electronics

Article

Analyzing TCP Performance in High Bit Error Rate Using Simulation and Modeling

Nurul I. Sarkar [1,*], Roman Ammann [1] and Salahuddin Muhammad Salim Zabir [2]

[1] Department of Computer Science and Software Engineering, Auckland University of Technology, Auckland 1010, New Zealand; roman.ammann@xxo.ch
[2] National Institute of Technology, Tsuruoka College, Tsuruoka 997-8511, Japan; szabir@tsuruoka-nct.ac.jp
* Correspondence: nurul.sarkar@aut.ac.nz

Abstract: While Transmission Control Protocol (*TCP*) works well with a low bit error rate (BER), the performance of TCP degrades significantly if the BER rises above a certain level. A study of the performance of TCP with high BER is required for the efficient design and deployment of such systems. In this paper, we address the problem of TCP performance in high BERs and analyze the issues by investigating the effect of BERs on system performance. We consider TCP Reno in our study to explore the system performance using extensive analysis of simulation and modeling. In the analysis, we consider the amount of datagram sent and retransmitted, mean throughput, link-layer overhead, TCP window size, *FTP* download response time, packet dropping and retransmission, and the TCP congestion avoidance mechanism. We validate simulation results by setting up a virtualized testbed using Linux hosts and a Linux router. The results obtained show that TCP throughput degrades significantly and eventually collapses at the packet drop probability of 10% ($BER = 10^{-5}$). The *FTP* download response time is about 32 times longer than that of a perfect channel (no packet dropping). We found that TCP Reno cannot handle such a high BER to operate in wireless environments effectively. Finally, we provide recommendations for network researchers and engineers confronted with the challenge of operating TCP over noisy channels.

Keywords: TCP Reno; performance; simulation; throughput; bit error rate (BER); noisy channel

Citation: Sarkar, N.I.; Ammann, R.; Zabir, S.M.S. Analyzing TCP Performance in High Bit Error Rate Using Simulation and Modeling. *Electronics* **2022**, *11*, 2254. https://doi.org/10.3390/electronics11142254

Academic Editor: Christos J. Bouras

Received: 17 June 2022
Accepted: 13 July 2022
Published: 19 July 2022

1. Introduction and Related Work

The Transmission Control Program was first introduced by Cerf and Kahn [1] as a networking model to share resources among connected nodes in packet-switching networks. TCP has undergone modification, refinement, improvement, and fine-tuning. The original control program incorporated both connection-oriented links and a connection-less approach (i.e., datagram) between the nodes. Soon after its introduction, it was divided into Transmission Control Protocol (TCP) at the transport layer and Internet Protocol (IP) at the network layer [2,3]. The pivotal idea of TCP is to provide a connection-oriented, reliable, ordered, and error-checked transmission between applications running on the sender and the receiver nodes over a packet-switching network [2]. On the other hand, the purpose of IP is to relay the actual data across networks [3]. Its routing capabilities empower internetworking and eventually become the backbone of the Internet.

TCP mitigates the applications and users from the underlying details of communication techniques and works as a facilitator for them to easily use the communication services without any hassle. TCP runs under the assumption that packet loss and unexpectedly prolonged delay are mainly because of congestion in the network, and tries to adapt itself to the changing conditions of the network with congestion control mechanisms [2,4–6]. There are various congestion control algorithms such as slow start, congestion avoidance, fast retransmission, and fast recovery [5,6]. Based on different techniques and ways to handle and avoid the network congestion problem, TCP has many variants. The original

congestion avoidance algorithm is known as TCP Tahoe [5,7], and some other variants are TCP Reno, TCP Vegas [8], FAST TCP [9,10], TCP New Reno [11], among others. TCP Tahoe was named after the 4.3BSD operating system (in which it appeared the first time) that was itself named after Lake Tahoe. TCP Reno is a modification of TCP Tahoe and is named after the 4.3BSD operating system where it appeared the first time, and named after the city of Reno, Nevada, NV, USA.

It should be noted that TCP was originally designed to work over wired networks with a low bit error rate (BER). However, TCP is being used for many wireless communication services such as mobile data services, wireless local area, wide area networks, and wireless communication in data centers with higher BERs. The wireless channels are inherently lossy and are subject to higher BER. There are many comparative studies on TCP variants in various network settings and scenarios [12–22]. Numerous research works on improving the performance of TCP are reported in the network literature (see, for example, [23–45]). This paper addresses the following research question.

What impact do high BERs have on TCP Reno performance for FTP download response time, packet retransmission attempt, and mean throughput?

To answer the research question posed, we carry out an extensive analysis of simulation and modelling.

The main contributions of this paper are summarized as follows.

- We analyze the performance of TCP Reno in high BERs over a wireless network environment using simulation and modelling. The effect of BERs on system performance is also investigated.
- We derive (analytically) the key performance metrics, such as expected throughput TTCP over a 10 Mbit/s Ethernet connection, FTP Download response time, packet drops and retransmission, and TCP congestion avoidance mechanism to estimate the system performance. We then perform an extensive simulation using Riverbed Modeler (formerly OPNET Modeler) to validate analytical models.
- We implement TCP Reno (in C++) and the corresponding process models in the Riverbed Modeler to study and analyze the system performance. We record various statistics for system analysis, such as FTP Download response time, network load, TCP Retransmission count, sent segment ACK number, traffic received, congestion window size, and remote receiver window size. This is a significant piece of work contributing towards the implementation of TCP in wireless and mobile networks.
- We validate simulation results by cross-checking through analysis, as well as using a testbed. To this end, we set up a virtualized testbed using two Linux hosts and a Linux router. The Linux router was configured to achieve the desired performance testing (see Section 4—FTP Download Response Time).

The rest of the paper is organized as follows. The network model and simulation scenarios are laid down in Section 2. Simulation results are presented in Section 3. Section 4 is dedicated to results validation and verification. In Section 5, we present the TCP Reno retransmission algorithm and results. Finally, the paper is concluded with recommendations for practitioners in Section 6.

2. Modelling the Network

A network channel is a physical medium in which a signal can travel. The medium can be guided (e.g., wired medium) or unguided (i.e., wireless). A noise in networks is a disturbance in the transmission medium that affects the characteristics of the signal and its quality. Thus, a noisy channel is a medium of transmission with disturbances. Independent of the source of disturbance, a random disturbance in a transmission channel results in random bit errors. The amount of noise can be expressed as the number of bit errors it introduces. A higher BER means more noises in the channel. We use the packet dropping capabilities of the Riverbed Modeler simulation tool to simulate the consequence of different BERs on system performance.

2.1. Bit Error Rate and Packet Error Rate

A bit error is defined as the total number of bits of a data stream over a communication channel that is affected by the noise. Bit error rate (*BER*) is defined as the number of bit errors divided by the total number of transferred bits [46].

$$BER = \frac{number\ of\ bit\ errors}{number\ of\ transfered\ bits} \tag{1}$$

The packet error rate (*PER*) is defined as the number of packets with errors divided by the number of packets transmitted.

$$PER = \frac{number\ of\ packet\ with\ errors}{number\ of\ packets\ transmitted} \tag{2}$$

The relation between *BER* and *PER* depends on the average packet length *L*. Assuming BER is evenly distributed, *PER* is the probability that there is a bit error in the packet with length *L*.

$$PER = BER \times L \tag{3}$$

Table 1 shows the relationship between *BER* and *PER* for two different packet lengths. Column 1 shows *BER*. The corresponding *PER* for packet lengths of 1250 bytes and 1500 bytes are shown in Columns 2 and 3, respectively.

Table 1. Relationship between *BER* and *PER* for two different packet lengths.

BER	***PER* (*L* = 1250 Bytes)**	***PER* (*L* = 1500 Bytes)**
10^{-1}	1000	1200
10^{-2}	100	120
10^{-3}	10	12
10^{-4}	1	1.2
10^{-5}	10^{-1}	1.2×10^{-1}
10^{-6}	10^{-2}	1.2×10^{-2}
10^{-7}	10^{-3}	1.2×10^{-3}
10^{-8}	10^{-4}	1.2×10^{-4}
10^{-9}	10^{-5}	1.2×10^{-5}

The first packet length (i.e., frame length) was chosen due to the simplicity of the relation it creates between BER and PER. The packet length of 1250 bytes leads to a 10^{-4} factor between BER and PER, and the average Ethernet packet length is likely a bit smaller than the maximum packet length. The second packet length (1500 bytes) was selected because it is the average size of an Ethernet frame. The average frame length heavily depends on the network environment (corporate or internet service provider) and the traffic direction (up- or down-stream) [6]. The Riverbed simulation in this study consists of one large file transfer from a server node to a client node. Because our simulation transfers the one file over the link from the server to the client with a maximum transmission unit (MTU) of length 1500 bytes, an average frame size of 1500 bytes is a realistic assumption.

We observe that a high BER (10^{-4}) leads to a PER of 1 or more. This means that every packet is dropped out and it is impossible to transfer data over the link. With an evenly distributed BER, only frames smaller than 1250 bytes have a chance to get sporadically through.

