

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

Case study

Chi-ho Ng, Jack Chow, and Ljiljana Trajković

Simon Fraser University

Roadmap

- Introducing the problem and the project scope
- TCP retransmission and window size
- Wireless LAN using TCP
- Snoop protocol
- OPNET implementation
- Conclusions

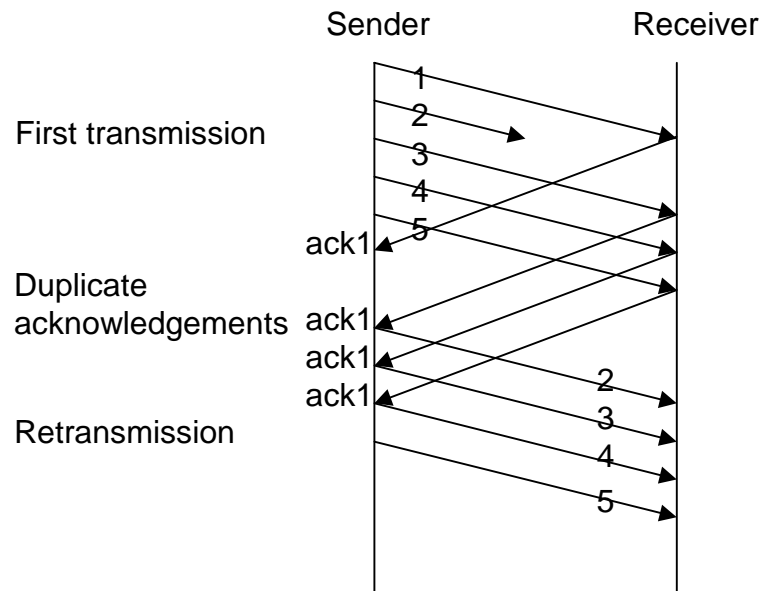
Problem formulation

- TCP is a reliable protocol with packet retransmission and congestion control (transmission window adjustment)
- Loss packets are seen as network congestion: packets are retransmitted using a smaller window size
- This scheme works well in wired networks
- In wireless networks, with the high bit error rate, TCP reduces window size excessively and under-utilizes the available bandwidth

Project goals

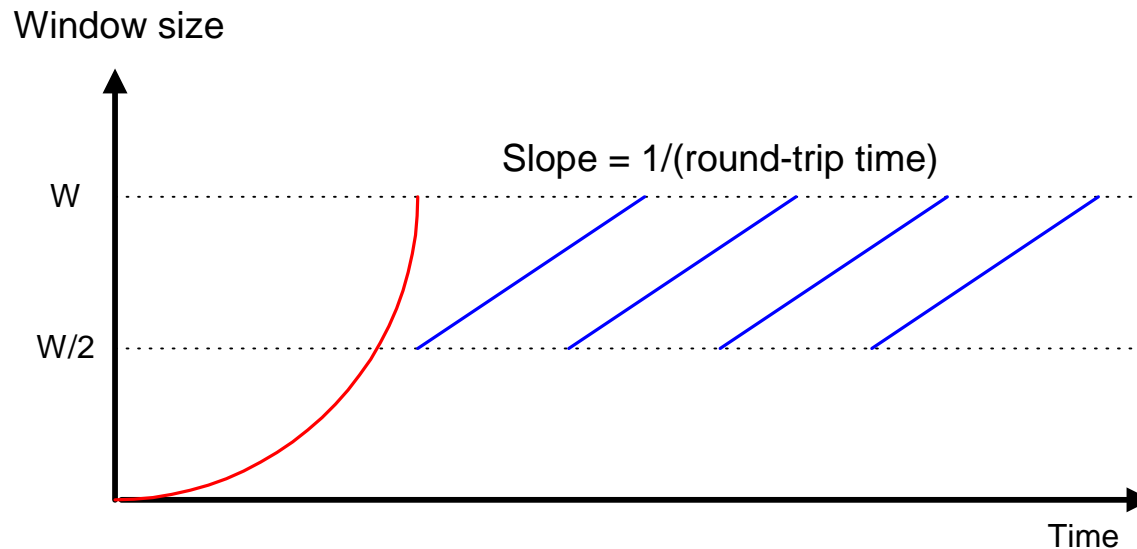
- Investigate the TCP congestion control policy and its performance in wireless LAN networks
- Find possible enhancing algorithms
- Investigate the **Snoop** protocol
- **Implement** Snoop protocol in OPNET
- Compare TCP performance **with** and without **Snoop**

TCP retransmission policy



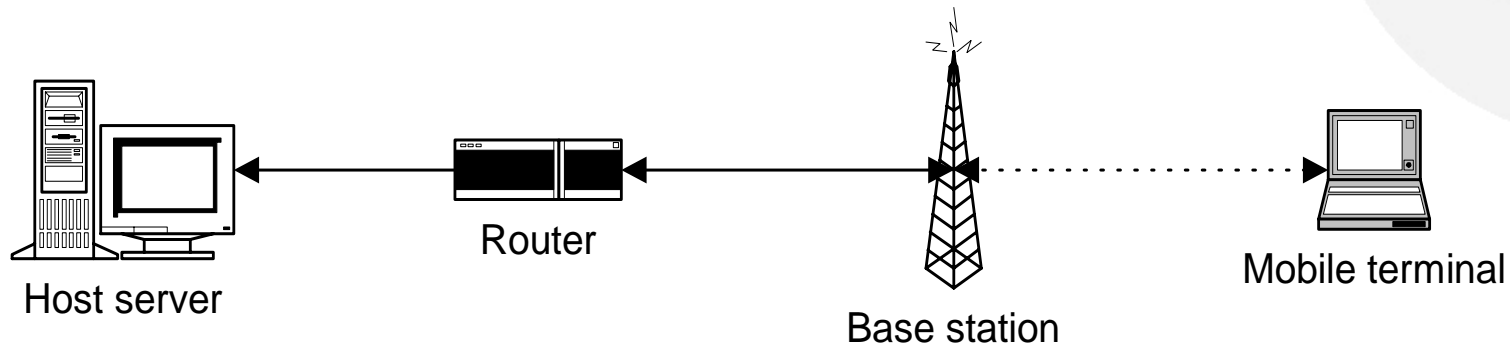
- For every packet received, the recipient returns an **Ack**
- Recipient sends **duplicate Acks** if a packet is lost
- Sender re-transmits lost packets

TCP transmission window



- TCP **increases** the window exponentially to determine the available bandwidth
- When the source fails to receive an acknowledgement, TCP **reduces** the window size by half.
- The source **increases** its window size by one unit every average round trip time

