2:

- It requires enough memory to simulate the topology, which means that it may not executive if there is insufficient space in memory.
- It is sometimes incapable of running large-scale systems, or of obtaining the results.
- It occasionally slows down due to bugs or crashes.

2.4.2 Gnuplot

Gnuplot is a portable command-line driven graphing utility for Linux, OS/2, MS Windows, OSX, VMS, and other operating systems. The source code is copyrighted but freely distributed [56]. It was designed to allow researchers and students to plot their mathematical equations and data interactively.

2.4.3 Network configuration

We consider two distinct configurations in our implementations. One configuration assumes having partially wireless network while the other configuration assumes all wireless network. There are many reasons that we need to consider the wireless network in different applications:

1) Sensor networks: sensor networks are composed of numerous minuscule immobile sensors that are randomly inserted to detect and transmit the current environment's physical attributes, which are accumulated and tallied on a "data-centric" basis[57] [58]. A common term used to describe this feature is "battlefield surveillance", whereby several sensors are dropped from an overhead aircraft in enemy territory. Machinery prognosis, biosensing, and environmental monitoring are other examples of potential commercial fields.

2) Personal healthcare includes multimedia networks that can monitor and analyze the behavior of elderly citizens in order to identify the cause of illnesses that afflict them (e.g., dementia) [59]. Networks comprised of an audible communication device that attaches to clothing, or those with video capability, can detect emergency events and immediately connect the elderly person with remote crisis services or family members[60].

3) Industrial process control includes imaging and temperature / pressure sensing features for instances of critical time-sensitive events, or for industrial or process control purposes. Assimilating machine vision systems with WMSNs can streamline and add flexibility to manufacturing processes in order to provide visual inspections and pre-set functions[61].

Chapter 3

TCP Congestion Control Enhancement of Random Loss (CERL)

In this chapter, we briefly summarize TCP CERL's algorithm, and compare it to TCP protocols previously mentioned in section 1.4.

3.1 TCP CERL Algorithm

TCP Congestion Control Enhancement of Random Loss (CERL) is an end-to-end mechanism that achieves immediate throughput improvement over wireless. Although it is similar to TCP Veno in terms of distinguishing between random loss and congestive loss, CERL techniques are equipped with different mechanisms.

RTT consists of two parts in TCP CERL [62]:

1. The bottleneck queuing delay.
2. Total round-trip propagation delay with service delay.

The delay of queuing is equal to $\frac{l}{B}$, where l is queue length, and B is the bandwidth of the bottleneck link [62].

The total round-trip propagation delay with service delay is indicated as T, where T is a constant value in TCP CERL.

The calculation of bottleneck queue length l uses the following equation [62]:

$$l = (\mathrm{RTT} - \mathrm{T})\ \mathrm{B} \qquad (4,1)$$

TCP CERL sets T to be smallest RTT detected by the sender. Usually, l changes with updated RTT measurements.

The difference between TCP Veno and TCP CERL is that TCP Veno measures the backlog of end-to-end, whereas TCP CERL measures the queue length of the router [62].

TCP CERL utilizes queue length in Eq. 4.1 to estimate the congestion state of the link. TCP CERL uses parameter N as a dynamic queue length threshold, where N is equal to Eq. 4,2:

$$\mathrm{N} = \mathrm{A} * l_{max} \qquad (4,2)$$

l_{max} is the largest value of l calculated by the sender, and A is a value equal to 0.55 [62]. When the sender detects packet loss through three duplicate acknowledgments, and l is less than N, the TCP CERL sender, therefore, assumes that the lost packet was due to random loss, and retransmits that packet without reducing the congestion window and slow start threshold. However, if $l > N$ at the time when the sender receives the duplicate acknowledgments, TCP CERL then assumes that the loss was due to congestion and therefore, the TCP sender decreases the congestion window and slow start threshold similar to TCP Reno. On occasion, when multiple losses occur during transmission, the sender reduces the window only once.

3.1.1 Congestion window inflation

TCP Reno receiver sends an acknowledgment when a packet is received. The Reno receiver sends a duplicate acknowledgment (ACK) when a packet is out of order. Therefore, if the TCP Reno sender receives the duplicate ACK, the sender uses the fast recovery algorithm to increase the congestion window and to keep the network channel full [62]. For each duplicate ACK received by the sender during the fast recovery phase, the congestion window is increased by one. When the sender receives the first ACK for the new data during the fast algorithm, the TCP Reno sender sets the congestion window to the slow start threshold value.

TCP CERL slightly modified window inflation and deflation. If the first loss occurs in the current window of data, the TCP CERL sender behaves in the same manner as TCP Reno in terms of window inflation. "Otherwise, the current value of the congestion window is saved. To deflate the window, the congestion window is set to this value when the first ACK acknowledges the new data to this value" [62].

Figure 3.1 illustrates the Reno congestion window inflation and deflation from one time to six. The TCP Reno sender receives the loss through three duplicate acknowledgments, after which the sender sets the slow start threshold to cwnd/2, and sets cwnd to ssthresh + 3 * SegmentsSize. From Figure 3.1, it is clear that any period between 1 and 2, between 3 and 4, or between 5 and 6, Reno received duplicate ACKs, and allows the congestion window to become inflated. In addition, in periods 2, 4 and 6, TCP Reno received ACKs for the new data and reduced the cwnd equal to ssthresh.

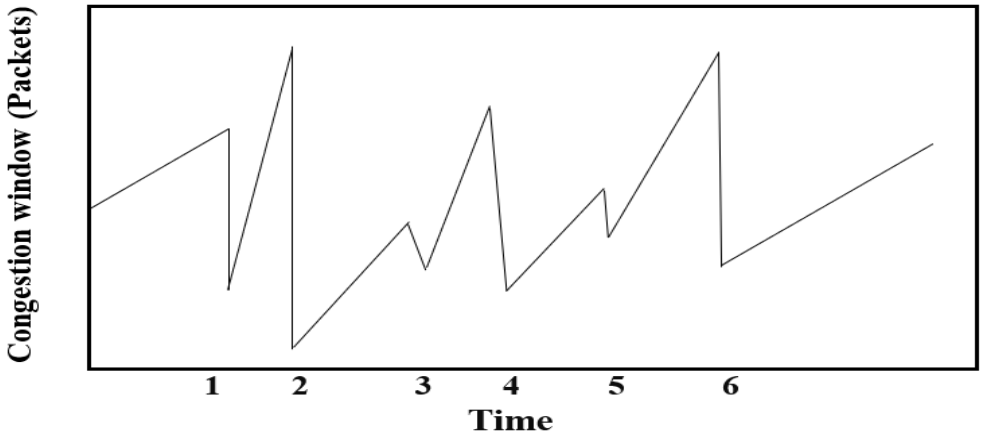

Figure 3.1: TCP Reno congestion window

Figure 3.2 shows that CERL detected the first loss at time 1 through three duplicate ACKs, the TCP CERL sender behaved similarly to Reno. However, when the TCP CERL sender detected another congestive loss at time 3, it behaved differently because it was not the first congestive loss, and therefore the sender reduced oldcwnd to cwnd, and set the cwnd to equal cwnd + 3* SegmentSize; CERL did not decrease the ssthresh value. In addition, during the period between 3 and 4, CERL received more duplicate ACKs, which resulted in CERL allowing the congestion window to become inflated. In time 4, CERL received ACK for the new data and deflated cwnd equal to oldcwnd. In time 5, CERL received another packet loss through three duplicates ACKs, therefore determining that it was due to random loss.

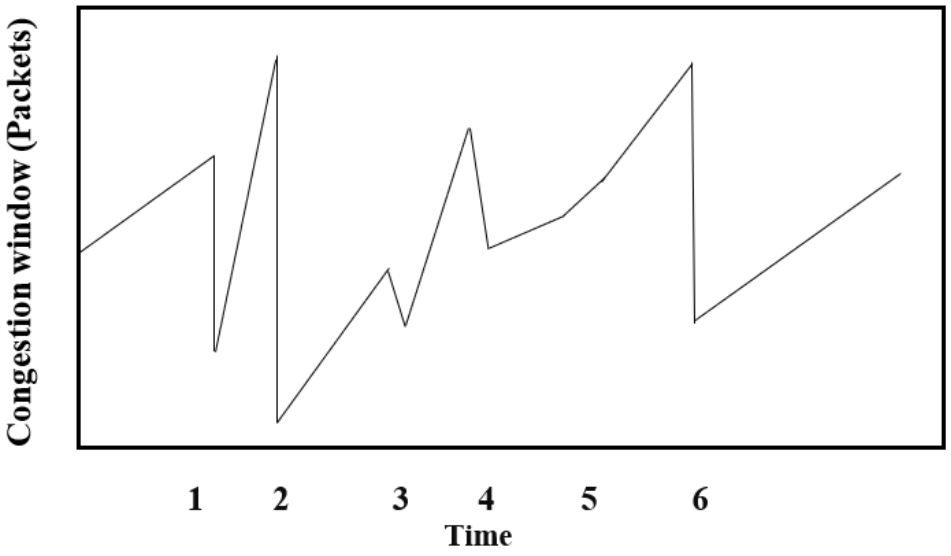

Figure 3.2: TCP CERL congestion window

3.1.2 Implementation of CERL

1 - Every time a new *RTT* is estimated by the sender, *T, l, lmax,* and *N* are updated.

2 - The TCP CERL sender changes to the fast recovery algorithm if the sender receives three duplicate acknowledgments. Therefore, the congestion window reduces to half if *l* is larger than or equal to *N,* or when the first-time congestion loss occurs only during this window of data.

The details of CERL mechanism is presented in Algorithm 1

Algorithm 1: CERL algorithm

```
While (true)
{
 WaitFor RTT arrival Event
 If (Event(RTT ArrivalNotification))
 {
   T ←—— Min (ArrivedRTT, OldRTT)

   A ←—— 0.55
   Calculate: l ←—— RTT—T
   lmax ←—— Max(Calculated l, Old l)
   N ←—— A*lmax
   If (Event(l >N&highstAck>lastDecMaxSentSeqno)
   {
    ssthresh ←—— min (cwnd, rwnd)/2
    If (ssthresh < 2*segsize)
       ssthresh ←—— 2*segsize
    end if
    cwnd ←—— ssthresh+3*segsize
    lastDecMaxSentSeqno= MaxSentSeqno
   else
    oldcwnd ←—— cwnd
    cwnd ←—— cwnd + 3 * segsize
  end if
 end if }
Return }
```

For example, in Figure 3.3, the TCP CERL sender transmits segments from 1 to 20, and segment 13 and 16 are lost. Eventually, three duplicate ACKs for segment 13 are received by the sender. Therefore, CERL causes the congestion window to be reduced, and the sender sets lastDecMaxSentSeqno to equal 20, and segment 13 to be retransmitted. However, when segment 16 is lost, the TCP CERL sender does not reduce the congestion window since it is not the first loss, and it is less than 20. Therefore, CERL simply retransmits only segment 16.

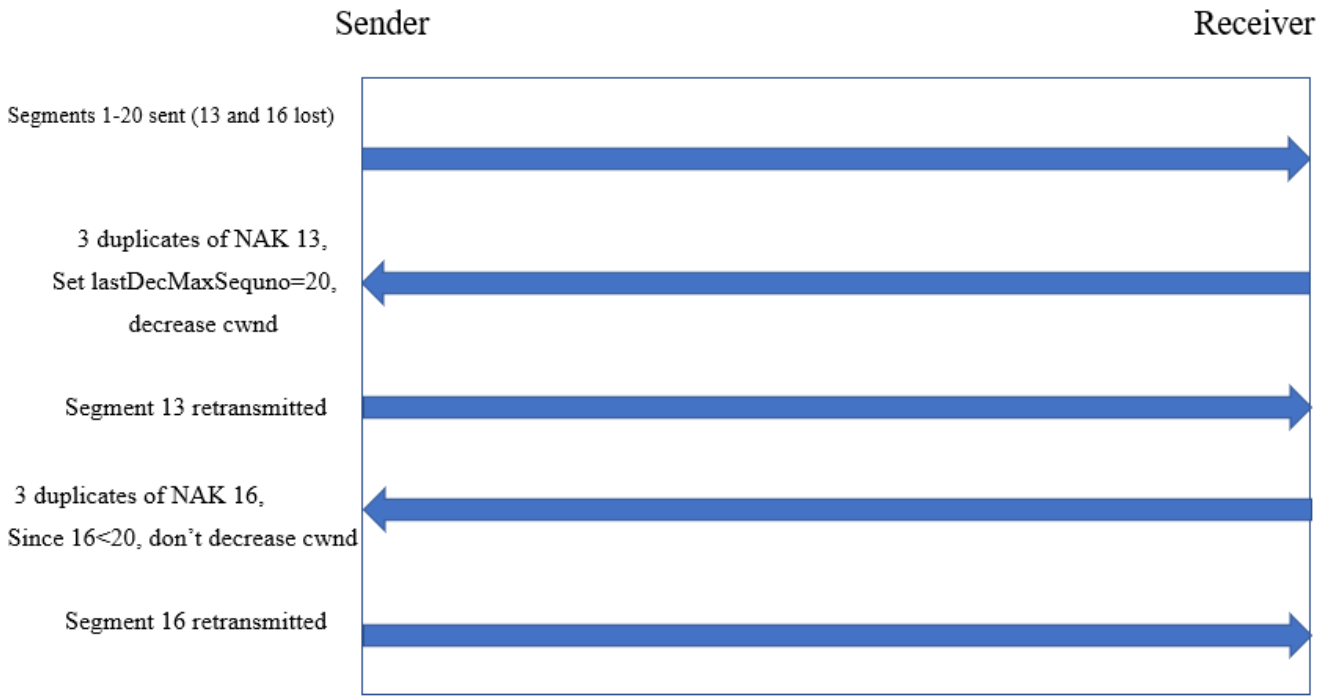

Figure 3.3: TCP CERL implementation

3.2 Network Configurations

In this chapter, we consider two configurations. The first configuration is a wired/wireless topology, and the second is all wireless topology. We will implement various scenarios in both topologies.

3.2.1 Network Configuration 1

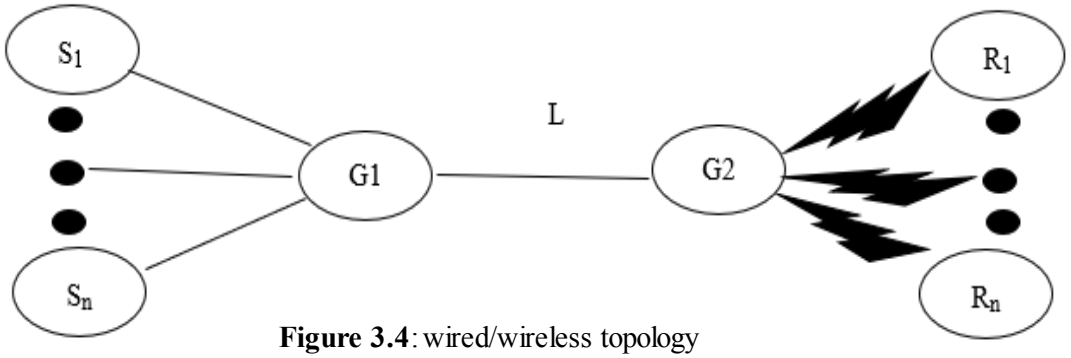

Figure 3.4: wired/wireless topology

Figure 3.4 illustrates the network configuration. We assume that N end-senders are S1 to SN, and N end-receivers are R1 to RN. We also assume that the network operates on two routers (G1 and G2). All transmission lines between end-senders are wired, and end-receivers are wireless. In addition, we assume that the transmission line between G1 and G2 is wired.

3.2.1.1 Scenario 1 : One-way transmission

Scenario 1 evaluates the throughput of CERL with a small number of users, using one-way transmission. Users will apply the same protocol during the simulation by sending FTP files at different times.

Figure 3.4 shows the network topology used for scenario 1.

- N = 10.
- S1 to S10 are TCP senders.
- R1 to R10 are TCP receivers.
- G1 and G2 are routers.
- S1 to S10 are connected to G1 via wire links; bandwidth and propagation delay for each link are 1 Mbps and 10 ms, respectively.
- R1 to R10 are connected to G2 via wireless links; bandwidth and propagation delay for each link are 1 Mbps and 10 ms, respectively.
- We denote L for the wired link between G1 and G2, the bandwidth and propagation delay for L are 8 Mbps and 50 ms, respectively.
- We consider the maximum segment size as 1460 bytes.

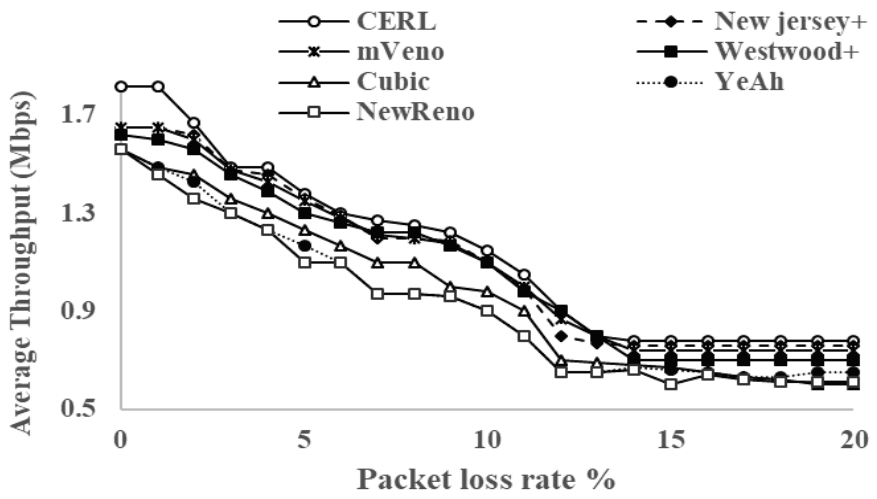

Figure 3.5: Average throughput versus packet loss

Figure 3.5 compares CERL with various TCPs when using one-way transmission, with the packet loss rate in L ranging between 0 and 20, Results show that CERL throughput is higher than other throughputs, when there is 0% loss rate in link between G1 and G2. When that random loss is greater than 10%, the network is quite lossy and as a result timeouts occur frequently. In such case, TCP variants perform similarly through retracting back to slow start and they in turn behave poorly. While random loss is below 10%, duplicate acknowledgments are the sign for the packets losses. In this range, CERL is able to effectively discriminate the random loss from congestion loss and therefore the congestion window does not decrement, as other TCPs, and gain higher throughput.