2.2. Packet Error Rate and Packet Drop Probability

Random bit errors in the link-layer frame will lead to errors in the Ethernet frame check sequence (FCS, also known as cyclic redundancy check) or in the IP header checksum or the TCP checksum. If a destination node receives an Ethernet frame with a wrong checksum, the destination node drops the frame immediately. In rare cases, the destination node does not detect the error in the Ethernet frame or the IP header, or in the TCP header in the very improbable case where several bit errors lead to the same cyclic redundancy check (CRC) code. The first router receiving the faulty packet will discard the packet after verifying its checksum [46]. The router will not inform the sender about discarding the packet. If the bit error occurs in the TCP segment, the receiver will detect it while verifying the TCP checksum. If the checksum does not match, the receiver will discard the segment and will not acknowledge receiving it [47]. Because a bit error in a packet will always lead to a silent packet drop (the sender is not informed about the packet drop), a noisy channel can be simulated by a network that drops packets with a certain probability p_{drop}. In the simulation, *PER* is simulated by the packet dropping probability p_{drop}, i.e.,

$$p_{drop} = PER \tag{4}$$

2.3. Simulation Model

The simulation was carried out using Riverbed Modeler version 16.0.A. The model is simplified to minimize the effect of other parameters on system performance. A File Transfer Protocol (FTP) server and an FTP client were set up to simulate network traffic. Figure 1 shows the simulated network model.

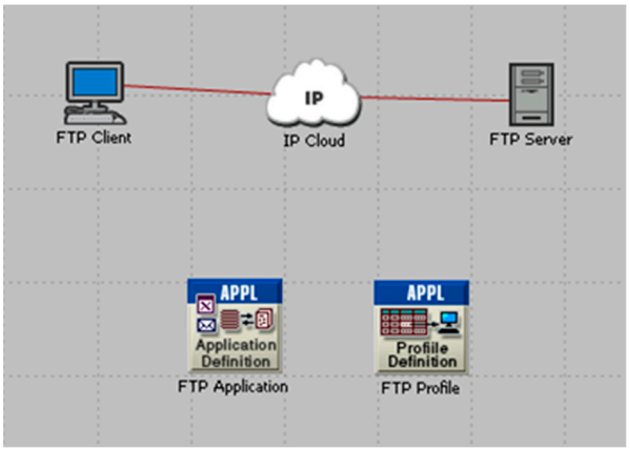

Figure 1. The Network Model.

FTP Client: The FTP client uses the Riverbed simulation module "ethernet_wkstn_adv" for client functionality. It connects to the IP cloud using an Ethernet cable. The FTP client uses the default TCP parameters, as shown in Table 2. The client supports the FTP Profile (bottom right box in Figure 1).

FTP Server: The FTP server uses the Riverbed module "ethernet_server_adv" for setting up server functionality. The server is linked to the IP cloud using an Ethernet cable. The FTP server also uses the simulator default TCP parameters (Table 2), and it supports the FTP application, as shown in Figure 1 (bottom left icon).

IP Cloud: The IP cloud uses the simulator module "ether-net4_slip8_cloud_adv" to set up its functionality. It is used to simulate an IP network. The Packet Latency was set to 0.02 s and the Packet Discard Ratio was set to different values depending on the scenarios studied.

FTP Application: The application simulates FTP traffic. The FTP parameters are set to:

- Command Mix (Get/Total): 100%
- Inter-Request Time (seconds): constant (3600)

- File Size (bytes): constant (10,000,000)
- Symbolic Server Name: FTP Server
- Type of Service: Best Effort (0)
- RSVP Parameters: None
- Back-End Custom Application: Not Used

FTP Profile: The FTP profile uses the FTP application and runs it once.

Table 2. Riverbed Default TCP Parameters.

Attribute	Value	Attribute	Value
Version/Flavor	Unspecified	Active Connection Threshold	Unlimited
Maximum Segment Size (bytes)	Auto-assigned	Nagle Algorithm	Disabled
Receiver Buffer (bytes)	8760	Karn's Algorithm	Enabled
Receive Buffer Adjustment	None	Timestamp	Disabled
Receive Buffer Threshold	0.0	Initial Sequence Number	Auto Compute
Delayed ACK Mechanism	Segment/Clock-based	Retransmission Thresholds	Attempts-based
Maximum ACK Delay (s)	0.200	Initial RTO (s)	3.0
Maximum ACK Segments	2	Minimum RTO (s)	1.0
Slow-Start Initial Count (MSS)	2	Maximum RTO (s)	64
Fast Retransmit	Enabled	Round Trip Time (RTT) Gain	0.125
Duplicate ACK Threshold	3	Deviation Gain	0.25
Fast Recovery	Reno	RTT Deviation Coefficient	4.0
Window Scaling	Disabled	Timer Granularity (s)	0.5
Selective ACK (SACK)	Disabled	Persistence Timeout (s)	1.0
ECN Capability	Disabled	Connection Information	Do Not Print
Segment Send Threshold	MSS Boundary	Acceleration	Disabled

2.4. Simulation Scenarios

We simulated five selected scenarios (Table 3) based on the packet drop probability (PDP) of 0, 0.01, 0.1, 1, and 10%. Column 1 lists the five scenarios of PDP. The corresponding PDP, BER, and the channel conditions are shown in Columns 2, 3, and 4, respectively.

Table 3. Packet drops probability and the corresponding BER and channel conditions defined.

Scenario	p_{drop}	*BER*	Channel Condition
PDP: 0%	0%	0	Perfect channel
PDP: 0.01%	0.01%	$\approx 10^{-8}$	Low noise channel
PDP: 0.1%	0.10%	$\approx 10^{-7}$	Medium noise channel
PDP: 1%	1%	$\approx 10^{-6}$	High noise channel
PDP: 10%	10%	$\approx 10^{-5}$	Very high noise channel

2.5. Simulation Parameters and Statistics

The simulation runtime was set to 45 min (2700 s) and 2700 sets of values (various statistics) were collected during the simulation. We have recorded the following seven statistics.

- Global Statistics—FTP Download Response Time (s)
- Server Node Statistics—TCP Connection—Retransmission Count (Packets)
- Server Node Statistics—Traffic Load (packets)

- Server Node Statistics—TCP Connection—Congestion Window Size (bytes)
- Client Node Statistics—TCP Connection—Sent Segment ACK Number
- Client Node Statistics—TCP Connection—Throughput (bytes/s)
- Client Node Statistics—TCP Connection—Receiver Window Size (bytes)

3. Results

In this section, we present our findings obtained from system analysis, as well as from simulation runs.

3.1. FTP Response Time

Table 4 shows the time needed by the FTP client to download a 10 MB file from the FTP server for the five selected scenarios. Column 1 lists the five selected scenarios. The corresponding response times are shown in column 2. The rightmost column compares the result of each scenario with that of p_{drop} = 0%. We observe that the download response time increases with increasing p_{drop}, as expected.

Table 4. FTP Download Response Time for Five Selected Scenarios.

Scenario	Response Time (s)	Response Time (s) Difference from p_{drop} = 0%
p_{drop} = 0%	60.21	0
p_{drop} = 0.01%	60.42	0.35
p_{drop} = 0.10%	61.46	2.08
p_{drop} = 1%	76.83	27.60
p_{drop} = 10%	1953.61	3144.66

3.2. FTP Server: Retransmission Count

Table 5 lists the five scenarios of packet drop probabilities and the corresponding retransmission count. We observe that the retransmission increases with increasing packet drop probabilities. For instance, 1092 TCP segments were retransmitted (highest number) by the FTP server at $p_{drop} = 10\%$.

Table 5. FTP Server: Retransmission Count for Five Selected Scenarios.

Scenario	Retransmission Count
p_{drop} = 0%	0
p_{drop} = 0.01%	1
p_{drop} = 0.10%	7
p_{drop} = 1%	62
p_{drop} = 10%	1092

3.3. FTP Server: Ethernet Frames Sent

Table 6 lists the scenarios (probabilities of packet dropping) and the corresponding total number of Ethernet frames transmitted by the FTP server. We observe that more Ethernet frames were sent because of increasing packet drop probabilities. For instance, 7946 Ethernet frames were retransmitted (highest number) by the FTP server at the packet dropping rate of 10%.

Table 6. FTP Server: Ethernet Frame Sent for Five Selected Scenarios.

Scenario	Total Ethernet Frame Sent
p_{drop} = 0%	6854
p_{drop} = 0.01%	6855
p_{drop} = 0.10%	6861
p_{drop} = 1%	6916
p_{drop} = 10%	7946

3.4. Effect of TCP Congestion Window

Figure 2 illustrates the effect of TCP congestion window size (bytes) on the packet dropping rate of the FTP server over 2700 s of simulation time. We observe that more TCP segments are dropped out by the Router when the TCP congestion window size is smaller. In contrast, no TCP segments are dropped by the Router when the size of the TCP congestion window is as big as 135,000 bytes.

