Performance Evaluation of TCP over WLAN 802.11 with the Snoop Performance Enhancing Proxy

Chi Ho Ng, Jack Chow, and Ljiljana Trajkovic
School of Engineering Science
Simon Fraser University
Vancouver, British Columbia
Canada V5A 1S6
cng@sierrawireless.com, jackchow@comeindustrial.com, ljilja@cs.sfu.ca

Abstract
The growing popularity of wireless devices used to access the Internet and an increasing use of the TCP/IP protocol suites, indicate that in the near future TCP protocol will be frequently used over the wireless links connecting wireless devices. The characteristics of wireless links are significantly different from characteristics of wired network links because data is frequently lost due to the volatile environment in which wireless links operate. TCP was originally designed for wired networks, where loss of data is assumed to be due to congestion only. This assumption leads to poor performance of TCP in a wireless environment. Hence, various mechanisms were proposed to improve TCP performance over wireless links. One such mechanism is a performance enhancing proxy, such as the Snoop protocol.

In this paper, we study the effect of the Snoop protocol on the performance of TCP over wireless links. We implemented and simulated Snoop protocol in OPNET wireless LAN (WLAN) devices. We measured the performance improvement by comparing the performance of an FTP session with and without the Snoop protocol. We found that the Snoop protocol significantly improves the performance of TCP (~ 68 times under 30% packet error rate). We show that this improvement is achieved by preventing the TCP layer from reducing the congestion window size.

1 Introduction
Transport Control Protocol (TCP) [1, 2] is a reliable protocol designed to perform well in networks with low bit-error rates, such as wired networks [3]. TCP assumes that all errors are due to network congestion, rather than to loss. When congestion is encountered, TCP adjusts its window size and retransmits the lost packets. In wireless networks, however, packet loss is mainly caused by high bit-error rates over air-links. Thus, the TCP window adjustment and retransmission mechanisms result in poor end-to-end performance.

A number of approaches have been proposed to improve the TCP efficiency in an unreliable wireless network. One of these approaches is the Snoop protocol [4, 5]. The protocol modifies network-layer software, mainly at the base station, and preserves end-to-end TCP semantics. Its main feature is to cache packets at the base station and to perform local retransmissions across the wireless links.

This paper is organized as follows. In Section 2, we present an overview of TCP and the Snoop protocol. Section 3 describes the OPNET implementation of the Snoop protocol. In Section 4, we provide simulation results and analyze the performance improvements. We present our conclusions in Section 5.

2 Overview of the Transport Control Protocol
TCP is a reliable transport protocol that performs efficiently in wire-line networks. It uses a Go-back-N protocol and a timer-based retransmission mechanism. The timer period (the timeout interval) is calculated based on the estimated round-trip delay. Packets whose acknowledgements are not received before the timer expires are retransmitted. In the presence of frequent retransmissions, TCP assumes that there is a congestion and invokes its congestion control algorithm. The algorithm reduces the transmission (also called congestion) window size. As the window size is reduced, the transmission rate is also reduced. This window size adjustment technique prevents the source from overwhelming the network with an excessive number of packets.

In the presence of high bit error rates in wireless links, TCP reacts the same way as in a wired link: it reduces the window size before packet retransmission. This adjustment results in an unnecessary reduction of the bandwidth utilization causing significant performance degradation (poor throughput and long delays). The bandwidth of the wired-link segments is especially under-utilized because the high error rates occur only on the wireless sections. Nevertheless, the window reduction affects all transmission links.

2.1 Performance Enhancing Proxies
Performance Enhancing Proxies (PEPs) are methods employed to reduce performance degradation due to distinct link characteristics [4]. The Snoop protocol is one such method [5]. Its major goal is to improve the performance of communication over wireless links without triggering retransmission and window reduction polices at the transport layer.

2.2 Snoop Protocol
The Snoop protocol runs on a Snoop agent that is implemented in a base station or a wireless device. The agent monitors packets that pass through the base station and caches the packets in a table. After caching, the agent forwards packets to their destinations and monitors the corresponding acknowledgements.

In TCP, a sequence number is associated with each acknowledgement (Ack). This number informs the sender of the index of the last byte successfully delivered to the receiver. If the sender receives the same acknowledgement sequence number more than once, this suggests that data sent since the last

received byte reported in the acknowledgement sequence number is lost. An Ack that contains a sequence number that is smaller than the sequence number of the last received Ack is called a duplicate Ack.

Consider a scenario where a TCP sender sends packets 1, 2, 3, 4, and 5 that are forwarded by a base station. Assume that packet 2 is lost due to an error on the wireless link. The Acks received for packets 3, 4, and 5 are thus considered duplicate Acks. When the sender receives these three duplicate Acks, it retransmits the packets and reduces the size of the congestion window, as shown in Figure 1.

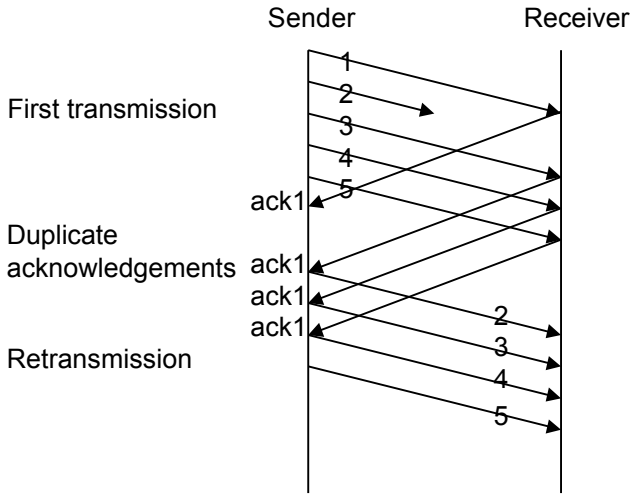

Figure 1: TCP retransmission mechanism.