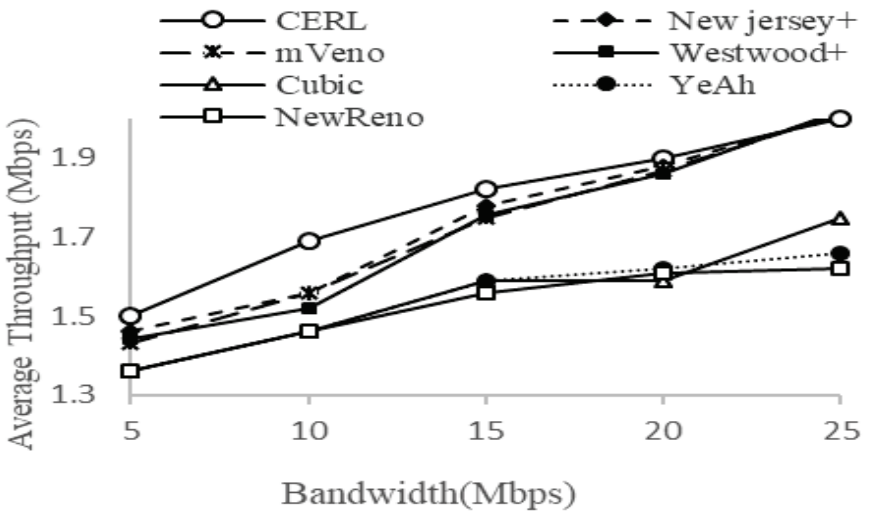

Figure 3.6: Average throughput L's bandwidth,

Figure 3.6 illustrates the throughput of TCP protocols with a bandwidth of L ranging from 5 to 25 Mbps, where q_L (random loss q in link L) is 1%. Results show that CERL gains a 169% and 87% throughput development over NewReno and Westwood+, respectively. With limited bandwidth, although that the bottleneck queue length will be relatively small for CERL according to equation (1) but based on its mechanism won't suffer from the unnecessary congestion window trimming that occurs frequently with other TCP variants. When the bandwidth increases, TCP variants including CERL will behave similarly as the queue length is long enough and congestion window won't decrement obviously. As indicated in [26], that square root formula [36] would impose an upper limit on the throughput of TCP connections.

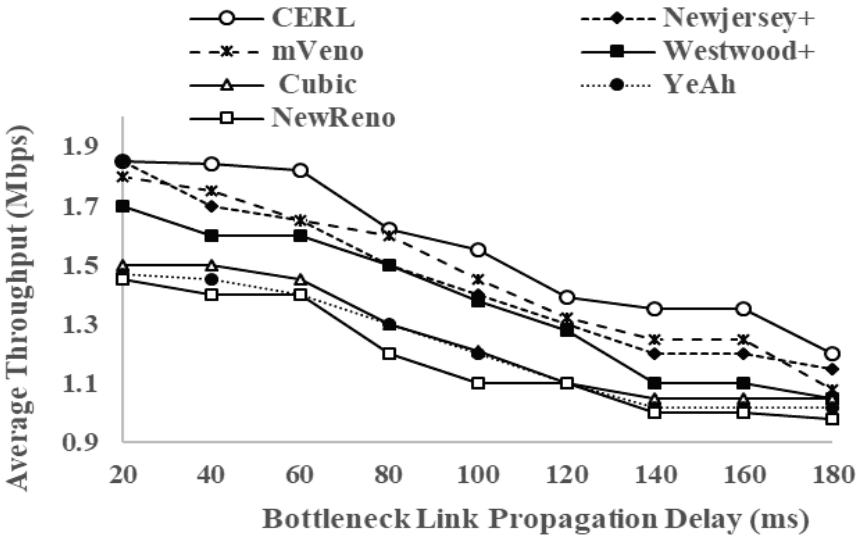

Figure 3.7: Average throughput versus bottleneck link

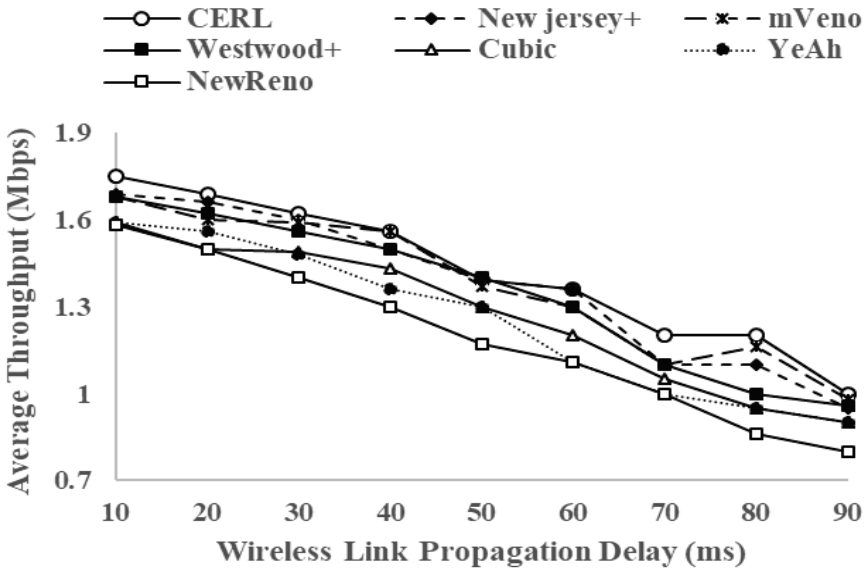

Figure 3.8: Average throughput versus wireless link propagation delay between receives and G2, where $q_L = 1\%$

Figure 3.7 clarifies the throughput of CERL with other protocols when T_p for L ranges from 20 to 180 ms, where q_L is 1%, and Fig 3.8 illustrates the throughput of TCP protocols, where the propagation delay between TCP receivers and G2 ranges between 10 and 90 ms, where q_L is 1%. Increasing the time delay would lead to a quicker excessive aggregation of packets in the bottleneck queue waiting for transmission. This would increase the possibility of congestion and random loss which in turn, would reduce the congestion window and, therefore, the throughput decreases. However, in Fig 3.7, CERL still gains higher throughput compared to other variants for its discrimination process. In fig 8, more timeouts would be produced when increasing the propagation time delay at the wireless portion so TCPs would behave worse.

3.2.1.2 Scenario 2 : Two-way transmission

Scenario 2 evaluates the throughput of CERL with a small number of users, using bidirectional flow. Users will apply the same protocol during the simulation sharing sending FTP files at different times.

Figure 3.4 shows the network topology used for scenario 2.

- N = 10.
- S1 to S10 are TCP senders.
- R1 to R10 are TCP receivers.
- G1 and G2 are routers.
- S1 to S10 are connected to G1 via wire links; bandwidth and propagation delay for each link are 1 Mbps and 10 ms, respectively.
- R1 to R10 are connected to G2 via wireless links; bandwidth and propagation delay for each link are 1 Mbps and 10 ms, respectively.
- We denote L for the wired link between G1 and G2, the bandwidth and propagation delay for L are 8 Mbps and 50 ms, respectively.

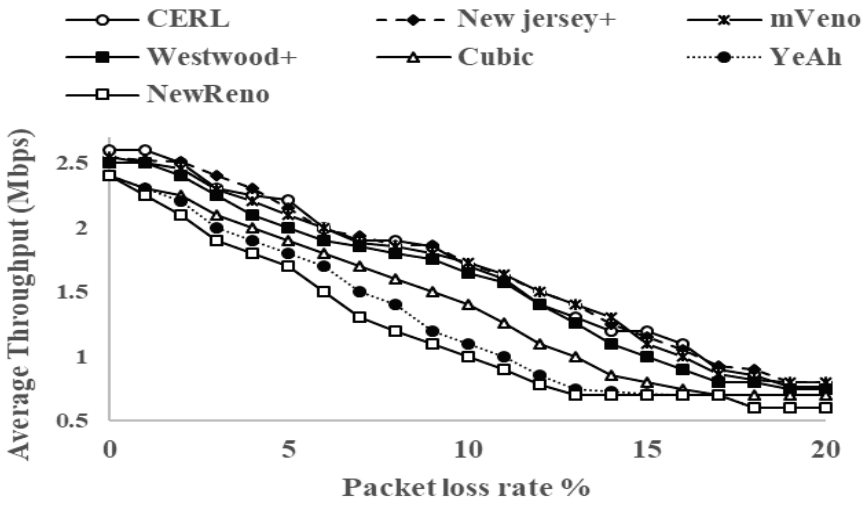

Figure 3.9: Average throughput versus packet loss in L

Figure 3.9 compares CERL with various TCPs when using two-way transmission, with the packet loss rate in L ranging between 0 and 20.It's obvious that CERL is not behaving well as the congestion window would diminish as a result of the heavy load with the two-way transmission. Particularly, mVeno and NewJersey+ have a quite competitive performance with CERL.

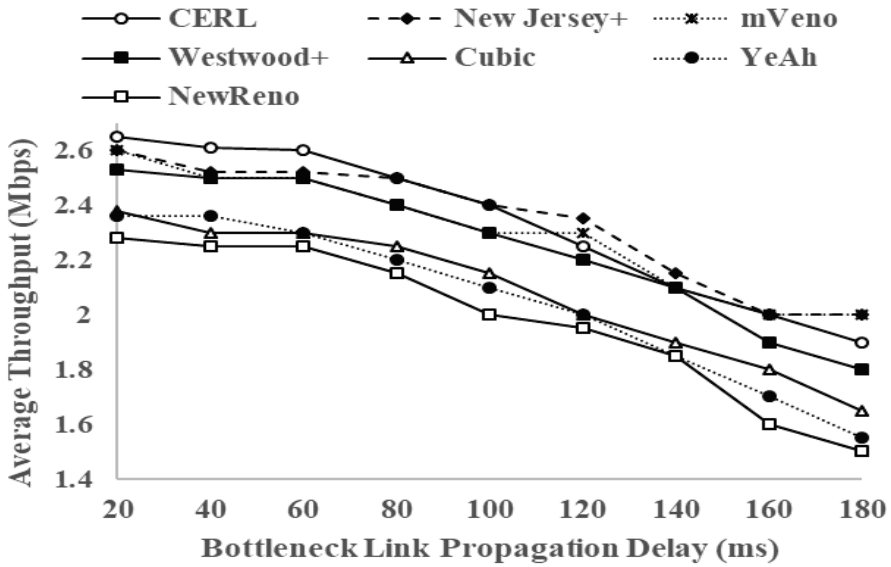

Figure 3.10: Average throughput versus bottleneck link propagation delays in L, where q_L=1%

Figure 3.10 CERL behaves better than other TCPs when T_p is short as the queue will have less segments accumulation and this alleviates the possibility of congestion and accordingly the congestion window will not decrease. In this case, the mechanism of the CERL will function properly and throughput is better than other TCPs. On the contrary, when increasing T_p, senders will send more segments before ACKs come back and therefore the queue will be more congested

and the mechanism of CERL is not working efficiently. Also, CERL will undergo more timeouts as well as other variants and the throughput is close for all TCPs.

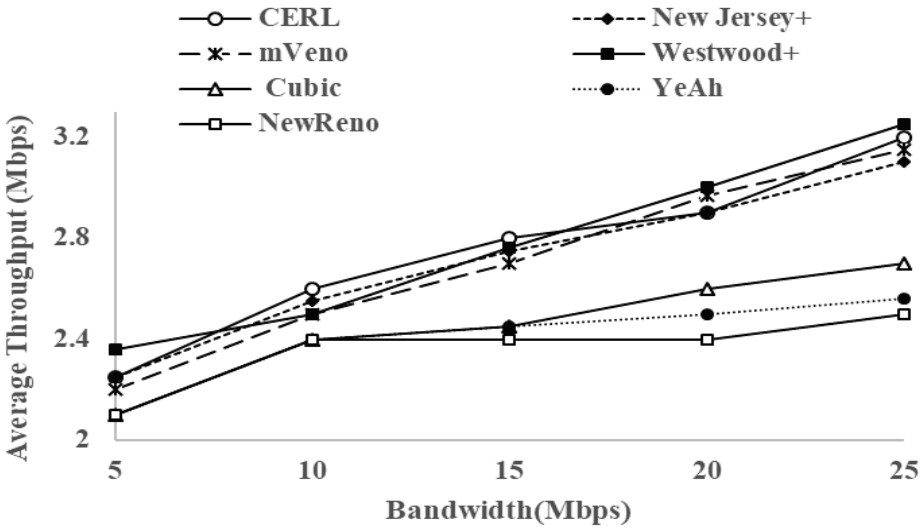

Figure 3.11: Average throughput versus L's

Figure 3.11 TCP variants are not performing that well with all ranges of bottleneck bandwidth. This implies that there are potentially unnecessarily drops of segments of the bottleneck queue while the mechanism of CERL is not functioning well. This is because of the heavy load of the network in case of two-way transmission.

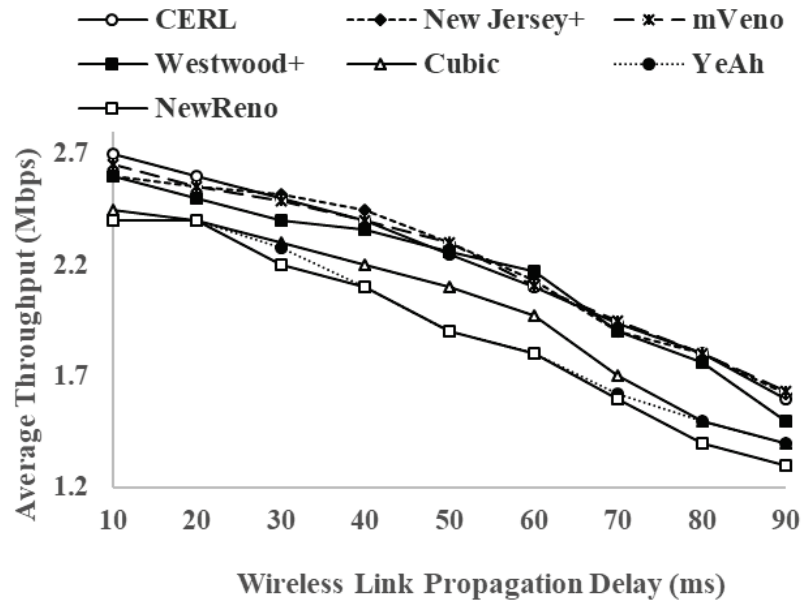

Figure 3.12: Average throughput versus wireless link propagation delay between receivers and G2,where

Figure 3.12 CERL behaves similarly as in Fig 3.10. CERL has a better performance at a wider range of T_p of the link L (between 20 to 60 ms) compared to links delay at the receiver side (between 10 to 20 ms). This shows that CERL is more flexible to the bottleneck delay than the receivers' side delays and this agrees with the one-way transmission in Figs 3.7 and 3.8, although that CERL performs better with the one-way transmission generally. Such discrepancy in CERL performance between tests in Figs 3.11 and 3.12 is because the limitation of the bandwidth at the receivers' side compared to that of L in addition to the increase of their time delay would lead to have an obvious congestion at the bottleneck queue.

3.2.2 Network configuration 2

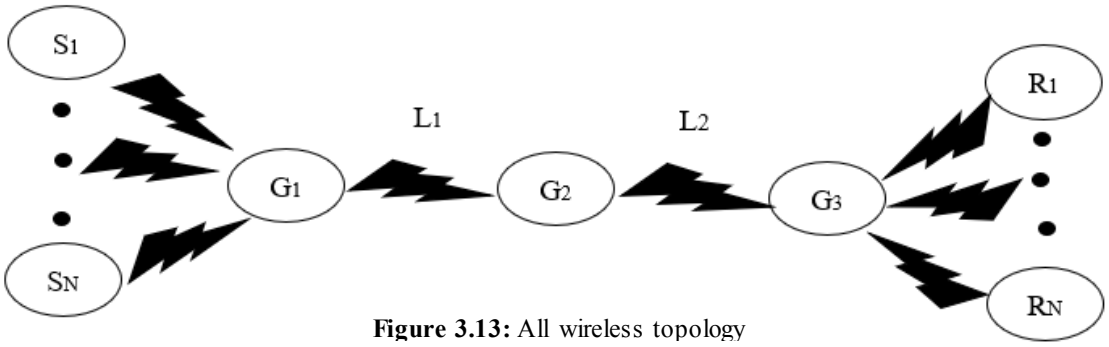

Figure 3.13: All wireless topology

Figure 3.13 illustrates the network configuration. We assume N end-senders S_1 to S_N as well as N end-receivers R_1 to R_N. We also assume that the network has three routers: G_1 to G_3. All transmission lines between end-senders and end-receivers (including between routers L1 and L2) are wireless. Random loss (q) may occur in L1 or L2, as will be shown in our results in various scenarios.

3.2.2.1 Scenario 1: Two-way transmission and heavy load, where q_{L1} is 1% and q_{L2} is 1% loss rate.

Scenario 1 evaluates CERL's throughput with a large number of users, using piggybacking flow. Users apply the same protocol during the simulation sharing FTP files at different times. Scenario 1 allows receivers to send data with ACK in the same frame. Note that we allowed receivers to send data with ACK by default in NS2.

Figure 3.13 shows the network topology used in scenario 1.

- N = 100.
- S1 to S100 are TCP senders.
- R1 to R100 are receivers.

- G1 to G3 are routers.
- S1 to S100 are connected to G1 via wireless links; bandwidth and propagation delay for each link are 1 Mbps and 20 ms, respectively.
- R1 to R100 are connected to G3 via wireless links; bandwidth and propagation delay for each link are 1 Mbps and 20 ms, respectively.
- We denote L1 for the wireless link to be between G1 and G2; bandwidth and propagation delay for L1 are 85 Mbps and 60 ms, respectively.
- We denote L2 for the wireless link to be between G2 and G3; bandwidth and propagation delay for L2 are 85 Mbps and 60 ms, respectively. We added a constant 1% loss rate for q_{L2}.