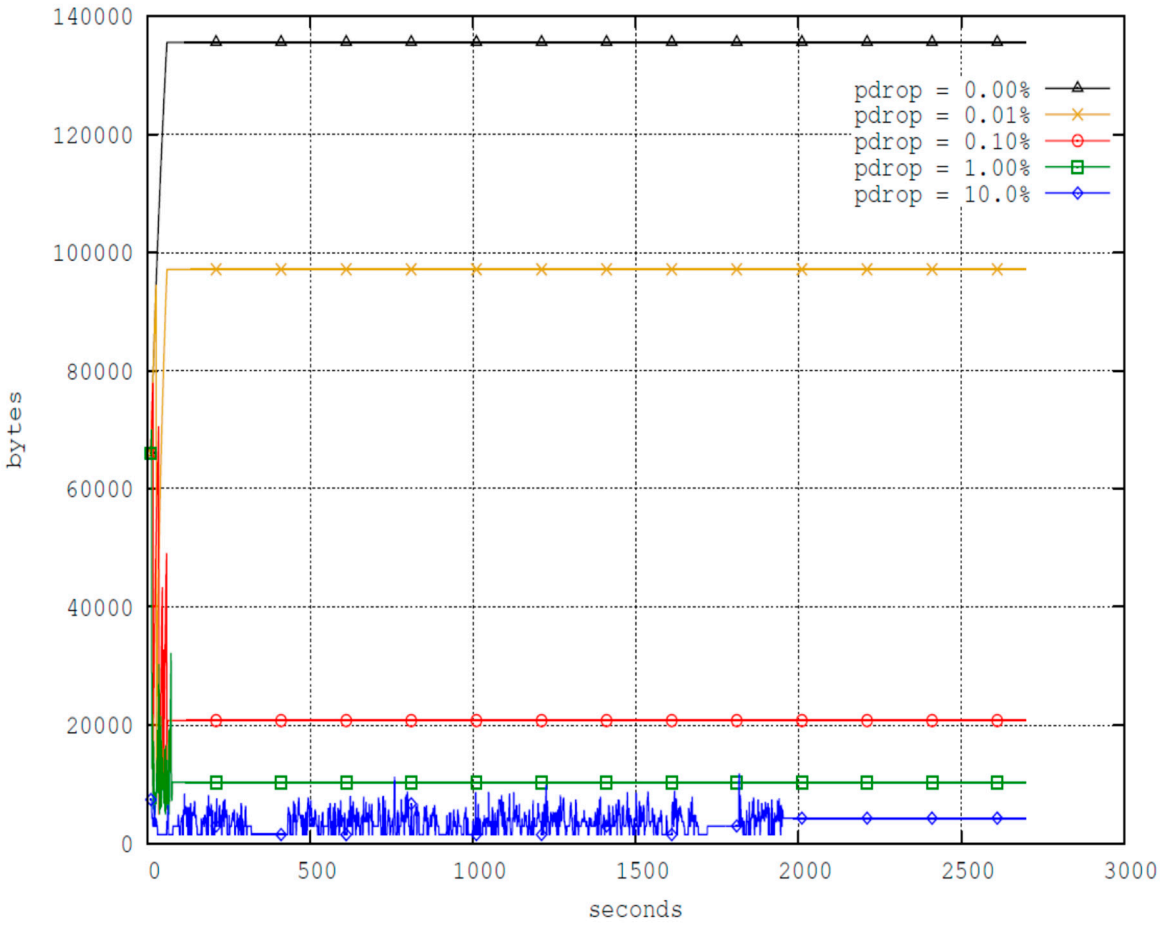

Figure 2. Effect of TCP Congestion Window Size on Packet Dropping.

Figure 3 illustrates the effect of TCP congestion window size on packet dropping over the first 80 s of simulation time. We observe that a bigger congestion window allows the FTP server to send more TCP segments before waiting for an ACK from the client.

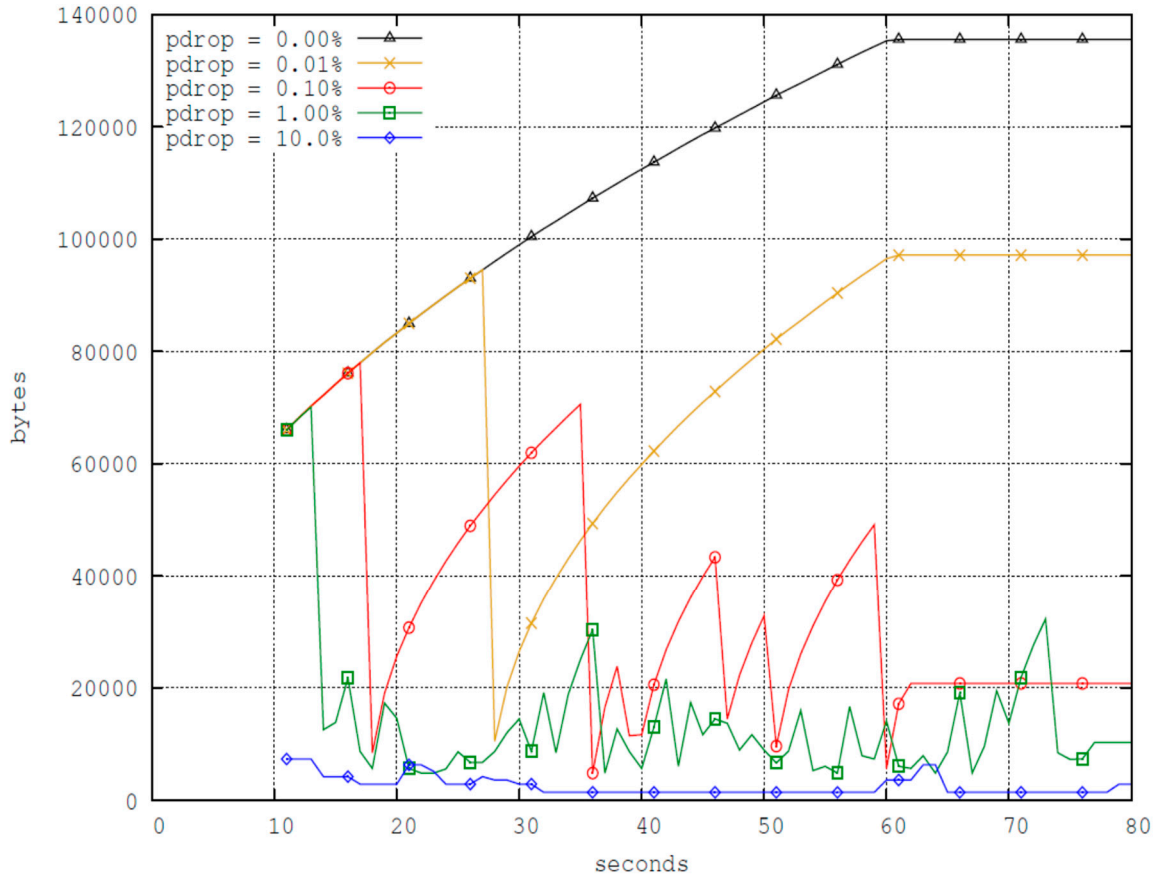

Figure 3. Effect of TCP Congestion Window Size on Packet Dropping (first 80 s).

3.5. FTP Client: Sent Segment ACK Number

Figures 4 and 5 illustrate the TCP Sent Segment ACK Number for an FTP client. Figure 4 shows the behavior of TCP Sent Segment ACK Number over 2700 sec simulation time. The graphs for $p_{drop} = 0\%$, $p_{drop} = 0.01\%$, $p_{drop} = 0.1\%$, and $p_{drop} = 1\%$ are almost congruent, and only the graph for $p_{drop} = 10\%$ rises at a slower rate and is clearly distinguishable from other graphs. In Figure 5, we have redrawn Figure 4 for the first 80 s of simulation time. A faster rising graph indicates that the FTP client acknowledged TCP segments at a faster rate.

3.6. FTP Client: Average Throughput

Figure 6 shows the average throughput for the five selected packet drop probabilities ($p_{drop} = 0\%$, $p_{drop} = 0.01\%$, $p_{drop} = 0.1\%$, and $p_{drop} = 1\%$, and $p_{drop} = 10\%$) that we considered in this study. The transferred file (10 MB) is divided by *FTP* download response time to get the average throughput. We observe the effect of packet drop probability on average throughputs. For instance, the lowest throughput is achieved at a packet drop probability of 10%. The main conclusion is that TCP does not perform well (in terms of throughputs) at high *BER*s.

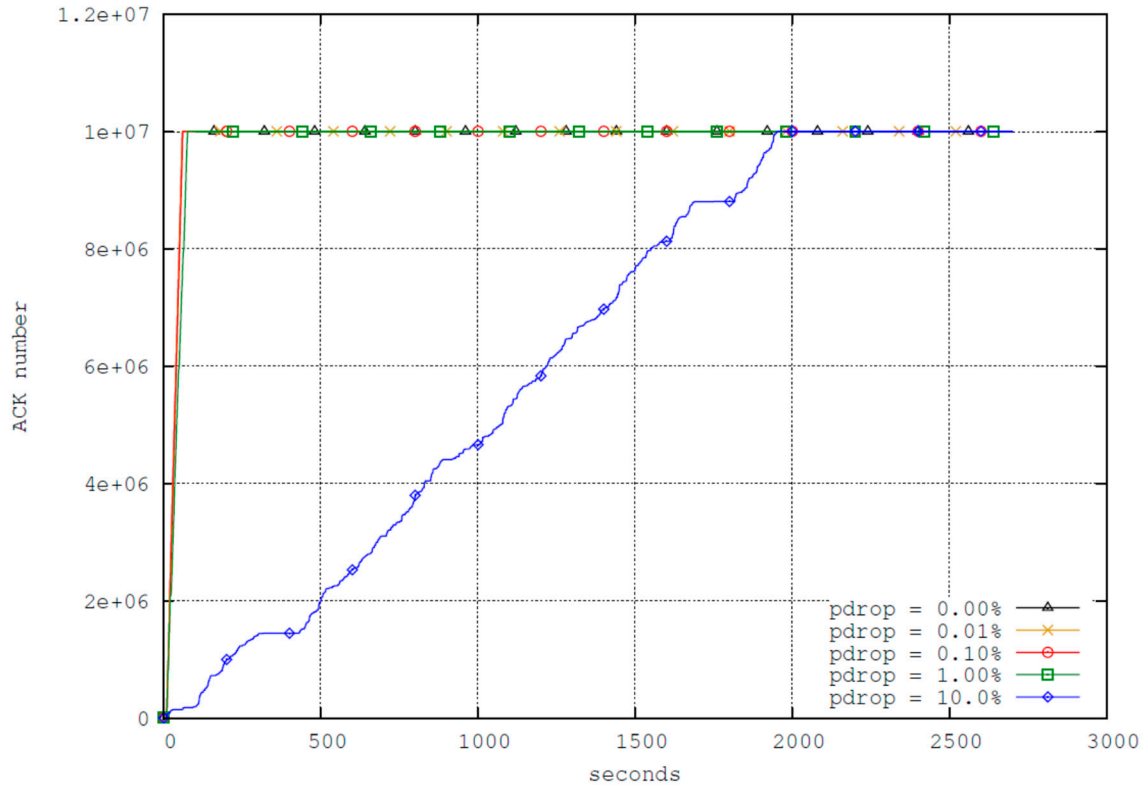

Figure 4. *TCP* Sent Segment ACK Number (*FTP* Client) over 2700 s of simulation time.