The role of the Snoop agent is to cache data packets for a TCP connection. When data packets are lost (indicated by the reception of duplicate Acks), the Snoop agent retransmits those packets. It retransmits them locally without forwarding the ACKs to the sender. Hence, since the TCP layer is not aware of the packet loss, the congestion control algorithm is not triggered. In addition, the Snoop agent starts a retransmission timer for each TCP connection. When the retransmission timer expires, the agent retransmits the packets that have not been acknowledged yet. This timer is called a *persist* timer because, unlike TCP retransmission timer, it has a fixed value. Figure 2 illustrates a Snoop agent implemented in a base station.

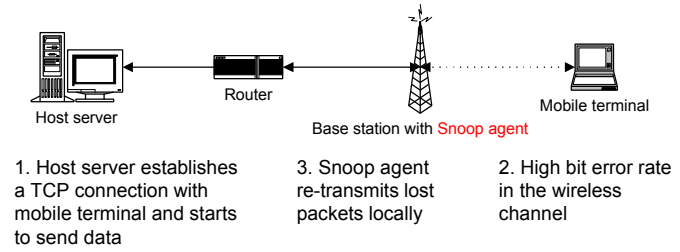

Figure 2: Snoop agent at a base station.

Snoop protocol intercepts TCP packets, analyzes them, and retransmits them if necessary. As a result, no additional packet formats are introduced into the protocol, and all packets sent and received still conform to the TCP protocol. Moreover, no changes to the other layers in the TCP/IP protocol stack are necessary. This is rather important because the TCP/IP protocol is designed for wire-link network where most network servers are located.

The goal of this project is to implement the Snoop agent in OPNET WLAN device. We expected that the implemented Snoop agent will improve the TCP performance and that it should not require modifying the default parameters.

3 OPNET WLAN Model

We implemented the Snoop protocol in the OPNET WLAN model. The wireless workstations modeling the medium access control (MAC) and the physical layer are comprised of the wirless_lan_mac process, transmitter, receiver, and the channel streams. The address resolution protocol (ARP) is an interface between the MAC and the upper layers. The ARP translates IP addresses into network interface addresses. This project focuses on the improvements of TCP performance over the wireless links and, hence, only the wireless workstation and the wireless server models (no wireless routers) are employed.

3.1 Model Modifications

The OPNET WLAN model does not allow the user to inject packet loss or bit error rate into the WLAN network. To facilitate testing of the Snoop protocol, we implemented an additional layer, called the Packet Error Generator (PEG), between the ARP and the IP layer. The PEG layer allows the user to specify the packet error rate generated according to a uniform distribution function. The Snoop agent is added on top of the PEG layer. The diagram of the modified WLAN layers is shown in Figure 3.

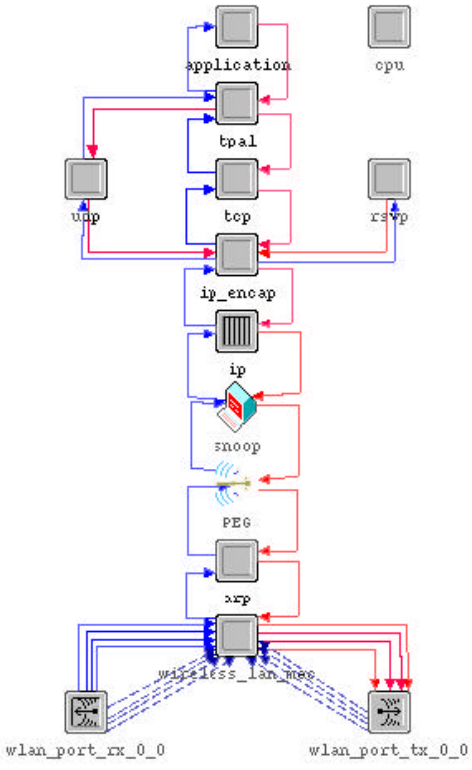

Figure 3: The modified WLAN device model.

The location of the additional process models is chosen to avoid modifications to both the upper and lower layer protocols. It only requires changes to the ARP model. The ARP model is modified to send and receive packets to and from the Snoop agent.

3.2 Packet Error Generator (PEG)

This process model simulates packet loss by dropping received packets. The number of packets to be dropped (as a percentage of the total packets received) can be exported as a model attribute.

3.3 PEG Process Model

Figure 4 shows the state diagram of the PEG process model. The Init state initializes the various global variables when the process is first instantiated. The process remains in the Wait state until a packet arrives either from the upper or the lower layer. Depending on the direction of the packet, the process enters the Send_Data or Recv_Data state. The instructions implemented in the Send_Data and the Recv_Data states are used to decide whether or not the current packet should be discarded.

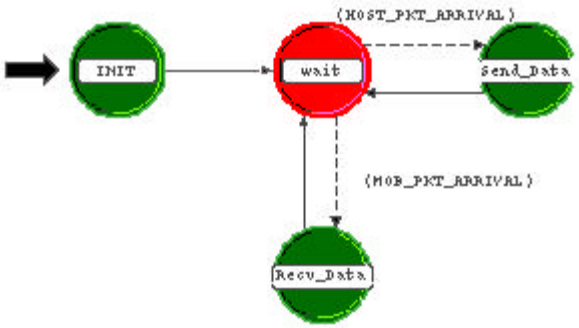

Figure 4: PEG process model.

3.4 Algorithm and its Implementation

The following pseudo-code is implemented in the Send_Data and Recv_Data states:

```
random_val = op_dist_uniform (100.0)
get TCP info of the packet
if (packet is a SYN or FIN packet)
{
  pass packets to the next layer
}
else if (random_val > packet_drop_rate)
{
  pass packets to the next layer
}
else
{
   destroy packet
}
```

The PEG model employs a uniform distribution variable to determine if a certain packet should be dropped. The uniform distribution implies that all packets have the same probability of being discarded. Thus, this model depicts scenarios where packet loss occurs randomly due to the characteristics of the environment. Figure 5 shows a typical packet loss.