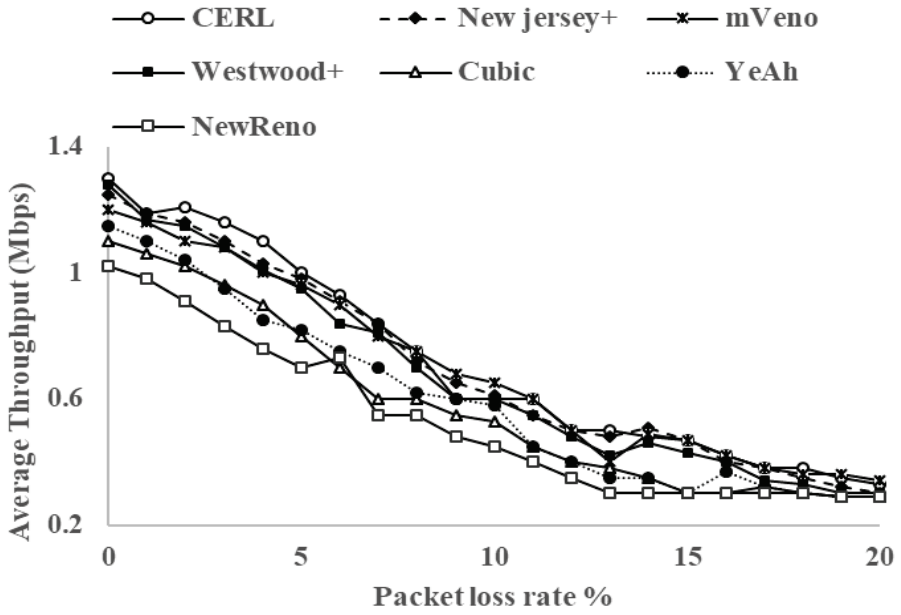

Figure 3.14: Average throughput versus packet loss in L1, where $q_{L2} = 1\%$

Figure 3.14 compares CERL throughput with various TCPs when q_{L1} ranges between 0% and 20%, where q_{L2} is 1%. Results show that CERL gets poor throughput when there are many users and ACK delays, compared to the throughputs achieved from New Jersey+ and mVeno when packet loss rates are more than 4% in L1.

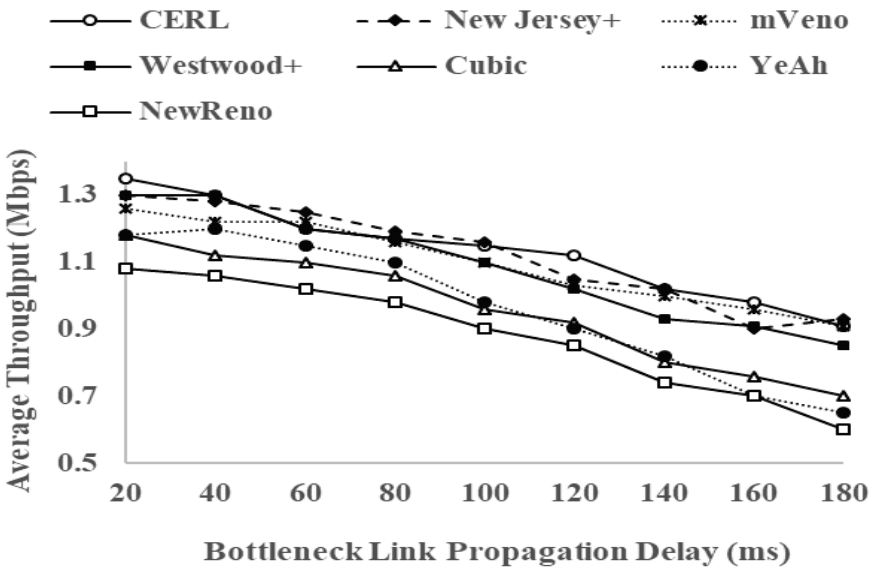

Figure 3.15: Average throughput versus bottleneck link propagation delay in L1, where propagation delay in L2 = 60 ms and q_{L1} = 1% and q_{L2} = 1%

Figure 3.15 compares CERL with other protocols when propagation delay for L1 ranges between 20 and 180 ms, where propagation delay for L2 is 60 ms and q_{L1} is 1% and q_{L2} is 1%. Results show that most throughput from tests such as CERL and New Jersey+ is fairly similar when bottleneck ranges between 20 and 120 ms. However, CERL throughput remains lower than that of New Jersey+ in most of the propagation delays in L1.

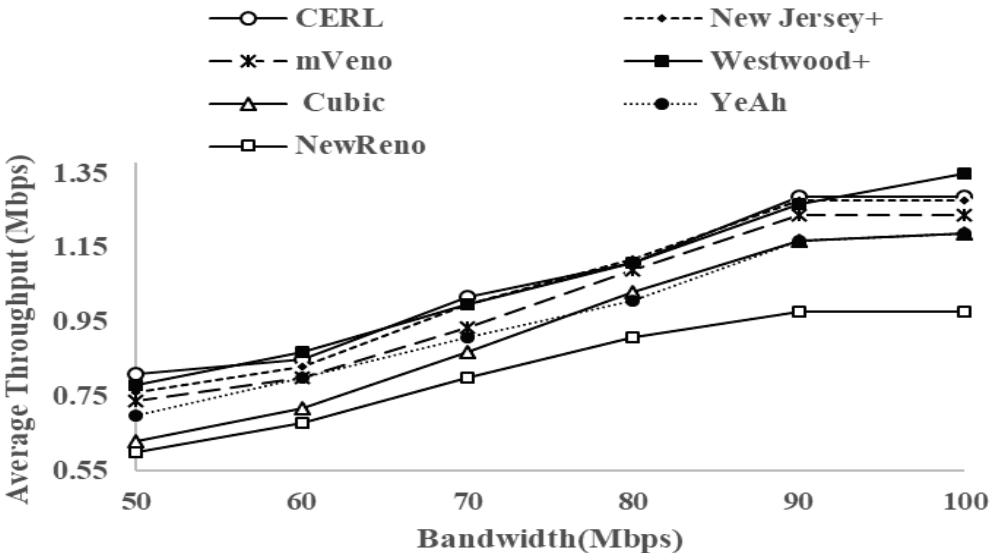

Figure 3.16: Average throughput versus L1's bandwidth, where bandwidth of L2 = 85 Mbps and q_{L1} = 1% and q_{L2} = 1%

Figure 3.16 compares the throughput of TCP protocols with the bandwidth of L1 ranging between 50 and 100 Mbps, where the bandwidth of L2 = 85 Mbps, q_{L1} is 1% and q_{L2} is 1%. Results indicate that throughput from all TCPs gradually increases until they reach a specific point, and some of them remain at the same level. It is evident that Westwood+ throughput outperforms on all protocols when bottleneck bandwidth is equal to 100 Mbps. However, CERL throughput is close to that of Westwood+ when L1's bandwidth equals 70 and 80 Mbps.

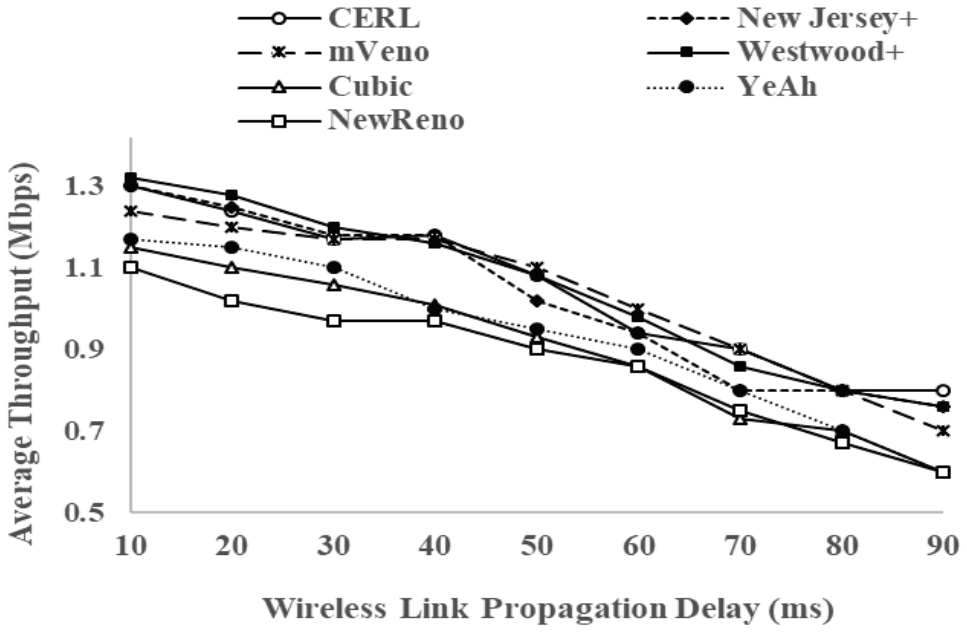

Figure 3.17: Average throughput versus wireless link propagation delay between senders and G1, where q_{L1} = 1% and q_{L2} = 1%.

Figure 3.17 illustrates the throughput of TCP protocols, where the propagation delay between TCP senders and G1 ranges between 10 and 90 ms, where q_{L1} is 1% and q_{L2} is 1%. Results show that the CERL throughput remains poor compared to those of Westwood+ and New Jersey+ when wireless propagation delays are small.

3.2.2.2 Scenario 2: Two-way transmission and heavy load, where q_{L1} is 1% and q_{L2} is 0% loss rate.

Scenario 2 evaluates CERL throughput with a large number of users with the bidirectional flow. Users will apply the same protocol during the simulation. Senders will transmit FTP files at different times, and the receivers are able to send data with ACK in the same frame. In this scenario, the duration will be lengthy.

Figure 3.13 shows the network topology used for scenario 2.

- N = 100.
- S1 to S100 are TCP senders.
- R1 to R100 are receivers.
- G1 to G3 are routers.
- S1 to S100 are connected to G1 via wireless links; bandwidth and propagation delay for each link are 1 Mbps and 20 ms, respectively.
- R1 to R100 are connected to G3 via wireless links; bandwidth and propagation delay for each link are 1 Mbps and 20 ms, respectively.
- We denote L1 for the wireless link to be between G1 and G2; bandwidth and propagation delay for L1 are 85 Mbps and 60 ms, respectively.
- We denote L2 for the wireless link to be between G2 and G3; bandwidth and propagation delay for L2 are 85 Mbps and 60 ms, respectively, with constant 0% loss rate for q_{L2}

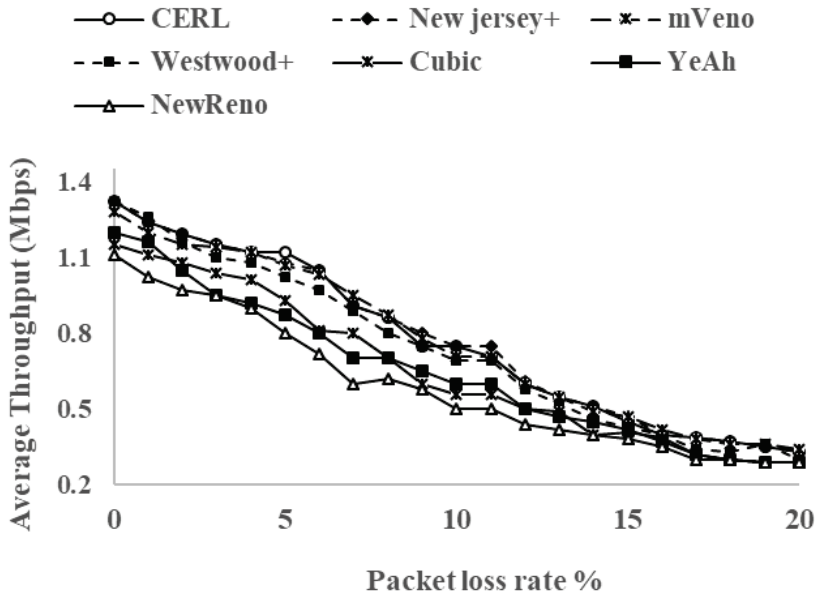

Figure 3.18: Average throughput versus packet loss in L1, where q_{L2} =

Figure 3.18 compares CERL with various TCPs when using piggybacking, and L1 ranges between 0 and 20, where q_{L2} is 0%. Results show that throughput of all protocols is higher than in figure 3.14 and that all protocols throughput increased by approximately 5%. In Figure 3.18, CERL throughput does not behave well compared to New Jersey+ and mVeno.

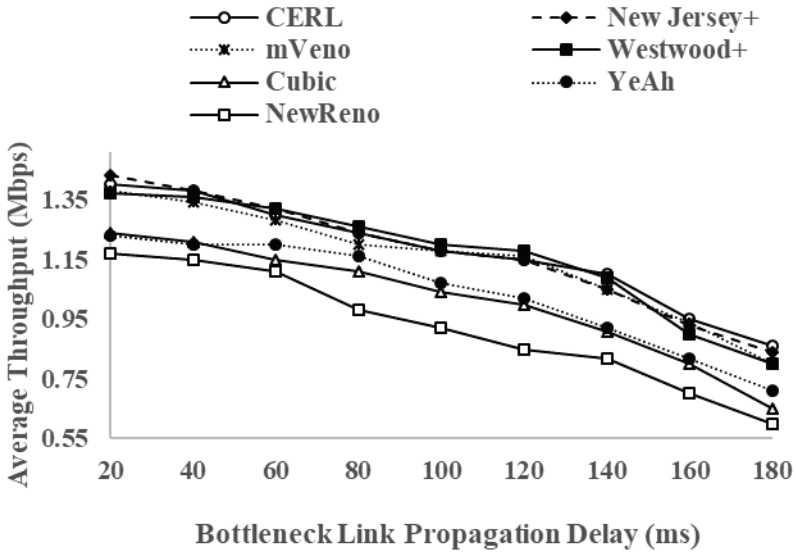

Figure 3.19: Average throughput versus bottleneck link propagation delay in L1,where $q_{L1} = 1\%$ and $q_{L2} = 0\%$

Figure 3.19 compares CERL throughput with that of other TCPs when propagation delay for L1 ranges between 20 and 180 ms, where propagation delay for L2 is 60ms q_{L1} is 1% and q_{L2} is 0%. Results show that CERL throughput remains lower than that of New Jersey+ and Westwood+. However, CERL performance growth improves by approximately 10% to 15% compared to its performance in scenario 1, as shown in figure 3.15. Throughput from New Jersey+, Westwood+, and mVeno is fairly similar to CERL in this scenario.

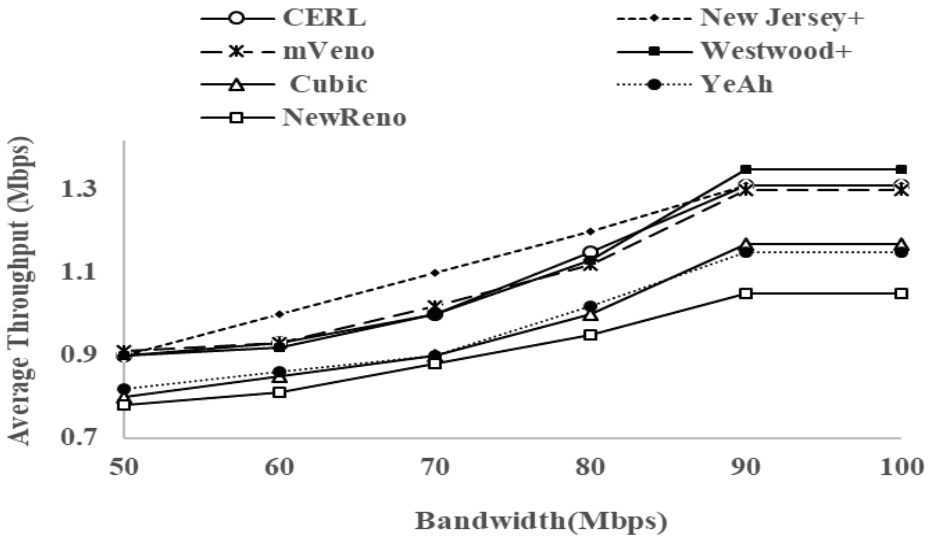

Figure 3.20: Average throughput versus L1's bandwidth, where bandwidth of L = 85 Mbps,$q_{L1} = 1\%$ and $q_{L2} = 0\%$

Figure 3.20 indicates that the throughput of TCP protocols with bottleneck bandwidth L1 ranges between 50 and 100 Mbps, where q_{L1} is 1% and q_{L2} is 0%. Results indicate that Westwood+ throughput still outperforms CERL and other protocols after 80 Mbps. However, CERL gains a 9% and 7% throughput over Westwood+ and New Jersey+, respectively, when bandwidth equals 70 Mbps

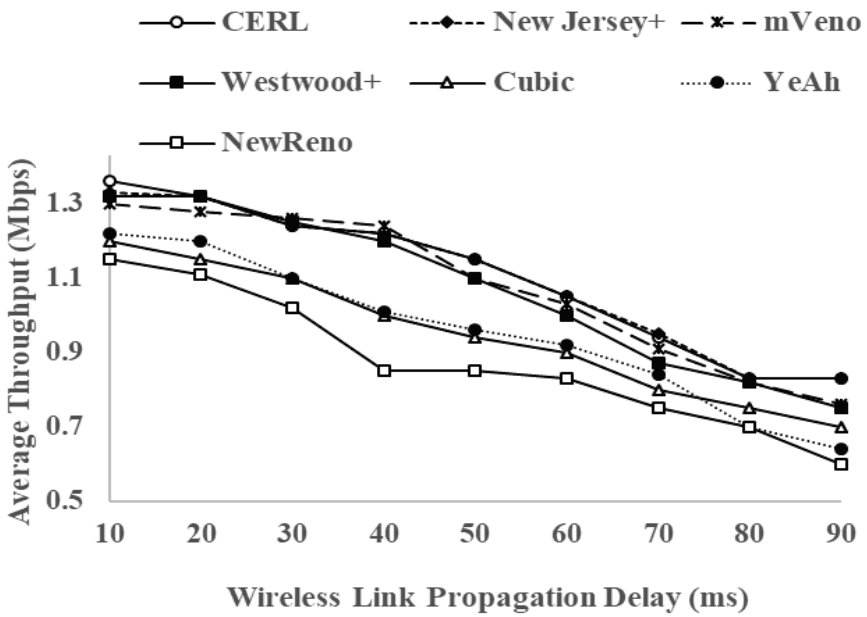

Figure 3.21: Average throughput versus wireless link propagation delay between senders and G1, where q_{L1} = 1% and q_{L2} = 0%

Figure 3.21 illustrates the throughput of TCP protocols, and propagation delay between TCP sources and G1, when the propagation delay ranges between 10 and 90 ms, and where q_{L1} is 1% and q_{L2} is 0%. Results indicate that the throughput of some protocols improve when L2 does not have a loss rate like NewReno, and Yeah, but they are still lower than other TCPs. Also, as depicted in Figure 3.21, CERL's throughput outperforms in all protocols at the beginning, but it gradually reduces with the rest of the propagation delays.