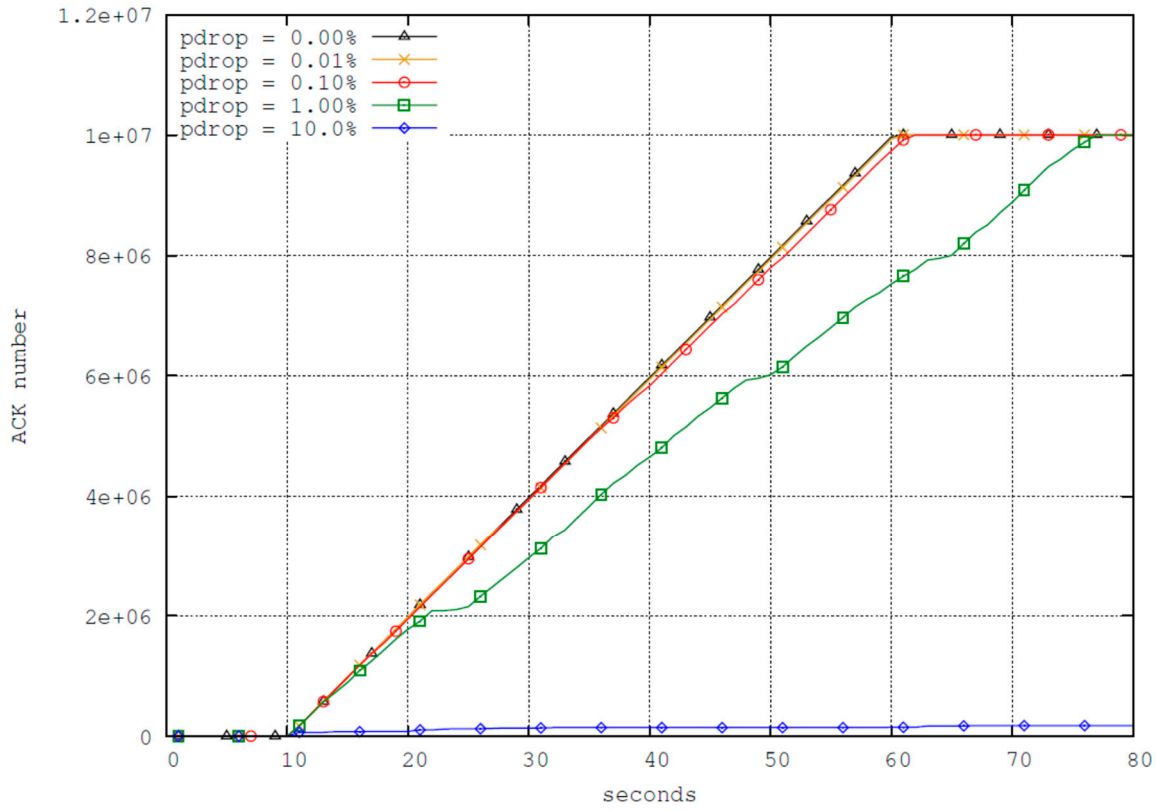

Figure 5. Figure 4 is redrawn for the first 80 s of simulation time.

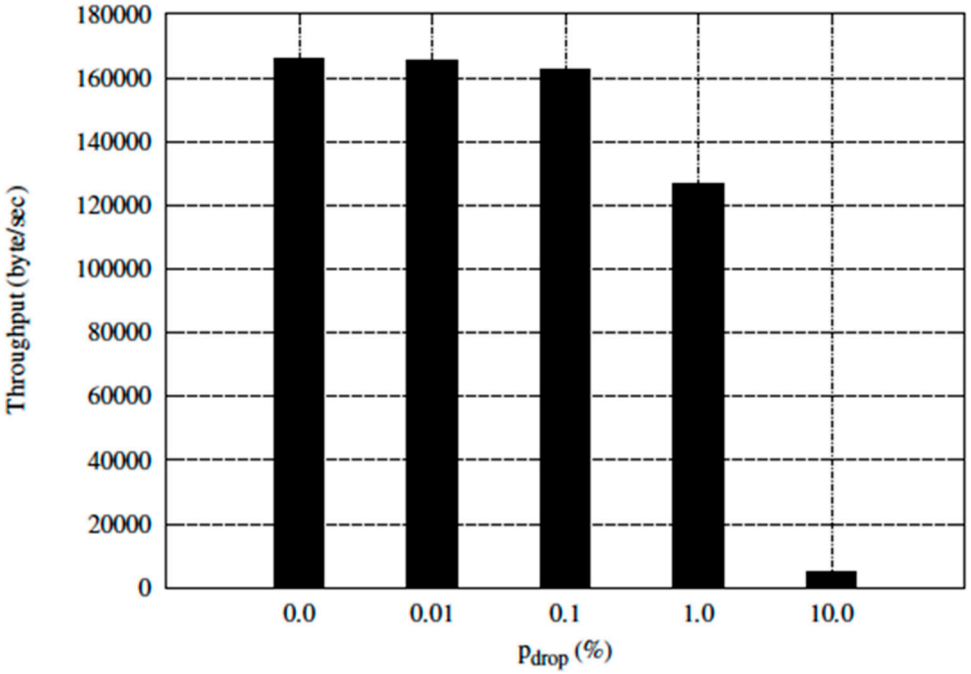

Figure 6. Average FTP Throughput for various probabilities of packet dropping.

3.7. FTP Client—Traffic Received

Figure 7 shows the traffic receiving rate characteristics of *FTP* clients over 2700 s (simulation time). We observe the effect of five selected packet drop probabilities ($p_{drop} = 0\%$, $p_{drop} = 0.01\%$, $p_{drop} = 0.1\%$, and $p_{drop} = 1\%$, and $p_{drop} = 10\%$) on system performance. It is interesting to note that FTP traffics are dropped out significantly at high packet drop probability ($p_{drop} \geq 1\%$). For instance, there were no traffics received at all (100% dropped out) at a packet drop probability of 10%. This low TCP performance is clear especially for a simulation time over 2000 s.

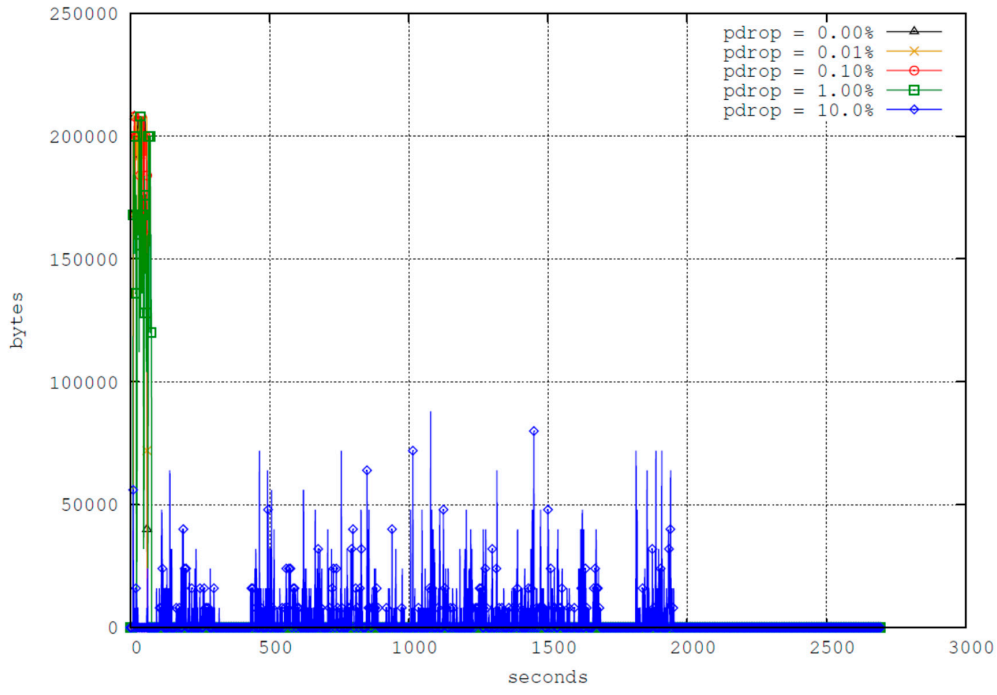

Figure 7. Traffic receiving rate of FTP Clients over 2700 s simulation.

