CS-298 REPORT

# ENHANCING TCP PERFORMANCE IN WIRED-CUM-WIRELESS NETWORKS


BY:

SHRUTHI B KRISHNAN

ADVISOR:

DR. MELODY MOH

DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY


FALL 2006

**APPROVED FOR SHRUTHI B KRISHNAN**

**THE DEPARTMENT OF COMPUTER SCIENCE**

---

**Prof. Melody Moh, Advisor,**

**Department of Computer Science**

---

**Prof. Teng Sheng Moh, Committee Member,**

**Department of Computer Science**

---

**Prof. Mark Stamp, Committee Member,**

**Department of Computer Science**

# ACKNOWLEDGEMENTS

First and foremost, I would like to thank my professor and advisor, Dr. Melody Moh, for accepting me as her protégé, and guiding me through this project. I believe that my association with her for over two years has been pivotal, albeit arduous, in preparing me for future professional challenges. She has encouraged me to perform better and test my limits. She has steered me in the right direction at every stage of the project. Her critical reviews of my research, results, and reports have been invaluable to my work.

I would like to extend my heartfelt thanks to my professors, Dr. Mark Stamp and Dr. Teng Sheng Moh, for agreeing to serve as my committee members. I sincerely appreciate them for providing crucial comments and advice during the course of my project, and patiently reviewing my final report.

I profoundly thank my husband, Arun Venkataraman, for his support and encouragement. My work would be impossible without his vital guidance and assistance. He is my constant source of inspiration.

I owe my gratitude to my family for their support and understanding. My parents have toiled all their lives to provide me with excellent education. They have taught me the value of hard work, and helped me believe in myself. My sister's unwavering belief in my abilities makes me give my best in whatever I do.

Finally, I would like to thank all my professors and friends at San José State University from the bottom of my heart. They have helped me better myself not only as a professional but also as a person. I will cherish every moment I spent at SJSU for the rest of my life.

# ABSTRACT

*Increasing popularity for mobile devices has prompted industrial and academic research towards improving the performance of wireless applications. Transmission Control Protocol (TCP) plays an important role in defining a network's performance, and its use in wireless networks has exposed several inadequacies in its operation. Tight coupling of TCP's error and congestion control mechanisms has proven to be incompatible with the unique characteristics of wireless channels. TCP, designed for wired networks, assumes any loss of packet to be an indication of congestion in the network. Wireless networks exhibit a higher bit error rate, low and varying bandwidth, and disconnections of hosts due to mobility. All of the aforementioned reasons can result in random packet loss which is misinterpreted as a sign of congestion by TCP. Such erroneous triggering of congestion control measures can unnecessarily reduce TCP throughput. In this report, we will delve deeper into TCP's operation, and discuss its performance issues in wired-cum-wireless networks. We also present a survey of existing schemes that tackle these issues, and introduce a new scheme called TCP-ECN to enhance TCP performance in wireless networks.*

*The essence of the new scheme is to use Explicit Congestion Notification to enable the wireless host to distinguish between wired and wireless losses. Another facet of our scheme is to allow the base station to "freeze" the sender when it notices an imminent disconnection of the mobile host. The objective of TCP-ECN is to insulate the TCP sender from the idiosyncrasies of the wireless channel. We have both simulated and implemented the new scheme. This report details the new scheme in depth, and analyzes the test results obtained.*