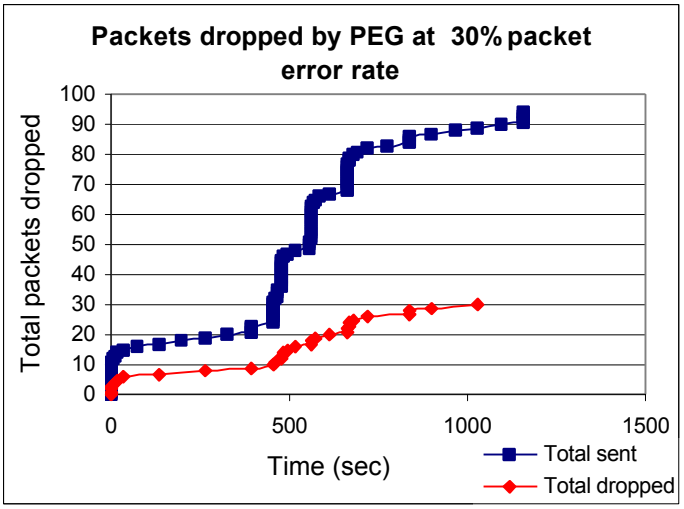

Figure 5: Packet loss at 30% packet error rate.

3.5 Implementation of the Snoop Protocol

In this section we describe the implementation of the Snoop agent as a process model. Figure 6 shows the five states and the state transitions in the Snoop agent process model.

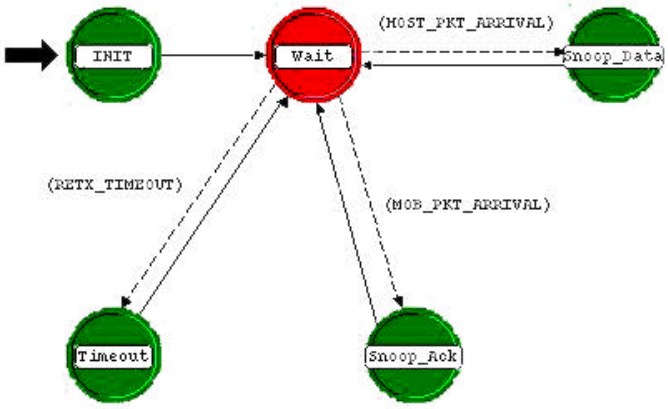

Figure 6: Snoop agent process model.

Snoop Cache: The Snoop cache is used to cache the packets that are received from upper layers. Packets are cached in a linear array. The C code definition of the Snoop cache is:

```
typedef struct
{
  /* Packet cache */
  Packet       cached_pkp[SNOOP_CACHE_SIZE];
  /* ICI content cache */
  int          ici_addr[SNOOP_CACHE_SIZE];
  /* Max number of packets that have been cached at
the same time */
  int          max_cached;
  /* Number of packets that have been cached */
  int          num_cached;
  /* Number of packets that have been removed */
  int          num_removed;
  /* Current number of packets cached */
  int          curr_cached;
} struct_snCache;
```

The cache table contains copies of packets and the Interface Control Information (ICI) content, as well as various counters such as the total number of packets that have been cached.

Snoop Connection Table:The Snoop agent maintains a table to keep track of the TCP connections. This table is required because the two connection ends may have multiple connections established, and each of these connections needs to be maintained separately (because sequence numbers are not unique among the distinct connections). The record format in the Snoop connection table is:

```
typedef struct
{
   /* Source IP */
   unsigned int    src_ip;
   /* Destination IP */
   unsigned int    dest_ip;
   /* Source Port */
   int             src_port;
   /* Destination Port */
   int             dest_port;
   /* Last sent seq number */
   unsigned int    last_seq_num;
   /* Last received seq number */
   unsigned int    last_ack_num;
   /* Determines if we have received repeat Acks */
   int             repeat_ack;
   /* Determines if we have received FIN packet */
   int             fin_flag;
   /* Seq num of the fin packet */
   unsigned        fin_seq_num;
   /* Timeout event handle */
   Evhandle   timeout_evt;
} struct_snTable;
```

Each record identifies a TCP connection by its source and destination IP address and the TCP port pairs. It also keeps track of the last sequence number and the last acknowledgement number recorded for the connection. The timeout_evt is used for the retransmission timer.

Init state: The Snoop process begins in the Init state where its cache table and the TCP connection table are initialized.

Wait state: After the initialization tasks have been performed, the process transits from the Init state to the Wait state. The process remains in the Wait state until:

- A packet arrives from the MAC layer
- A packet arrives from the upper TCP/IP layer
- The retransmission timer expires.

All events arrive as interrupts. The packet arrival is represented as a stream interrupt. The retransmission timer expiration event is implemented as a self-scheduled interrupt.

Snoop_Data state: When a packet from the upper TCP/IP layer arrives, the process transits from the Wait state to the Snoop_Data state. The Snoop agent keeps track of the last sequence number seen from the upper layer. The packet is processed depending on its sequence number:

- *A new packet in the normal TCP sequence*: This is the normal case when a new packet with a higher sequence number arrives. The packet is added to the Snoop cache and is forwarded to the lower layer.
- *An out-of-sequence packet that has been cached earlier*: This occurs when lost packets cause timeouts at the sender. If the sequence number is higher than the last acknowledgement seen by Snoop agent, it is assumed that the packet is lost. The packet is therefore forwarded to the destination. If the sequence number is lower than the last acknowledgement, the packet is discarded.

After processing the packet, the Snoop process transits back to the Wait state and awaits the next event. Figure 7 shows the flow chart of the algorithm implementation.

