future internet

MDPI

Article

Performance Analysis of a Novel TCP Protocol Algorithm Adapted to Wireless Networks

Gonzalo Olmedo [1], Román Lara-Cueva [1,*], Diego Martínez [1] and Celso de Almeida [2]

[1] Grupo de Investigación en Sistemas Inteligentes (WiCOM-Energy), and Centro de Investigaciones de Redes Ad-Hoc (CIRAD), Departamento de Eléctrica, Electrónica y Telecomunicaciones, Universidad de las Fuerzas Armadas ESPE, Sangolquí 171103, Ecuador; gfolmedo@espe.edu.ec (G.O.); dxmartinez@espe.edu.ec (D.M.)

[2] School of Electrical and Computer Engineering, University of Campinas (UNICAMP), Campinas-SP 13083-970, Brazil; celso@decom.fee.unicamp.br

* Correspondence: ralara@espe.edu.ec; Tel.: +593-9-92537182

Received: 21 April 2020; Accepted: 4 June 2020; Published: 9 June 2020

Abstract: As telecommunication systems evolve towards new-generation architectures, likewise, new protocols are created in order to improve efficiency. One of these protocols is Transmission Control Protocol (TCP), which controls the transmission bit rate in function of network congestion. Nevertheless, in wireless communications, there appear problems such as noise and interference, for which TCP was not designed. Based on these problems, there exist some methods trying to mitigate congestion, such as explicit loss notifications and the use of end-to-end codification. The aim of this work was to propose a wireless TCP protocol improvement, considering a negative acknowledgment (NACK), which allows to differentiate between losses due to congestion and losses due to wireless channel issues. NACK employs a small protocol packet and produces improvement in the quality of service metrics. The experiments were carried out in in-door and out-door environments, over an online video game scenario, and over a long-distance wireless link between two islands. The average results show a 25-percent delay improvement and a 5-percent jitter improvement when compared to the original TCP Reno protocol, while for throughput a 90-percent improvement was achieved for distances between 100 and 414 m.

Keywords: CWDN; NACK; Linux kernel; wireless TCP

1. Introduction

The Transmission Control Protocol (TCP) was originally designed for wired networks, by interpreting packet loss (PL) as congestion consequence; meanwhile, in wireless networks, PL is also caused by issues in the communication channel, such as interference, multi-path fading, mobility, and reflections, which may produce a PL misinterpreting [1]. In order to avoid congestion, TCP employs cumulative Acknowledgments (ACK); in this sense, the receptor sends ACKs to indicate that a packet was successfully received and to specify that it is expecting the next one. If the transmitter undetected an ACK, the packets are re-sent or the connection is disrupted [2]. TCP can use four algorithms for Congestion Window (CWND) control: Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery. These algorithms seek to achieve a high network performance and to avoid collapse by congestion. Slow Start increases exponentially the transmission bit rate until a packet loss is detected. From this point, the Congestion Avoidance algorithm increases the transmission bit rate linearly, and when three duplicate ACKs are detected, TCP reduces the transmission bit rate by half. Meanwhile, Fast Retransmit and Fast Recovery allow to resend packets immediately when three duplicate ACK (DUPACK) are detected. The transition from one algorithm to another occurs when PL is detected [3]. Although CWND control algorithms achieve their goal in wired networks, in wireless networks,

unnecessary executions of the four algorithms occur frequently, which, in turn, causes low transmission bit rates, as seen in the results presented in [4].

TCP has different versions seeking CWND optimization by an ACK notification rate analysis. Two of the most popular TCP versions are TCP Reno and TCP Westwood. The first one reduces CWND by half after three DUPACK, while the second selects a Slow Start threshold (ssthresh) and a CWND consistent with the effective connection bit rate [5]. Several efforts to improve TCP protocol in wireless networks have been developed, for instance, Explicit Loss Notifications (ELN) which adds explicit bits to the packet as feedback to the TCP source to identify the loss cause. For example, in [6], ELN gives a superior throughput ($\overline{\theta}$) than TCP Reno and TCP-Tahoe, reaching as high as 80 to 90% in an error free simulated scenario. Similarly, in [7], by using ELN, reduced response times of almost 30% were obtained for high error probability web traffic in a simulated scenario. Another proposal for wireless TCP improvement is the end-to-end coding approach [8], in which redundancy is added to the TCP packet to facilitate loss recovery, which outperforms in simulation TCP New Reno and TCP with the selective acknowledgment option (TCP SACK) over a range of loss probability and propagation delays.

Additionally, other modification approaches exist for TCP functionality, such as the use of a fast end-to-end retransmission scheme [9]. This proposal avoids long pauses in communication during cellular handoffs. It requires minimal changes to end terminals and improves $\overline{\theta}$ values from 1400 to 1490 kb/s for non-overlapped cells with zero seconds between cells, and from 1100 to 1380 kb/s for non-overlapped cells with 1 second between cells. In [10], the Indirect TCP protocol (I-TCP) splits the network in a wired-cum-wireless connection by adding a Mobility Support Router, and then it attempts to alleviate mobility related performance problems by adapting the transport layer protocols on the wireless link, modifications are required only on the Mobile Host (MH). This solution achieved better $\overline{\theta}$ than TCP Reno for a local area network under a Bit Error Rate (BER) of up to 2×10^{-6}, and twice θ for a wide area network under BER of up to 5×10^{-6}. In [11], a sender-only TCP modification based on dynamic bandwidth estimation in wired-cum-wireless networks is presented as TCP Prairie. This protocol fairly estimates the bandwidth to set the CWND and the *ssthresh* after three DUPACK or after a timeout. The protocol was simulated by ns-2, where the θ gain over TCP Westwood was 17 percent at 0.01 BER and 542% over TCP New-Reno at 0.5 BER. Finally, another wireless link problem to be addressed is the Retransmission Timeout (RTO) which is calculated based on the Round-Trip Time (RTT), which is fluctuating in wireless networks. This RTT dependence can be appreciated on in-door environments, where θ unbalances, hidden terminals problems, and interference among Basic Service Sets frequently occur. As a solution, in [12], a MAC layer contention window control is proposed using an analytical model of nonlinear equations, achieving a fair θ distribution with a confidence interval of 99.98% at a 2 Mb/s bit rate.

Several solutions for wired-cum-wireless network improvement with a wireless local retransmission approach have been also proposed, for example, in [13], the snoop protocol caches unacknowledged packets from the fixed host at the Base Station (BS) and performs local retransmissions across the wireless link. For a wireless link BER of over 5×10^{-7}, the snoop protocol achieves a $\overline{\theta}$ improvement of 1 to 20 times than TCP Reno for data transfer from the fixed host to the MH, requiring modifications at the BS and the MH. In [14], a TCP extension called Delayed Congestion Response TCP protocol (TCP-DCR) was proposed. Such protocol delays the fast retransmit/recovery algorithm when the sender receives the first DUPACK by starting a delayed response timer in order to allow a link level retransmission at the BS to recover a lost packet. With some modifications at the sender host, TCP-DCR reaches better θ than TCP-SACK for several BER values. Meanwhile, in [15], the authors propose a trend topic in communications by using machine learning based loss discrimination algorithm (ML-LDA) for wireless TCP congestion control. ML-LDA learns how to distinguish packet losses due to congestion and wireless channel environment using multi-layer perceptron (MLP). Based on the learning results, the congestion control classifies the cause of losses and does not reduce congestion window in case of random losses. The algorithm was implemented in Linux kernel and configured a testbed where packet loss occurs randomly. They compared the experimental results with TCP-Reno

and TCP-Westwood and showed that the proposed ML-LDA has 98% packet loss classification accuracy in wireless channel environments, and average throughput is greatly improved compared to the others congestion controls.