# TABLE OF CONTENTS

# ACRONYMS

| | | |
|---|---|---|
| **ACK** | : | Acknowledgement |
| **AIMD** | : | Additive Increase, Multiplicative Decrease |
| **AP** | : | Access Point |
| **ARQ** | : | Automatic Repeat reQuest |
| **BER** | : | Bit Error Rate |
| **BS** | : | Base Station |
| **Cwnd** | : | Congestion window |
| **CRC** | : | Cyclic Redundancy Check |
| **DupACK** | : | Duplicate Acknowledgement |
| **ECN** | : | Explicit Congestion Notification |
| **ELN** | : | Explicit Loss Notification |
| **FEC** | : | Forward Error Correction |
| **FA** | : | Foreign Agent |
| **FH** | : | Fixed Host |
| **HA** | : | Home Agent |
| **IP** | : | Internet Protocol |
| **MAC** | : | Medium Access Control |
| **MH** | : | Mobile Host |
| **MSS** | : | Maximum Segment Size |
| **MTU** | : | Maximum Transmission Unit |
| **RTO** | : | Retransmission Time Out |
| **RTT** | : | Round Trip Time |
| **SACK** | : | Selective Acknowledgement |
| **SNR** | : | Signal to Noise Ratio |
| **TCP** | : | Transmission Control Protocol |
| **WLAN** | : | Wireless Local Area Network |
| **WWAN** | : | Wireless Wide Area Network |
| **ZWA** | : | Zero Window Advertisement |

# LIST OF FIGURES

# Chapter 1. INTRODUCTION

Years of research, experience, and fine tuning has resulted in the current design of one of the most important protocols in the networking stack—Transmission Control Protocol (TCP). TCP has proven to be essential for reliable communication, and resilient to the uncertainties in the network. Today, the world's biggest network, the Internet, relies heavily on TCP for sustainable performance. Although TCP is indispensable for reliable communication in the wired network, its performance is sub-par when used in wired-cum-wireless networks. TCP's error and congestion control mechanisms have been found wanting in the face of unique challenges posed by wireless networks. In this report, we will study the intricacies of TCP's operation, and the effect of wireless network characteristics on its performance. Several enhancements and workarounds have been proposed to overcome TCP's inadequacies in wireless networks. We will present a survey of some of these proposals in this report.

The knowledge we gained by surveying several wireless TCP enhancement proposals aided us in developing a new scheme we call TCP-ECN. Our main focus was to utilize the principles of local recovery techniques, and overcome their main problem—competing local and TCP retransmissions. We achieve this by adapting the Explicit Loss Notification technique in order to provide Explicit Congestion Notification (ECN) to the wireless hosts. The information provided by ECN enables the wireless host to differentiate between wired and wireless losses, and hence prevents it from sending unnecessary duplicate acknowledgements. Our scheme also tries to solve the problem of long disconnections of wireless hosts by using sender freezing by the base station. We primarily concentrate on the scenario where the wireless host is idle for a while and then immediately disconnects. We have simulated TCP-ECN using ns2 [24], and implemented part of the scheme in the Linux kernel. Our tests have shown marked TCP performance improvement when TCP is coupled with TCP-ECN.

The report has been organized as follows: chapter 2 provides an overview of TCP's important functions; chapter 3 lists a few important characteristics of wireless networks pertinent to the topic; chapter 4 discusses the interplay between wireless networks and TCP operation; an extensive survey of schemes proposed to solve TCP issues in wireless environment is presented in chapter 5; design of the new scheme, TCP-ECN, is described in chapter 6; details of simulation and implementation of TCP-ECN, along with the analyses of test results are presented in chapters 7 and 8; and finally, in chapter 9, we conclude the report and discuss future work.

# Chapter 2. TCP SEMANTICS

Transmission Control Protocol (TCP) [1][7][19][22][29] is the most widely used layer-4 protocol in the Internet today, handling about 95% of the total traffic [33]. Decades of experience with the internet has helped fine tune TCP's operation to provide reliable communication between application layer processes. Any protocol designed to provide reliability must clone TCP in some form. In addition, TCP is fundamental to controlling congestion in the internet. TCP's indispensability for reliable communication and congestion control can be better understood by taking a deeper look into its operation. The primary responsibilities of TCP are multiplexing/de-multiplexing segments from/to application processes, connection set up and management, error control, flow control, and congestion control [20]. We will look at each of these aspects briefly, emphasizing on error and congestion control, as they are crucial to our discussion.