Wireless LAN using TCP

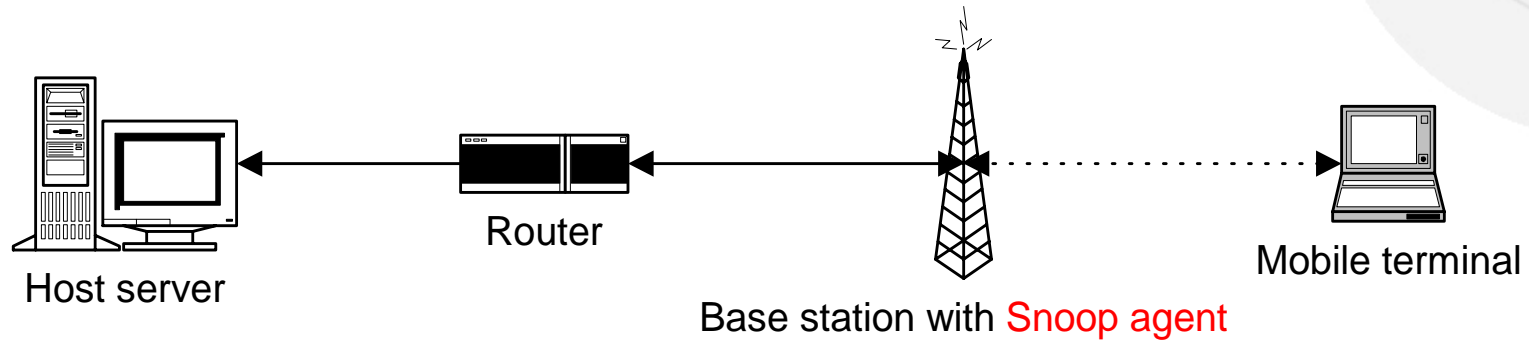


1. Host server establishes a TCP connection with mobile terminal and starts to send data

3. Missing acknowledgements trigger congestion control at host server

2. High bit error rate in the wireless channel

Adding the Snoop agent



1. Host server establishes a TCP connection with mobile terminal and starts to send data

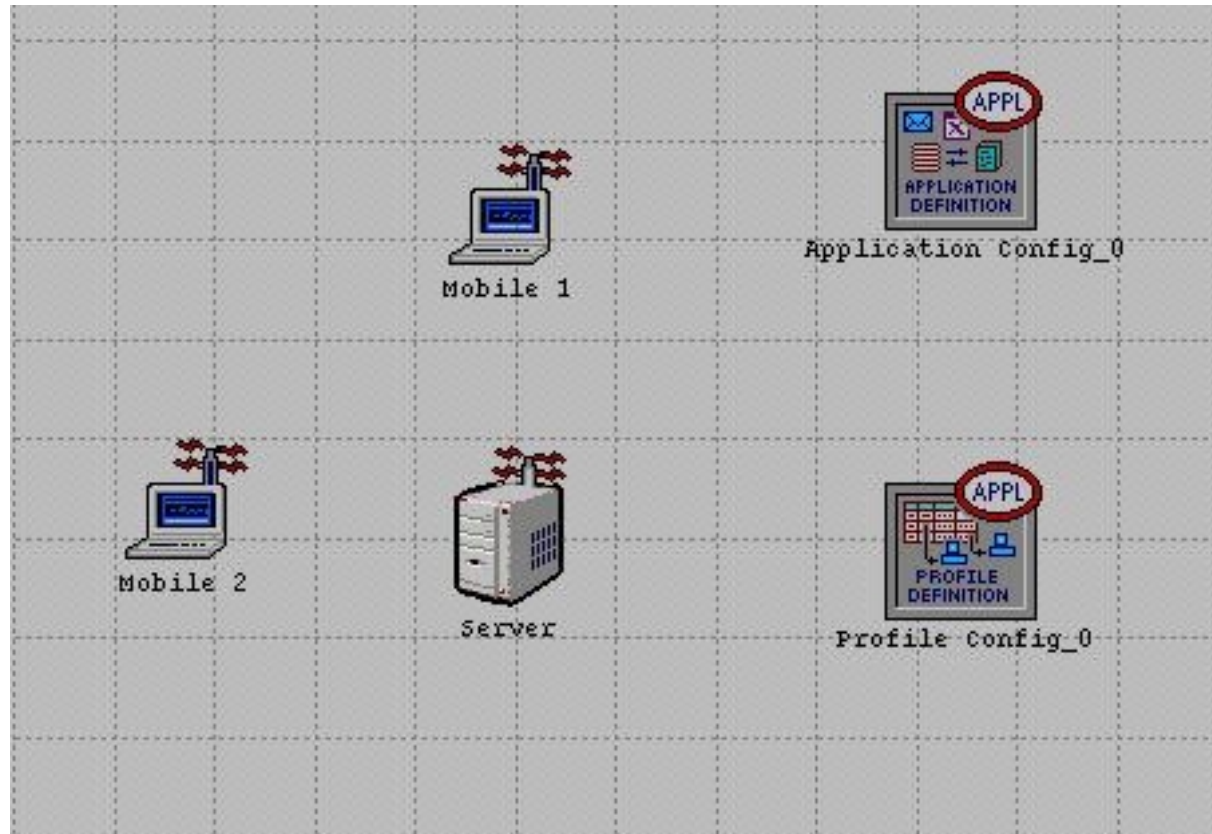
3. Snoop agent re-transmits lost packets locally

2. High bit error rate in the wireless channel

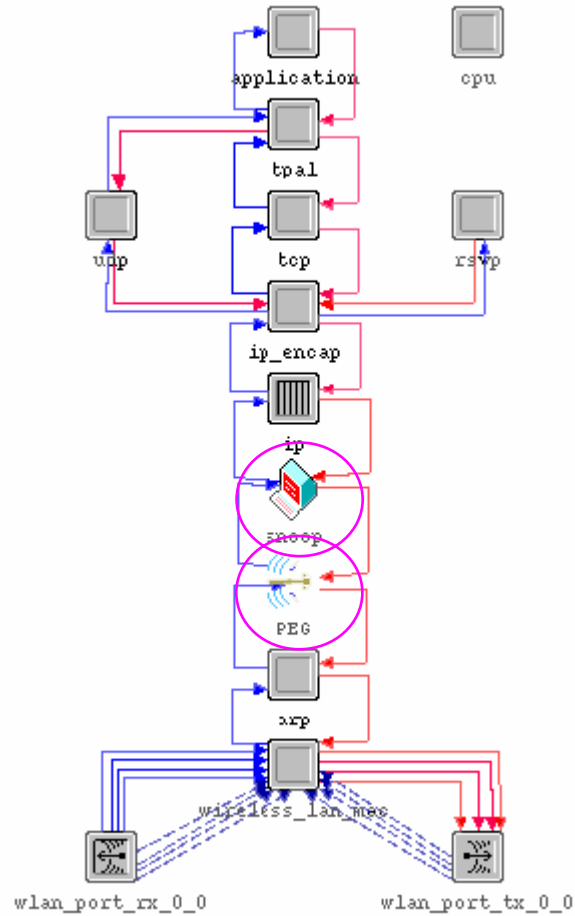
Snoop protocol

- Snoop copies packets to its **cache**
- It starts a retransmission timer
- It retransmits the packets if a duplicate Ack is received or if the timer expires
- When an **Ack** is received, Snoop **deletes** the cache entry