In Figure 8, we have redrawn Figure 7 for the first 80 s of simulation time. We observe that the FTP traffics started to drop out at $p_{drop} \geq 1\%$. A faster fall of traffic delivery indicates that the FTP client traffic receiving rate is diminishing over time. The main conclusion is that TCP does not perform well (in terms of traffic delivery) at high BERs.

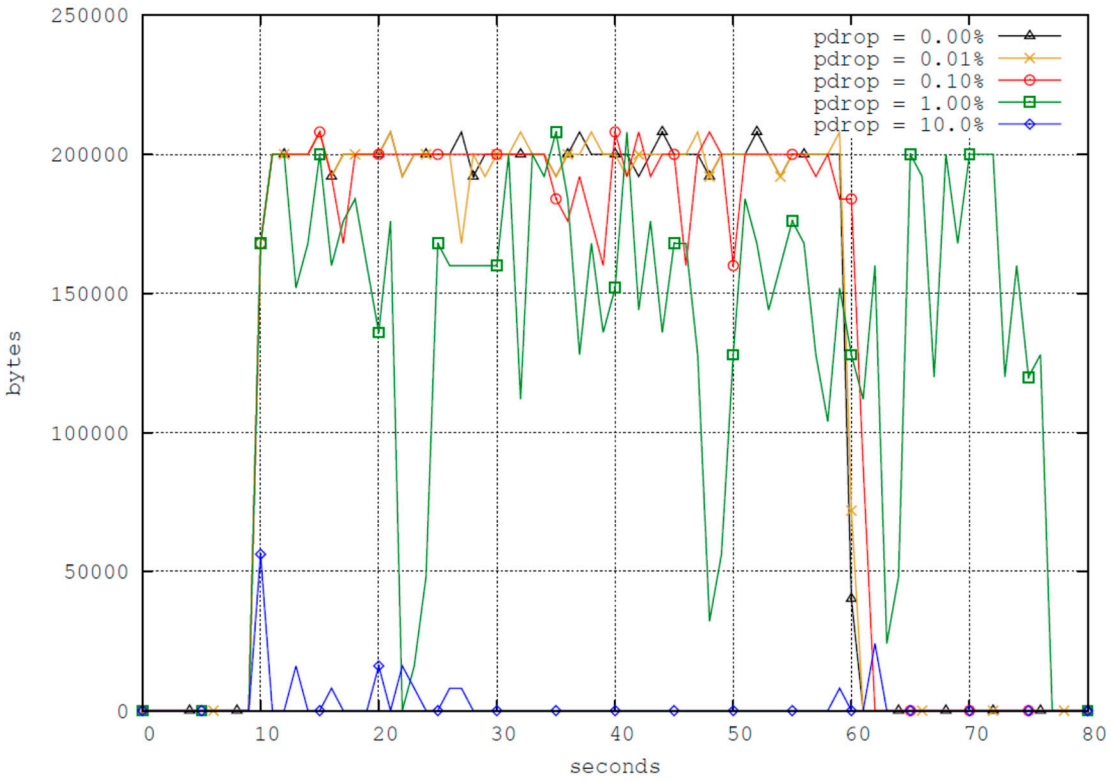

Figure 8. Figure 7 is redrawn for the first 80 s of simulation time.

4. Analysis and Validation of Results

Several aspects of the simulation are verified either by calculating the result based on the protocol behavior or by rerunning the tests in a virtualized network environment.

4.1. Amount of Datagram Sent

This section verifies that the number of datagrams necessary to transport the file corresponds with the number of datagrams sent by OPNET. The size of the transferred file in the OPNET simulation is 10,000,000 bytes. The standard Ethernet maximum transmission unit (MTU) is 1500 bytes. When the IP header without any options (20 bytes) and the TCP header without any options are removed (20 bytes), the TCP maximum segment size (*MSS*) is 1460 bytes. The number of datagrams n required to transport the file is given by

$$n = \frac{s}{MSS} = \frac{10,000,000}{1460} = 6849.32 \quad (5)$$

Thus, about 6850 datagrams are required to transfer the file from *FTP* server to the *FTP* client. The Riverbed Modeler simulator recorded 6854 frames sent by the *FTP* server for the scenario $p_{drop} = 0\%$ (Table 6).

The difference between the four packets can be explained as follows. The first packet is related to the TCP handshake. It is the answer (TCP SYN/ACK) to the TCP connection request of the FTP client. The next two frames are related to the TCP teardown. One is the TCP FIN and the other one is the TCP ACK. Both are sent from the server to the client. The last frame is related to the link-layer address resolution. The server needs to determine the link-layer address of the next-hop and sends, therefore, an Address Resolution Protocol

(ARP) request. This adds up to the four frames, the difference between the calculated and the recorded value. Thus, the number of datagrams sent during the simulation are the same as the calculated expected number of datagrams necessary to transfer the file of size s = 10,000,000 bytes with MSS = 1460 bytes.

To conclude, the simulation matches the amount of datagram (packets) expected.

4.2. Amount of Packet Retransmitted

It is expected that a link with a packet drop probability p_{drop} will retransmit at least $n \times p_{drop}$ packets. This assumes that only the dropped packets need to be retransmitted. Table 7 shows the calculated number of packets drops and the actual retransmissions sent by the *FTP* server from Table 5.

Table 7. Packet Retransmission (Calculated and Simulated).

Scenario	Packet Retransmission		Difference between Calculated and Simulated (%)
	Calculated	Simulated	
p_{drop} = 0.00%	0	0	No difference
p_{drop} = 0.01%	0.68	1	47.06%
p_{drop} = 0.10%	6.85	7	2.19%
p_{drop} = 1.00%	68.49	62	−9.48%
p_{drop} = 10.00%	684.93	1092	59.43%

The packet retransmissions should be at least as high as the number of calculated packet drops. A packet dropped because of a bit error needs to be retransmitted to finally transmit the whole file. Thus, at least the dropped packets need to be retransmitted. A higher amount of retransmission might be because of protocol mechanisms.

While the Table 7 shows that the retransmission count is within the range of the expected packet drops, it also shows that the observed retransmission count is higher, and in one case lower, than the calculated value. We expect that the reason for the difference is in the implementation of the dropping algorithm in Riverbed Modeler and how it tries to achieve a certain drop probability. This assumption is based on the observation that a modification of the *FTP* download offset (when the download should start relative to the simulation start) also changed the amount of retransmission the *FTP* server sends.

4.3. Theoretical Maximum Throughput

This section compares the theoretical maximal throughput obtained from TCP protocol window size with the achieved throughput in the simulation.

4.3.1. Simulation

For the duration of the simulation run, Riverbed Modeler recorded the bytes received by the FTP client. For the scenario p_{drop} = 0.00%, the first value was recorded at $t_0 = 10$ s and the last value at $t_1 = 60$ s. The throughput T_{TCP} is the transferred file size s divided by the time that was needed to transfer the file:

$$T_{TCP} = \frac{s}{t_1 - t_0} \tag{6}$$

$$T_{TCP} = \frac{10,000,000}{60 - 10} = 200,000 \text{ bytes} \tag{7}$$

This calculation contains a small error as the simulation just records the values every second. Therefore, t_0 is not exactly 10 s, but between 9 s and 10 s, and t_1 is between 59 s and 60 s. For the order of the throughput, however, it does not have a significant impact.

4.3.2. Medium (Ethernet) Limitation

Each TCP segment sent over an Ethernet connection will have some overhead because of the lower layer protocols involved in the communication [46]. Different protocol layers include additional headers (see below).

TCP header (20 byte): TCP header without options
IP header (20 byte): IP header without options
Ethernet overhead (38 byte), preamble (8 byte), CRC (18 byte), inter-packet gab (12 byte)

Thus, a TCP segment has an overhead of 78 bytes. The expected throughput T_{TCP} over a 10 Mbit/s Ethernet connection will be:

$$T_{TCP} = \frac{MSS}{MSS + overhead} \times T_{Ethernet} \tag{8}$$

$$T_{TCP} = \frac{1460}{1460 + 78} \times 1,250,000 = 1,186,605.98 \text{ bytes/s} \tag{9}$$

This is significantly higher than the throughput achieved in the simulation because most TCP feathers, such as timeout, acknowledgement mechanism and retransmission, are ignored and only the protocol overhead is considered.

4.3.3. TCP Window Size

TCP throughput T_TCP depends on the round-trip time (*RTT*) and the TCP receiver window (*rwnd*). It is not possible to go any faster than the window size offered by the receiver, divided by the RTT [46].

$$T_{TCP} \leq \frac{rwnd}{RTT} \tag{10}$$

Thus, with an *RTT* of 0.04 s (as used in the simulation) and *rwnd* of six MSS (8760, see Table 2, Receive Buffer).

$$T_{TCP} \leq \frac{8760}{0.04} = 219,000 \text{ bytes/s} \tag{11}$$

This correlates well with the throughput of 200,000 bytes/s (see Equation (7)), as recorded by the Riverbed Modeler. *RTT* in the simulation is expected to be slightly higher because of the processing delays on the end systems.

4.4. FTP Download Response Time

The download response time, which is the time to download the file over FTP client, is verified in two ways. First, the duration is calculated based on the theoretical throughput (see Equation (11)). The expected response time for the scenario $p_{drop} = 0\%$ is the size of the transferred file s divided by the theoretical throughput T_{TCP}.

$$t_{download} = \frac{s}{T_{TCP}} \tag{12}$$

$$t_{download} = \frac{10,000,000}{219,000} = 45.66 \text{ s} \tag{13}$$

This does not correlate very well with the 60.21 s obtained with OPNET. Furthermore, it also does not correlate well with the graphs in Figures 3, 5 and 8. They all suggest that traffic gets sent from around $t = 10$ to $t = 60$, which would mean roughly 50 s for the scenario $p_{drop} = 0\%$.