In this context, the aim of this paper is to describe the development of a novel simple TCP protocol algorithm adapted to wireless networks. Specifically, we address improvements in the wireless TCP. In this paper, we will consider all stages related to performance evaluation, including a mathematical analysis, configuring our proposal by modifying the linux kernel, emulations, and finally testbed analysis in real scenarios. Our primary hypothesis is that by inserting a negative acknowledgment (NACK) flag inside the TCP header, naming this TCP modification as "TCP-NACK", we will prevent the misinterpretation of PL due to channel instability, as well as PL due to congestion, thus reaching a better performance. In this sense, we consider the ELN concept of retransmitting corrupted packets without the CWND reduction.

The rest of the paper is organized as follows: In Section 2, we describe the methods and materials used for TCP-NACK design, emulation, and evaluation. Section 3 presents the experimentation process and how the results were obtained. Section 4 presents the discussion of the results when compared to generic TCP algorithms. Finally, our conclusion and future works are presented in Section 5.

2. Materials and Methods

This section shows the tools and techniques used for the TCP-NACK protocol development, including its mathematical analysis, implementation and testbed experimentation.

2.1. Mathematical Analysis

We start the analysis from a mathematical modeling of the Generic TCP behavior in order to develop an adaptation for corrupted packets retransmission without the CWND size reduction (i.e., the TCP-NACK performance idea) [16]. For this purpose, Slow Start and Congestion Avoidance algorithms were used for modeling the Generic TCP protocol in their Reno and Westwood version.

At the TCP layer, we use the maximum packet size (MSS) of N_{TCP} bits. The sender releases packets into a limited FIFO buffer that can hod up B packets. The packets are then sent over a single bottleneck link with a speed of $R_p = R_b/N_{TCP}$ packets per second, where R_b is the bit rate of the TCP layer.

Figure 1 shows the ω-th cycle evolution of the congestion window between two indications of acknowledgements (ACKs). If W_0 is the initial congestion window, the first burst contains exactly W_0 packets, the second $W_0 + 1$ packets and so on. The congestion window is increased by one at the end of each burst. The index of the first packet lost in the cycle is denoted by n_l. If x is the burst where the lost one occurs, in the burst $x + 1$ the congestion window is reduced for $W_{0(\omega+1)}$ and this value depends on the estimate bandwidth for TCP Reno or TCP Westwood [5].

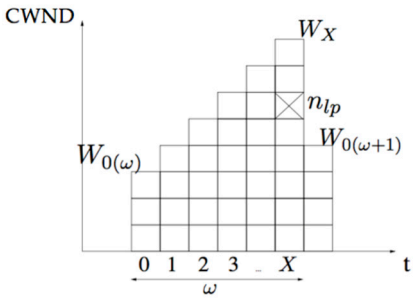

Figure 1. Evolution of CWND.

We considered that the congestion window is increased by one at the end of each burst. Therefore, the congestion window for the x-th burst is given by

$$W_x = W_0 + x. \tag{1}$$

When the congestion does not exist, the congestion window has a linear increase for every round-trip times (*RTT*) seconds, given by

$$RTT = \frac{1}{R_p} + \tau_p, \tag{2}$$

where τ_p is the propagation time.

During a cycle, ACKs in the same burst arrive $1/R_p$ seconds apart while consecutive bursts arrive *RTT* second apart, until the pipe capacity, $W_C = R_p RTT$, is reached and $c = W_C - W_o$. After this point, ACKs arrive continuously every $1/R_p$ seconds.

Since a packet loss would occur one the buffer is full, the maximum window size a connection can achieve is given by

$$W_{max} = W_C + B, \tag{3}$$

for $x = W_C - W_0 + B$.

Let n_x the index of the first packet in the x-th burst, given by

$$n_x = 1 + \sum_{j=0}^{x+1} W_j \;\; = \; 1 + xW_0 + \frac{x(x-1)}{2}, \tag{4}$$

The index of the packet dropped due to buffer overflow is denoted by n_{of} and is given by:

$$n_{of} = n_{cb} + 2(W_C + B) \tag{5}$$

where n_{cb} is the first packet in the burst $x = c + B$, for the maximum window size W_{max}.

Let x_n be the number of burst that contain the packet number n and r_{x_n} the offset of the packet in the burst x_n. These parameters for $1 \le n \le n_{of}$ are expressed as

$$x_n = \left\lfloor -W_0 + \frac{1}{2} + \sqrt{W_0^2 - W_0 - \frac{7}{4} + 2n} \right\rfloor \tag{6}$$

and

$$r_{x_n} = n - n_{x_n} \tag{7}$$

The instant in which the n-th ACK is received is given by

$$\Delta t(W_0, n) = \begin{cases} RTT(x_n - 1) + \frac{r_{x_n}}{R_p} & n \le n_c, \\ RTT(c+1) + \frac{n - n_c}{R_p} & n > n_c, \end{cases} \tag{8}$$

where $c = W_C$-W_0x^o is the burst where the pipe capacity occurs.

For the TCP Reno at the beginning of a generic cycle ω, the start threshold according to the value of the estimate bandwidth at the end of the previous cycle for congestion avoidance is given by

$$\begin{aligned} W_{0(\omega)} &= f_\omega\left(W_{0(\omega-1)}, n_{l(\omega-1)}\right) \\ &= \max\left\{2, \left\lfloor \frac{W_{0(\omega-1)} + x_{n_l(\omega-1)}}{2} \right\rfloor\right\}, \end{aligned} \tag{9}$$

where n_l is the ACK missed at the end of the cycle $\omega - 1$.

The TCP Westwood sets the slow start threshold according to the value of the estimated bandwidth at the end of the previous cycle using the following expression [5]:

$$W_{0(\omega)} = \max\left\{2, \left\lfloor \frac{W_{n_l(\omega-1)} RTT}{2} \right\rfloor\right\} \tag{10}$$

where

$$W_{n+1} = \alpha W_n + (1-\alpha)\frac{b_{n+1}+b_n}{2}, \tag{11}$$

where α is related to the cut-off frequency of the low pass filter and b_n is the bandwidth information carried by the n-th ACK, expressed as

$$b_n = \begin{cases} \frac{R_p}{W_C - W_{x_n}+1}, & x_n < x_C \\ R_p & \text{otherwise} \end{cases} \tag{12}$$

The evolution of the initial congestion window size, in each cycle, can be modelled as a Markov process. $W_{0(\omega)}$ depends only on $W_{0(\omega-1)}$ and $n_{l(\omega-1)}$. The transition probability from $W_{0(\omega-1)} = i$ for $W_{0(\omega)}$ = j is given by

$$\begin{aligned} p_{i,j} = \mathrm{P_r}\left\{W_{0(\omega)} = j \middle| W_{0(\omega-1)} = i\right\}; \quad i,j \in \{2,3,\ldots\ldots,W_C\} \\ = \sum_{n \epsilon n_l} \Pr\{n|W_0 = i\} \end{aligned} \tag{13}$$

where the probability that the packet n is dropped, given that the initial window $W_0 = i$, and is expressed as

$$\Pr\{n|W_0 = i\} = \begin{cases} P_{packet,TCP}\left(1 - P_{packet,TCP}\right)^{n-1}, & n < n_{of}(i) \\ \left(1 - P_{packet,TCP}\right)^{n-1}, & n = n_{of}(i) \end{cases} \tag{14}$$

where $P_{packet,TCP}$ is the TCP packet error probability.