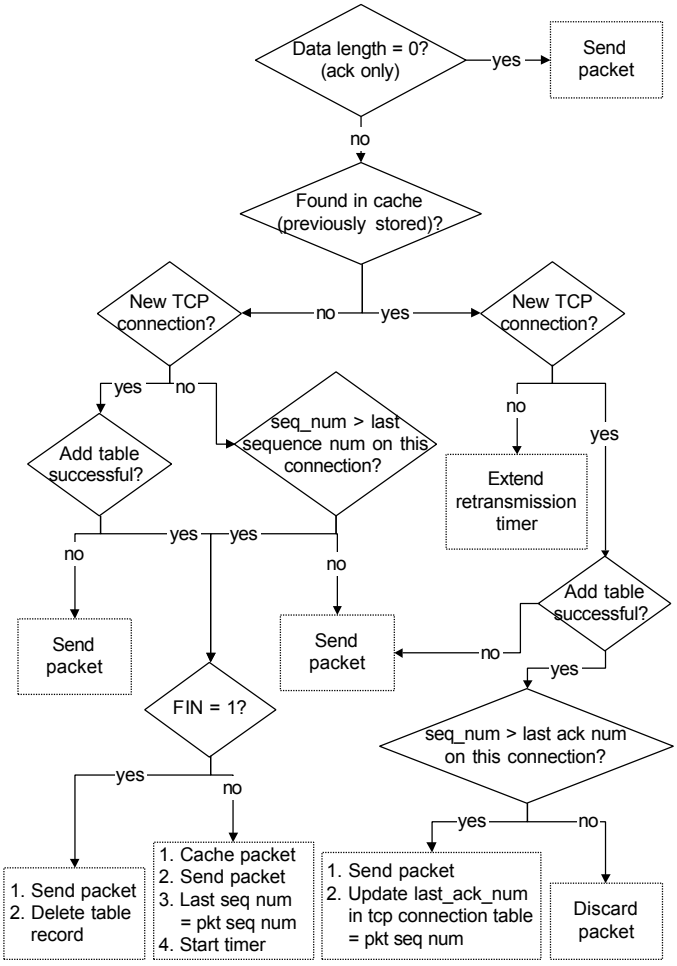

Figure 7: Snoop_Data implementation flow chart.

In addition to implementing the Snoop algorithm, the OPNET implementation also cleans the connection table entry when the last packet (FIN) of the connection is detected. In addition, the test shown at the top of the flow chart is used to ensure that the Snoop agent does not mistakenly caches packets that contain data relevant to the opposite direction (e.g., Acks packets for acknowledged data packets in the opposite direction).

Snoop_Ack state: When a packet from the MAC layer arrives to the Snoop agent, the process transits from the Wait state to the Snoop_Ack state. The packet is processed depending on the acknowledgement sequence number:

- *A new acknowledgement*: This is an acknowledgement packet with an acknowledgement sequence number higher than

the last received. This initiates the cleaning of the corresponding cache entries. The acknowledgement is passed to the upper TCP/IP layers.

- *A false acknowledgement*: This is an acknowledgement packet with an acknowledgement sequence number lower than the last received. This rarely happens and the acknowledgement is discarded.
- *A duplicate acknowledgement*: This is an acknowledgment packet that is identical to the last received. This causes the Snoop agent to assume that packets that have been sent with higher sequence number were lost. The Snoop agent will retransmit all packets starting with the first lost packet.

Figure 8 shows the flow chart of the steps executed by the Snoop_Ack state.

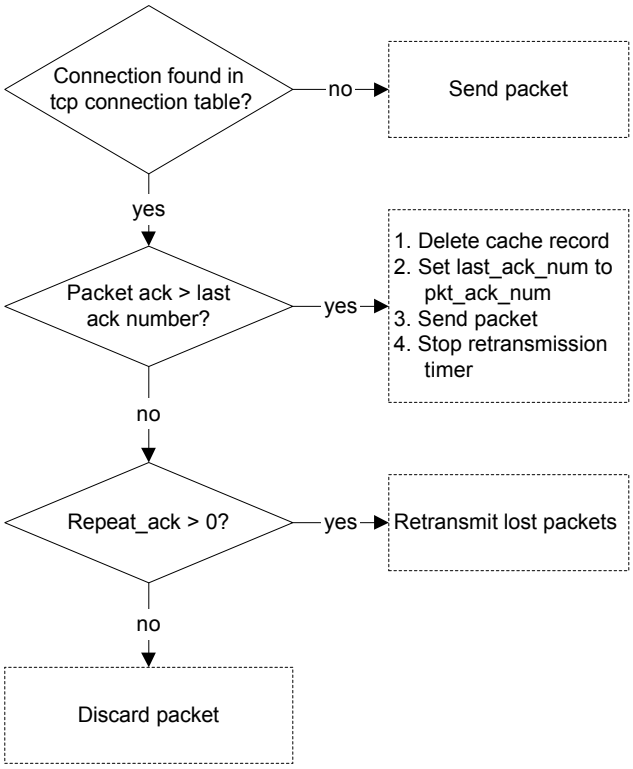

Figure 8: Snoop_Ack implementation flow chart.

Timeout state: When the retransmission timer of a connection expires, the process transits from the Wait state to the Timeout state. Processing during the Timeout state is very similar to the processing of receiving a duplicate acknowledgement in the Snoop_Ack state: all packets that have not been acknowledged are retransmitted.

The retransmission timer of a connection is extended when a new data packet is received from the upper layer, a new acknowledgement packet is received from the lower layer, or a retransmission timer has just expired.

4 Performance Comparison

This section describes how are the models, described in Section 3, used to measure the protocol performance.

Setup: We simulated three scenarios to validate the models, to measure the performance of a WLAN device with Snoop protocol, and to compare it to a device with unmodified TCP.

Devices: Each scenario consists of two main types of WLAN nodes: mobile workstations and servers. The Snoop protocol model is implemented in both devices and it is enabled or disabled depending on the various scenarios. Device with Snoop protocol are called enhanced devices, while devices without Snoop are called the original devices. The remaining nodes that are used in the scenarios are Application Configuration and the Profile Configuration nodes. They enable the devices to generate traffic using the OPNET Application Model paradigm.

TCP Model Parameters: TCP parameters used for the TCP models in WLAN nodes are listed in Table 1.

TCP Parameters	Values
Maximum Segment Size (bytes)	2,264
Receiver Buffer Size (bytes)	8,760
Receiver Buffer Usage Threshold	0.0
Delayed Ack Mechansim	Segment/Clock Based
Maximum Ack Delay	0.2
Fast Transmit	Enabled
Fast Recovery	Enabled
Selective Ack (SACK)	Disabled
Nagle SWS Algorithm	Disabled
Karn's Algorithm	Enabled
Retransmission Threshold	Attempt Based
Initial RTO	1.0
Minimum RTO	0.5
Maximum RTO	64
RTT Gain	0.125
Deviation Gain	0.25
RTT Deviation Coefficient	4.0
Timer Granularity	0.5
Persist Timeout	1.0

Table 1: TCP model parameters.