Second, a small, virtualized network was setup to verify the download response time with an experiment. The setup consists of two Linux hosts and a Linux router. The Linux router runs netem (part of the Linux kernel for network emulation) to introduce a 20 milli-seconds delay in each direction and a certain drop probability p_{drop}. The Linux router was configured with the commands as shown in Figure 9.

```
#  enable routing
echo 1 > /proc/sys/net/ipv4/ip_forward

#add delay in both directions
tc qdisc add dev eth1 root netem delay 20ms
tc qdisc add dev eth2 root netem delay 20ms

#add certain loss (e.g. 1%)
tc qdisc replace dev eth1 root netem delay 20ms loss 1%
```

Figure 9. Linux router configuration commands.

Table 8 shows the measured TCP download response time for various scenarios. A reevaluation of the simulation model revealed a 5 s start time in the FTP profile and another 5 s start time offset in the FTP application settings. Unfortunately, removing these two offsets did not change the results of the FTP download response time but merely moved the previously mentioned Figure 10 to the left. The authors believe that the error originates from a setting within the Riverbed Modeler simulator but were not able to determine the setting responsible for this offset.

Table 8. Download Response Time in Virtualized Network Environment.

Scenario	Response Time (s)	Difference from p_{drop} = 0%
p_{drop} = 0.00%	50.25	0%
p_{drop} = 0.01%	50.25	0%
p_{drop} = 0.10%	51.97	3.42%
p_{drop} = 1.00%	75.84	50.65%
p_{drop} = 10.00%	259.665	416.75

Assuming the response times in Table 4 are 10 s shorter, the virtualized setup shows the same behavior as the Riverbed Modeler. A low p_{drop} does not influence the download time significantly. A drop probability of 1% increases the download time significantly, and a drop probability of 10% leads to an immensely longer download time.

The authors believe that the difference in the scenarios with $p_{drop} = 0\%$ and $p_{drop} = 10\%$ comes from the difference in the number of packets dropped in the simulation and the virtualized network environments. In the scenario with $p_{drop} = 1\%$, the simulation drops fewer packets than expected based on the calculations. In addition, the simulation is about 9 s faster with the file download than in the virtualized environment. In the scenario with $p_{drop} = 10\%$, the file download is significantly slower, but the number of dropped packets is also drastically higher than the expected number of dropped packets based on the p_{drop} probability.

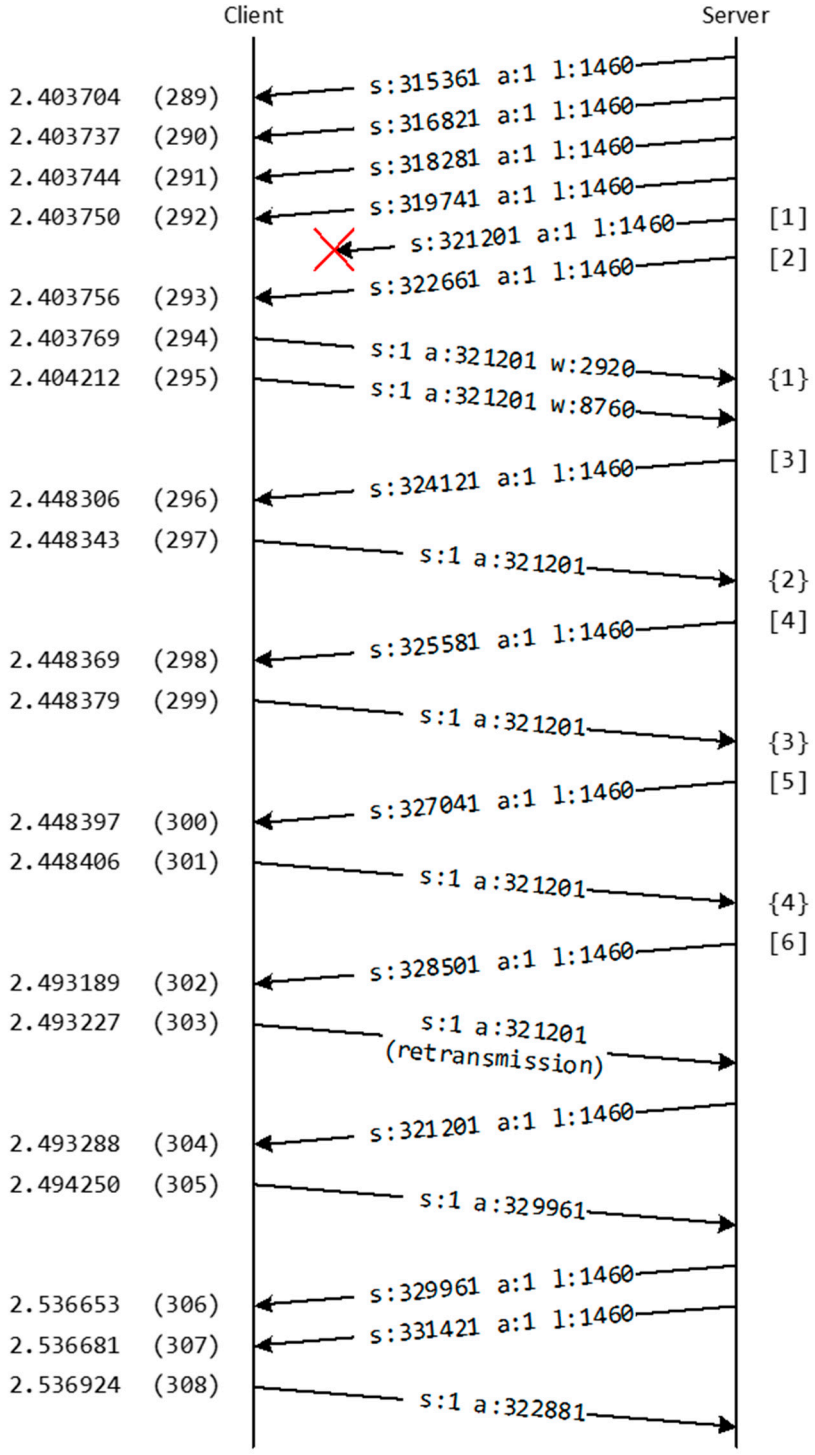

Figure 10. Packet Exchange for Packet Loss and Retransmission.

5. Discussion and Interpretation of Results

Anyone unfamiliar with TCP could assume that in the case of a packet drop, TCP would just need an additional *RTT* to retransmit the packet. The average packet rate for $p_{drop} = 0.00\%$ is:

$$\frac{6854 \text{ pkts}}{60.24 \text{ s}} = 113.778 \text{ pkts/s} \tag{14}$$

Thus, if 1% of the packets are dropped, that would mean that about 68.54 packets need to be retransmitted.

With the above-average packet rate, that would add:

$$\frac{68.54 \text{ pkts}}{113.778 \text{ pkts/s}} = 0.602 \text{ s} \tag{15}$$

Adding 0.602 s to the transfer time of $p_{drop} = 0\%$ would result in 60.82 s. This is still far less than the simulated transfer time of 66.83 s ($p_{drop} = 1\%$, 10 s subtracted from simulation result, see Table 4). This indicates that other factors need to be considered to explain the observed behavior.

5.1. Packet Drop and Retransmission

Figure 10 shows the packet exchange for the scenario $p_{drop} = 1\%$ around the first packet loss until the packet is retransmitted. The graph is based on traffic captured in the *virtualized network environment*, used to verify the results, and not on the Riverbed simulation. It was captured on the client-side. Therefore, the packet order on the server side does not necessarily represent the correct sequence.

In Figure 10, the numbers on the left-hand side are timestamps in seconds. Numbers on the left-hand side in round brackets are frame numbers as they were recorded. Numbers on the right-hand side in square brackets count the segments sent without an ACK received, and in curly brackets are the numbers of ACKs received for the same sequence number.

Before the first segment was lost, the server reached the receiver's advertised window (*rwnd*) of 8760 bytes (six segments' of 1460 bytes) with a slow start. The server was sending six segments when the fifth segment (between 292 and 293) was dropped. The client immediately replied with an ACK (294) as defined in RFC 5681 [6]. Because the FTP client did not have a chance to read the data from the TCP receiving buffer (289–292), the client set the receiving window to 2920 bytes. Shortly after, the FTP clients read the data and the client sent a TCP window adjustment (295) to open the window again.

After receiving the ACK (294), the server sent another four segments to the client and filled the window with six segments 1–6. The first three segments 3 to 5 were sent almost at the same time. The figure might be a bit misleading here. For easier reading, the figure has been stretched to avoid overlapping arrows. Many packets were sent almost simultaneously. The packets sent from the server and the client crossed each other somewhere between the two systems.

Because the client was missing the segment with sequence number 321,201, the client kept sending ACKs for sequence numbers 1–4. When the server received the fourth ACK (or the third duplicated ACK), it went into fast retransmission [6] and retransmitted the segment (303). Again, the figure might be misleading. The third duplicated ACK was received by the server around 2.468 (sent time plus half of RTT), and the packet was retransmitted around 2.473 (arrival time minus half RTT).

When the next (new) ACK (305) was received, the congestion window (*cwnd*) was set to slow start threshold (ssthresh). In this case, it is set to three *MSS* or 4380 bytes. The server then sent two additional segments (306, 307) and the client acknowledged them (308).