However, as per the results in scenario 1, figure 3.16 indicates that CERL's throughput was more inconsistent, resulting in a 1% loss rate in L2.

3.3 Conclusion

From the results, it is evident that CERL performance's in a small number of users is high compared to other protocols. In a packet loss test, CERL obtains 13%, 10%, 23%, 46%, 35%, and 83% throughput gains over New Jersey+, mVeno, Westwood+, Cubic, Yeah, and NewReno, respectively. However, throughputs from New Jersey+ and Westwood+ were close to that of CERL in several test results.

On the other hand, CERL does not achieve a good throughput when the network has a large number of users, because users utilize the same threshold A, therefore, the percentage of decrease the

congestion window is high. Also, when we decrease the size of the segments in piggybacking tests, New Jersey+, mVeno, and Westwood+ perform well during two-way sending tests. Throughput from Cubic, Yeah and NewReno remain low in both one-way and piggybacking simulations.

We evaluate CERL with different simulations and conclude that CERL is not capable of achieving high performance. Therefore, the constant value in CERL is not sufficient for piggybacking tests.

Chapter 4

TCP Congestion Control Enhancement of Random Loss plus (CERL +)

In this chapter, we will briefly summarize about TCP CERL+ and compared with TCP protocols that previously mentioned in section 1.4

4.1 TCP CERL +

TCP Congestion Control Enhancement of Random Loss plus (CERL+) is an end to end mechanism to improve the performance over wireless and wired networks, particularly when there are a large number of nodes. TCP CERL+ is the evolution of TCP CERL [62] to distinguish between random loss and congestion loss. TCP CERL+ has a similar implementation of TCP CERL measuring bottleneck and inflation of congestion window, but CERL+ behaves differently when calculating the dynamic queue length threshold N that in CERL.

4.1.1 Distinguish random loss from congestion loss

TCP CERL utilizes A, used in equation 3,2 of chapter 3, a constant value that is equal to 0.55 to measure the dynamic queue length threshold N. When CERL uses this value, it gains a higher throughput comparing to some protocols, but when A values are more than 0.55, throughputs are the same [52]. Results in chapter 3 assuming piggybacking transmission and a heavy load of users showed poor behavior of CERL compared to other techniques. In this regard, we revisit CERL and consider modifying the queue length threshold N.

TCP CERL+ makes use of the average of RTTs and minimum RTT and this is measured by the sender. As a result, CERL+ makes a dynamic queue length threshold N more flexible for every sender. Which means that sender will estimate average RTTs and minimum in every RTT is measured, so the transmission for data between users will be various. TCP CERL+ calculates N according to equation 4,1:

$$N = \frac{RTTavg}{T} * lmax \qquad (4,1)$$

Where RTTavg is the average of RTT measured in each time, T is minimum RTT observed by the sender, and $lmax$ is the largest value of l calculated by the sender.

In the results, we will show how TCP CERL+ improved the performance of throughput comparing to CERL, when CERL + users use average and minimum round-trip time.

The details of CERL+ mechanism is presented in Algorithm 1

Algorithm 1: CERL+ algorithm

```
While (true)
{
 WaitFor RTT arrival Event
 If (Event(RTT ArrivalNotification))
 {
  T ←— Min (ArrivedRTT, OldRTT)
  AvegRTT ←— Average of RTT over simulation time
  A ←— AvegRTT/T
  Calculate: l ←— RTT—T
  lmax ←— Max(Calculated l, Old l)
  N ←— A*lmax
  If (Event(l >N&highstAck>lastDecMaxSentSeqno)
  {
   ssthresh ←— min (cwnd, rwnd)/2
   If (ssthresh < 2*segsize)
      ssthresh ←— 2*segsize
    end if
    cwnd ←— ssthresh+3*segsize
    lastDecMaxSentSeqno= MaxSentSeqno
  else
   oldcwnd ←— cwnd
   cwnd ←— cwnd + 3 * segsize
  end if
 end if }
Return }
```

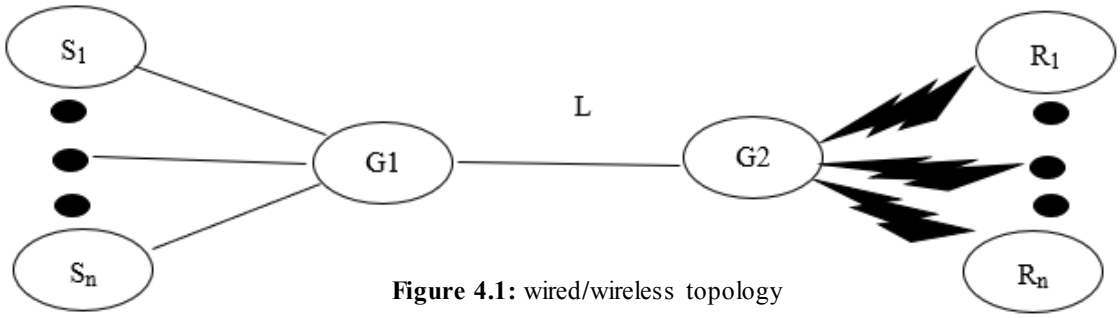

Figure 4.1: wired/wireless topology

In figure 4.1, the network configuration is shown. We assume N end-senders S_1 to S_N and as well N end-receivers R_1 to R_N. We assume that the network two routers G1 and G2. All transmission lines between end-senders are wired, and end-receivers are wireless. Also, we assume that the transmission line between G1 and G2 is wired.

4.1.2 Evaluation

In this test, we evaluate the congestion window of NewReno and CERL+ when the link between G1 and G2 in figure 4.1 has a random loss and without a random loss. We assume that from figure 4.1, there is one sender and one receiver. The bandwidth and propagation delay of links S1G0 and G1R1 are set to 15 Mbps and 20 ms, respectively. Figure 4.2 illustrates the congestion widow evolution of TCP NewReno, and figure 4.3 illustrates the congestion window evolution of TCP CERL+.

Results show that congestion window of TCP NewReno and CERL+ are close without random loss. In figure 4.3, we can notice that congestion window drops to 2 segments at 17 s at 90 s, because when time out occurs CERL+ changes to slow start phase. With 1% random loss, the average throughput of CERL+ is 0.89 Mbps, while the average throughput of NewReno is 1.79 Mbps. Which means that CERL+ gains a 115% throughput improvement over NewReno in this test.

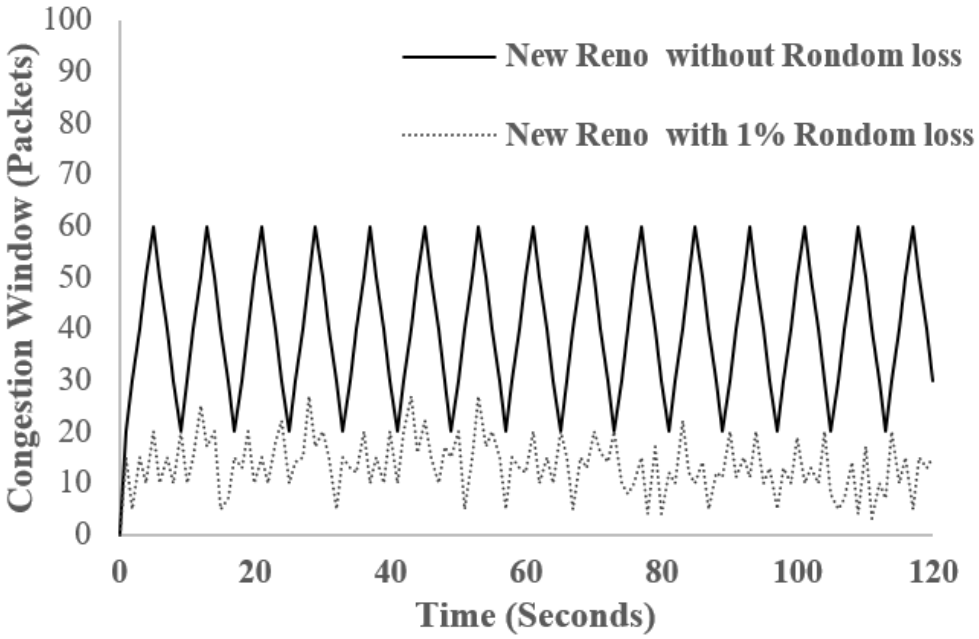

Figure 4.2: NewReno Congestion Window Evaluation

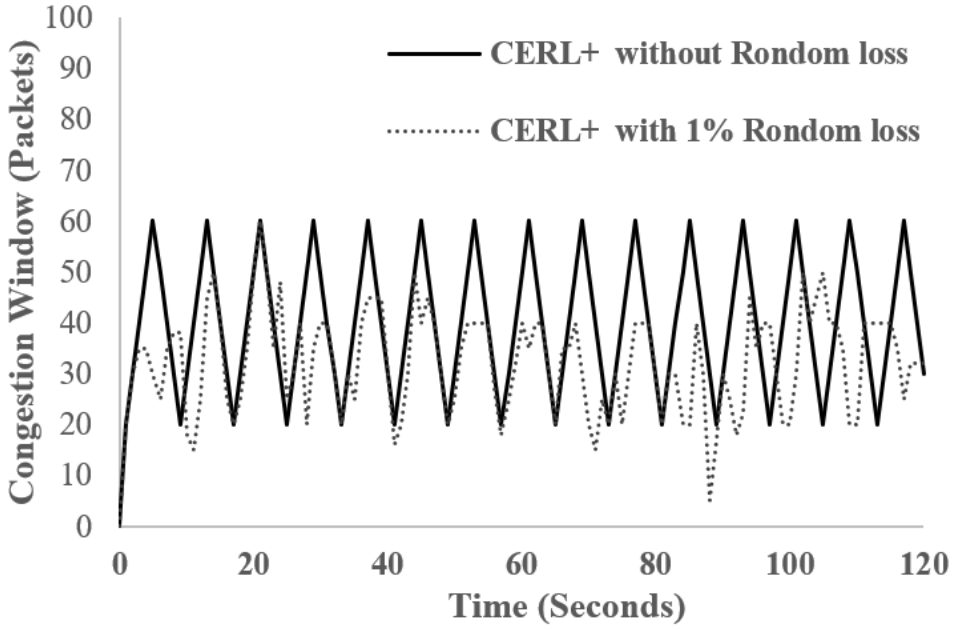

Figure 4.3: CERL+ Congestion Window Evaluation

4.1.3 CERL+ behavior in absence of random loss

In this section, we demonstrate that the CERL+ random loss distinguishing mechanism does not affect the throughput of CERL+ when the random loss is not present. To facilitate the demonstration, we define CERL+2 as a modified version of CERL+ in which the code preventing multiple segment losses in one window of data from reducing the congestion window more than once is removed.

In figure 4.1, we set 10 connections between senders and G1, in addition, 10 connection between G2 and receivers. The bandwidth and propagation delay between G1and G2 are 8 Mbps and 50 ms, respectively. The bandwidth between all senders and G1 and between all receivers and G2 is set to 1 Mbps. S_1 to S_{10} are TCP senders running either CERL+, CERL+2,NewReno or Reno. Each sender initiates an ftp transfer to one of the receivers, R_1 to R_{10}. In order to test the TCPs under a wide range of traffic conditions, we randomly set the remaining simulation parameters as described below. The propagation delay of links Si G1 and G2 Ri, where i is an integer value ranging from 1 to 10, is set to a randomly generated number between 1 and 15 ms. The ftp transfer start time of each connection between senders S_1 to S_{10} and receivers R_1 to R_{10} is set to a randomly generated number between 1 and 150 s. The random values are generated using a uniformly distributed random variable and a combined multiple recursive generator. We chose 25 different seeds from among the 64 recommended seeds listed in the file rng.cc of the ns-2 source code and we number the seeds from 0 to 24. Figure 4.4 illustrates the ftp transfer start and end times that resulted when the seed was set to 0.

In Figure 4.5(a), we measure the throughput with 10 connections of Reno, CERL+ or CERL+2 and in figure 4.5(b), we measure the throughput with 10 connections of NewReno, CERL+ or CERL+2. These measurements are made as the seed ranges from 0 to 24 and q = 0%. In Figure 4.5(a), the performance of CERL+2 is almost identical to that of Reno at every seed value. However, CERL+ is sometimes at some seed values somehow different from NewReno as shown in figure 4.5(b) as CERL+ is a sender-side modification of Reno. We can notice that the performance of CERL+ implementation mostly performs better than Reno and NewReno owing to the CERL+ mechanism that prevents multiple window decrement strategy.

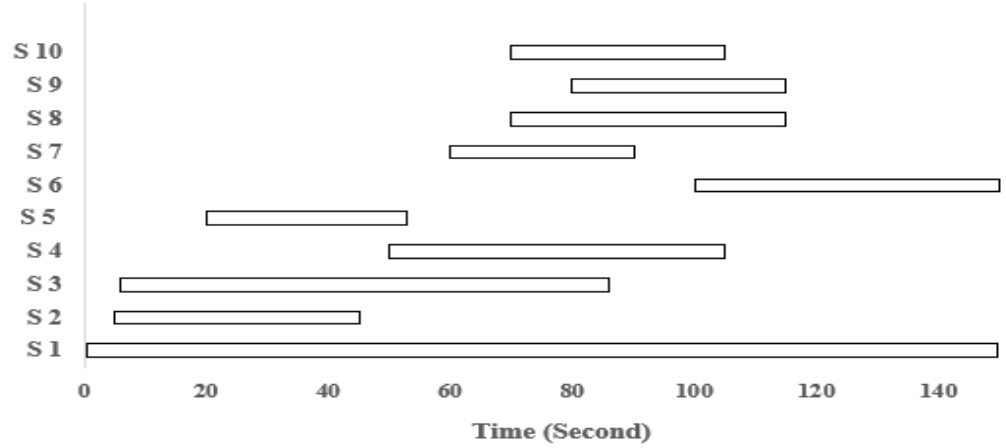

Figure. 4.4 The ftp transfer start and end time

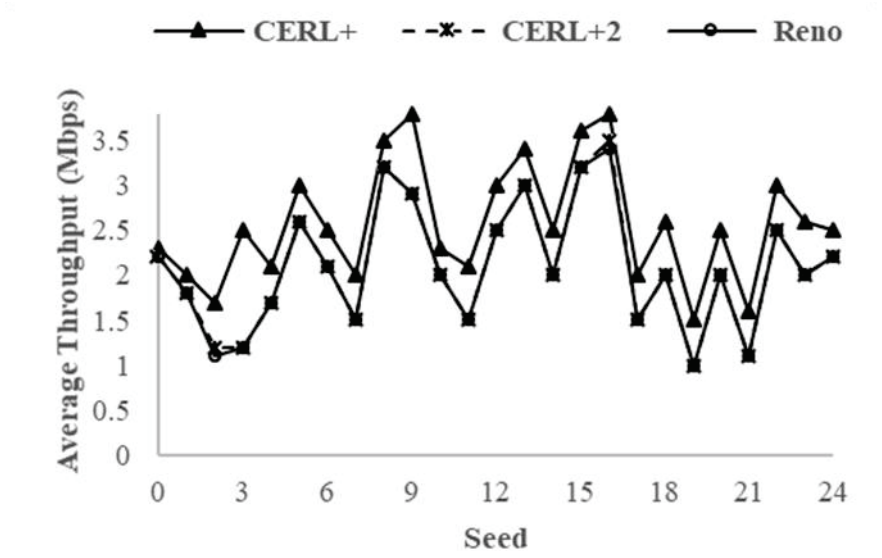

Figure. 4.5(a) Average throughput Reno versus seed when B = 8 Mbps, Tp = 50 ms

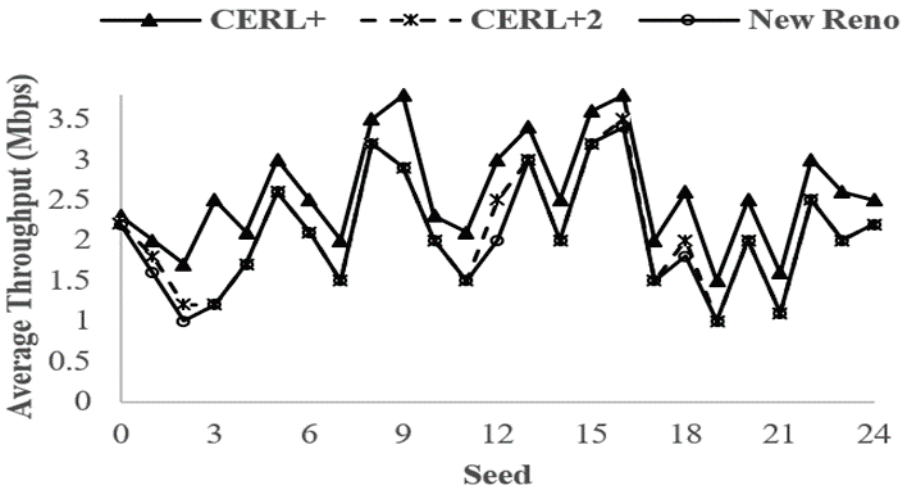

Figure. 4.5(b) Average throughput NewReno versus seed when B = 8 Mbps, Tp = 50 ms

Results show that congestion window of TCP New Reno and CERL+ are close without random loss. But, when there is a random loss in the link, TCP CERL+ is better than TCP New Reno because TCP CERL+ had a smaller number of dropped segments.