Finally, using (8) and (14) the throughput realized by the system is given by [5]

$$\overline{\theta} = \sum_{i=2}^{C} \pi_{(W_0=i)} \sum_{n=2}^{n_{of}(W_0=i)} \frac{n-1}{\Delta_t(W_0=i,n)} \Pr\{n\,|W_0 = i\} N_{TCP}, \ [\text{bits/s}] \tag{15}$$

where $\pi_{(W_0=i)}$ is the asymptotic probability of the initial window size W_0.

Then, we added in (16) the packet loss probability effect due to wireless issues in order to fit the model to the TCP-NACK protocol behavior proposed. Therefore, if the packets are corrupted due to the wireless channel, these errors are recognized by NACKs, and the CWND increases normally (i.e., until the depletion of the receptor buffer). Finally, the TCP-NACK $\overline{\theta}$ is defined as

$$\overline{\theta}_{NACK} = \sum_{i=2}^{C} \pi_{(W_0=i)} \sum_{n=2}^{n_{of}(W_0=i)} \frac{(n-1)\left(1 - P_{packe,TCP}\right)}{\Delta_t(W_0=i,n)} \Pr\{n\,|W_0 = i\} N_{TCP}, \ [\text{bits/s}] \tag{16}$$

2.2. TCP-NACK Linux Implementation

We chose and modified the TCP/IP stack from Linux 2.6.32 source code for the TCP-NACK implementation based on TCP-Reno protocol, since it is an open access and open source operating system [17]. Specifically, we modified the files inside the "include" and "net" directories, which are distributed within the Linux kernel as showed in Figure 2.

2.3. TCP Sockets

In Linux, the different communication protocols are implemented by sockets, which works as a common interface between the user and the different systems files and device systems. Three data structures for the sockets handling are used, the first is called "socket buffer", which store the packet information, the second is called "socket", which register the open connections, and the last is called "sock", which maintains the open connections state [17].

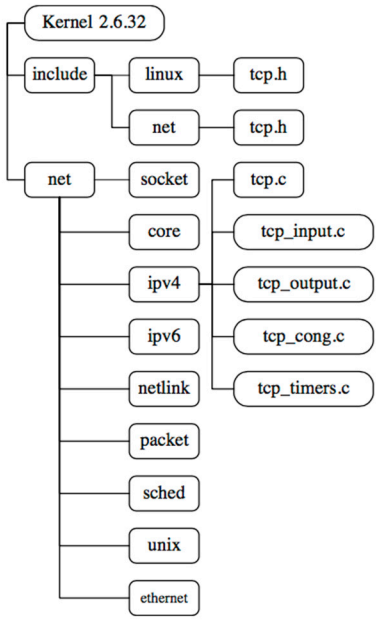

Figure 2. Linux source code architecture.

2.3.1. Data Sending by TCP

TCP is a protocol that ensures the reliable transmission by executing a data flow management. Therefore, there are three principal functions inside the Linux kernel that intervene in the data sending process and flow management [17].

1. tcp_sendmsg: which copies the user space data to the Linux kernel space, then it is assigned to the "socket buffer" and divided in smaller packets.
2. tcp_send_skb: which organizes the data in "socket buffer" for the transmission queue, and decides whether the transmission can take place or not.
3. tcp_transmit_skb: which builds the TCP header and sends the packets to the network layer.

2.3.2. Data Reception by TCP

Received packets must be transferred from the network layer to the transport layer for the TCP header and data processing by the following functions.

1. tcp_v4_rcv: which verifies the packet integrity, it checks if the packet is properly destined, process the transport layer checksum and removes the IP header.
2. tcp_v4_do_rcv: which verifies that the received packet has a complete header and checks the current TCP connection state.
3. tcp_rcv_established: If the current TCP connection state is "ESTABLISHED", it processes the received packets and copies the data to the user space. If the packets are error-free, a fast process denominated "fast path" is executed, otherwise a "slow path" process is executed.

2.3.3. TCP Header Modifications

We used one of the TCP header reserved bits for the NACK implementation, as showed in Figure 3. If the NACK bit is "1", it means that a "corrupted" packet was received, otherwise it corresponds to an ACK. In the same way that ACK header, NACK header has a NACK information field and a NACK number.

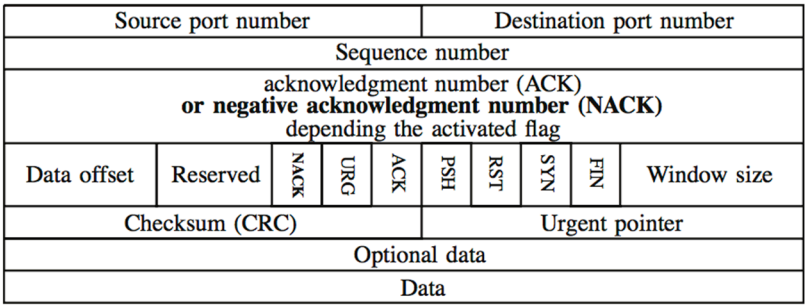

Figure 3. TCP-NACK header.

The code that defines the TCP header is in the /include/Linux/tcp.h file. Inside the tcphdr structure, we created the nack attribute for the NACK flag creation, and we reduced to three bits the attribute res1 that corresponds to the reserved TCP field. Algorithm 1 shows the tcp.h file modifications implemented.

Algorithm 1 tcphdr structure modifications for the NACK flag insertion

```
res1 = 4 bits
if little or big endian bitfield is used then
define nack = 1 bit
    res1 = res1 - size(nack)
                end if
```

The TCP flag position in the header is defined inside the tcp_flag_world structure with the TCP_FLAG_FLAGNAME = __cputobe32(0xXXXXXXXX) format, where "X" corresponds to the bit flag position. Hence, we added the TCP_FLAG_NACK = cpu_to_be32 (0x01000000) line in order to assign the NACK flag position in the TCP header.

Moreover, we added the #define TCPCB_FLAG_NACK 0x100 line inside the tcp_skb_cb structure in order to assign the value that NACK flag takes when it is assigned to a packet. Additionally, it was necessary to increase the flags attribute type from u8 (8 bits) to u16 (16 bits).

2.3.4. Sending NACK Notifications

We created and implemented the NACK sending function based on the tcp_send_ack function, located inside the /net/ipv4/tcp_output.c file, due to the similarity between NACK and ACK approach.

- New tcp_send_nack function: This function, declared inside the /include/net/tcp.h file, sends NACK notifica- tions by four steps. First, it checks if the connection was restarted, if this is true, then the NACK notification is annulled. After that, it calls the skb_reserve function, that creates a socket buffer with a memory space of the maximum TCP header size. Then, the tcp_init_nondata_skb function sends the posi- tive NACK flag state and the sequence number to be retransmitted to the control buffer. Finally, the packet is transmitted to the network layer by the tcp_transmit_skb function. Algorithm 2 shows the description of the NACK sending function.