Between the first segment (280) arriving at the client and the last segment (307) arriving at the client, 0.132977 s passed, which is roughly three RTTs (0.042 s). During this period, 12 segments were sent to the client. Without a packet drop, 18 packets would have been sent to the client (three times the full window size) at the same time. In other words, each packet drop led to a delay of one RTT. This is exactly what anyone unfamiliar with TCP would expect. Unfortunately, it does not account for all the additional delays observed in the scenario $p_{drop} = 1\%$.

5.2. Congestion Avoidance

The previous section explained that the file transfer needed a longer time because of the retransmission of the dropped packets. It also showed that the retransmission only accounts for a small amount of the additional time the file transfer needed in the scenario $p_{drop} = 1\%$ compared to the scenario $p_{drop} = 0\%$. This section focuses on the TCP

congestion avoidance mechanism and shows how the mechanism is responsible for the rest of the additional time needed in the scenario $p_{drop} = 1\%$.

With segment (309), the server enters the congestion avoidance phase. RFC 5681 [6] defines two possibilities of fast expansion of *cwnd* for a TCP host. The first option is to expand *cwnd* by byte-acknowledged *N* or by one *MSS*.

$$cwnd+ = min(N,\ MSS) \tag{16}$$

The second expansion option is:

$$cwnd+ = \frac{MSS \times MSS}{cwnd} \tag{17}$$

Both formulas can be used by a TCP implementation. Table 9 shows the theoretical increase with a start value of three *MSS* based on (17). The second column shows the results in bytes, and the third column shows results in the number of *MSS*.

Table 9. *Cwnd* increased per RTT.

RTT	*Cwnd* (Bytes)	*Cwnd* (No. of *MSS*)
0	4380	3
1	4867	3
2	5305	3
3	5707	3
4	6080	4
5	6431	4
6	7762	4
7	7077	4
8	7379	5
9	7667	5
10	7945	5
11	8214	5
12	8473	5
13	8725	5
14	8969	6

After 14 RTTs, the *cwnd* should again reach *rwnd* (8760 bytes or six MSS). The actual traffic capture shows a slightly different image. During 15 RTTs, the server keeps sending three segments, and the client replies with an ACK. Following that, the server sends nine times alternately four and five segments, which are acknowledged by the client. Then the server starts again by sending the full *rwnd* of six MSSs.

Overall, the TCP connection in the *virtualized environment* needs 27 RTTs to recover from one packet drop. In these 27 RTTs, only 98 segments are transmitted. With a *cwnd* equal to the *rwnd* (which would be the case without packet drops), the server would have sent 162 segments at the same time. Consequently, the server must transmit the other 64 segments later. The server transmits six segments per RTT. Accordingly, the server will need 10.67 RTTs or, based on an RTT of 0.04 s, 0.4268 s longer because of the one packet drop.

In the scenario $p_{drop} = 1\%$, 68 segments should have been retransmitted. This adds up to 29.02 s, which is the additional time overhead required by the server to transfer the file to the client because of the packet drops. This is in the same range of time as the FTP download in the scenario $p_{drop} = 1\%$ (see Table 4) in the Riverbed simulation.

Figure 3 shows that the server is not able to scale-up the congestion window because of the dropped packets. This indicates that the issue is not only related to the number of dropped packets, but it is also an important issue the way TCP handles packet drops and scales down the window.

6. Conclusions and Recommendations

In this paper, we analysed TCP performance in a high bit error rate (BER) using simulation and modelling. Results obtained have shown that TCP throughput collapses for $p_{drop} = 10\%$. The problem is that TCP was not designed to handle such high BER wireless environments. The assumption of the TCP algorithm (fast recovery) is that packet loss caused by damage is very small (much less than 1%) [47]. Fast recovery algorithm is not good in recovering multiple dropped packets from the same packet streams [6]. This is because the receiver must wait for the TCP timeout of the sender. The waiting for the timeout brings much higher delays than the fast retransmission algorithm. Therefore, it is no surprise that both the simulation and the emulation show a much longer download time at $p_{drop} = 10\%$ than any other scenarios. For instance, FTP download response time is about 32 times longer than that of a perfect channel (no packet dropping). The findings reported in this paper provide some insights into TCP performance in high BER wireless and mobile network environments, which can help network researchers and engineers contribute further towards the development of smart transport layer protocols for the next-generation Internet. However, analyzing TCP performance over the Internet of Things for life-saving emergency applications is suggested as future work.

In this paper, we provide three specific recommendations. Specifically, network researchers and engineers should: (1) avoid the combination of a very high noisy channel (BER > 10^{-6}) and TCP; (2) employ either an optimized TCP or other transport layer protocols for wireless connections; and (3) design a new transport protocol, e.g., Datagram Congestion Control Protocol (DCCP, RFC 4340), or leave the task of a reliable end-to-end connection to the application layer by choosing the User Datagram Protocol. Adopting these recommendations will help network researchers and engineers to produce better TCP performance in practice, especially in wireless environments.

Author Contributions: Conceptualization, R.A. and N.I.S.; methodology, R.A. and N.I.S.; software, N.I.S.; validation, R.A. and N.I.S.; formal analysis, R.A. and N.I.S.; investigation, R.A. and N.I.S.; resources, N.I.S.; data curation, N.I.S. and R.A.; writing—original draft preparation, R.A.; writing—review and editing, S.M.S.Z.; visualization, R.A.; supervision, N.I.S.; project administration, N.I.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest. We also confirm that our paper has not been previously published and is not currently under consideration by any other journals. All authors involved have approved the contents of this paper and have agreed to the Journal of MDPI Electronics submission policies.

References

1. Cerf, V.; Kahn, R. A Protocol for Packet Network Intercommunication. *IEEE Trans. Commun.* **1974**, *22*, 637–648. [CrossRef]
2. Postel, J. Transmission Control Protocol. 1981. Available online: https://datatracker.ietf.org/doc/rfc793/ (accessed on 1 July 2022).
3. Postel, J. Internet Protocol—DARPA Internet Program Protocol Specification. 1981. Available online: https://tools.ietf.org/html/rfc791 (accessed on 1 July 2022).
4. Karn, P.; Partridge, C. Improving round-trip time estimates in reliable transport protocols. *SIGCOMM Comput. Commun. Rev.* **1987**, *17*, 2–7. [CrossRef]
5. Jacobson, V. Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev.* **1988**, *18*, 314–329. [CrossRef]
6. Allman, M.; Paxson, V.; Blanton, E. TCP Congestion Control. 2009. Available online: https://tools.ietf.org/pdf/rfc5681.pdf (accessed on 1 July 2022).
7. Jacobson, V.; Karels, M.J. Congestion Avoidance and Control: L. B. N. L. (LBNL). 1988. Available online: http://ee.lbl.gov/papers/congavoid.pdf (accessed on 1 July 2022).