## *2.1 Multiplexing and de-multiplexing data*

TCP provides logical communication between two application processes, as Internet Protocol (IP) provides logical communication between two hosts. Data addressed to a host can be destined to any of the several processes currently executing on that host. TCP is responsible for delivering segments arriving at the host to respective processes. This task is called de-multiplexing. Similarly, multiplexing refers to the task of processing segments from different sockets (processes), like attaching headers etc., and passing them down to IP [20].

## *2.2 Connection set up and management*

TCP uses three-way handshaking to establish a full-duplex, point-to-point, logical connection between two processes. TCP on the host initiating the connection sends a SYN segment, which is acknowledged by the peer TCP in a segment that also includes a SYN for the other direction (SYN + ACK). Finally, the initiator acknowledges the SYN from the receiver to complete the connection establishment. The handshaking is used to set up buffers and initial parameters such as sequence numbers, maximum segment size (MSS) etc., on both the ends. Once the connection is set up, TCP manages these parameters in both the directions for the entire lifetime of the connection. A connection is torn down by exchanging FIN segments and their respective ACKs.

## *2.3 Error control*

Since IP, and hence, the underlying network provides only best-effort service (no guarantees), the onus of providing reliable communication service to the application layer rests upon TCP. When data, broken into segments, is sent from the source, there are several scenarios in which the segments may fail to arrive at the destination intact. Some segments could be dropped in the network; segments could arrive with a few of their bits garbled; the destination may receive the segments out-of-order, as each segment can take a different path in a packet-switched network; or the destination may receive segments more than once (duplication). TCP must account for all of these contingencies. Positive cumulative acknowledgements (ACKs), acknowledgement numbers, along with a timer are used by TCP

to recognize loss of packets, or their ACKs. Checksums are used to verify the integrity of segments, and sequence numbers are used to distinguish original segments from their duplicates.

When a source sends a segment, it numbers it with the next sequence number for that direction, and includes the correct checksum. The source TCP also sets a timer for the oldest unacknowledged segment. Upon receiving a segment, the destination accepts the segment only if the checksum is correct. It then checks the sequence number against the number it was expecting. If it is an in-order segment, it sends an ACK. If the segment is duplicated, the receiver resends the ACK, but discards the segment. If the received segment is out-of-order (previous segment(s) missing), it buffers the segment, and resends an ACK to the most recent in-order segment. At the sender's end, segment loss is assumed if the timer expires, or if it receives three duplicate acknowledgements (value of three was chosen heuristically). Such an event automatically triggers a retransmission of the oldest unacknowledged segment from the sender.

TCP timer management is very subtle, and couples the functions of error and congestion control tightly. The combined time taken by a segment to reach the receiver and its ACK to reach the source is termed as round trip time (RTT). A single timer is used, and is set only for the oldest unacknowledged segment. TCP maintains a running, weighted average of the RTTs using:

$$\text{SmoothRTT} = (1 - \alpha)\ \text{SmoothRTT} + \alpha * \text{RTT}$$

where $\alpha = 0.125$. In addition to average RTT, RTT deviation is also maintained using:

$$\text{DevRTT} = (1 - \beta)\ \text{DevRTT} + \beta * |\text{SmoothRTT} - \text{RTT}|$$

Where $\beta = 0.25$. The retransmission timeout (RTO) value is computed as:

$$\text{RTO} = \text{SmoothRTT} + 4 * \text{DevRTT}$$

This subsection provided a brief introduction to TCP error control. For a more detailed discussion, please refer to [1][7][19][20][29][22].

## *2.4 Flow control*

The sender and the receiver applications could be processing data at different speeds. To avoid a fast sender from swamping a slow receiver with more segments than its buffer capacity, TCP has been designed to provide the receiver with full control of the flow rate. The *window* field in the TCP header is used to implement a sliding window scheme to achieve this. Every ACK sent by the receiver contains a value in the window field which reflects the available buffer capacity at its end (this value could also be zero, if the buffer is full). The sender paces its transmission rate based on this value. Window scale factor is decided during connection establishment. Amount of data (bytes) that can be sent in a full-sized segment—maximum segment size (MSS)—is also decided during the connection establishment.