4.2 Network configuration

In this chapter, we consider two configurations. The first configuration is a wired/wireless topology, and the second is all wireless topology. We will implement various scenarios in both topologies

.

4.2.1 Network Configuration 1

4.2.1.1 Scenario 1 : One-way transmission

Scenario 1 evaluates the throughput of CERL+ with a small number of users, using one-way transmission. Users will apply the same protocol during the simulation by sending FTP files at different times.

Figure 4.1 shows the network topology used for scenario 1.

- N = 10.
- S_1 to S_{10} are TCP senders.
- R_1 to R_{10} are TCP receivers.
- G1 and G2 are routers.
- S_1 to S_{10} are connected to G1 via wire links; bandwidth and propagation delay for each link are 1 Mbps and 10 ms, respectively.
- R1 to R10 are connected to G2 via wireless links; bandwidth and propagation delay for each link are 1 Mbps and 10 ms, respectively.
- We denote L for the wired link between G1 and G2, the bandwidth and propagation delay for L1 are 8 Mbps and 50 ms, respectively.

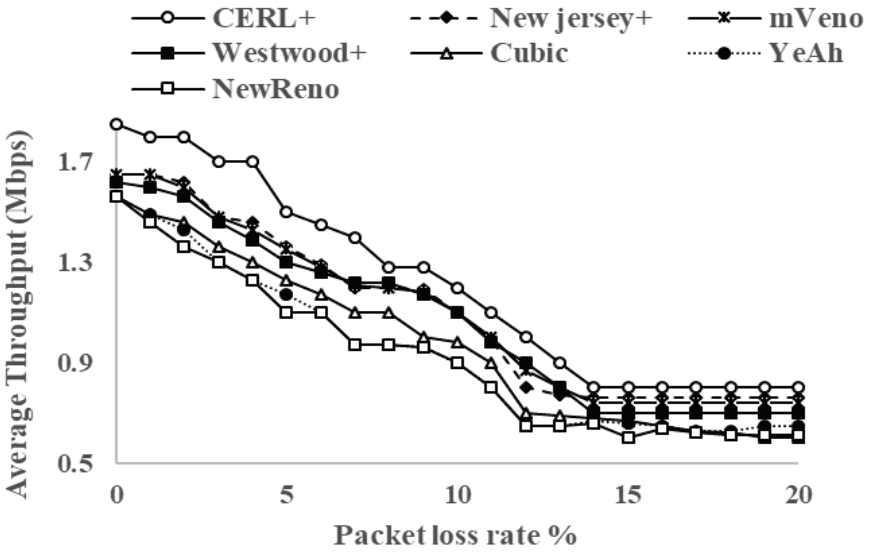

Figure 4.6: Average throughput versus packet loss in L

In Figure 4.6, CERL+ improves obviously the throughput compared to Figure 3.5. This implies that unnecessary packets drop off the bottleneck queue is reduced. This is because the variation in the amount of packets sent by each sender based on the condition of the traffic at G_1. In other words, one sender might increase its window while the other reduces its window based on the RTT measurement in CERL+ so possibility of congestion reduces. However, in case of CERL, all senders might have same window size leading to increasing the possibility of congestion which in turn would cut the congestion window and therefore decreases the throughput.

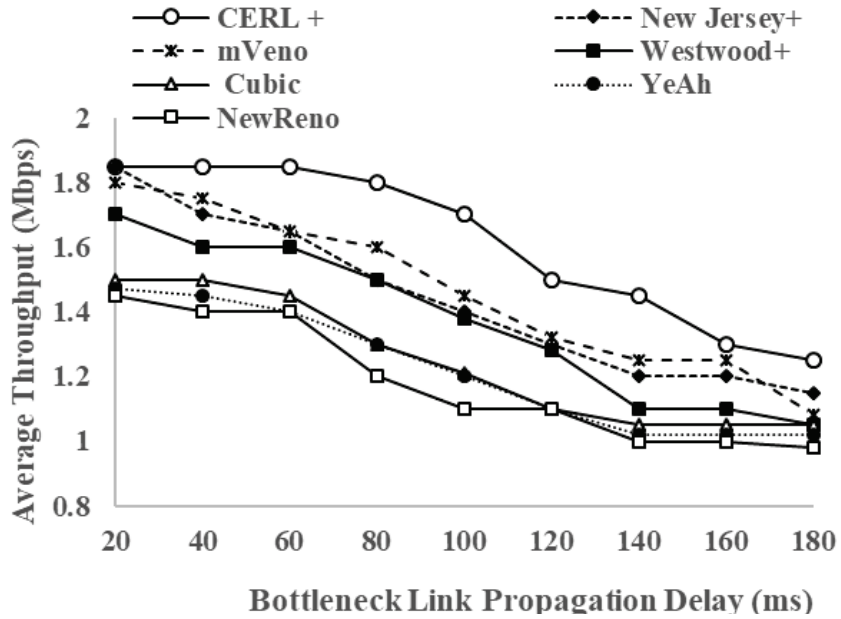

Figure 4.7: Average throughput versus bottleneck link propagation delay in L, where $q_L = 1\%$

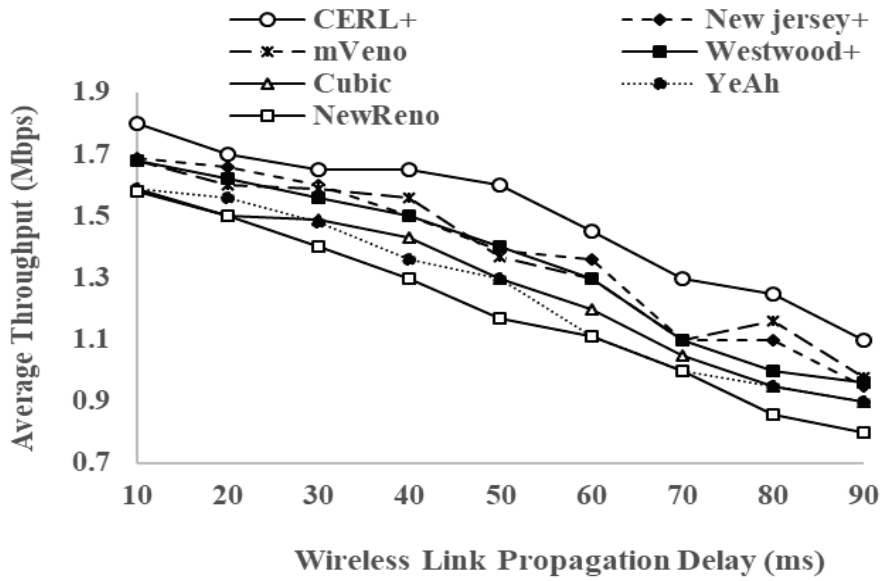

Figure 4.8: Average throughput versus wireless link propagation delay between receivers and G2, where $q_L = 1\%$.

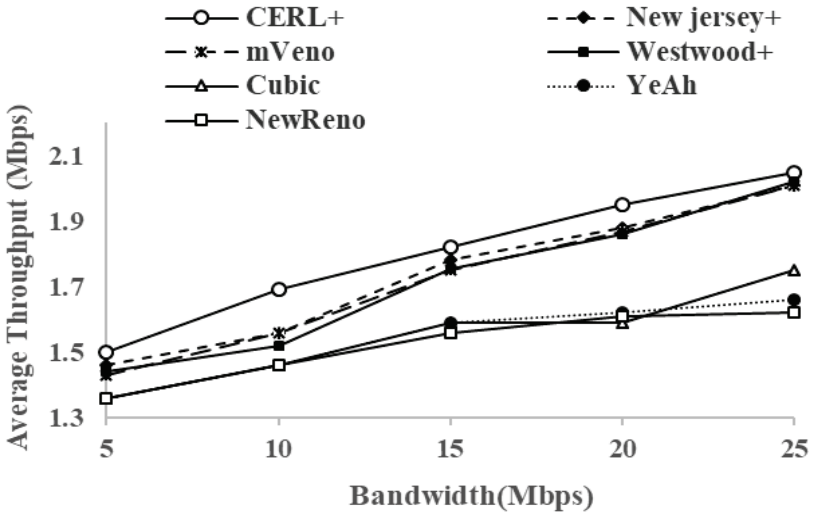

Figure 4.9: Average throughput versus bandwidth in L, where q_L=1%.

In Figures 4.7, 4.8 and 4.9, CERL+ performs better than that CERL in Figures 3.6, 3.7 and 3.8, respectively. The good achievement of CERL+ mechanism is its ability to control the congestion in a better way even in case of the increase of the propagation time delay at the receivers' side assuming its bandwidth limitation as shown in Fig 8. CERL+ throughput gains over other TCP variants are quite obvious.

4.2.1.2 Scenario 2: Two-way transmission

In scenario 2 test, we consider same assumptions of scenario 1 but for two-way transmission to evaluate the throughput of CERL+ vs other TCP variants. We consider the maximum segment size as 1024 bytes.

Figure 4.1 shows the network topology used for scenario 2.

- N=10
- S_1 to S_{10} are TCP senders.
- R_1 to R_{10} are TCP receivers.
- G1 and G2 are routers
- S_1 to S_{10} are connected to G1 via wire links, bandwidth and propagation delay for each link are 1 Mbps and 10 ms, respectively.
- R_1 to R_{10} are connected to G2 via wireless links with bandwidth and propagation delay for each link are 1 Mbps and 10 ms, respectively.
- We denote L for the wired link between G1 and G2, and the bandwidth and propagation delay for L are 8 Mbps and 50ms, respectively.

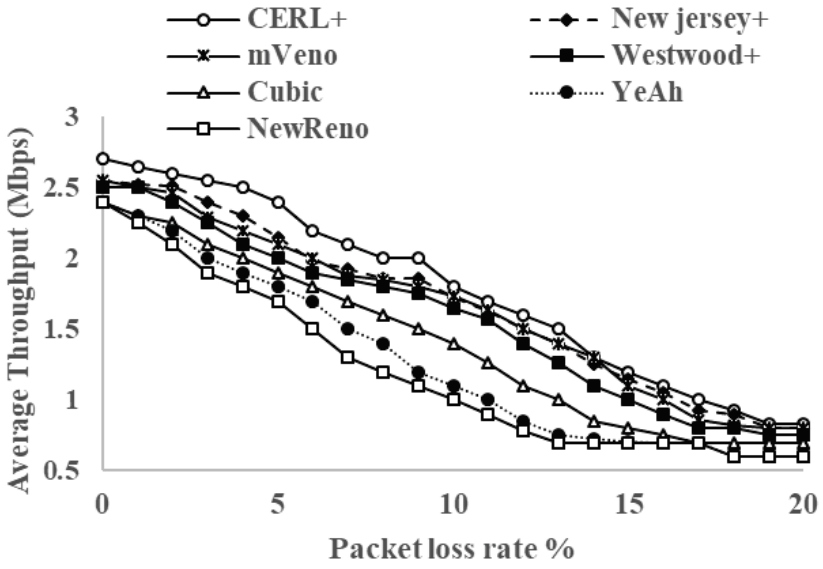

Figure 4.10: Average throughput versus packet loss in L

Our motivation to probe a better throughput of CERL compared to other variants was because of its poor performance in the two-way transmission shown in figure 3.9. In figure 4.10, CERL+ improves greatly the throughput compared to Figs 3.9 . The performance enhancement achieved by CERL+ is particularly shown in the relatively low random loss in the range below 10%. This proves that the mechanism of CERL+ which is to monitor the data traffic in the network through using the RTT measurement by each user is successful. As increasing the random loss rate above 10%, more timeouts occur which in turn would have protocols to go slow start phase and they all behave worse as indicated early.

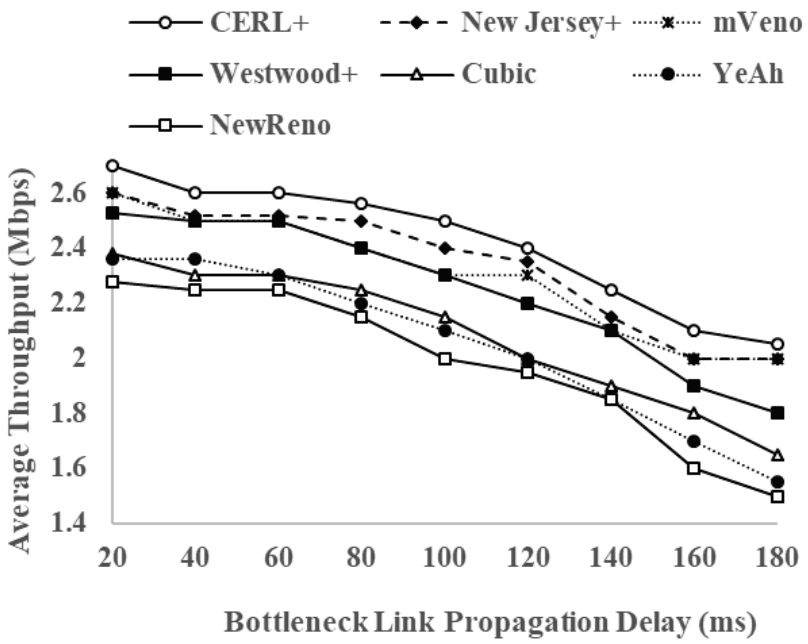

Figure 4.11: Average throughput versus bottleneck link propagation delay in L, where q_L=1%

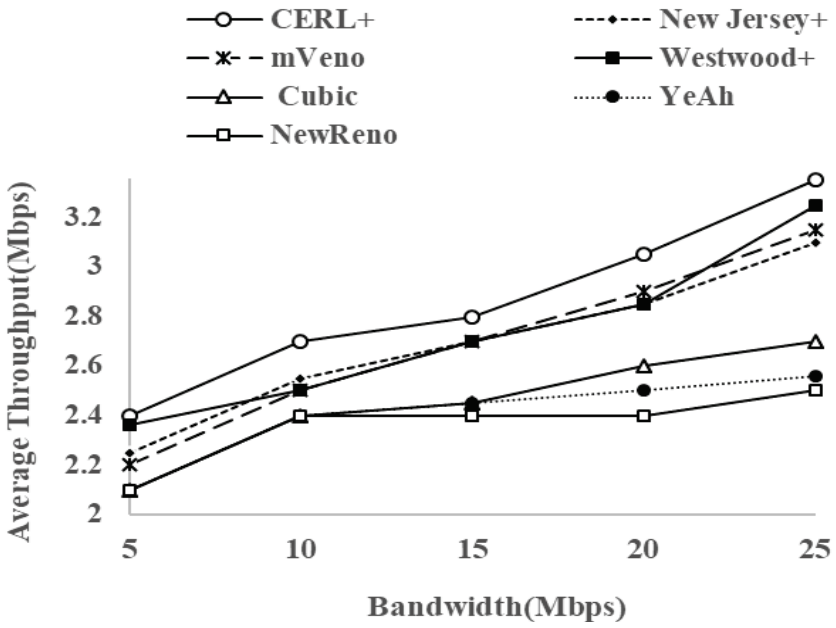

Figure 4.12: Average throughput versus bandwidth in L, where q_L=1%

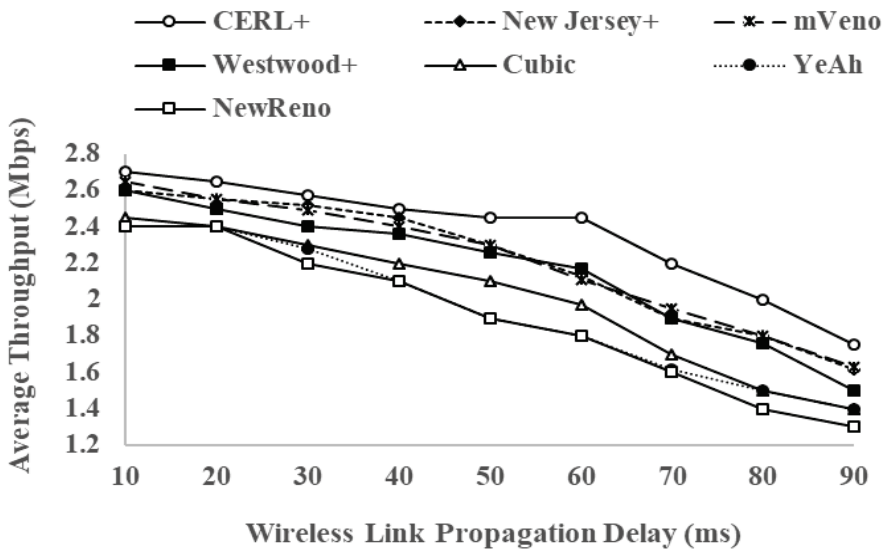

Figure 4.13: Average throughput versus wireless link propagation delay between receivers and G2,where q_L=1%.

In Figures 4.11, 4.12 and 4.13, CERL+ performs way better than that CERL in Figures 3.10, 3.11 and 3.12, respectively, compared to other TCPs. The good achievement of CERL+ mechanism is not only its ability to discriminate between random loss and congestion loss, but as well for its capability to alleviate the congestion in the bottleneck router queue. This is particularly in case of propagation delay increase at the receivers' side links shown in Fig 4.13 compared to Figure 3.12.

4.2.2. Network configuration 2

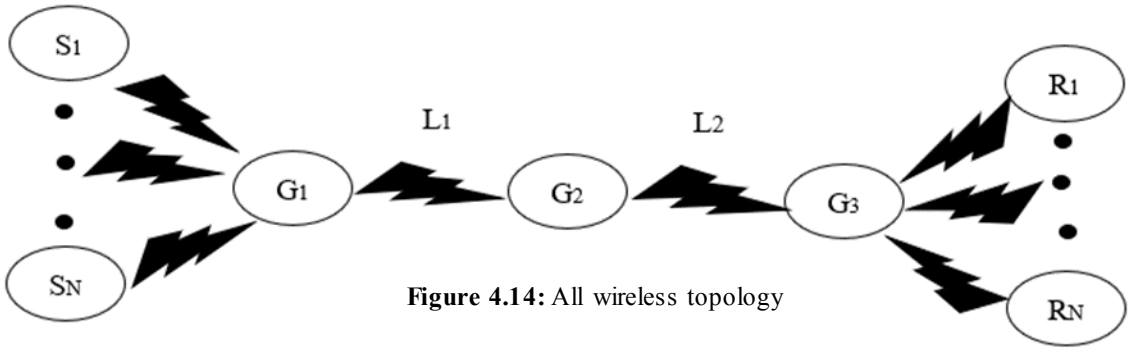

Figure 4.14: All wireless topology

In figure 4.14, the network configuration is shown. We assume N end-senders S_1 to S_N and as well N end-receivers R_1 to R_N. We assume that the network has three routers: G_1 to G_3. All transmission lines between end-senders and end-receivers including between routers (L1 and L2) are wireless. Random loss (q) may occur in L1 or L2 as we will show in our results in different scenarios.