Most values are the default OPNET values and they do not affect the results of our simulation experiments. They are chosen because the Snoop protocol is used as a PEP, and TCP parameters should be left unaltered on wireless devices so that they can be tuned to handle congestion cases.

Parameters that directly affect the experiment results are the Maximum Segment Size and the Receiver Buffer Size. Choosing smaller values for both parameters will causes a larger number of TCP messages. Because the Snoop protocol is most effective when more packets are lost, it is expected that choosing smaller values for these parameters will enhance the Snoop performance.

Traffic: All simulation scenarios use the OPNET Application Model to generate traffic. In order to study the effects of the Snoop protocol, we used a simple traffic pattern: FTP transfers of 100,000-byte files. Figure 9 illustrates a typical message sequence of an FTP file transfer. During the data transfer, each message contains 2,264 bytes of data. This value corresponds to the WLAN TCP window size.

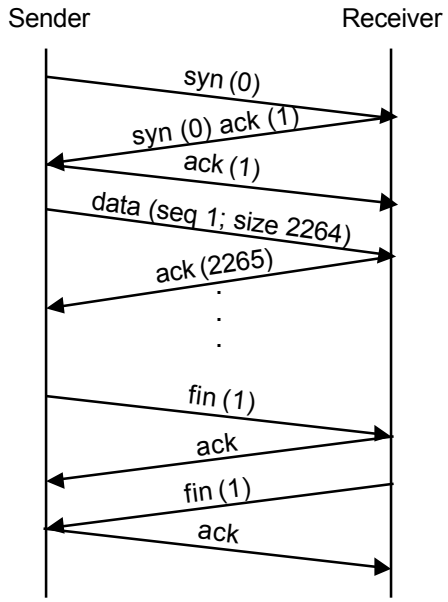

Figure 9: FTP transfers message sequence.

Wireless LAN Model Parameters: In our simulation scenarios, we use the default values of the WLAN parameters. The goal of the simulation is to evaluate the performance of an enhanced device and compare it to the performance of an original device. Therefore, it is important that all devices use identical WLAN parameters. Their values are listed in Table 2.

Wireless LAN Parameters	Values
Rts Threshold	None
Fragmentation Threshold	None
Data Rate	1 Mbps
Physical Characteristics	Frequency Hopping
Buffer Size (bits)	256,000
Maximum Receive Lifetime (sec)	0.5

Table 2: Wireless LAN parameters.

The use of the PEG process model embedded inside the device protocol stack minimizes the effects of the WLAN parameters on the simulation results. Therefore, most parameters were set to the default OPNET values.

4.1 Simulation Scenario 1: Single Mobile Upload

This scenario is designed to verify the basic operation of the Snoop protocol and to illustrate the improved network performance with the Snoop protocol. In this scenario, the mobile performs an FTP upload of 100,000 bytes to the server. The simulation settings are listed in Table 3.

In this scenario, shown in Figure 9, the mobile is the host and the server is the receiver. The send_data_drop_rate and the recv_data_drop_rate are identical. This simulates the case where both TCP data and TCP acknowledgement messages may be lost. The values of the send_data_drop_rate and the recv_data_drop rate are called the packet error rate (PER). The value of the Snoop protocol retransmission timeout is set to the initial retransmission timeout (RTO) value of the TCP model.

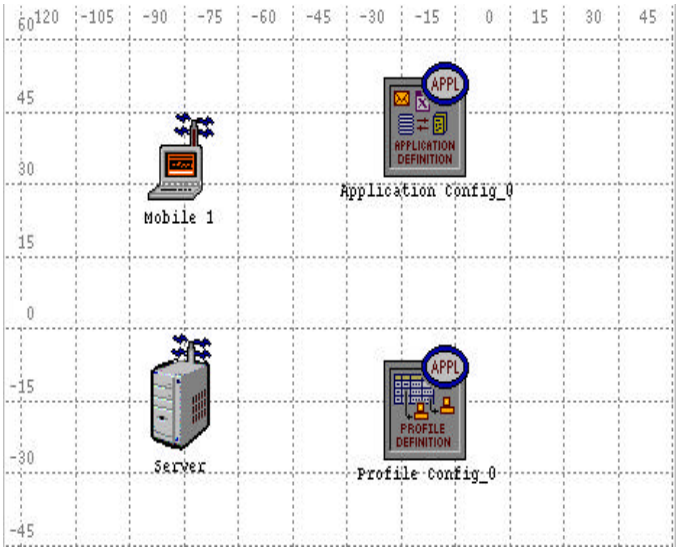

Figure 10: Network model used in Scenario 1.

Parameters Setup	Values
Snoop Model Attribute in Mobile 1	
Snoop Enabled	1 (Enabled)
Snoop Retransmission Timeout	1 sec
PEG Model Attribute in Mobile 1	
Send_Data_Drop_Rate	Varied
Recv_Data_Drop_Rate	Varied
Snoop Model Attribute in Server	
Snoop Enabled	0 (Disabled)
PEG Model Attribute in Server	
Send_Data_Drop_Rate	0
Recv_Data_Drop_Rate	0

Table 3: Simulation parameters used in Scenario 1.

4.1.1 Results and Observations

Figure 11 shows the upload response time of the 100,000-byte file transfer. The response time is measured from the first SYN packet sent from Mobile 1 to the last acknowledgement packet received for the last FIN packet.

Simulation results indicate that the performance improvement of Snoop protocol increases as packet error rate increases. At a packet loss rate of 30%, the Snoop protocol improves the TCP performance approximately 68 times. This large improvement is due to the combination of two effects: packet caching and local retransmission timeout.