Algorithm 2 tcp_send_nack function algorithm

```
if TCP state == close then
    return
end if
skb_reserve(maximum TCP header size)
tcp_init_nondata_skb(NACK flag state, sequence number)
tcp_transmit_skb()
```

- *tcp_transmit_skb* modification: If the packet to be transmitted has raised the TCP_FLAG_NACK flag, the TCP confirmation number field must contain the sequence number of the corrupted packet. Algorithm 3 shows this modification.

Algorithm 3 *tcp_transmit_skb* function modification

```
if NACK flag == 1 then
    TCP header <- sequence number of the corrupted packet
else
    TCP header <- sequence number of the next expected packet
end if
```

The negative confirmations must be triggered by reception of the corrupted TCP packets, thus *tcp_send_nack* function is then called inside *tcp_rcv_established* function for detecting an error.

2.3.5. Receiving NACK Notifications

There are two possible paths for the received packets treatment, "Fast Path" and "Slow Path". All the packets with a raised NACK flag use the "Slow Path". First the packet integrity is checked by a checksum to consequently verify the NACK flag state, if this is positive, the *tcp_retransmit_skb* function is called for an immediate retransmission of the requested packet, which is determined by the *tcp_write_queue_head* function.

3. Results

We evaluated the proposed TCP-NACK protocol under different configuration environments and distances by using laptops with the modified TCP-NACK protocol under the Ubuntu kernel, and applying the necessary communication standards for each experiment. We choose to evaluate the general Quality of Service (QoS) provisioning parameters from a networking perspective using the $\overline{\theta}$, the delay, δ, and jitter metrics [18]. Hence, those parameters were obtained by the intrusive traffic technique, for which we employed the Distributed Internet Traffic Generator (D-ITG) software. D-ITG is a platform capable of producing IPv4 and IPv6 traffic, following stochastic models for packet size and inter departure time [19]. The details of our fulfilled experiments are described next.

3.1. Wireless Link Emulation

In this experiment, we evaluated TCP-NACK performance by the Network Emulator (Netem) software, which is an utility available within the Linux kernel from the 2.6.7 version, and allows emulating the link properties by specifying bandwidth parameters, δ, losses, and traffic control using statistical probability [20].

This experiment required a computer with the Ubuntu Operating System and with two network interfaces configured in bridge mode to emulate the wireless link. As user equipment, we employed two computers with the modified TCP-NACK Ubuntu Operating System, which acted as a client and a server, respectively. In each user equipment, we used the "Iperf" and "Tcpprobe" tools, available within the Linux repositories, those tools allowed to obtain the CWND and $\overline{\theta}$ values.

The scenario consisted of two computers interconnected by an emulated wireless link, which was managed in an intermediate computer by the Netem software, as showed in Figure 4. The link emulation was required to maintain a fixed error rate for multiple traffic injections.

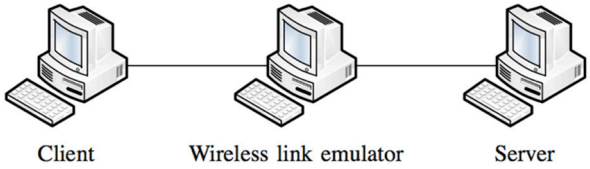

Figure 4. Emulation test topology.

Netem is a utility available within the Linux kernel, it allows to emulate a communication link by the specification of its parameters [20]. Table 1 shows the parameters used for a 25-second emulation of the wireless link, with high packet errors probability, greater than 0.5% ($P_{seg,TCP} \geq 5 \times 10^{-3}$). The commands used to define Netem parameters in the Linux terminal were:

1. sudo tc qdisc add dev ethX root handle 1:0 tbf rate Bandwidthkbit
2. sudo tc qdisc add dev ethX parent 1:1 handle 10: netem corrupt Y% delay Z

where ethX is the network interface name, Bandwidth is the link bandwidth in kb/s, Y is the error probability in percent, and Z is the δ in milliseconds.

Table 1. Wireless Link emulation Paramenters.

Parameter	Value
Bandwidth	5000 kb/s
δ	70 ms
Packet error probability	0.5%, 1% and 5%

3.1.1. Iperf and Tcpprobe Configuration

Iperf allowed to do a client-server TCP connection and to size its $\overline{\theta}$. Table 2 shows the necessary configuration commands for both user equipment. It usually runs in the emitter side with Tcpprobe tool, which allowed us to extract parameters as CWND, ssthresh and Congestion Avoidance threshold, and then to store the results in the data.out file.

Table 2. IPERF Configuration Commands.

TCP Entity	Linux Command
Emitter (server)	iperf -c IPdirection -t time
Receiver (client)	iperf -s

3.1.2. $\overline{\theta}$ Evaluation

We did tests with the Iperf tool to obtain the $\overline{\theta}$ values under different error probabilities. Each test consisted in establishing a TCP connection and to perform a data transference for 1 minute.

3.1.3. CWND Evaluation

For the CWND evaluation, we started a TCP connection between both terminal equipment and transmitted the maximum possible raw bit rate through the emulated link for 25 seconds.

3.2. Short Distance Link Whit a Real-Time Video Game

We evaluated the TCP-NACK performance over a video gaming scenario with two players and one server wirelessly connected by an AP, where players and server were laptops with the modified TCP-NACK Ubuntu Operating System. The game at the server was "Quake IV" and the employed topology was Point-to-Multipoint (PtM), as Figure 5 shows.

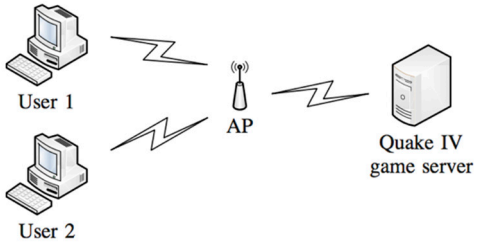

Figure 5. Short distance network topology.

We perform this experiment over two scenarios. The first one was an in-door scenario with a 7-m distance between user equipment and the Netgear N600 wireless router configured as AP, using IEEE 802.11n as communication standard. The second one was an out-door scenario with 160 m' distance between the user equipment and the Ubiquiti PowerStation2 antenna configured as AP, using IEEE 802.11b/g as communication standard. The experiments were carried out with a 128.8 kb/s bandwidth between the user equipment and the server (uplink), and a 222 kb/s bandwidth between the server and the user equipment (downlink). Additionally, we introduced the OpenVPN tool for the adequacy of the multiplayer communication system. OpenVPN allows to connect multiple remote clients to the server, where any client can communicate with the rest of them [21].

3.2.1. Server Equipment Configuration

After the OpenVPN software was installed, we configured some parameters in the server.conf file with the data shown in Table 3.

Table 3. Server configuration parameters.

Parameters	Data
Transmission model	TCP
Interphase to implement	TUN
Keys importation	.crt, .key, .ca
Receiver (client)	IP Direction

3.2.2. User Equipment Configuration

At the user side of the network, we modified the *client.conf* file for the connectivity between equipment, with the parameters shown in Table 4 Additionally, we copied to the */etc/openvpn* path, the user keys created at the server.

Table 4. Client configuration parameters.

Parameters	Data
Interface	dev tun
Interphase to implement	ca.crt, client.crt, client.key
IP direction	remote 10.0.0.7 port 1194
Compression	TCP

After configuring the client and server, the clients executed the sudo./quake4-linux-1.4.2.x86.run command in a terminal, the command displayed the principal menu for the game, giving them the option to enter to the multiplayer mode. More details about this experiment are presented in [22].