4.2.2.1 Scenario 1: Two-way transmission with heavy load, where q_{L1} is 1% and q_{L2} is 1% loss rate.

Scenario 1 evaluates the throughput of CERL+ with a large number of users, using two-way transmission.

Figure 4.14 shows the network topology used for scenario 1.

- N=100
- S_1 to S_{100} are TCP senders.
- R_1 to R_{100} are receivers.
- G1 to G3 are routers.
- S_1 to S_{100} are connected to G1 via wireless links, bandwidth and propagation delay for each link are 1 Mbps and 20 ms, respectively.
- R_1 to R_{100} are connected to G3 via wireless links, also, bandwidth and propagation delay for each link are 1 Mbps and 20 ms respectively
- We denote L1 for the wireless link between G1 and G2, and the bandwidth and propagation delay for L1 are 85Mbps and 60ms, respectively.

- We denote L2 for the wireless link between G2 and G3, and the bandwidth and propagation delay for L1 are 85Mbps and 60ms, respectively. We add a constant 1% loss rate for L2.
- We consider the maximum segment size as 1024 bytes.

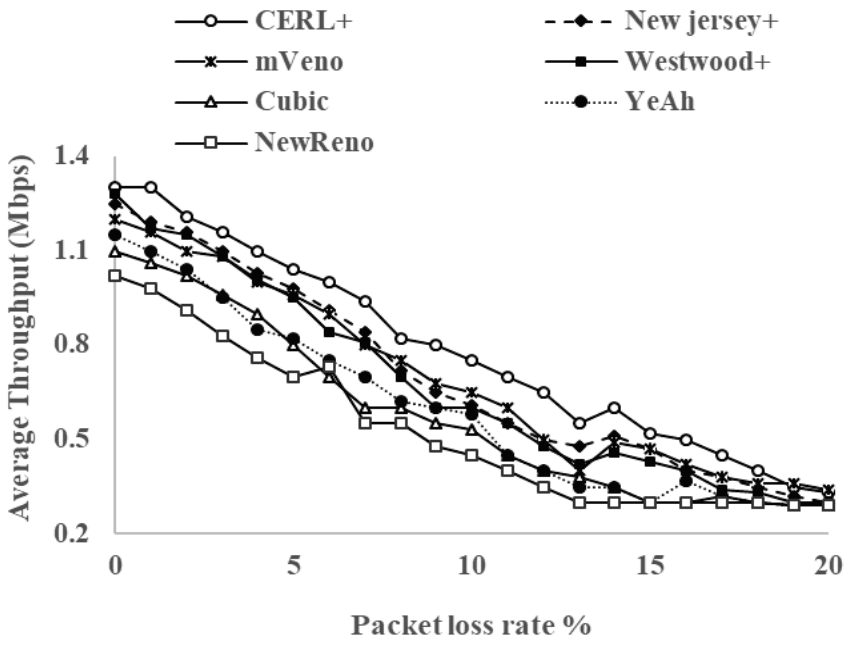

Figure 4.15: Average throughput versus packet loss in L1(q_{L1}), where q_{L2} =1%

In Figure 4.15, CERL+ outperforms other TCP variants for such heavy load of users on a two-way transmission. Some of the users in this large pool would reduce their *cwnd* based on the RTT measurement so other users would take advantage of that and increase their *cwnd* so the total throughput would be improved compared to other TCPs. Having all wireless network configuration would result in more random loss which would affect other TCPs *cwnd* negatively and more drops of the bottleneck queue. On the contrary, CERL+ would have a relatively minimal random loss effect compared to other TCPs. This proves that CERL+ better perform in wireless network such as WSN compared to those networks having a combination of wired/wireless links. Results show that TCP CERL+ throughput outperforms TCP Newjersey+ throughput by 60%, TCP Westwood+ by 73%, TCP mVeno by 41%, TCP Cubic by 90%, TCP YeAh by 105% and TCP NewReno by 118%.

In Figures 4.16, 4.17 and 4.18, CERL+ outperforms clearly other TCPs. Compared to wired/wireless topology, CERL+ behaves similarly in all wireless network considering the bandwidth and T_p variations. However to understand the whole picture, we should consider our discussion about the random loss rate in Figure 15.

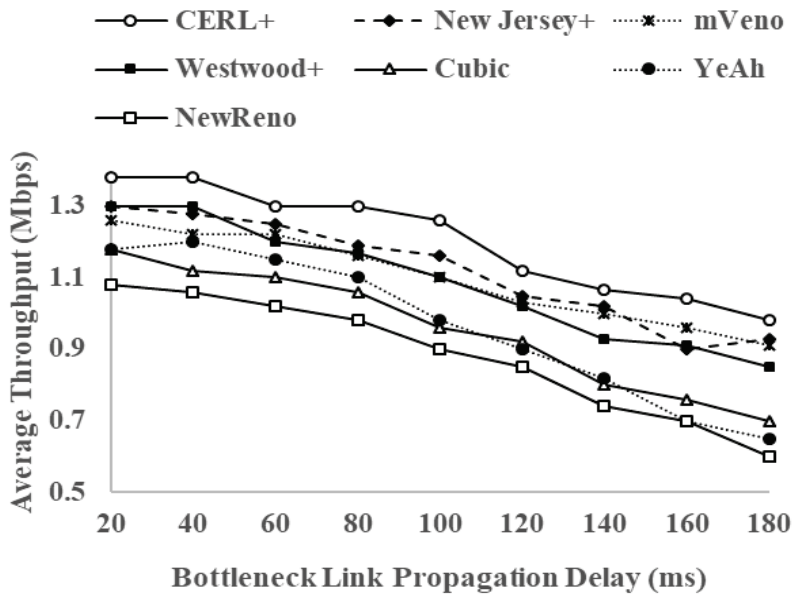

Figure 4.16: Average throughput versus bottleneck link propagation delay in L1, where propagation delay in L2 = 60 ms and q_{L1} = 1% ,and q_{L2} = 1%.

Figure 4.16 clarifies the throughput of CERL+ with other protocols when propagation delay for L1 ranges from 20 to 180 ms, where propagation delay for L2 is 60ms and q_{L1} is 1% and q_{L2} is 1% (random loss q in link L1 and random loss q in link L2). Results show that most of the throughputs are fairly similar such as between CERL + and NewJersey+, or CERL+ and Westwood+. But, CERL+ throughput still higher than all the mentioned protocols.

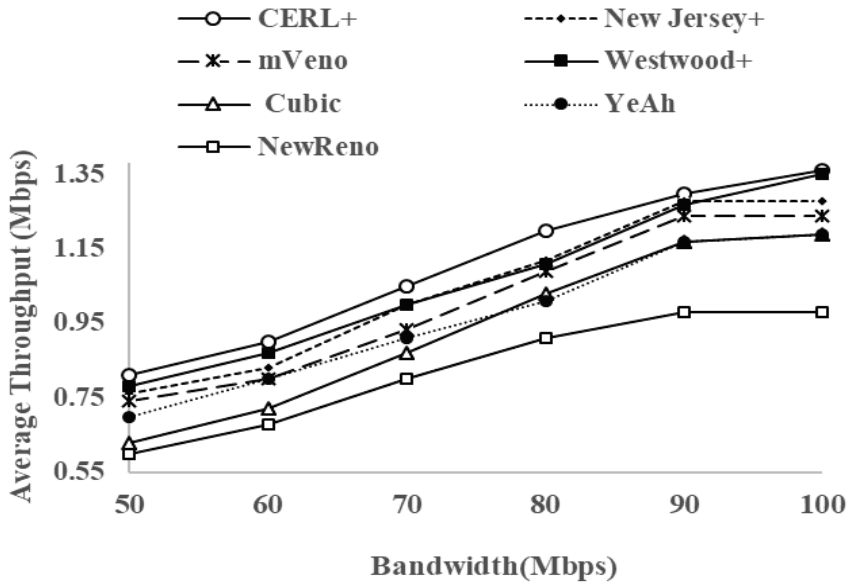

Figure 4.17: Average throughput versus in L1' bandwidth, where bandwidth of L2 = 85Mbps and q_{L1} = 1% and q_{L2} = 1%.

Figure 4.17 illustrates the throughput of TCP protocols with a bandwidth of L1 ranges from 50 to 100 Mbps where the bandwidth of L2 = 85Mbps and q_{L1} is 1% and q_{L2} is 1% (random loss q in link L1 and random loss q in link L2). Results show that CERL + gains a 148%, 125%, 117% ,60%, and 80% throughput development over NewReno, YeAh, Cubic, Westwood+, and Newjersey+ respectively, also, CERL+ increases gradually when the bandwidth increases in the link.

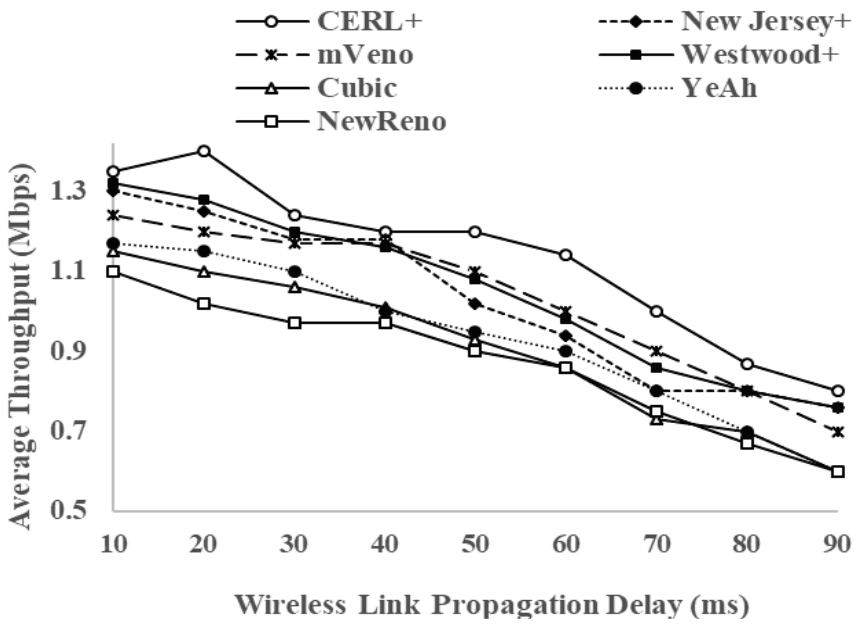

Figure 4.18: Average throughput versus wireless link propagation delay between senders and G1 where q_{L1} = 1%. and q_{L2} = 1%.

Figure 4.18 illustrates the throughput of TCP protocols and propagation delay between TCP sources and G1. The propagation delay ranges from 10 to 90 ms, where q_{L1} is 1% and q_{L2} is 1%. Results indicate that throughput CERL+ outperformance on other TCP protocols. However, there is a 1% loss rate in L2, because each user changes the transmission rate according to its average of rtts, therefore, the percentage of sending at the same time is low. When propagation delay equals to 90ms, CERL+ remains higher than other various TCPs.

4.2.2.2 Scenario 2: Two-way transmission with heavy load, where is q_{L1} is 1% and q_{L2} is 0% loss rate

Scenario 2 evaluates the throughput of CERL+ with a large number of users and use bidirectional flow. Users will use same protocol during the simulation, and senders will send FTP file at different times, and the receivers are able to send data with ACK, also the duration will be big for this scenario.

Figure 4.14 shows the network topology used for scenario 2.

- N=100
- S_1 to S_{100} are TCP senders.
- R_1 to R_{100} are receivers.
- G1 to G3 are routers.
- S_1 to S_{100} are connected to G1 via wireless links, bandwidth and propagation delay for each link are 1 Mbps and 20 ms respectively.
- R_1 to R_{100} are connected to G3 via wireless links, also, bandwidth and propagation delay for each link are 1 Mbps and 20 ms respectively.
- We denote L1 for the wireless link between G1 and G2, and the bandwidth and propagation delay for L1 are 85Mbps and 60ms, respectively.
- We denote L2 for the wireless link between G2 and G3, and the bandwidth and propagation delay for L2 are 85Mbps and 60ms, respectively. We add constant 0% loss rate for L2.

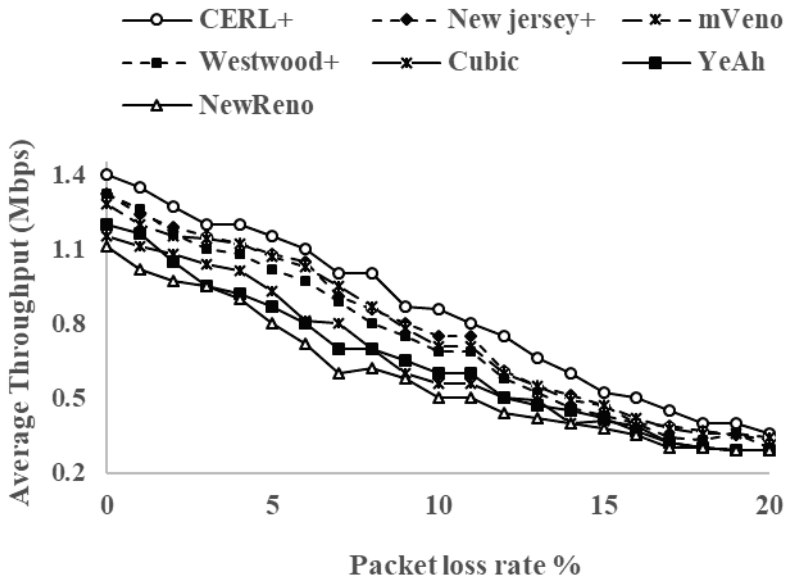

Figure 4.19: Average throughput versus packet loss in L1 (q_{L1}),where q_{L2} =0%

Figure 4.19 compares CERL+ with various TCPs when q_{L1} ranges from 0 to 20, where q_{L2} is 0%. Results show that throughput of CERL+ increases slightly similar to other protocols when there is no loss in L2, but CERL+ remains higher when loss decreases in the link. CERL+ is 8% high than CERL as shown in figure 3.13 when L2 has no rate loss.

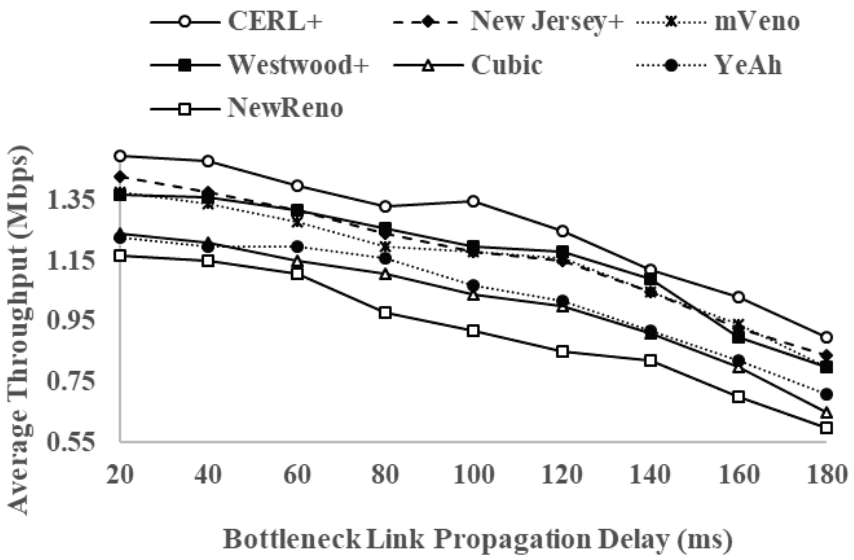

Figure 4.20: Average throughput versus bottleneck link propagation delay in L1 and q_{L1} =1%, where L2= 60ms and q_{L2} =0%.

Figure 4.20 shows that the throughput of CERL+ with different TCP throughputs when propagation delay for L1 ranges from 20 to 180 ms, where propagation delay of L2 is 60ms and q_{L2} is 0% (random loss q in link L1 and random loss q in link L2). Results show that CERL+ throughput outperforms Newjersey+ by 9%, mVeno by 7%,Westwood+ by 5%, Cubic by 23 % ,Yeah by 20%,and NewReno by 30% when propagation delay equals to 40 ms. Also, the CERL+ throughput reduces gradually when propagation delay increases in the bottleneck.

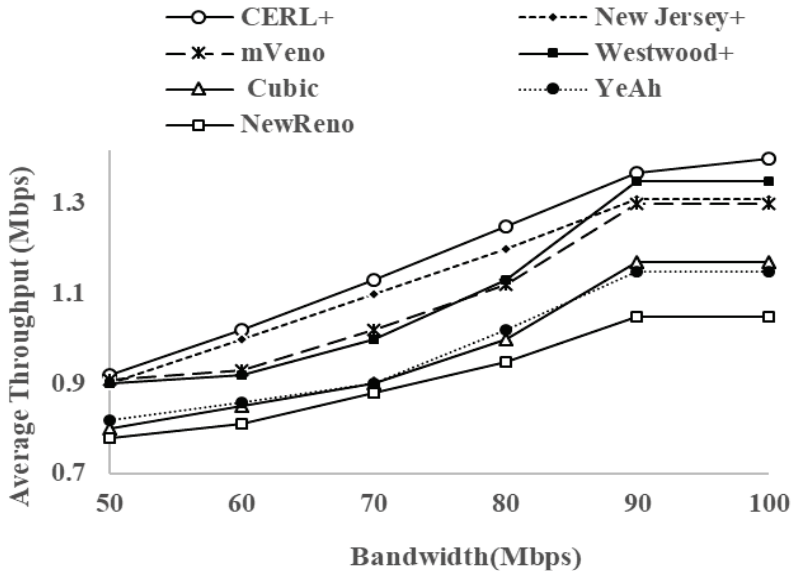

Figure 4.21: Average throughput versus L1' bandwidth , where bandwidth of L2=85Mbps and q_{L1} = 1% and q_{L2} = 0%.