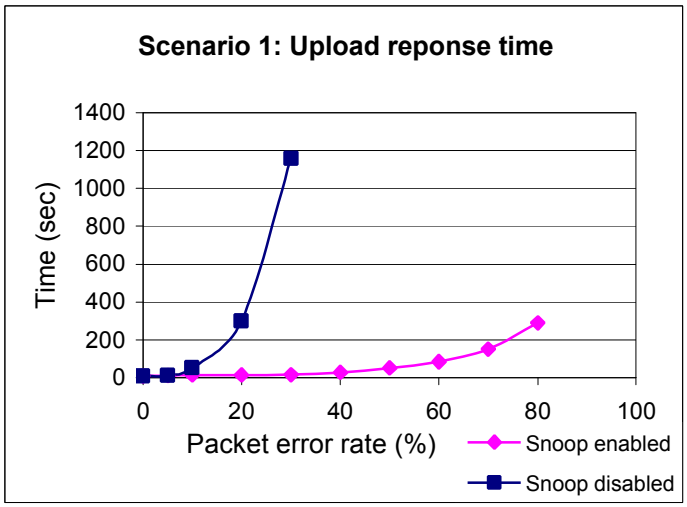

Figure 11: Upload response time of a 100,000-byte file.

4.1.2 Caching Packets

The Snoop protocol caches packets and retransmit them on behalf of TCP once it detects packet loss due to the received duplicate acknowledgement sequence.

Figure 12 shows the number of packets cached during the file upload. The number of cached packets increases as the mobile packets are transmitted to the server without receiving acknowledgement packets. A maximum of four packets can be cached because the mobile can transmit up to four data messages before filling up the receive buffer. (The receiver window buffer size is 8,760 and the maximum segment size is 2,264.) The number of packets in the cache decreases as the Snoop protocol receives the acknowledgements of those packets. The number of cached packets is zero when the connection ends. This verifies that all packets are cleared at the end of the connection.

Figure 13 shows the size of the congestion window during file upload at 20% packet error rate. TCP shrinks its congestion window when it detects that packets are lost because TCP assumes packet loss occurs due to congestion in the network. While this is often true for wired line networks, it is not true for wireless networks where packets loss is mainly caused by bit errors resulting from environmental effects (e.g., background noise, Doppler effect, multi-path delays).

Packet caching prevents TCP congestion window from shrinking. It is the most critical factor in improving the performance of data transfers. Shrinking the congestion windows has a "cyclic" effect. It causes TCP to send less data in each packet, which implies sending more data packets across the network. Since sending more data packets increases the chances of packets getting corrupted, this, in turn, will shrink the congestion window further, and the cycle repeats.

Figure 14 shows the sequence number (recorded at the TCP layer) sent across the network during the file transfer. The sequence number ends at 100,001 because the sequence number starts from 0, and is incremented for every byte of data sent. In addition, there is one extra byte for the SYN packet and one for the FIN packet. It is evident that without the Snoop protocol, the TCP layer has to retransmit the same sequence number (and, hence, same data bytes) several times.

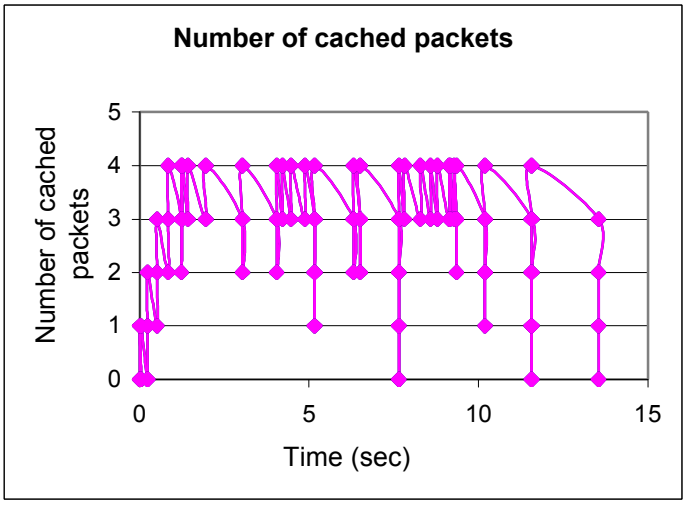

Figure 12: Number of cache packets during file upload.

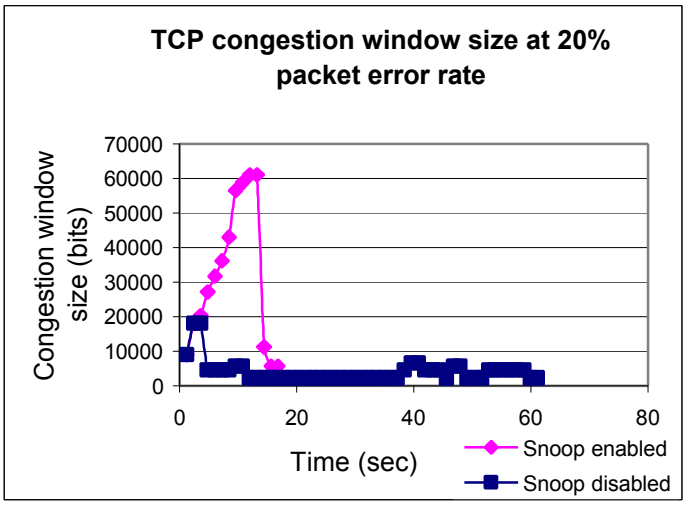

Figure 13: Congestion window of the TCP connection at 20 % packet error rate.

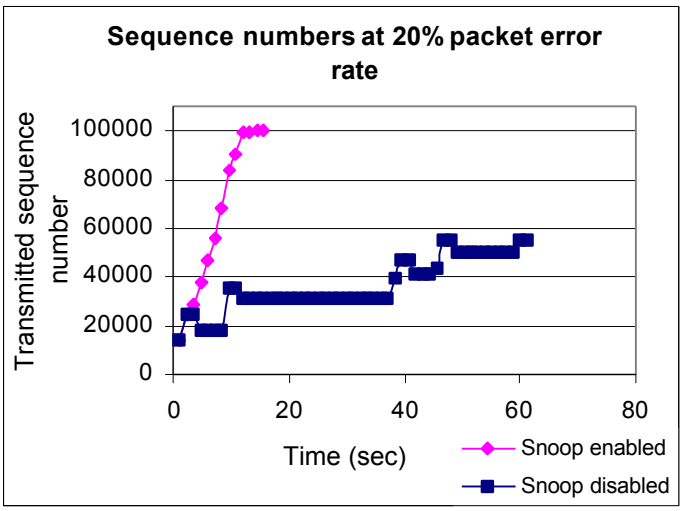

Figure 14: Transmitted sequence numbers.