3.2.3. Short-Medium Distance Scenario

This experiment included short distance IEEE 802.11b/g WiFi links (6, 12, 100, 150, 60 and 260 m) using an Ubiquiti PowerStation2 antenna configured as AP, and medium distance IEEE 802.16-2009 WiMAX links (312, 364, 414 and 826 m) using an Albentia System ARBA550 BS and two CPE150

Customer Premises Equipment (CPE). Both links used laptops with the modified TCP-NACK Ubuntu Operating System as user equipment in a Point-to-Point (PtP) topology. Figure 6a shows the used WiFi link topology and Figure 6b shows the used WiMAX link topology.

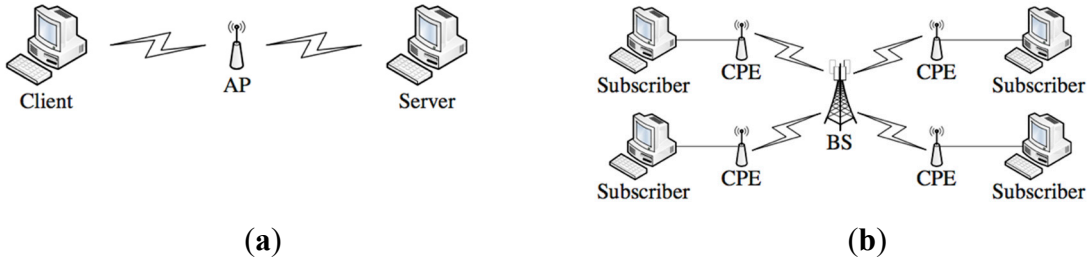

Figure 6. Short-medium distance topology: (**a**) WiFi test; (**b**) WiMAX test.

We performed tests under different bandwidth values by the modification of the number of packets per second and the frame size, as shown in Table 5 More details about this experiment are presented in [23].

Table 5. Data injection configuration.

Packets/s	Frame Size [Bytes]	Resulting Bandwidth [kb/s]
1500	512	6624
2500	512	11,880
1000	1500	12,320
1500	1024	12,768

3.3. Long Distance Scenario

For this experiment, we considered a 92 km PtP link with line-of-sight between the Crocker hill in Santa Cruz island (0°38′50.0′′ S 90°19′21.0′′ W) and San Joaquin tower in San Cristóbal island (0°53′26.9′′ S 89°29′40.0′′ W), using the IEEE 802.16-2009 WiMAX as communication standard, and employed 1, 2, 3 and 4 Mb/s bandwidth. First, we validated the link by simulation through the "SImulation of Radio-Electric NETworks" (SIRENET) software and the Motorola PTP LINKPlanner software, with the aim of eliminating most design assumptions. Eventually, we used two parabolic antennas with 24 dBi gain, two computers with the TCP-NACK modified kernel, and the Motorola PTP 58,600 proprietary radio equipment to implement the network. Figure 7 shows the PtP topology used for this experiment. More details about this experiment are presented in [24].

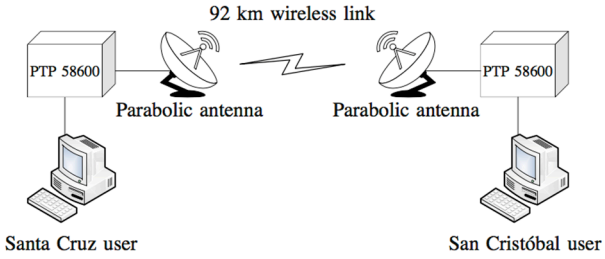

Figure 7. Long-distance network topology.

4. Discussion

Several tests were executed in each experiment scenario with the aim of obtaining average results. We obtained, from emulation experiment, $\overline{\theta}$ values by the computation of the area under the CWND performance curve. Likewise, we obtained from field experiments, $\overline{\theta}$, δ, and jitter values by D-ITG software.

4.1. Results from Mathematical Analysis

A normalized throughput ($\eta\overline{\vartheta}$) was obtained for TCP Reno, TCP Westwood, TCP-NACK Reno and TCP-NACK Westwood. The TCP Reno and Westwood results were calculated by (16), and the TCP-NACK results by (17). Figure 8 shows a comparison of the generated curves for each protocol as a function of the error packet probability. We found that both TCP-NACK versions had a similar behavior among them, and a better performance that its generic versions. Therefore, based on this analysis, for the next experiments, we only compare TCP-NACK with TCP Reno, declared as Generic TCP.

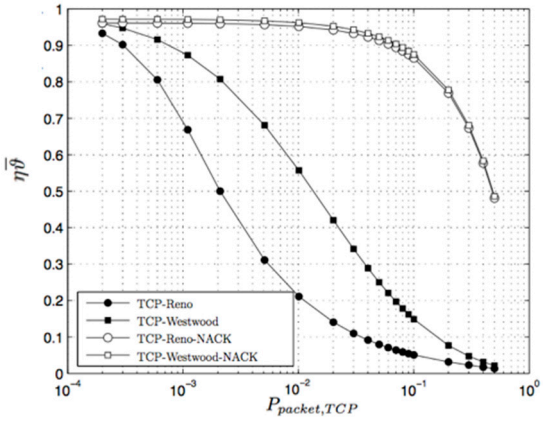

Figure 8. Theoretical $\eta\overline{\vartheta}$ as a function of the error packet probability.

4.2. Results from Emulation Experiment

For this experiment, we obtained a $\eta\overline{\vartheta}$ comparison between Generic TCP and TCP-NACK as a function of the error packet probability. Figure 9 shows the generated curves for each emulated protocol. We obtained the CWND size as a function of the BER value through time and calculated the area under each CWND curve. Hence, we obtained the $\overline{\theta}$ values shown in Table 6. We found that the CWND size of TCP-NACK surpass the CWND size of Generic TCP in 2 average packets, leading to a higher area under the curve, or a $\overline{\theta}$ increment. The throughput of the TCP-NACK was better than TCP-Reno and if we compare the results presented in Figure 8, the throughput of the TCP-NACK also appear better than TCP-Westwood, and the algorithm of the TCP-NACK has less complexity than TCP-Westwood. Therefore, TCP-NACK will be used in practical experiments to validate its performance.

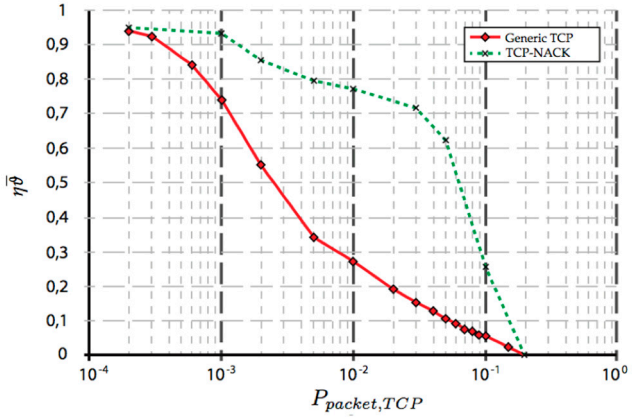

Figure 9. Emulated $\eta\overline{\vartheta}$ as a function of the error packet probability.

Table 6. Emulated link results.