## *2.5 Congestion control*

Congestion occurs in a network if a router's queue is full, and it is forced to drop any new incoming packets. In order to counter congestion, all the sources in the network must collaborate and reduce their transmission rates. TCP plays a pivotal role in controlling congestion in the internet. TCP bases its congestion control decisions upon its perception of congestion, or in other words, its estimation of the degree of congestion [20]. To make this estimation, it relies on the error control mechanisms discussed earlier—timeouts and duplicate-acknowledgements. Thus, error and congestion control mechanisms are tightly coupled. The sender's transmission rate is not only controlled by receiver's buffer availability, but also by another variable called the congestion window (*cwnd*). Congestion window is measured in terms of segments, unlike receiver window which is measured in terms of bytes.

```
Current sending window = min (receiver window, congestion window)
```

The congestion control algorithm of TCP is characterized by an *additive increase, multiplicative decrease (AIMD)* of the congestion window. Other features include *Slow start, reaction to timeouts, Fast retransmit,* and *Fast recovery*. We will discuss each of these briefly.

### 2.5.1 Additive increase, multiplicative decrease (AIMD)

Whenever TCP detects a loss of segment, it is evident that the sending rate has to be reduced, as the loss is perceived to be a mark of congestion. For this purpose, TCP uses a multiplicative reduction strategy, wherein it halves the current congestion window for every loss. Hence, for several consecutive losses, congestion window reduces exponentially. Nevertheless, *cwnd* size is never less than one segment. In contrast, TCP adopts a more conservative approach when increasing the congestion window. The congestion window is increased by ($MSS^2$/*cwnd*) bytes for every ACK [20], or in other words, *cwnd* is increased by one after all the ACKs for a *cwnd* worth of segments arrive. This linear increase is also called *congestion avoidance*.

### 2.5.2 Slow start

Immediately after connection establishment, *cwnd* is set at one segment, and another variable called *threshold* is set at 64Kbytes. Even though, receiver's buffer is empty initially, sender cannot start bursting segments, as it does not know the network's capacity (capacity of intermediate routers). So the sender starts with *cwnd* set to one. But, unlike the congestion avoidance phase, during the slow start phase, TCP increases *cwnd* by one segment for every ACK received. This effectively increases *cwnd* exponentially for every RTT. The exponential increase continues until *cwnd* = *threshold* or *cwnd* = receiver window.

### 2.5.3 Reaction to timeouts

If a loss of segment is detected via a timeout, TCP sets the *threshold* to *cwnd*/2 (multiplicative decrease), and sets *cwnd* to one. In addition, the RTO value is doubled. Hence, successive losses exponentially increase the value of RTO. Slow start phase is then initiated. As stated in the previous discussion, cwnd is exponentially increased in the slow

start phase until *cwnd = threshold*. After this, congestion avoidance, or additive increase of *cwnd*, is initiated.

### 2.5.4 Fast retransmit and Fast recovery

Loss of segments detected through timeouts and three duplicate ACKs are treated differently by TCP. We have just seen the reaction to timeouts. In this subsection, we will take look at TCP's behavior when it receives three duplicate ACKs (dupACKs). Since duplicate ACKs are sent by the receiver whenever it receives out of order segments, the sender deduces that the segments are still getting across to the receiver, and the network is not fully congested. Unlike in the case of a timeout, invoking slow start would be too drastic here, and would unnecessarily reduce the throughput. Hence, in response to the reception of three dupACKs, the sender first retransmits the lost segment, then increases *cwnd* by the number of dupACKs received, and continues to increment *cwnd* for every dupACK received. As soon as an ACK for new data arrives, incremental addition to *cwnd* is stopped. The sender then reduces cwnd by half (multiplicative decrease) of it original value (before receiving 3 dupACKs), and invokes congestion avoidance (additive increase). The process of resending the lost packet upon the reception of 3 dupACKs, and not waiting for a timeout to occur, is termed as *Fast retransmit*. The aforementioned incremental addition and halving of the *cwnd* and invocation of congestion avoidance instead of slow start is termed as *Fast recovery*.