4.1.3 Local Retransmission Timeout

The retransmission timeout mechanism in TCP is designed to handle network congestion. Therefore, the TCP timeout values are modified according to the level of network congestion that is inferred by the amount of packet loss. Similar to the congestion window size, this assumption is often false in the case of wireless networks. The Snoop protocol uses persist retransmission timeout period of 1 sec. It increases the performance by using a more appropriate timeout period for the wireless link, and prevents TCP from changing its own retransmission timeout period due to packet loss. The timeout period can be tuned to the wireless link characteristics by employing algorithms such as Karn's algorithm for updating the period over time. The important aspect, however, is that the Snoop protocol allows a retransmission timeout value to be adjusted for the wireless links separately.

Figure 15 shows the changes of the retransmission timeout period recorded at the TCP layer. As a result of caching packets, the Snoop protocol prevents TCP from incorrectly adjusting the retransmission timeout due to packet loss.

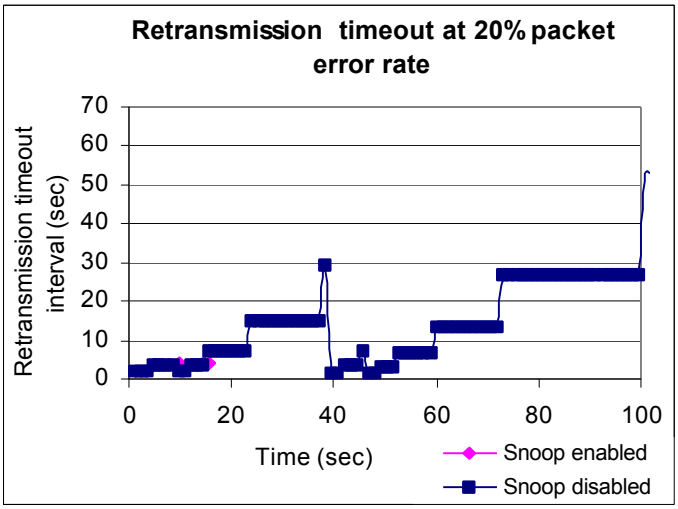

Figure 15: Retransmission timeout of TCP models for file upload at 20% packet error rate.

4.2 Simulation Scenario 2: Multiple Mobile Downloads

This scenario is designed to verify that the Snoop protocol can handle multiple connections simultaneously. In this scenario, the Snoop protocol in the server is enabled. The OPNET Application Model is configured so that each mobile downloads a 100,000-byte file from the server. Figure 16 shows the network model for this scenario. The simulation settings are listed in Table 4.

4.2.1 Results and Observations

Figure 17 shows the average download time for the two mobiles. Similar to Scenario 1, the Snoop protocol improves the performance of the 100,000-byte file transfer. The fact that the mobiles can download simultaneously verifies that the Snoop protocol can handle multiple connections. The decrease in response time of the original mobile for packet error rates between 15% and 20% is most likely caused by the random packet loss generated by the PEG process model.

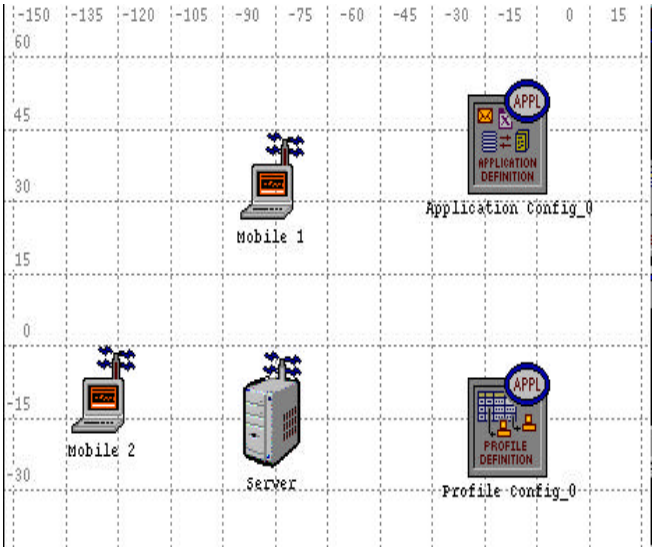

Figure 16: Network model used in Scenario 2.

Parameters Setup	Values
Mobile 1	
Snoop Enabled	0 (Disabled)
PEG Model Attribute in Mobile 1	
Send_Data_Drop_Rate	0
Recv_Data_Drop_Rate	0
Mobile 2	
Snoop Enabled	0 (Disabled)
PEG Model Attribute in Mobile 2	
Send_Data_Drop_Rate	0
Recv_Data_Drop_Rate	0
Server	
Snoop Enabled	1 (Enabled)
Snoop Retransmission Timeout (sec)	1
PEG Model Attribute in Server	
Send_Data_Drop_Rate	Varied
Recv_Data_Drop_Rate	Varied

Table 4: Simulation parameters used in Scenario 2.

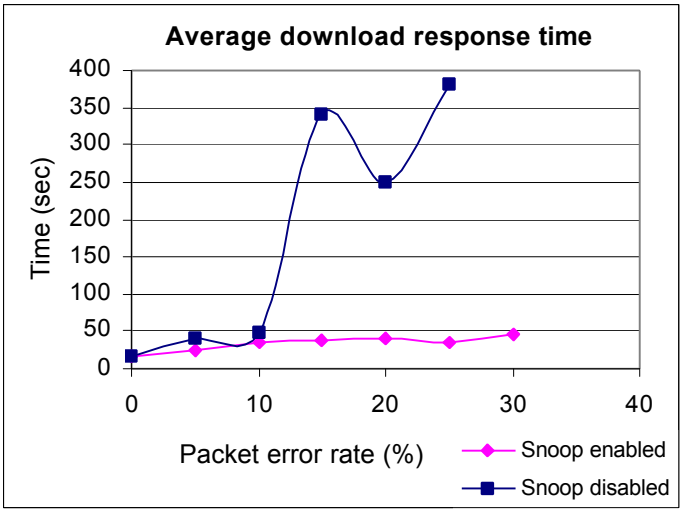

Figure 17: Average download response time for a 100,000-byte file.