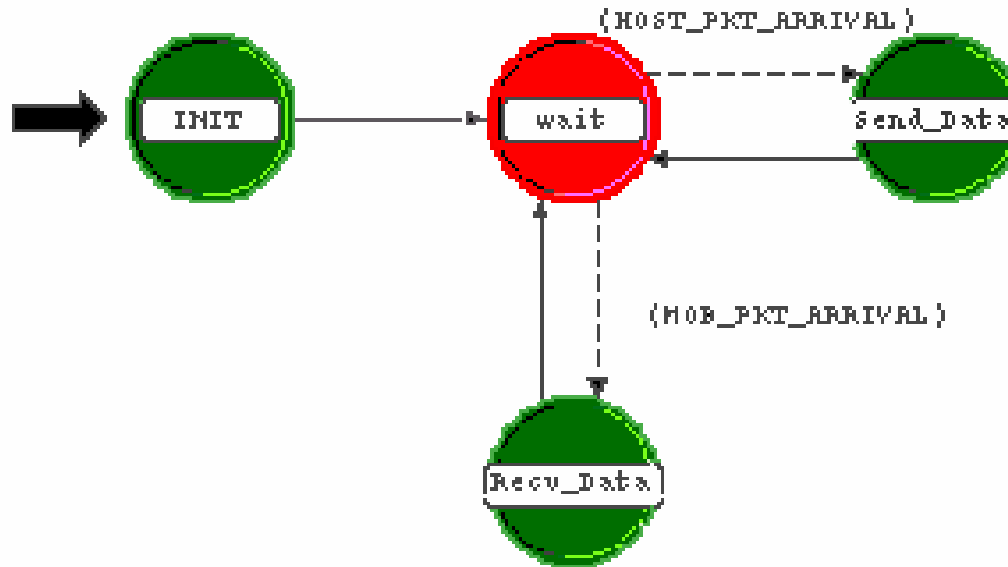
OPNET implementation



Two additional protocol layers

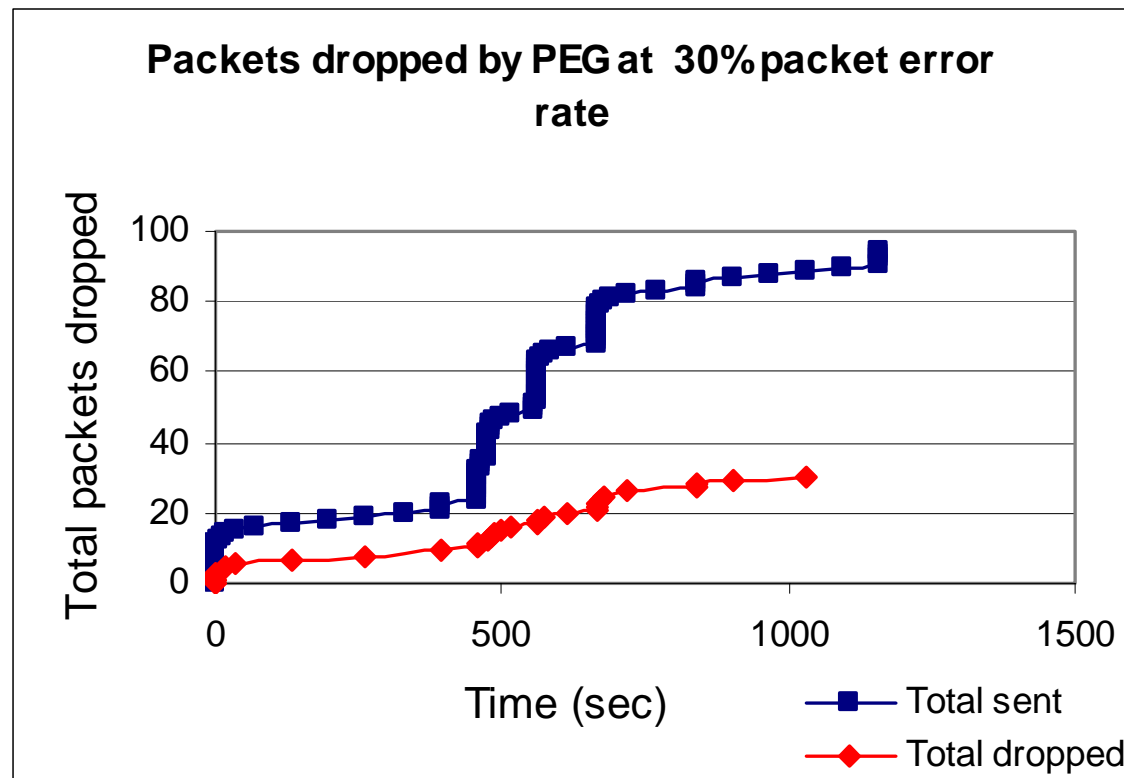


PEG process model

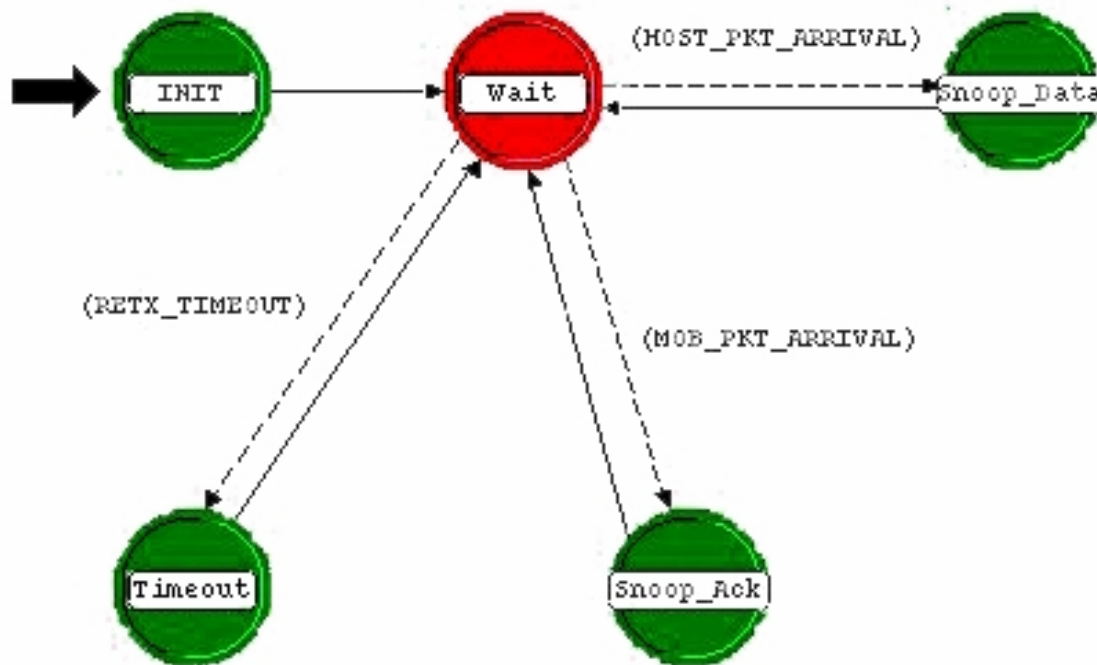


Packet error generator: PEG

- Generates packet loss (uniformly distributed)



Snoop agent process model



Snoop cache

typedef struct

```
{  
    unsigned int  src_ip;  
    unsigned int  dest_ip;  
        int      src_port;  
    int          dest_port;  
    unsigned int  seq_num;  
    unsigned int  ack_num;  
    unsigned int  rcv_win;  
    int          urgent_pointer;  
    int          data_len;  
    int          urg;  
    int          ack;  
    int          push;  
    int          rst;  
    int          syn;  
        int      fin;
```

```
} TcpInfo;
```