BER	Generic TCP		TCP-NACK	
	$\bar{\theta}$ (Mb/s)	$\eta\vartheta$ [%]	$\bar{\theta}$ (Mb/s)	$\eta\vartheta$ [%]
0.5%	1.53	30.6	1.79	35.8
1%	1.27	25.4	1.46	29.2
5%	0.52	10.4	0.74	14.8

4.3. Results from Indoor Short Distance Experiment

For this experiment by using WiFi technology in indoor scenarios, we obtained $\overline{\theta}$, δ, and jitter as a function of the distance and the traffic direction (uplink or downlink). Figure 10 shows the performance results obtained by using the D-ITG software. The results show a lower $\overline{\theta}$ performance for TCP-NACK, but a better δ and jitter performance when compared to Generic TCP.

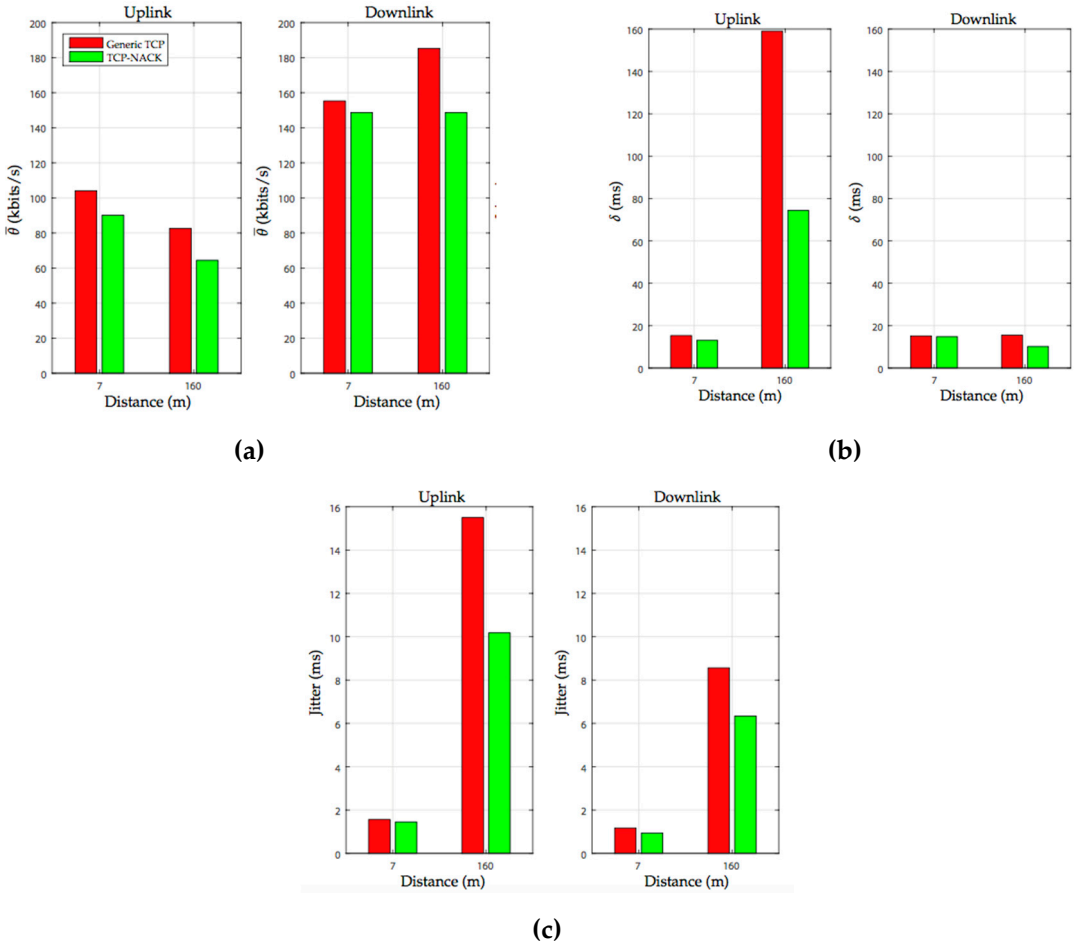

Figure 10. Short distance results: (**a**) θ; (**b**) δ; (**c**) jitter.

4.4. Results from Outdoor Short-Medium Distance Experiment

For this experiment in outdoor scenarios, we obtained $\overline{\theta}$, δ, and jitter as a function of the distance and bandwidth, therefore we decided to represent the obtained results as 3D surfaces, as Figure 11 shows. Figure 11a,c,e show the $\overline{\theta}$, δ, and jitter results, respectively. Hence, in order to show which protocol has a better performance, Figure 11b corresponds to a top view of Figure 11a, Figure 11d corresponds to a bottom view of Figure 11c, and Figure 11f corresponds to a bottom view of Figure 11e, the better protocol corresponds to the one that encloses a long area. Additionally, the left side of the yellow surface in all the subfigures inside Figure 11 corresponds to outdoor scenarios with short

distances (to 300 m) by using WiFi Technology, and the right side to medium distances (from 300 m to 826 m) by using WiMAX technology. Figure 11b,d,f suggest that TCP-NACK has best performance than the Generic TCP in the majority of the cases. We observed that TCP-NACK had a better performance in terms of throughput, delay and jitter. We found that the algorithm TCP-NACK, by not decreasing the congestion window, as we could observe in Figure 11, for WiFi technology TCP-NACK in mean has offer an improvement over the 70% in terms of throughput, delay and jitter, meanwhile for WiMAX technology over the 70% for throughput and jitter, and almost 100% in terms of delay.