It can be noticed that the average download time of the two mobiles is larger than the upload time shown in Scenario 1. For packet error rate of 0%, Scenario 1 requires 9.4 sec, while Scenario 2 requires 17 sec. The reason is that the two mobiles are competing resources on the radio link. This effect will be shown again in Scenario 3.

4.3 Scenario 3: Multiple Mobile Uploads

This scenario is designed to illustrate the effect of coexistence of an enhanced and an original mobile in the same network. Mobile 1 is configured with Snoop protocol disabled, and Mobile 2 is configured with Snoop enabled. The Application Model is configured to have both mobiles upload a 100,000-byte file to the server. The network model is shown in Figure 18. The simulation settings are listed in Table 5.

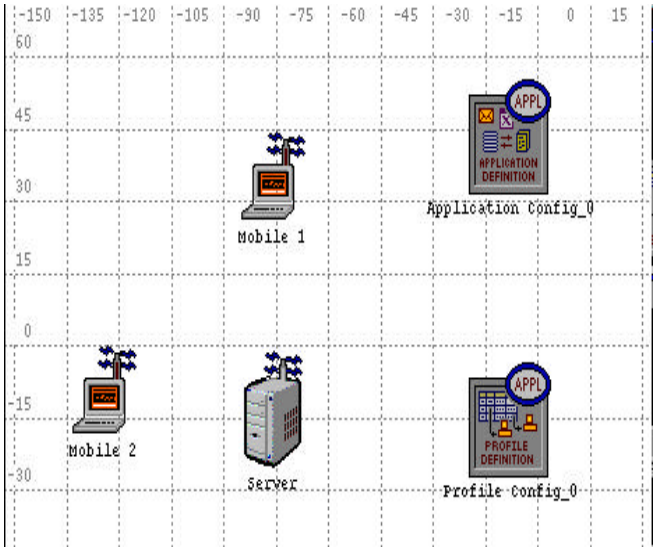

Figure 18: Network model used in Scenario 3.

Parameters Setup	Values
Mobile 1	
Snoop Enabled	0 (Disabled)
PEG Model Attribute in Mobile 1	
Send_Data_Drop_Rate	0
Recv_Data_Drop_Rate	0
Mobile 2	
Snoop Enabled	1 (Enabled)
Snoop Retransmission Timeout (sec)	1.0
PEG Model Attribute in Mobile 2	
Send_Data_Drop_Rate	0
Recv_Data_Drop_Rate	0
Server	
Snoop Enabled	0 (Disabled)
PEG Model Attribute in Server	
Send_Data_Drop_Rate	Varied
Recv_Data_Drop_Rate	Varied

Table 5: Simulation parameters used in Scenario 3.

4.3.1 Results and Observations

The upload response time of the two mobiles are shown in Figure 19. As expected, the mobile using the Snoop protocol outperforms the mobile without it. It is interesting to note that the time required for uploading a 100,000-byte file in Scenario 3 for Mobile 2 (Snoop protocol disabled) is larger than that required in Scenario 2. This suggests that the Mobile 1 (Snoop enabled) takes a larger portion of the available bandwidth (i.e., it retransmits more often) and, hence, reduces the bandwidth that can be allocated to Mobile 2.

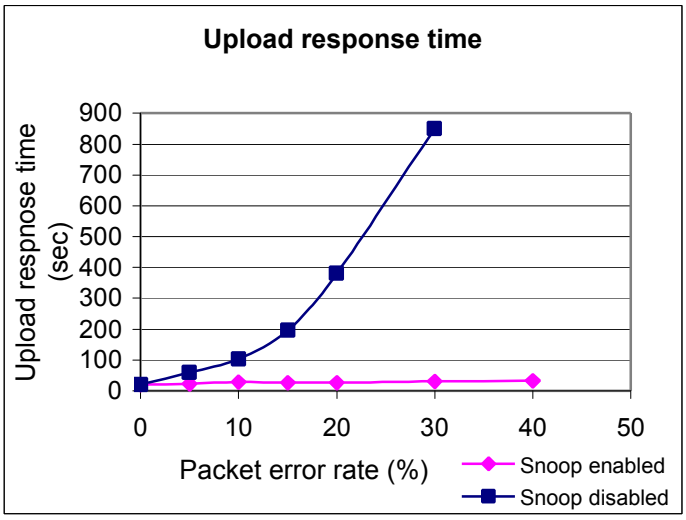

Figure 19: Upload response time of the two mobiles used in Scenario 3.

5. Conclusions

Our simulation results indicate that the Snoop protocol is implemented correctly (it can support multiple connection simultaneously) and can significantly increase the performance of TCP transfers (~ 68 times at 30% packet error rate) without modifying other layers in the protocol stack. The Snoop protocol significantly improves the TCP performance by preventing the transport layer from reducing the congestion window size.

Our results also show that devices with the Snoop protocol (enhanced devices) can co-exist with those without it (original devices). However, when transferring data simultaneously, the performance of original devices is affected because the enhanced devices are more active in competing for resources (they retransmit more often).

Improvements to the current implementation may include a mechanism to calculate the retransmission timer value based on the round-trip delay estimated from the time of sending a data packet to the time the acknowledgement is received. Currently, the retransmission timer value is set to a fixed value.

References

[1] W. R. Stevens, *TCP/IP Illustrated,* Volume 1. Reading, MA: Addison Wesley, Professional Computing Series, 1984.

[2] A. S. Tanenbaum, *Computer Networks*, Third Edition. Englewood Cliffs, NJ: Prentice-Hall Press, 1996.

[3] IEEE 802.11 Workgroup: http://grouper.ieee.org/groups/802/11/index.html (last accessed in August 2002).

[4] Performance Enhancing Proxy (PEP) Request for Comments: http://community.roxen.com/developers/idocs/drafts/draft-ietf-pilc-pep-04.html (last accessed in August 2002).

[5] Improving TCP/IP Performance over Wireless Networks: http://www2.cs.cmu.edu/~srini/Papers/publications/1995.mobicom/mobicom95.pdf (last accessed in August 2002).