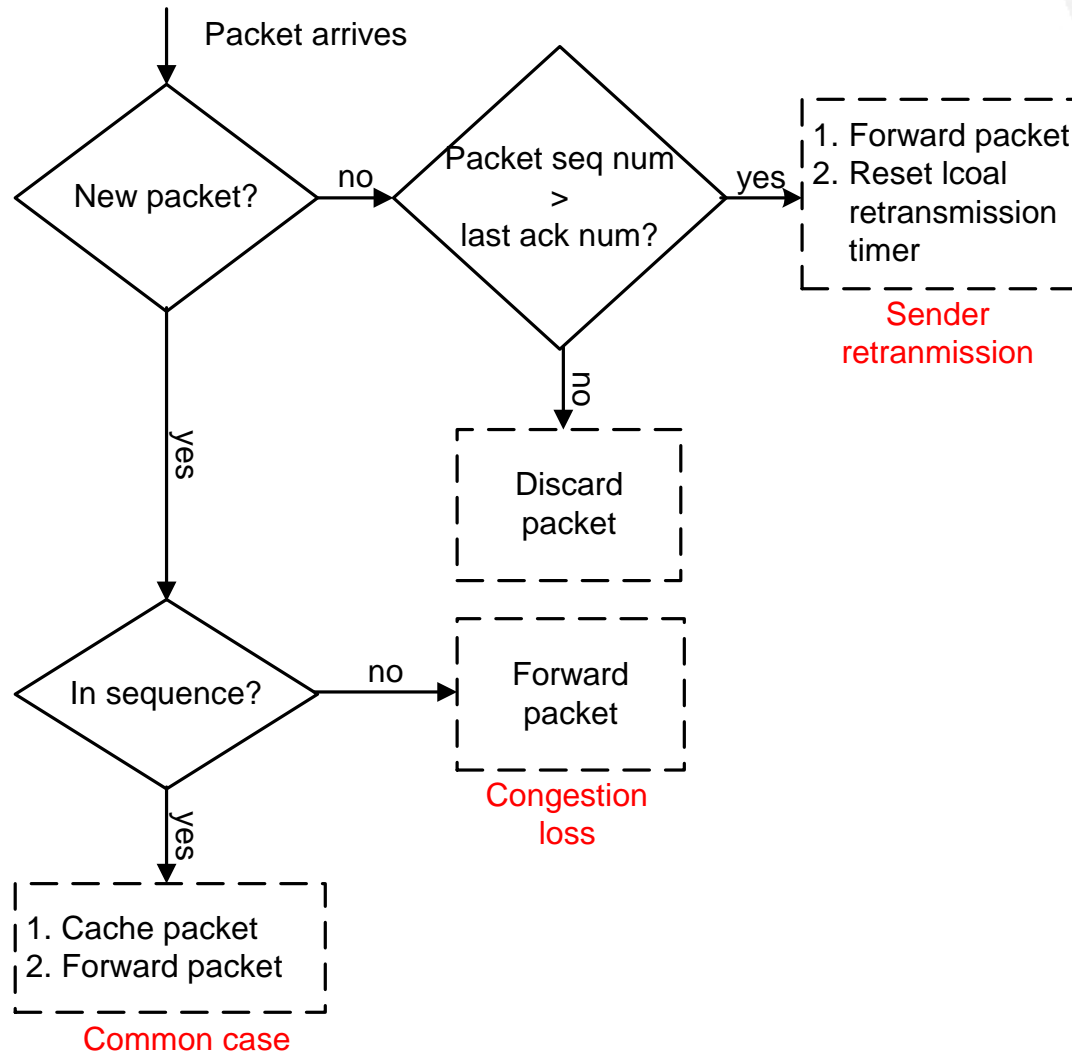
Cache functions

Function	Description
<code>snCacheInit</code>	Initializes the cache records
<code>snCachedPkt</code>	Copies a packet into the cache
<code>snCacheRetrieve</code>	Retrieves a packet from the cache
<code>snCacheDestroy</code>	Deletes a packet from the cache

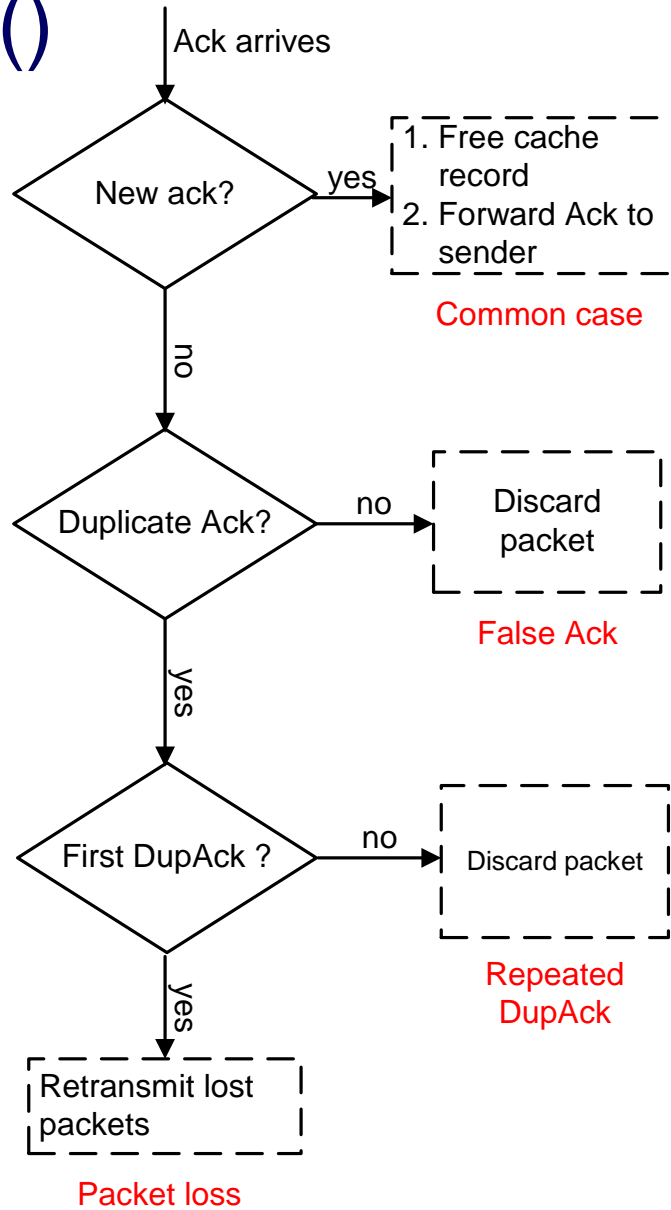
Snoop connection table

```
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 int   fin_seq_num;
    /* Timeout event handle */
    Evhandle       timeout_evt;
} struct_snTable;
```

Snoop_Data()



Snoop_Ack()