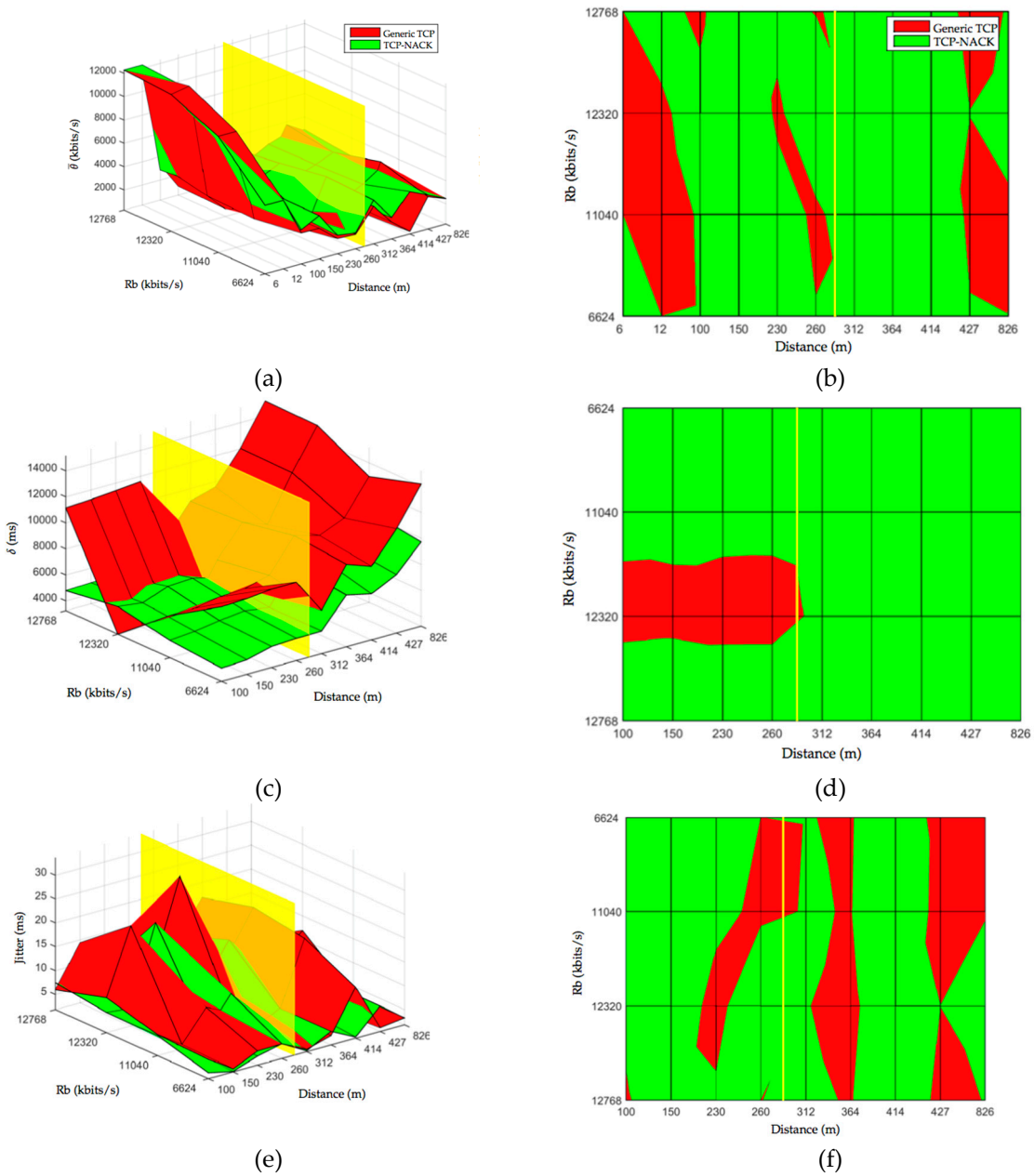

Figure 11. Short-medium distance results: (**a**) $\overline{\theta}$; (**b**) upper $\overline{\theta}$; (**c**) δ; (**d**) bottom δ; (**e**) jitter; (**f**) bottom jitter.

4.5. Results from Long Distance Experiment

For this experiment, we obtained $\overline{\theta}$, δ, and jitter as a function of the bandwidth for a 94 km distance between terminals. Figure 12 shows $\overline{\theta}$, δ, and jitter results obtained by using the D-ITG software. The results show that there are low $\overline{\theta}$ and jitter variance between TCP-NACK and Generic TCP, but a better δ performance for TCP-NACK for bandwidth up to 3000 kbps.

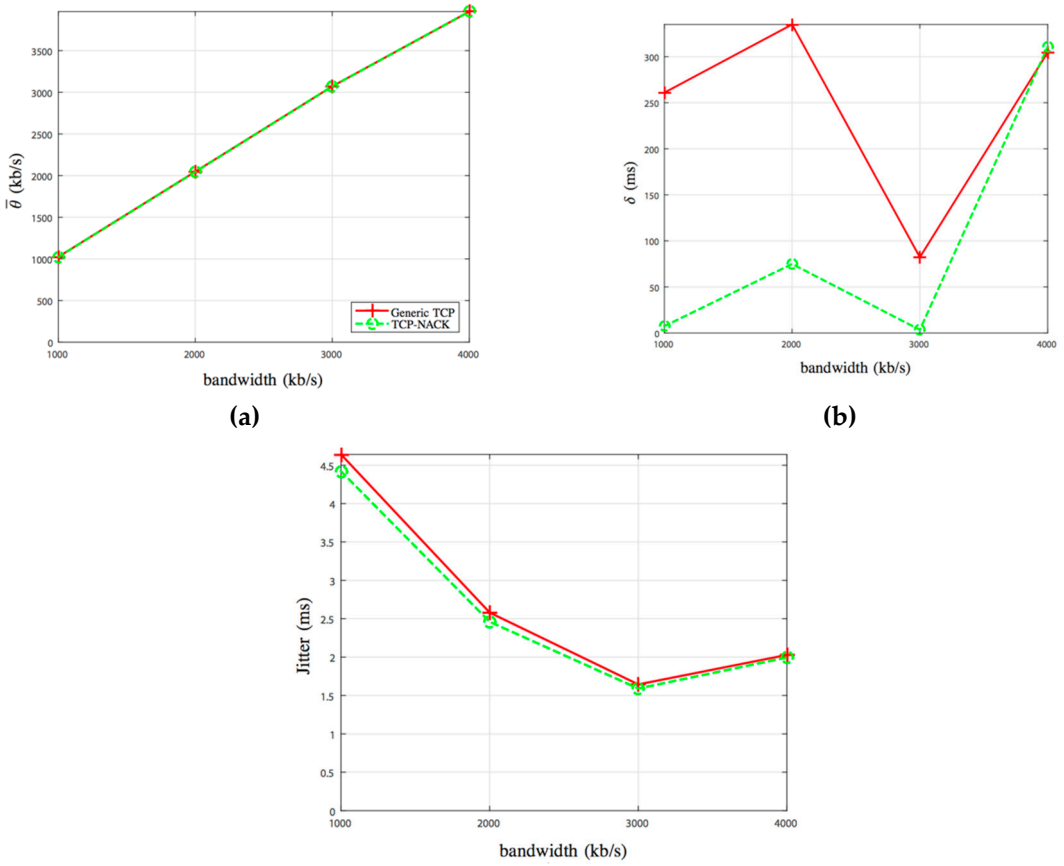

Figure 12. Long distance results: (**a**) $\overline{\theta}$; (**b**) δ; (**c**) jitter.

We determined that this was not improved in terms of throughput and jitter, but in terms of delay, we observed a significantly reduction; this is due to the fact that we considered to modify the ACK timeout value for MAC layer in order to reach long distance [24], but we did not consider changing the main parameters related to ACK timeout at Transport layer as this is out of this study's scope.

5. Conclusion and Future Work

Since the TCP protocol was designed to work in wired networks, its performance is not adequate for wireless networks, where the packets corrupted for interference, obstacles or fading are discarded and taken as a fictional congestion con- sequence. This assumption entails to an unnecessary CWND reduction which decreases the transmission bit rate.

In this paper, we proposed a TCP-NACK protocol that endures wireless performance problems by the incorporation of an error notification or "NACK" to the Generic TCP protocol functionality. This notification indicates to the transmitter the arrival of corrupted packets, proceeding to an immediate re- transmission without the CWND reduction. For its validation, we exposed the mathematical analysis, configurations to the Linux source code, emulation, and the testbed analysis in real scenarios.

The results obtained demonstrate a better performance of TCP-NACK protocol against its generic counterpart (TCP Reno) under different scenarios including different distances, bandwidths, packet error rates and technologies employed. Regardless of the fact that the experimental results obtained are lower than the mathematical results, we achieved an average improvement of 54% in $\overline{\theta}$, 26% in δ, and 5% in jitter when compared to Generic TCP. It is important to highlight that we obtained the best $\overline{\theta}$ results between 100 and 414 m, where, on average, there is an improvement of 90%. The principal differences among our work and those described in the literature are the performance evaluation as a function of the distance, and the experimentation tests over networks with exclusively wireless links.

In summary, TCP-NACK is a good option for wireless networks, and it is perfectly attachable to any personal computer as a modified Linux operating system. This proposal was evaluated under several real conditions and demonstrated high improvements when compared with TCP Reno protocol.