Figure 4.21 shows the throughput of TCP protocols with the bandwidth of L1 ranges from 50 to 100 Mbps where L2 = 85Mbps and q_{L1} is 1% and q_{L2} is 0% (random loss q in link L1 and random loss q in link L2). Results indicate that throughput of TCPs are a bit close when the bandwidth is not high, but again CERL+ throughput still more than other protocols. Also, results show that CERL + performance outperforms CERL in figure 3.14 when L2 does not have the loss rate.

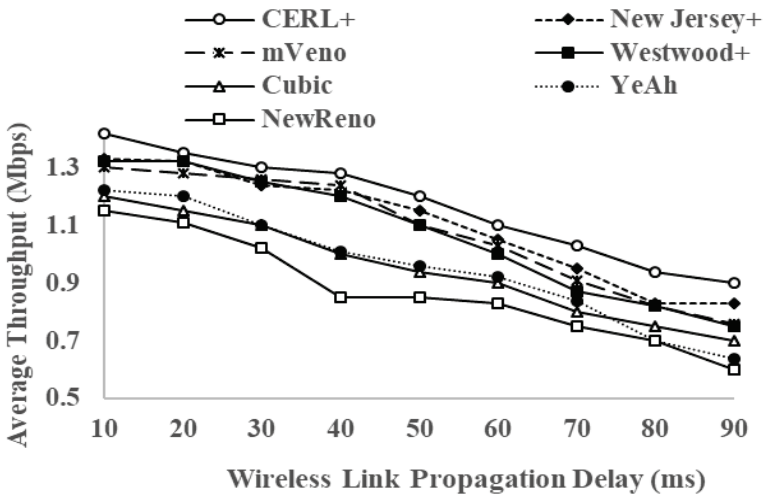

Figure 4.22: Average throughput versus wireless link propagation delay between senders and G1 where q_{L2} = 1% and q_{L2} = 0%.

Figure 4.22 illustrates the throughput of TCP protocols and propagation delay between TCP sources and G1, the propagation delay ranges from 10 to 90 ms, where q_{L2} is 0%. Results show that CERL+ throughput performance good comparing to other TCPs when propagation delay increases. CERL + gains a 38%, 30%, 18%,13%, and 7 % throughput development over NewReno, Yeah, mVeno, Cubic, Westwood+, and Newjersey+ respectively when propagation delay equals to 70ms.

4.2.2.3 Scenario 3: Two-way transmission with heavy load, where is q_{L1} is 1% and q_{L2} is 1% loss rate

Scenario 3 evaluates the throughput of CERL+ with a large number of users and use bidirectional flow. Users will use same protocol during the simulation, and senders will send FTP file at different times, and the receivers are able to send data with ACK, also the duration will be big for this scenario.

Figure 4.14 shows the network topology used for scenario 3.

- For Figure 4.23, N=50. For figures 4.24 and 4.25, N=100.
- S_1 to S_N are TCP senders.

- R_1 to R_N are receivers.
- G1 to G3 are routers.
- S_1 to S_N are connected to G1 via wireless links, bandwidth and propagation delay for each link are 1 Mbps and 20 ms respectively.
- R1 to RN are connected to G3 via wireless links, also, bandwidth and propagation delay for each link are 1 Mbps and 20 ms respectively.
- We denote L1 for the wireless link between G1 and G2, and the bandwidth and propagation delay for L1 are 85Mbps and 60ms, respectively for N=100. For N=50, the bandwidth and propagation delay for L1 are 70Mbps and 60ms, respectively.
- We denote L1 for the wireless link between G1 and G2, and the bandwidth and propagation delay for L1 are 85Mbps and 60ms, respectively for N=100. For N=50, the bandwidth and propagation delay for L1 are 70Mbps and 60ms, respectively.

In real applications, Reno and NewReno are still implemented so we assume that we have connections are still using these protocols while coexisting with others who use newer protocols. In figure 4.23, we test the throughput where 20 connections are of TCP New Reno coexisting with 30 connections of NewReno, YeAh, Cubic, Westwood+, mVeno, New Jersey+, or CERL+. Results show that all TCPs throughputs slightly increased compared to figure 4.15. Because in figure 4.15. wireless performance affects when there are a large number of users in the network and the bandwidth is limited [66][67]. But, CERL+ throughput remains higher than other TCPs in this is test.

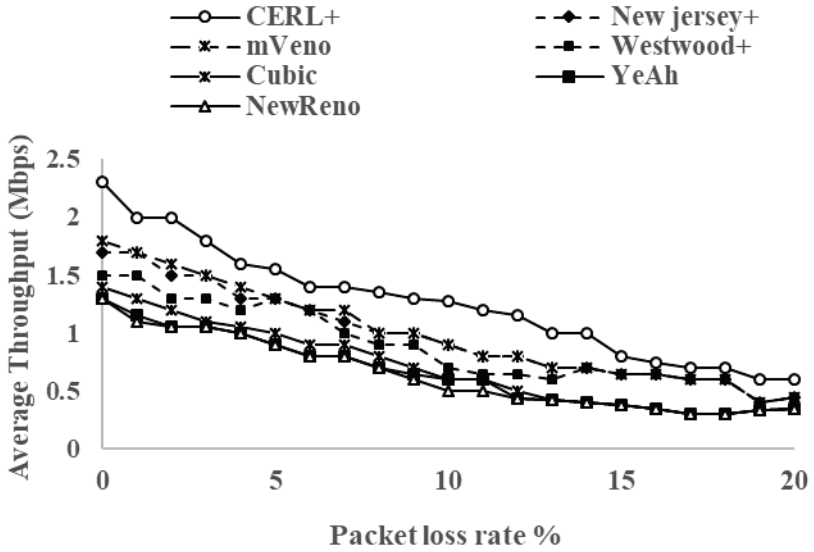

Figure 4.23: Average throughput of multiple coexisting connections of 20 New Reno with 30 other TCPs versus packet loss in L1 (q_{L1}),where q_{L2} =1%

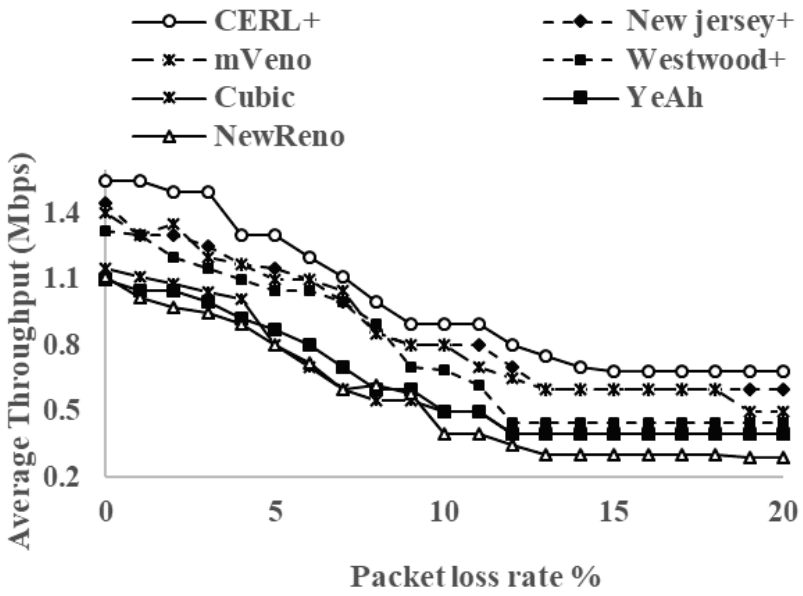

Figure 4.24: Average throughput of multiple coexisting of 10 connection other TCPs with 90 connections of TCP New Reno versus packet loss in L1 (q_{L1}),where q_{L2} =1%

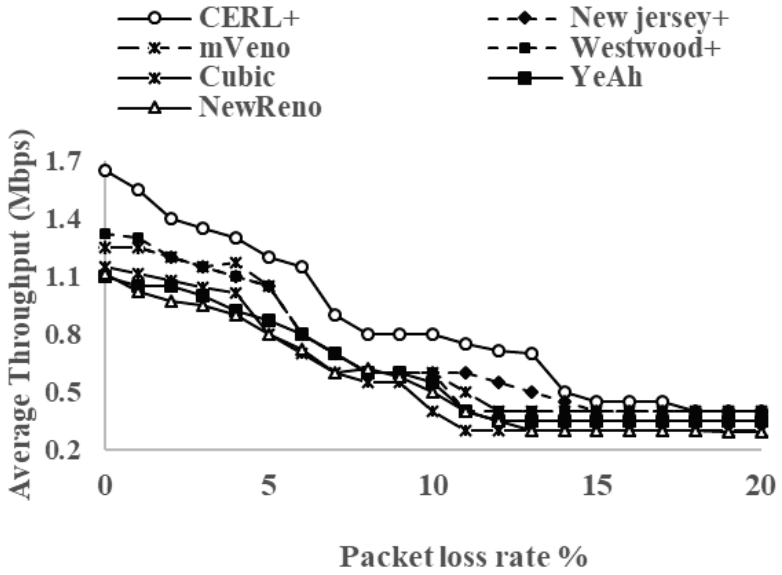

Figure 4.25: Average throughput of multiple coexisting of 40 connections other TCPs with 60 connections of TCP New Reno versus packet loss in L1 (q_{L1}),where q_{L2} =1%

In figure 4.24, we test the throughput where 90 connections are of TCP New Reno coexisting with 10 connections of YeAh, NewReno, Cubic, Westwood+, mVeno, New Jersey+, or CERL+**,** and figure 4.25, we test the throughput where 60 connections are of TCP New Reno coexisting with 40 connections of YeAh, NewReno, Cubic, Westwood+, mVeno, New Jersey+, or CERL+**.** Results show from both figures CERL+ throughput outperformance on other TCPs due to This is due to the fact that CERL is able to utilize some of the extra bandwidth left by the degradation of the NewReno connections.

4.3 Fairness

In this part, we prove the fairness of CERL+ and compare it with NewReno. We assume that there are two connections, and two ftp flows start to send at the same time. The bandwidth and propagation delay of links S1 G1 and S2G1 are set to 15 Mbps and 2 ms, respectively. In addition, the bandwidth and propagation delay of links G2 R1 and G2 R2 are set to 15 Mbps and 2 ms, respectively.

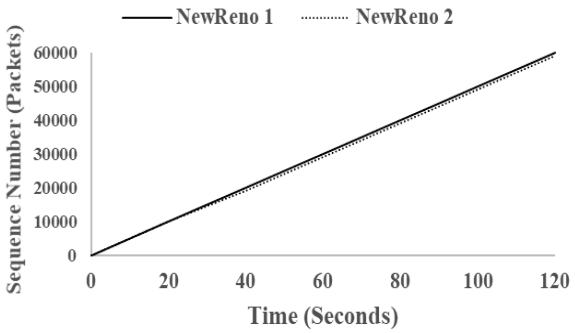

Figure 4.25: Two NewReno connections without random loss

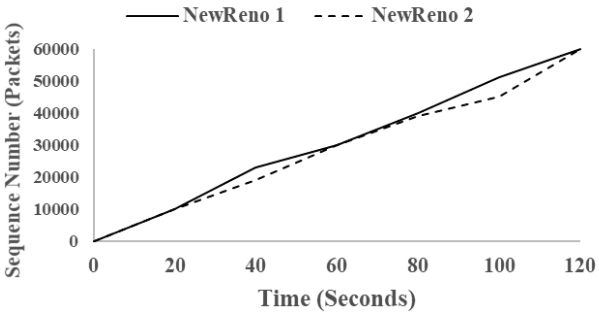

Figure 4.26: Two NewReno connections with 1% random loss

Figure 4.25 illustrates sequence number growth two NewReno connections without 1% random loss rate in between G1 and G2, and figure 4.26 illustrates sequence number growth two NewReno connections with 1% random loss. It is clear to see that both NewReno connections are close to each other when the link does have loss rate.

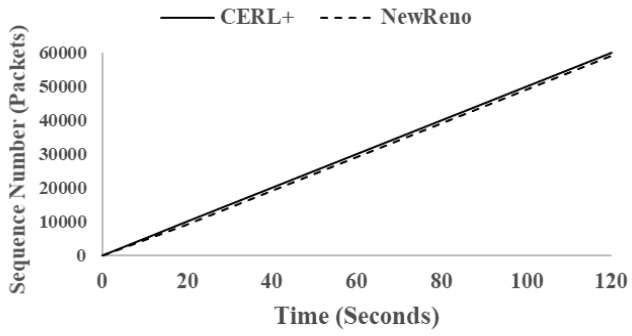

Figure 4.27: One CERL+ connection and one NewReno connection without random loss

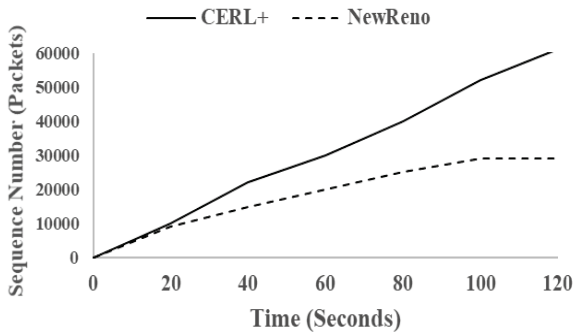

Figure 4.28: One CERL+ connection and one New Reno with 1% random loss

Figure 4.27 illustrates sequence number growth one CERL+ connection and one NewReno connection without 1% random loss rate in between G1 and G2 and figure 4.28 illustrates sequence number growth one CERL+ connection and one NewReno connection with1% random loss rate in between G_1 and G_2. It is clear to see that in figure 4.27, both TCPs are close to each other, and both TCPs increase parallelly. However, CERL+ and NewReno growth differently when there is a loss rate in the link.

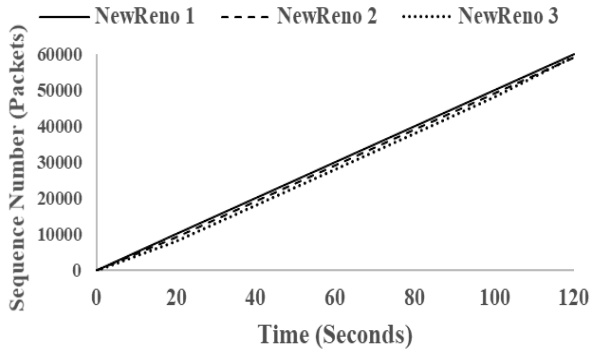

Figure 4.29: Three NewReno connections without Random loss

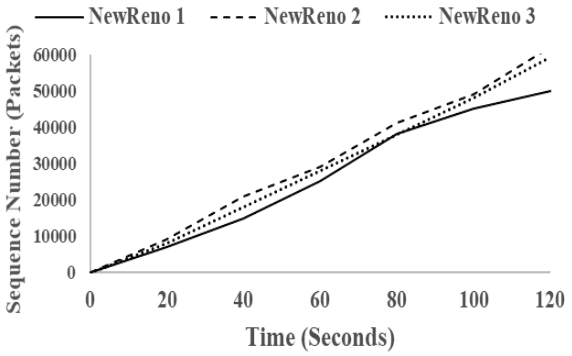

Figure 4.30: Three NewReno connections with1% Random loss

Figure 4.29 illustrates sequence number growth three NewReno connection without random loss rate in between G_1 and G_2, and figure 4.30 illustrates sequence number growth NewReno connection three with 1% random loss rate in between G1 and G2

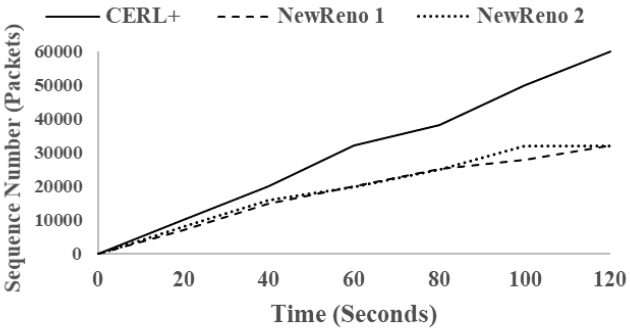

Figure 4.31: One CERL+ and two NewReno connections with 1% Random loss

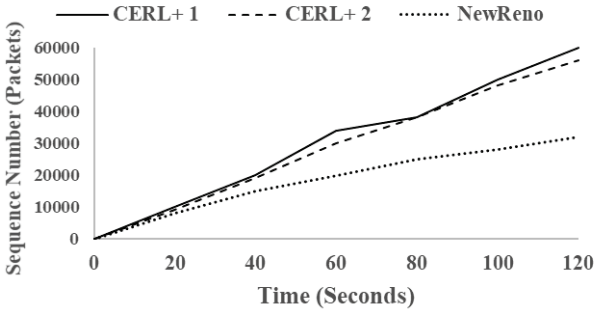

Figure 4.32: Two CERL+ and one NewReno connections with Random loss

Figure 4.31 illustrates sequence number growth one CERL+ connection and Two NewReno connections with 1% loss rate in between G_1 and G_2, and figure 4.32 illustrates sequence number growth two CERL+ connections and one NewReno connection with1% random loss rate in between G1 and G2. It is clear to see that in both figures, the sequence number of CERL+ is higher than NewReno sequence number even when there are two connections of CERL+. Figure 4.33

illustrates sequence number growth Two CERL+ connections with 1% loss rate in between G_1 and G_2.