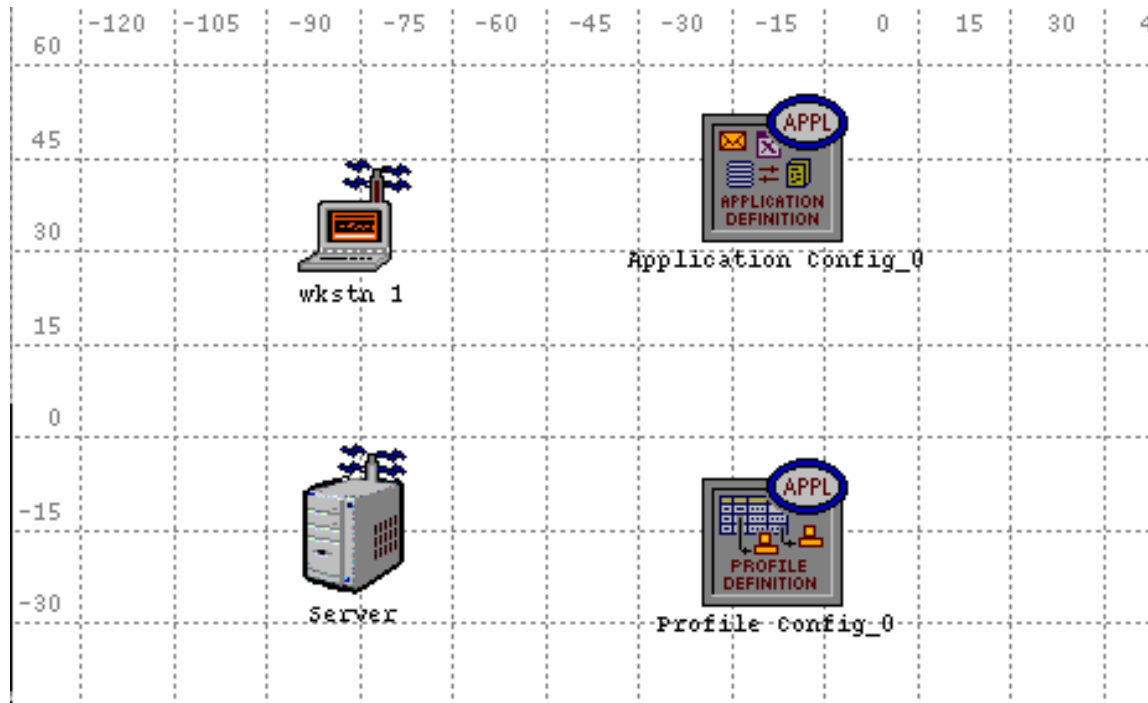


TCP parameters

TCP Parameters	Values
Maximum Segment Size (bytes)	2264
Receiver Buffer Size (bytes)	8760
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