As future works we are interested in considering the communication standard, frequency and the running application (e.g., VoIP) as variables. We are also interested in performing tests with larger distances and bandwidths evenly distributed. We are also planning to include tests with an optimization of the RTT calculation, by applying the mathematical modeling described in [24] with the aim of improving the θ values for wireless links with short and long distances. Finally, we are interested in comparing with other proposals which are available to be implemented in open hardware.

Author Contributions: Conceptualization, G.O. and C.d.A.; methodology, R.L.-C. and G.O.; validation, R.L.-C., G.O. and D.M.; formal analysis, R.L.-C.; investigation, G.O., R.L.-C.; writing—original draft preparation, D.M.; writing—review and editing, R.L.-C.; visualization, G.O.; funding acquisition, R.L.-C. and G.O.; All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Universidad de las Fuerzas Armadas ESPE, grants number 2010-PIT-008 and 2012-PIT-020.

Acknowledgments: The authors would like to thank grateful to Dr. Diego Benítez, Colegio de Ciencias e Ingenierías, Universidad San Francisco de Quito USFQ, for proofreading this document. And last but not least, this work might not have been possible without the efforts of the Grupo de Investigación de Sistemas Inteligentes (WiCOM energy), and Centro de Investigaciones de Redes Ad-Hoc (CIRAD).

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

References

1. Xylomenos, G.; Polyzos, G.C. Internet Protocol Performance over Networks with Wireless links. *IEEE Netw.* **1999**, *13*, 55–63. [CrossRef]
2. Postel, J. RFC 793—Transmission Control Protocol. Available online: https://tools.ietf.org/html/rfc793 (accessed on 4 June 2020).
3. Allman, M.; Paxson, V.; Ethan, B. TCP Congestion Control. Available online: https://goo.gl/CE5m3V (accessed on 4 June 2020).
4. Tian, Y.; Xu, K.; Ansari, N. TCP in wireless environments: Problems and solutions. *IEEE Commun. Mag.* **2005**, *43*, S27–S32. [CrossRef]
5. Zanella, A.; Procissi, G.; Gerla, M.; Sanadidi, M.Y. TCP Westwood: Analytic model and performance evaluation. In Proceedings of the GLOBECOM'01, IEEE Global Telecommunications Conference, San Antonio, TX, USA, 25–29 November 2001; Volume 3, pp. 1703–1707.
6. Ding, W.; Jamalipour, A. A New Explicit Loss Notification with Acknowledgement for Wireless TCP. In Proceedings of the 12th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, San Diego, CA, USA, 30 September–3 October 2001; Volume 1, pp. B-65–B-69.
7. Buchholcz, G.; Ziegler, T.; Do, T.V. TCP-ELN: On the protocol aspects and performance of explicit loss notification for TCP over wireless networks. In Proceedings of the First International Conference on Wireless Internet, Budapest, Hungary, 10–15 July 2005; pp. 1–8.
8. Cui, Y.; Wang, L.; Wang, X.; Wang, Y.; Ren, F.; Xia, S. End-to-end coding for TCP. *IEEE Netw.* **2016**, *30*, 68–73. [CrossRef]
9. Caceres, R.; Iftode, L. Improving the performance of reliable transport protocols in mobile computing environments. *IEEE J. Sel. Areas Commun.* **1995**, *13*, 850–857. [CrossRef]
10. Bakre, A.; Badrinath, B. Implementation and performance evaluation of indirect TCP. *IEEE Trans. Comput.* **1997**, *46*, 260–278. [CrossRef]
11. Parvez, N.; Hossain, E. TCP Prairie: A sender-only TCP modification based on adaptive bandwidth estimation in wired-wireless networks. *Comput. Commun.* **2005**, *28*, 246–256. [CrossRef]
12. Hung, K.L.; Bensaou, B. TCP performance optimization in multi-cell WLANs. *Perform. Eval.* **2011**, *68*, 806–824. [CrossRef]
13. Balakrishnan, H.; Seshan, S.; Amir, E.; Katz, R.H. Improving TCP/IP performance over wireless networks. In Proceedings of the 1st Annual International Conference on Mobile Computing and Networking, Berkeley, CA, USA, 13–15 November 1995; pp. 2–11.

14. Bhandarkar, S.; Sadry, N.E.; Reddy, A.L.; Vaidya, N.H. TCP-DCR: A Novel Protocol for Tolerating Wireless Channel Errors. *IEEE Trans. Mob. Comput.* **2005**, *4*, 517–529. [CrossRef]
15. Han, K.; Lee, J.Y.; Kim, B.C. Machine-Learning based Loss Discrimination Algorithm for Wireless TCP Congestion Control. In Proceedings of the 2019 International Conference on Electronics, Information, and Communication (ICEIC), Auckland, New Zealand, 22–25 January 2019.
16. Olmedo, G. Controle de Congestionamento do Protocolo TCP em Sistemas de Comunicação Sem Fio CDMA Usando Estrategia de Detecção Multiusuario, Arranjo de Antenas e Correção de Erro FEC. Ph.D. Thesis, Universidade Estadual de Campinas, Campinas, Brazil, 2008. Available online: https://goo.gl/r8tWtT (accessed on 4 June 2020).
17. Herbert, T.F. *The Linux TCP/IP Stack: Networking for Embedded Systems*; Charles River Media Ed.: Needham Heights, MA, USA, 2004.
18. Campbell, A.; Coulson, G.; Hutchison, D. A Quality of Service Architecture. *ACM SIGCOMM Comput. Commun. Rev.* **1994**, *24*, 6–27. [CrossRef]
19. Botta, A.; Donato, D.W.; Dainotti, A.; Avallone, S.; Pescapé, A. *D-ITG 2.8.1 Manual*. 2013, pp. 1–35. Available online: https://goo.gl/UZPJtg (accessed on 4 June 2020).
20. Hemminger, S. Network Emulation with NetEm. In Proceedings of the 6th Australia's National Linux Conference (LCA 2005), Canberra, Australia, 18–23 April 2005; pp. 1–9. Available online: https://goo.gl/XGEHBN (accessed on 4 June 2020).
21. Meyers, P. Systems and Methods for Massively Multi-Player Online Role Playing Games. U.S. Patent Application 10/754,069, 22 July 2004. Available online: https://goo.gl/BHydmL (accessed on 4 June 2020).
22. Panchana, M.A. Implementación de un Escenario de Juegos en Red a Través de una Comunicación Inalámbrica con Protocolo TCP para Soportar Aplicaciones en Tiempo Real. Univ. las Fuerzas Armadas ESPE, Sangolquí, Ecuador. 2014. Available online: https://goo.gl/QpzyRN (accessed on 4 June 2020).
23. Lara-Cueva, R.; Olmedo, G.; Calvopina, K. Performance evaluation of a new wireless TCP algorithm in out-door and in-door environments on WiFi and WiMAX links. In Proceedings of the 2015 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies, Santiago, Chile, 28–30 October 2016; pp. 313–318.
24. Lara-Cueva, R.; Olmedo, G.; Acosta, M.; Sandoval, J. Performance evaluation of the new algorithm of the TCP protocol for a long distance wireless link in the Galapagos Islands. In Proceedings of the 2016 IEEE International Engineering Summit, II Cumbre Internacional de las Ingenierias (IE-Summit), Boca del Rio, Mexico, 2–5 March 2016; pp. 1–5.