8. Low, S.; Peterson, L.; Wang, L. Understanding TCP Vegas: Theory and Practice: P. University. 2000. Available online: http://www.cs.princeton.edu/research/techreps/TR-616-00 (accessed on 1 July 2022).
9. Cheng, J.; Wei, D.X.; Low, S.H. FAST TCP: Motivation, architecture, algorithms, performance. In Proceedings of the INFOCOM 2004, Twenty-Third Annual Joint Conference of the IEEE Computer and Communications Societies, Hong Kong, China, 7–11 March 2004; Volume 4, pp. 2490–2501. [CrossRef]
10. Cheng, J.; Wei, D.; Low, S.H.; Bunn, J.; Choe, H.D.; Doylle, J.C.; Newman, H.; Ravot, S.; Singh, S.; Paganini, F.; et al. FAST TCP: From theory to experiments. *IEEE Netw.* **2005**, *19*, 4–11. [CrossRef]
11. Henderson, T.; Floyd, S.; Gurtov, A.; Nishida, Y. The NewReno Modification to TCP's Fast Recovery Algorithm: I. E. T. F. (IETF). 2012. Available online: https://tools.ietf.org/pdf/rfc6582.pdf (accessed on 1 July 2022).
12. Kumar, A. Comparative performance analysis of versions of TCP in a local network with a lossy link. *IEEE/ACM Trans. Netw.* **1998**, *6*, 485–498. [CrossRef]
13. Sikdar, B.; Kalyanaraman, S.; Vastola, K.S. Analytic models and comparative study of the latency and steady-state throughput of TCP Tahoe, Reno and SACK. In Proceedings of the Global Telecommunications Conference (GLOBECOM 2021), Madrid, Spain, 7–11 December 2021; pp. 1781–1787. [CrossRef]
14. Anjum, F.; Tassiulas, L. Comparative study of various TCP versions over a wireless link with correlated losses. *IEEE/ACM Trans. Netw.* **2003**, *11*, 370–383. [CrossRef]
15. Bashir, K. Comparative study on advance TCP stacks and their performance analysis. In Proceedings of the INMIC 2004—8th International Multitopic Conference, Lahore, Pakistan, 24–26 December 2004; pp. 256–263. [CrossRef]
16. Salleh, M.; Bakar, A.Z.A. Comparative performance of TCP variants on self-similar traffic. In Proceedings of the 1st International Conference on Computers, Communications, & Signal Processing with Special Track on Biomedical Engineering, Kuala Lumpur, Malaysia, 14–16 November 2005; pp. 85–90. [CrossRef]
17. Caini, C.; Firrincieli, R.; Lacamera, D. Comparative performance evaluation of TCP variants on satellite environments. In Proceedings of the 2009 IEEE International Conference on Communications, Dresden, Germany, 14–18 June 2009; pp. 1–5. [CrossRef]
18. Bhanumathi, V.; Dhanasekaran, R. TCP variants—A comparative analysis for high bandwidth–delay product in mobile ad hoc network. In Proceedings of the 2nd International Conference on Computer and Automation Engineering (ICCAE), Singapore, 26–28 February 2010; Volume 2, pp. 600–604. [CrossRef]
19. Kotsiolis, A.; Antonopoulos, C.; Koubias, S. Performance evaluation of TCP algorithms on hybrid wired/wireless LAN test-bed. In Proceedings of the International Conference on Data Communication Networking (DCNET), Athens, Greece, 26–28 July 2010; pp. 1–7.
20. Waghmare, S.; Nikose, P.; Parab, A.; Bhosale, S.J. Comparative analysis of different TCP variants in a wireless environment. In Proceedings of the 2011 3rd International Conference on Electronics Computer Technology (ICECT), Kanyakumari, India, 8–10 April 2011; pp. 158–162. [CrossRef]
21. Tahiliani, R.P.; Tahiliani, M.P.; Sekaran, K.C. TCP variants for data center networks: A comparative study. In Proceedings of the 2012 International Symposium on Cloud and Services Computing (ISCOS), Mangalore, India, 17–18 December 2012; pp. 57–62. [CrossRef]
22. Devaraj, S.A.; Anita, R.H.V.; Christa, J.J. Comparative analysis of random based mobility models using TCP variant in MANETs. In Proceedings of the 2014 International Conference on Communication and Network Technologies (ICCNT), Sivakasi, India, 18–19 December 2014; pp. 324–329. [CrossRef]
23. Byung-Gon, C.; Byeong Gi, L. Auxiliary timeout and selective packet discard schemes to improve TCP performance in PCN environment. In Proceedings of the 1997 IEEE International Conference on Communications: Towards the Knowledge Millennium, ICC 1997, Montreal, QC, Canada, 8–12 June 1997; pp. 381–385. [CrossRef]
24. Goyal, R.; Jain, R.; Kalyanaraman, S.; Fahmy, S.; Vandalore, B.; Kota, S. TCP selective acknowledgments and UBR drop policies to improve ATM-UBR performance over terrestrial and satellite networks. In Proceedings of the Sixth International Conference on Computer Communications and Networks, Las Vegas, NV, USA, 22–25 September 1997; pp. 17–25. [CrossRef]
25. Akyildiz, I.F.; Joe, I. TCP performance improvement over wireless ATM networks through a new AAL protocol. In Proceedings of the Global Telecommunications Conference, GLOBECOM 1998, Rome, Italy, 15–16 January 1998; The Bridge to Global Integration. IEEE: New York, NY, USA, 1998; Volume 1, pp. 508–512. [CrossRef]
26. Pan, J.; Mark, J.W.; Shen, X. TCP performance and its improvement over wireless links. In Proceedings of the Global Telecommunications Conference, GLOBECOM '00, San Francisco, CA, USA, 27 November–1 December 2000; IEEE: New York, NY, USA, 2000; Volume 1, pp. 62–66. [CrossRef]
27. Chinta, M.; Helal, A.; Lee, C. ILC-TCP: An interlayer collaboration protocol for TCP performance improvement in mobile and wireless environments. In Proceedings of the Wireless Communications and Networking, WCNC 2003, New Orleans, LA, USA, 16–20 March 2003; IEEE: New York, NY, USA, 2003; Volume 2, pp. 1004–1010. [CrossRef]
28. JianXin, Z.; BingXin, S.; Ling, Z. Improve TCP performance in ad hoc network by TCP-RC. In Proceedings of the 14th IEEE 2003 International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC 2003, Beijing, China, 7–10 September 2003; Volume 1, pp. 216–220. [CrossRef]
29. Biaz, S.; Vaidya, N.H. "De-Randomizing" congestion losses to improve TCP performance over wired-wireless networks. *IEEE/ACM Trans. Netw.* **2005**, *13*, 596–608. [CrossRef]

30. Qixiang, P.; Liew, S.C.; Leung, V.C.M. Performance improvement of 802.11 wireless network with TCP ACK agent and auto-zoom backoff algorithm. In Proceedings of the 2005 IEEE 61st Vehicular Technology Conference, Stockholm, Sweden, 30 May–1 June 2005; pp. 2046–2050. [CrossRef]
31. Le, D.; Guo, D.; Wu, B. WLC47-6: TCP Performance improvement through inter-layer enhancement with mobile IPv6. In Proceedings of the IEEE Globecom 2006, San Francisco, CA, USA, 27 November 2006–1 December 2006; pp. 1–5. [CrossRef]
32. Daniel, L.; Kojo, M. Using cross-layer information to improve TCP performance with vertical handoffs. In Proceedings of the Second International Conference on Access Networks & Workshops, AccessNets '07, Ottawa, ON, Canada, 22–24 August 2007; pp. 1–8. [CrossRef]
33. Komatireddy, B.; Vokkarane, V.M. Source-ordering for improved TCP performance over load-balanced optical burst-switched (OBS) networks. In Proceedings of the Fourth International Conference on Broadband Communications, Networks and Systems, BROADNETS 2007, Raleigh, NC, USA, 10–14 September 2007; pp. 234–242. [CrossRef]
34. Mahmoodi, T.; Friderikos, V.; Holland, O.; Aghvami, A.H. Cross-Layer Design to Improve Wireless TCP Performance with Link-Layer Adaptation. In Proceedings of the 66th IEEE Vehicular Technology Conference, Baltimore, MD, USA, 30 September–3 October 2007; pp. 1504–1508. [CrossRef]
35. Scalia, L.; Soldo, F.; Gerla, M. PiggyCode: A MAC layer network coding scheme to improve TCP performance over wireless networks. In Proceedings of the IEEE GLOBECOM 2007—IEEE Global Telecommunications Conference, Washington, DC, USA, 26–30 November 2007; pp. 3672–3677. [CrossRef]
36. Prajapati, H.B.; Bhatt, B.S.; Dabhi, V.K. An ELFN-based adaptive probing strategy to improve TCP performance in ad hoc networks. In Proceedings of the International Conference on Computer and Communication Engineering, ICCCE 2008, Kuala Lumpur, Malaysia, 13–15 May 2008; pp. 570–573. [CrossRef]
37. Quan, Z.F.; Kai, M.L.; Park, Y.J. Reasonable TCP's Congestion Window Change Rate to Improve the TCP Performance in 802.11 Wireless Networks. In Proceedings of the Third International Conference on Convergence and Hybrid Information Technology, ICCIT '08, Busan, Korea, 11–13 November 2008; Volume 1, pp. 808–812. [CrossRef]
38. Yide, Z.; Gang, F. A new method to improve the TCP performance in wireless cellular networks. In Proceedings of the International Conference on Communications, Circuits and Systems, ICCCAS 2009, San Jose, CA, USA, 23–25 July 2009; pp. 246–250. [CrossRef]
39. Shetty, S.; Tang, Y.; Collani, W. TCP Venoplus—A cross-layer approach to improve TCP performance in wired-cum-wireless networks using signal strength. In Proceedings of the 2010 International Conference on Networking, Sensing and Control (ICNSC), Chicago, IL, USA, 10–12 April 2010; pp. 693–697. [CrossRef]
40. Tabash, I.K.; Ahmad, N.; Beg, S. A congestion window control mechanism based on fuzzy logic to improve TCP performance in MANETs. In Proceedings of the 2011 International Conference on Computational Intelligence and Communication Networks (CICN), Washington, DC, USA, 7–9 October 2011; pp. 21–26. [CrossRef]
41. Wang, Z.; Xie, X.; Zhao, D. Cross-layer design for TCP performance improvement in vehicular communication networks. In Proceedings of the 2012 14th International Conference on Advanced Communication Technology (ICACT), Pyeongchang, Korea, 19–22 February 2012; pp. 400–405.
42. Douga, Y.; Bourenane, M. A cross layer solution to improve TCP performances in ad hoc wireless networks. In Proceedings of the 2013 International Conference on Smart Communications in Network Technologies (SaCoNeT), Paris, France, 17–19 June 2013; Volume 1, pp. 1–5. [CrossRef]
43. Yang, R.; Chang, Y.; Xu, W.; Yang, D. Hybrid multi-radio transmission diversity scheme to improve wireless TCP performance in an integrated LTE and HSDPA networks. In Proceedings of the 2013 IEEE 77th Vehicular Technology Conference (VTC Spring), Dresden, Germany, 2–5 June 2013; pp. 1–5. [CrossRef]
44. Krishnaprasad, K.; Tahiliani, M.P.; Kumar, V. TCP Kay: An end-to-end improvement to TCP performance in lossy wireless networks using ACK-DIV technique & FEC. In Proceedings of the 2015 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), Bangalore, India, 10–11 July 2015; pp. 1–6. [CrossRef]
45. Mohammadani, K.H.; Memon, K.A.; Memon, I.; Hussaini, N.N.; Fazal, H. Preamble time-division multiple access fixed slot assignment protocol for secure mobile ad hoc networks. *Int. J. Distrib. Sens. Netw.* **2020**, *16*, 1550147720921624. [CrossRef]
46. Kumar, B. *Broadband Communications*; McGraw-Hill: New York, NY, USA, 1998.
47. Stevens, W.R. *TCP/IP Illustrated: The Protocols*; Addison-Wesley: Boston, MA, USA, 1994; Volume 1.