Scenario 1: Single mobile upload

- Mobile 1 uploads a 100,000-byte file to the server

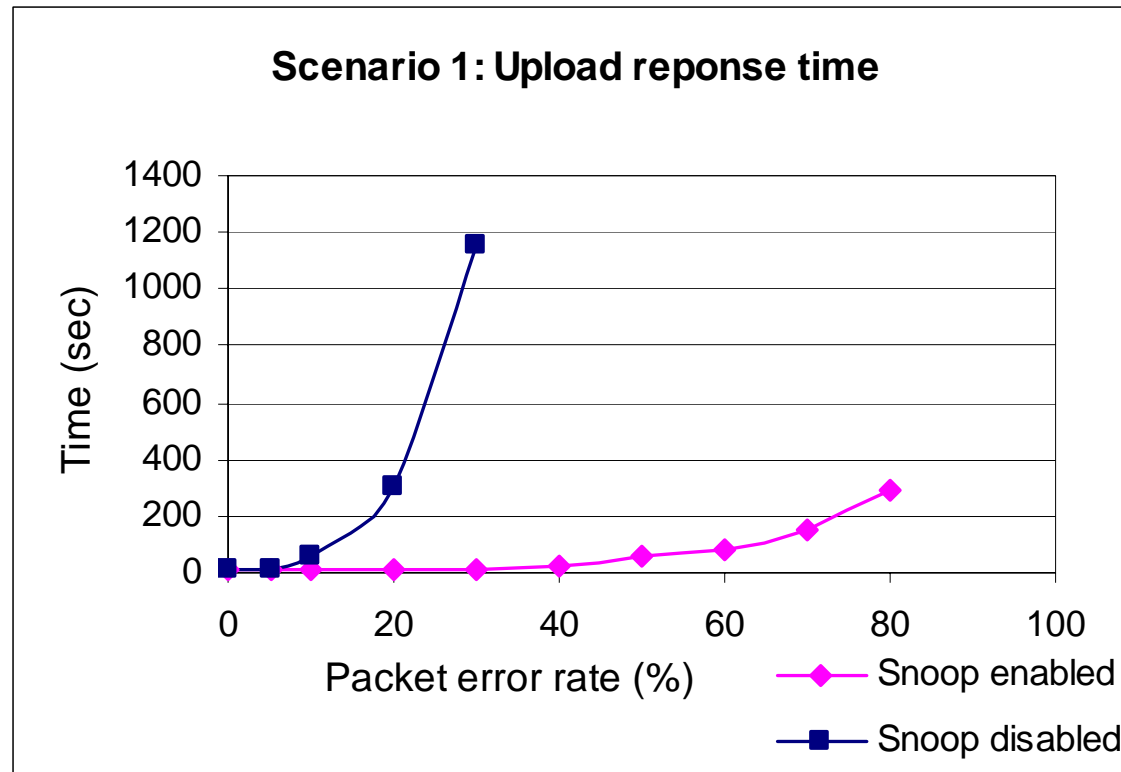


Scenario 1: parameters

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

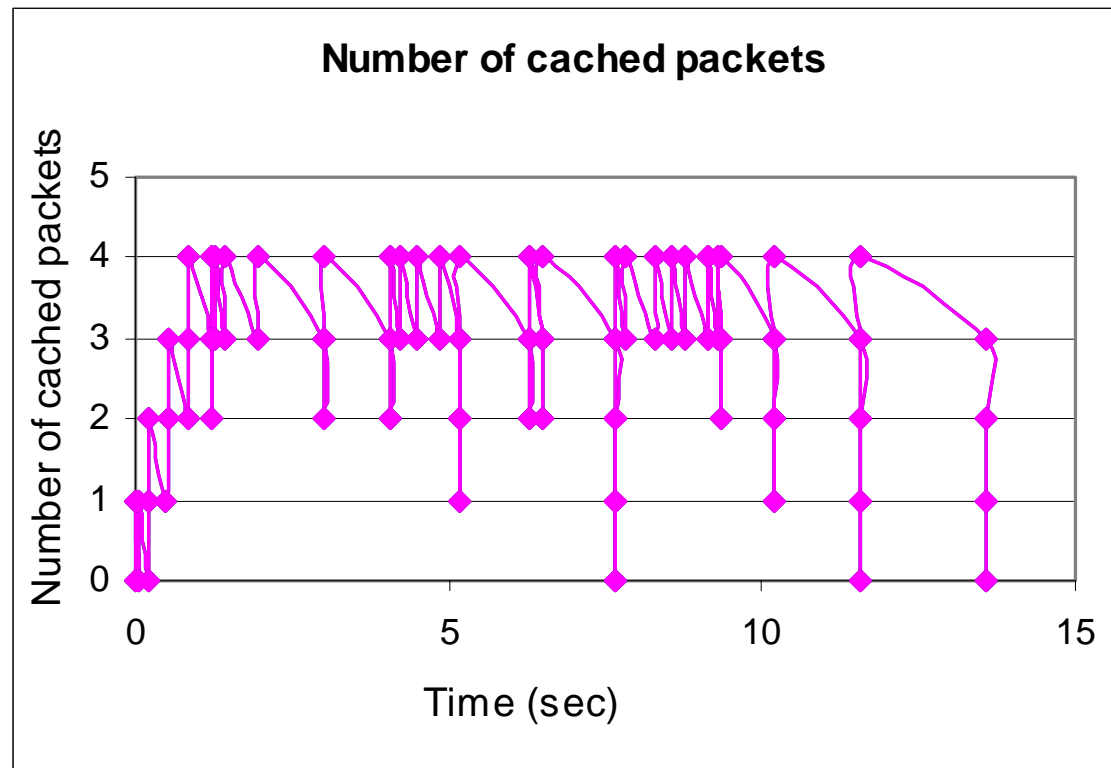
Upload response time

- ~ 68 times improvements at 30% error rate



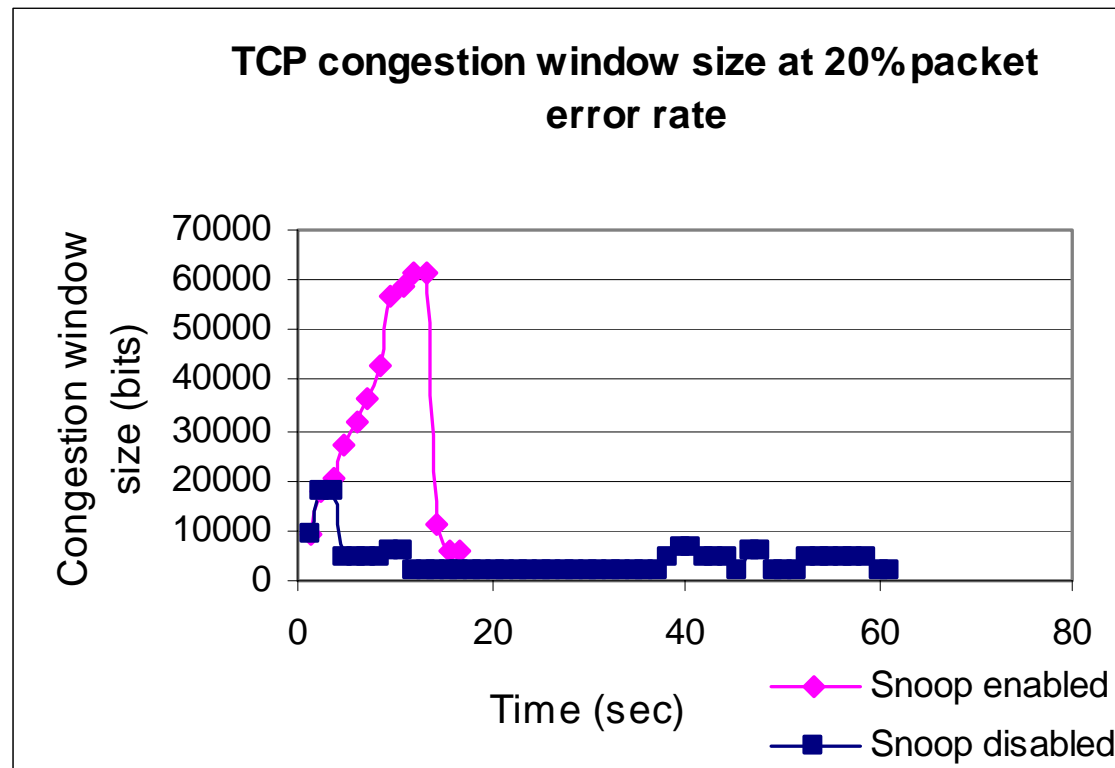
Packet caching

- Transmission window: four packets
- Number of packets cached at the end: zero



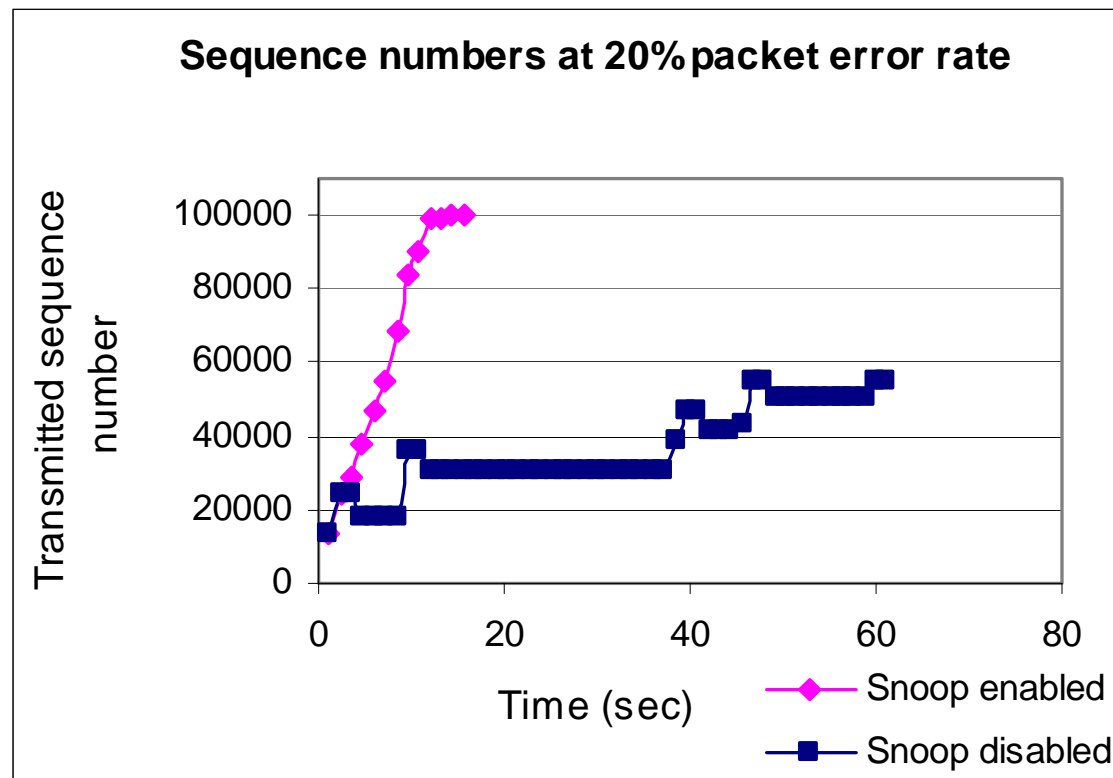
Congestion window size

- Varies with time



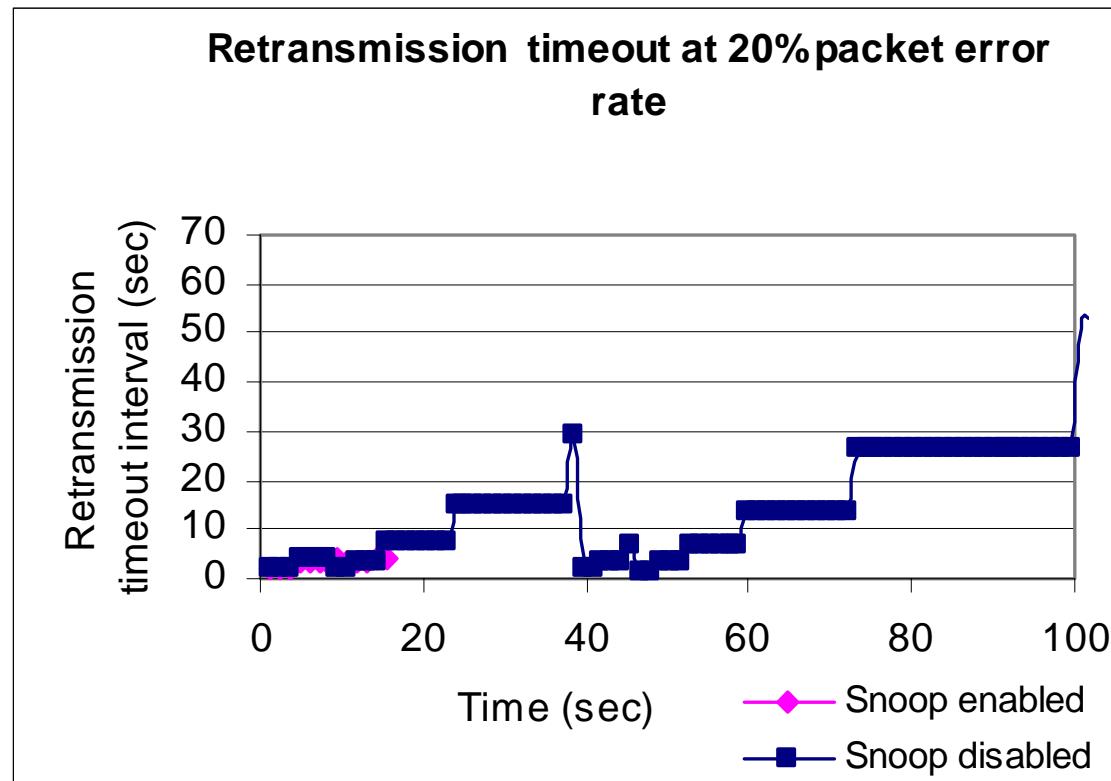
Sequence numbers

- Each data byte is represented by a sequence number



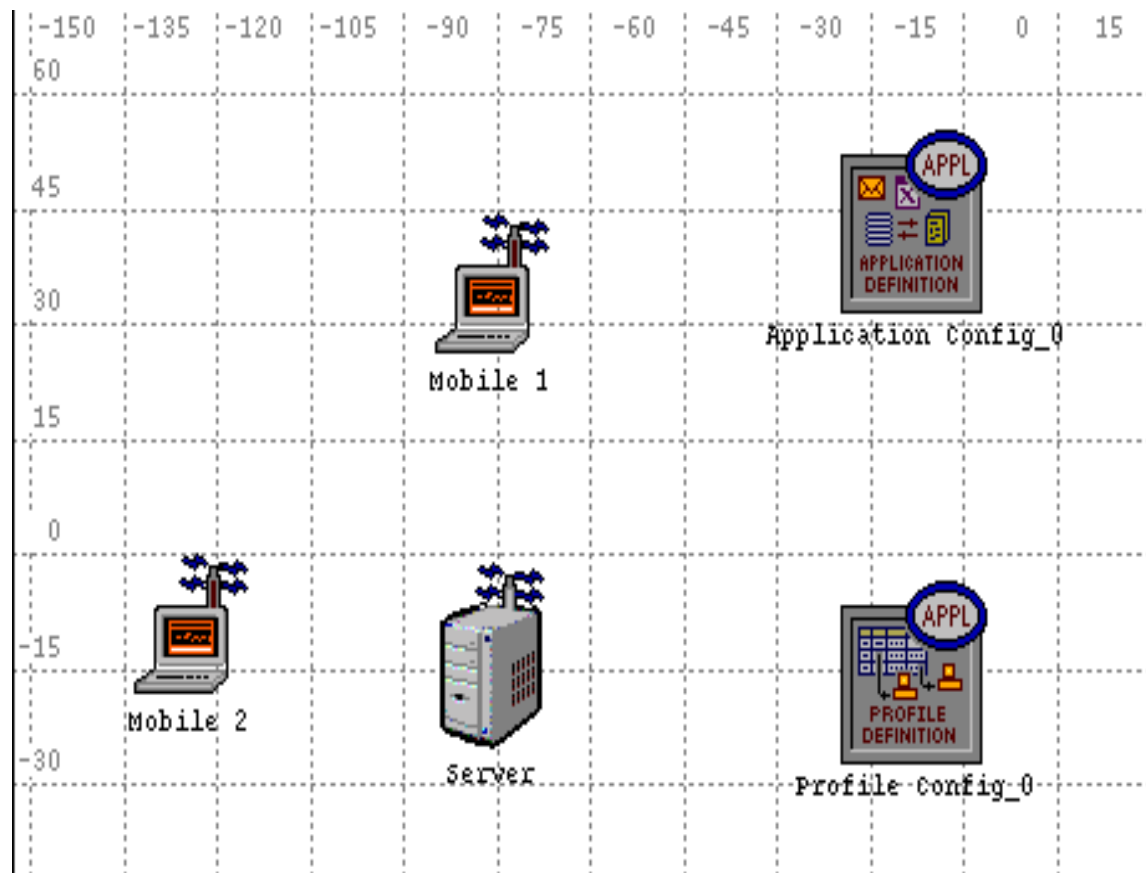
Retransmission timeout

- Persist retransmission timeout period = 1 sec