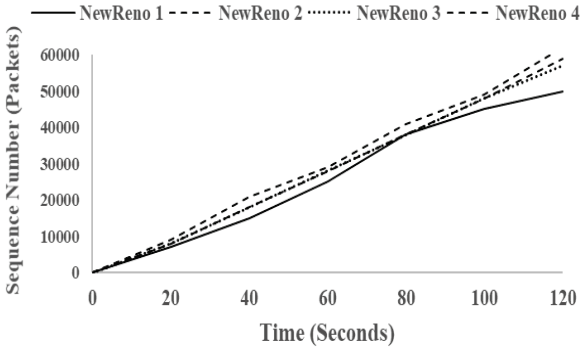

Figure 4.33: Four NewReno connections with 1% random loss

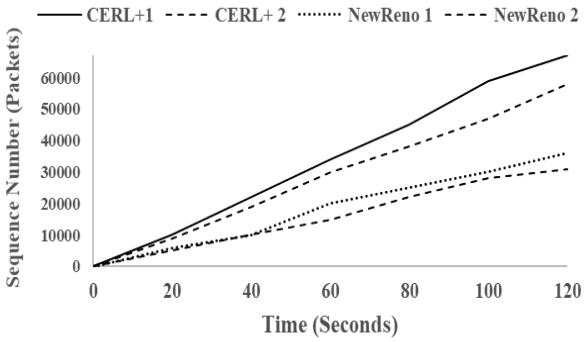

Figure 4.34: Two CERL+ connections and Two NewReno with 1% random loss

Figure 4.33 illustrates sequence number growth Four NewReno connections with 1% loss rate in between G_1 and G_2, and figure 4.34 illustrates sequence number growth two CERL+ connections and Two NewReno connection with1% random loss rate in between G1 and G2. It is clear to see that in both figures, the sequence number of CERL+ is higher than NewReno sequence number even when there are two connections of CERL+. Results show that in figure 4.34 there was a small amount of competition between the connections, but the two CERL+ connections still display a superior segment sequence evolution to either of the two New Reno connections.

4.4 Conclusion

It is clear to see that when we make dynamic value instant to fixed value in dynamic queue length threshold N, we increase the performance of network. TCP CERL+ improves the throughput, when nodes have different values of A, particularly when there are delays for the ACKs. Therefore, it helps more to distinguish between random and congestion loss through relies on round trip time.

TCP CERL + achieves a good throughput when there a large number of nodes in the network, and the data flow in both directions. Sometimes, there are congestions in the bottlenecks, but TCP CERL+ maintains the traffic of packets symmetrical between stations.

From the results, TCP CERL+ demonstrates that it has a high throughput comparing to Newjersey, mVeno, Westwood+, Cubic, Yeah, and NewReno. Newjersey, and Westwood+ throughputs are good, but they still less than TCP CERL+ throughput, when L2 has loss rate or not.

Also, from the results, TCP CERL+ throughput outperforms CERL throughput around 8% to 15 % , when using piggybacking comparing to results in section 3.4.

Chapter 5

Conclusion & Future work

In this thesis, we proposed a new version of TCP CERL called TCP CERL+. The idea of CERL+ is to improve the performance of wireless networks when there are a large number of connections and random loss in the link. We have examined CERL+ and compared to mVeno, New Jersey+, Westwood+, Cubic, YeAh, and NewReno in one-way and two-way transmissions. Our preliminary TCP CERL version throughput is good, but it is not that high compared to mVeno, and New Jersey+ connections.

CERL+ is similar in its implementation to CERL in the sense that it would not behave less than NewReno under any condition. Additionally, CERL+ achieves excellent performance in terms of throughput in different network system configurations. CERL+ is fair enough not to embezzle traffic resources such as bandwidth from other coexisting links that use NewReno mechanism. Instead, simulated results prove that CERL+ is efficiently fair with NewReno assuming having no random loss while it has a quite limited effect on those connections using NewReno and sharing the bottleneck link.

In case of two-way transmission with a heavy load, CERL+ gains an 148%, 130%, 110%, 92%, 125% and 105% over throughput improvement over NewReno, YeAh, Westwood+, mVeno, Cubic , NewJersey+, and, respectively

Future Work

TCP CERL+ is one of the significant protocols to distinguish between random loss, and congestion loss, particularly with the development of wireless networks. From the tests, CERL+ proves can reach the highest throughput as possible comparing several protocols. The simulation tool used for these tests was open source NS2. NS2 is a useful simulation for network researches, but the issue is not able to deal with big networks, and sometimes does not run the simulation correctly. My future work, I want to add more features to CERL+ and implement them by using new simulation tools like Tossim or NS3.

References

[1] Y. Zhang, P. Chowdhury, M. Tornatore, and B. Mukherjee, "Energy efficiency in telecom optical networks," IEEE Commun. Surveys Tuts., vol. 12, no. 4, pp. 441–458, 2010.

[2] Neha Trivedi, G. Kuamr, Teena Raikwar, "Survey onMAC protocol for Wireless Sensor Network", International Journal of Emerging Technology and Advanced Engineering, Vol. 3, Issue 2, pp.558-562, February 2013

[3] Michele Zorzi, James Zeidler, A. Lee Swindlehurst, Michael Jensen, "Cross-Layer Issues in MAC protocol Design for MIMO Ad Hoc Networks", IEEE Wireless Communication, pp. 62-76. August 2006

[4] Sharma,M.,& Sarmah,.M, "Technical Issues and Challenges involve in designing a MAC protocol for Wireless Ad hoc Network". International Journal of Computer Networks and Wireless Communications (IJCNWC), ISSN: 2250-3501Vol.5, No.1, February 2015.

[5] Z. Li, Y. Liu, M. Li, J. Wang, Z. Cao, "Exploiting ubiquitous data collection for mobile users in wireless sensor networks," IEEE Trans.Parallel and Distributed Systems, vol. 24, no. 2, pp. 312-326, 2013.

[6] Y. Zhang, Y.-P. Tian, "Consensus of data-sampled multi-agent systems with random communication delay and packet loss", IEEE Trans. Autom. Control, vol. 55, no. 4, pp. 939-943, Apr. 2010

[7] Y. Liang, H. V. Poor, S. Shamai, "Secure communication over fading channels", *IEEE Trans. Inform. Theory*, Nov. 2006.

[8] L.-P. Tung, W.-K. Shih, T.-C. Cho, Y.S. Sun, M.C. Chen, "TCP Throughput Enhancement over Wireless Mesh Networks", *IEEE Communications Magazine*, vol. 45, no. 11, pp. 64-70, Nov. 2007.

[9] Saeed V. Vaseghi, "Noise And Distortion". Advanced Digital Signal Processing and Noise Reduction, Second Edition. ISBNs: 0-471-62692-9 (Hardback): 0-4 70-84162-1 (Electronic)

[10] A. Tsertou and D. Laurenson, "Revisiting the hidden terminal problem in a CSMA/CA wireless network," IEEE Transactions on Mobile Computing, vol. 7, no. 7, pp. 817–831, July 2008

[11] M. C. Domingo, "Magnetic Induction for Underwater Wireless Communication Networks," IEEE Trans. Antennas and Propagation, vol. 60, no. 6, 2012, pp. 2929–39

[12] N. Parvez, A. Mahanti, C. Williamson, "An Analytic Throughput Model for TCP NewReno", *Proc. of IEEE/ACM TON*, vol. 18, no. 2, pp. 448-461, April 2010

[13] J. Sing and B. Soh, "TCP New Vegas: improving the performance of TCP Vegas over high latency links," in Proc. Fourth IEEE Int. Symp. on Netw. Comput. and Appl., Cambridge, MA, July 2005, pp. 73–82

[14] K. Shi, Y. Shu, O. Yang, and J. Luo, "Receiver-assisted congestion control to achieve high throughput in lossy wireless networks," IEEE Trans. Nucl. Sci., vol. 57, no. 2, pp. 491–496, Apr. 2010

[15] Nikitinskiy, M.A. and Chalyy, D.Ju., Performance analysis of trickles and TCP transport protocols under high-load network conditions, Automatic Control and Computer Sciences, 2013, vol. 47, pp. 359–365

[16] Bathla, N.; Kaur, A.; Singh, G., Relative Inspection on TCP variants Reno, NewReno, Sack, Vegas in AODV. International Journal of Research in Engg. & Applied Science, vol. 4,2014, pp. 1-12

[17] S. Shin, D. Han, H. Cho, J. M. Chung, I. Hwang, and D. Ok, "Tcp and mptcp retransmission timeout control for networks supporting wlans," IEEE Communications Letters, vol. 20, no. 5, pp. 994–997, May 2016

[18] B. Soelistijanto and M. P. Howarth, "Transfer reliability and congestion control strategies in opportunistic networks: A survey," IEEE Communications Surveys & Tutorials, vol. 16, 2014

[19] F. Zafar, Z. Mahmood, O. Ayoub, Z. Zhao, "Throughput Analysis of TCP SACK in comparison to TCP Tahoe Reno and New Reno against Constant Rate Assignment (CRA) of 2500 and 4500 bps", *Journal of Computer Science & Computational Mathematics*, vol. 2, July 2012

[20] R. Kaur, G. S. Josan, "Performance Evaluation Of Congestion Control Tcp Variants In Vanet Using Omnet++", International Journal of Engineering Research and Applications (IJERA), Vol. 2, No. 5, pp. 1682-1688, 2012

[21] W. Bao, V. W. S. Wong, V. C. M. Leung, "A model for steady state throughput of TCP CUBIC", *Proc. IEEE GLOBECOM*, pp. 1-6, Dec. 2010

[22] A. Pradeep, N. Dinakaran, P. Angelin, "Comparison of drop rates in different TCP variants against various routing protocols", *International Journal of Computer Applications*, vol. 20, no. 6, 2011

[23] Yuvaraju B. N, Niranjan N Chiplunkar - "Scenario Based Performance Analysis of Variants of TCP using NS2-Simulator" International Journal of Advancements in Technology, ISSN 0976-4860 Vol 1, October 2010.

[24] Hanaa Torkey, Gamal Attiya and Ibrahim Z. Morsi, "Modified Fast Recovery Algorithm for Performance Enhancement of TCP-New Reno" International Journal of Computer Applications (0975 – 8887) Volume 40– No.12, February 2012

[25] Biaz S, Vaidya NH. 'De-randomizing' congestion losses to improve TCP performance over wired-wireless networks. IEEE/ACM Transactions on Networking, 2005

[26] M. Ivanovich, P. Bickerdike, and J. Li, "On TCP performance enhancing proxies in a wireless environment," IEEE Communications Magazine, vol. 46, pp. 76–83, September 2008

[27] Keshav S, Morgan S. Smart retransmission: performance with overload and random losses. INFOCOM'97

[28] SAMARAWEERA, N. AND FAIRHURST, G. April 1998. Reinforcement of TCP error recovery for wireless communication. ACM SIGCOMM Computer Communication Review 28, 2, 30–38

[29] S. Floyd. "TCP and Explicit Congestion Notification". ACM Computer Communication Review, 24(5):10– 23, Oct. 1994

[30] PHANISHAYEE, A., KREVAT, E., VASUDEVAN, V., ANDERSEN, D. G., GANGER, G. R., GIBSON, G. A., AND SESHAN, S. Measurement and analysis of tcp throughput collapse in cluster-based storage systems. In Proceedings of the 6th USENIX Conference on File and Storage Technologies (2008), pp. 12:1–12:14.

[31] Callegari, C., Giordano, S., Pagano, M., Pepe, T., A survey of congestion control mechanisms in linux tcp. In: Distributed Computer and Communication Networks. Vol. 279 of Communications in Computer and Information Science. Springer, pp. 28–42,2014

[32] S. Gangadhar, T. A. N. Nguyen, G. Umapathi, and J. P. Sterbenz, "TCP Westwood+ protocol implementation in ns-3," in Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques, pp. 167–175, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013.

[33] Q. Peng, A. Walid, J. Hwang, and S. H. Low, "Multipath TCP: Analysis, design, and implementation," IEEE/ACM Trans. Netw, vol. 24, no. 1, pp. 596–609, Feb. 2016.

[34] Dalal, P.; Sarkar, M.; Dasgupta, K.; Kothar, N. Link Layer Correction Techniques and Impact on TCP's Performance in IEEE 802.11 Wireless Network. Commun. Netw. 2014, 6, 49–60.

[35] Katre.A.(2015, August 21). Explanation of the Three-Way Handshake via TCP/IP. Retrieved from URL

<https://swapnilkatre.wordpress.com/2015/08/21/explanation-of-the-three-way-handshake-via-tcpip/ >

[36] Kristoff,J . The Transmission Control Protocol. Retrieved from URL

https://condor.depaul.edu/jkristof/technotes/tcp.html

[37] B. Forouzan, Data Communications and Networkings, 4th ed. New York, NY, USA: McGraw-Hill, 2006

[38] Baiocchi A, Castellani AP, Vacirca F. YeAH-TCP: yet another highspeed TCP. In: Proceedings of the PFLDnet. Marina Del Rey (Los Angeles, California): ISI; February 2007

[39] V. Jacobson. Congestion avoidance and control. SIGCOMM Comput. Commun. Rev., 18(4):314–329, 1988

[40] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," in Proc. IEEE INFOCOM, Hong Kong, 2004, pp. 2514–2524

[41] Y.-T. Li, D. Leith, and R. N. Shorten, "Experimental evaluation of TCP protocols for high-speed networks," IEEE/ACM Transactions on Networking, vol. 15, no. 5, pp. 1109–1122, 2007

[42] S. Ha, I. Rhee, and L. Xu. Cubic: A new tcp-friendly high-speed tcp variant. ACM SIGOPS Operating Systems Review, 42, 2008

[43] N. Kaur, S. Umrao, R. K. Gujral, "Simulation based Analysis of TCP Variants over MANET Routing Protocols using NS2." International Journal of Computer Applications (0975 – 8887) Volume 99– No.16, August 2014

[44] Shivaranjani, M., Shanju, R., Joseph Auxilius Jude, M., and Diniesh, V.C.: 'Analysis of TCP's micro level behaviour in wireless multi-hop environment'. IEEE Conf. on Computer Communication and Informatics, Coimbatore, India, January 2016, pp. 1–6, doi: 10.1109/ ICCCI.2016.7480006

[45] K. Ong, S. Zander, D. Murray, T. McGill, "Experimental Evaluation of Less-than-Best-Effort TCP Congestion Control Mechanisms" in 42nd IEEE Conference on Local Computer Networks (LCN), Singapore:IEEE, 2017

[46] S. Sathya Priya and K. Murugan, " Enhancing TCP Fairness in Wireless Networks using Dual Queue Approach with Optimal Queue Selection", Wireless Personal Communications, vol. 83, no 2, pp. 1359-1372, Jul. 2015.

[47] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast congestion control for TCP," in Proc. ACM CoNEXT, 2010, p. 13

[48] Wang, J., Wen, J., Zhang, J., Han, Y.: 'TCP-FIT: an improved TCP congestion control algorithm and its performance'. INFOCOM, 2011 Proc. IEEE, 2011, pp. 2894–2902

[49] P. Dong, J. Wang, J. Huang, H. Wang, and G. Min, ''Performance enhancement of multipath TCP for wireless communications with multiple radio interfaces,'' IEEE Trans. Commun., vol. 64, no. 8, pp. 3456–3466, Aug. 2016

[50] Milad. A, Lafi.S, Algaet.M, "Piggyback Scheme over TCP in Very High Speed Wireless LANs: Review" International Journal of Data Science and Analysis. Vol. 3, No. 6, 2017, pp. 69-76

[51] Lee, T.H., Y.W. Kuo, Y.W. Huang and Y.H. Liu, 2010. "To Piggyback or not to piggyback acknowledgments?" Proceedings of the IEEE 71th Vehicular Technology Conference, May 16-19, IEEE Xplore Press, Taipei, pp: 1-5

[52] P. Rastin, S. Dirk, and M. Daniel, "Performance Evaluation of Piggyback Requests in IEEE 802.16," in Proc. IEEE Vehicular Technology Conf., Baltimore, MD, 2007, pp. 1892-1896

[53] J. He, K. Yang, K. Guild, and H.-H. Chen, "On bandwidth request mechanism with piggyback in fixed IEEE 802.16 networks," Wireless Communications, IEEE Transactions on, vol. 7, no. 12, pp. 5238-5243, 2008

[54] Gupta K, "Comparison based Performance Analysis of UDP/CBR and TCP/FTP Traffic under Routing Protocol in MANET", International Journal of Computer Application, VoL-56, 2012

[55] ns-2 network simulator. LBL, Retrieved from URL: http://www.isi.edu/nsnam/ns

[56] Gnuplot. Retrieved from URL http://www.gnuplot.info/

[57] Malladi, R., & Agrawal, D. P. (2002). Current and future applications of mobile and wireless networks. Communications of the ACM, 45(10)

[58] Kahn, J.M., et al. Next Century Challenges: Mobile Networking for Smart Dust. ACM Mobicom, 1999.

[59] A. A. Reeves, "Remote monitoring of patients suffering from early symptoms of dementia", in Proc. Int. Workshop Wearable Implantable Body Sensor Netw., London, U.K., Apr. 2005.

[60] F. Hu and S. Kumar, "Multimedia query with QoS considerations for wireless sensor networks in telemedicine", in Proc. Soc. Photo-Optical Instrum. Eng. Int. Conf. InternetMultimedia Manage. Syst., Orlando, FL,Sep. 2003.

[61] I. F. Akyildiz, T. Melodia, and K. R. Chowdury, "Wireless multimedia sensor network: A survey," IEEE Wireless Commun., vol. 14, no. 6, pp. 32–39, Dec. 2007

[62] H. El-Ocla, "TCP CERL: Congestion control enhancement over wireless networks," Wireless Netw., vol. 16, no. 1, pp. 183–198, Jan. 2010

[63] Sreekumari, P., & Chung, S. H. (2011). Tcp nce: A unified solution for non-congestion events to improve the performance of tcp over wireless networks. EURASIP Journal on Wireless Communications and Networking, 2011, 1–20. doi:10.1186/1687-1499-2011-23

[64] Stevens, W. R. (1994). TCP/IP illustrated, vol. 1. Addison Wesley

[65] J. Wu and H. El-Ocla, "TCP congestion avoidance model with congestive loss," in Proc. 12th IEEE ICON, Nov. 2004, vol. 1, pp. 3–8

[66] G. Y. Li et al., "Energy-efficiency Wireless Communications: tutorial, Survey and Open Issues," IEEE Wireless Commun., Dec. 2011, pp. 28–35

[67] R. Cavallari, F. Martelli, R. Rosini, C. Buratti, and R. Verdone, "A survey on wireless body area networks: Technologies and designchallenges," IEEE Communications Surveys &Tutorials, vol. 16, no. 3,pp. 1635–1657, Third Quarter, 2014