Scenario 2: Multiple mobile downloads

- Snoop enabled at the server

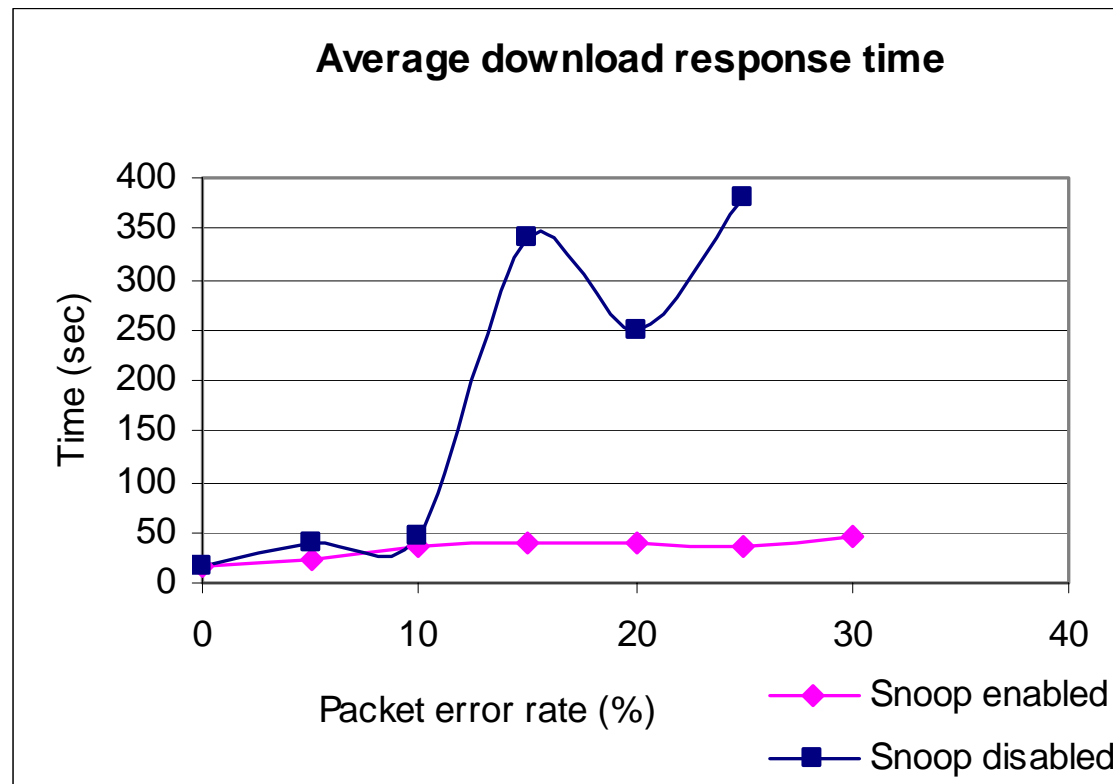


Scenario 2: parameters

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

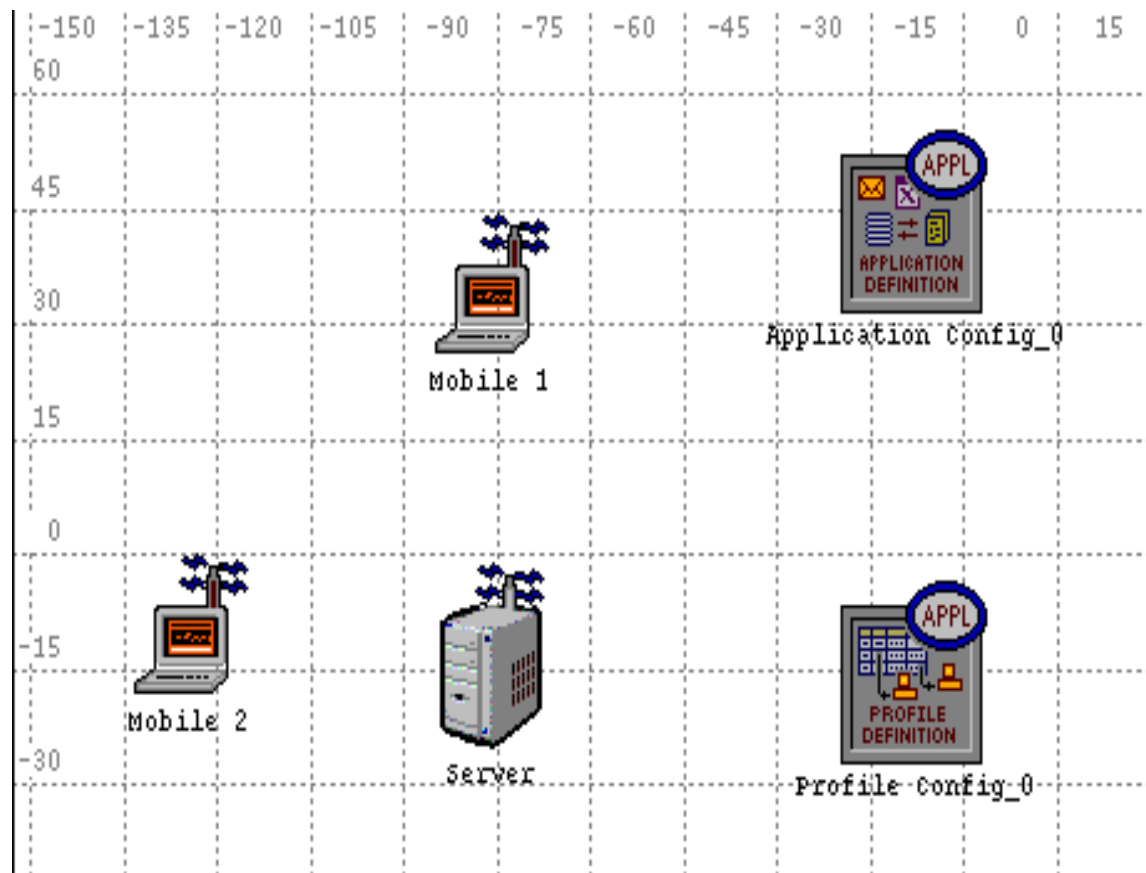
Download response time

- Snoop can handle multiple connections



Scenario 3: Multiple mobile uploads

- Snoop disabled on Mobile 1 and enabled on Mobile 2

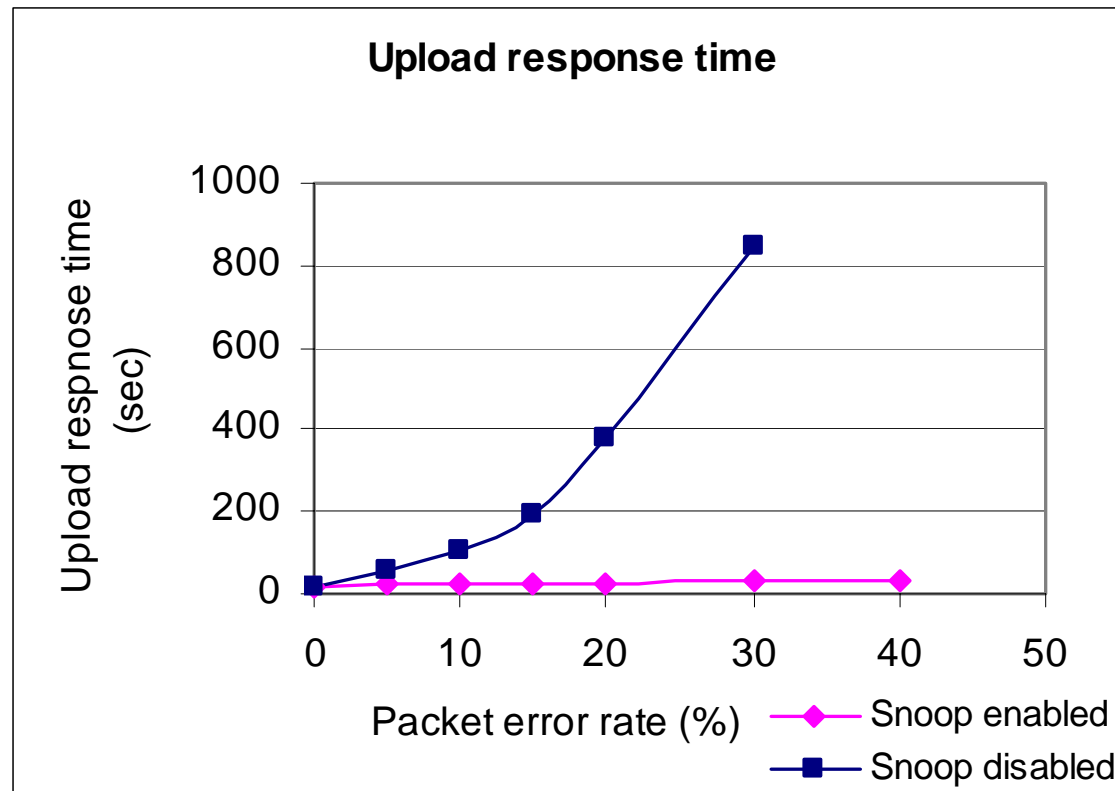


Scenario 3: parameters

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

Upload response time

- Mobile with Snoop has shorter upload time



Conclusions

- We implemented Snoop and packet error generator OPNET models
- They were implemented between the ARP and IP OPNET nodes.
- Implementation required examination of the ARP and IP node source code and their separation
- **Future work:**
 - Vary the retransmission timer based on calculated round trip delays in wireless links

References

- [1] IEEE 802.11 Workgroup: <http://grouper.ieee.org/groups/802/11/index.html>.
- [2] Performance Enhancing Proxy (PEP) Request for Comments (RFC): <http://community.roxen.com/developers/idocs/drafts/draft-ietf-pilc-pep-04.html>.
- [3] Improving TCP/IP Performance over Wireless Networks: <http://www2.cs.cmu.edu/~srini/Papers/publications/1995.mobicom/mobicom95.pdf>.
- [4] W. Stevens, *TCP/IP Illustrated*, Volume 1. Reading, MA: Addison Wesley, Professional Computing Series, 1984.
- [5] A. S. Tanenbaum, *Computer Networks*, Third edition. Englewood Cliffs, NJ: Prentice-Hall, 1996.
- [6] Wireless LAN Model Description, OPNET Manual.