## *2.6 Congestion control in different versions of TCP*

One of the earliest versions of TCP, *TCP Tahoe*, included Slow start, congestion avoidance and Fast retransmit algorithms. *TCP Reno* was an improvement over TCP Tahoe in terms of throughput, due to the inclusion of the Fast recovery algorithm. The next version of TCP, *TCP New-Reno*, corrected the problems in TCP Reno for multiple losses in the same window. TCP Reno invokes Fast retransmit and Fast recovery upon receiving 3 dupACKs, and exits the phase as soon as it receives a new ACK. If several losses are encountered close to each other, the sender continually halves it congestion window due to repeated invocation of Fast recovery. A situation may be reached, wherein the congestion window is so small that the sender cannot send enough packets for the receiver to send dupACKs. This will lead to the sender and the receiver waiting for each other, and network resources being wasted. New-Reno overcomes this problem by making the sender set the congestion window to one, initiating slow start upon the reception of 3 dupACKs. This is called the Fast retransmission phase, wherein the sender retransmits unacknowledged segments. The sender exits this phase only after all the segments sent before the start of the phase are acknowledged. After the sender retransmits the first lost segment, the receiver sends a *partial ACK* indicating the next loss. New-Reno results in redundant retransmissions, but efficiently handles multiple losses in the same window [13].

*TCP SACK* or TCP with Selective Acknowledgements was proposed to handle multiple losses efficiently. SACK options are included in the duplicate acknowledgements sent by the receiver, upon detection of a loss of segment followed by the reception of a block of in order segments. The SACK option includes the sequence numbers of the first and the last in-order segment blocks buffered at the receiver. After receiving the third dupACK with the SACK option, the sender retransmits only the lost segments. Unlike the older versions of TCP, *TCP VEGAS* monitors the actual segment flow rate, and also maintains an estimated segment flow

rate. It uses the difference in the two rates to estimate the degree of congestion, and hence, deduce the sending window. The closer the actual flow rate is to the estimated rate, lesser the perceived congestion. TCP VEGAS tries to be proactive in countering congestion, rather than being reactive [13].

# Chapter 3. UNIQUE CHARACTERISTICS OF WIRELESS NETWORKS

Wireless communication is fast gaining popularity due to its flexibility, and convenience of on-the-fly access to network resources. Cellular telephony has already established itself as an indispensable service, and wireless data communication is catching up quickly. Wireless Local Area Networks (WLAN) are being installed in business establishments and homes to enhance flexibility. Similarly cellular networks are being used to carry data traffic to allow wider range of accessibility. Wireless networks are characterized by some unique features that pose several restrictions on applying the existing wired networking principles on them.

- ***Limited bandwidth availability:*** WLANs usually employ 802.11 wireless networking standards. Currently two of the standards are more prevalent than the others: 802.11b operating at 2.4-2.485 GHz, providing a bandwidth of up to 11Mbps; and 802.11g operating at the same frequency range, but allowing a maximum bandwidth of 54Mbps. Cellular networks provide long range communication facility at the cost of much lower bandwidth availability. Current 2.5G systems using GPRS provide data rates in the vicinity of 50Kbps. 3G systems using either UMTS or CDMA-2000 standards provide bandwidths that range from 144Kbps to 2Mbps [20]. In contrast, a typical Ethernet connection operates at 100Mbps, and Gigabit Ethernet is fast gaining ground.

- ***High Bit Error Rate (BER):*** Wireless links are more prone to random bit errors, and bursty losses. Since 802.11 networks operate within unlicensed frequency bands, signals from other devices such as a 2.4GHz telephone, or electromagnetic noise from a microwave can interfere with the signals from an 802.11b or 802.11g device causing errors. Radio waves taking different paths after being reflected off objects can cause a decrease in signal intensity at the receiver (multipath propagation) and bit errors. In addition, as a wireless device moves away from the base station, signal intensity decreases causing loss of data [20]. While the BER in wired networks is around $10^{-6}$, wireless networks exhibit a BER in the vicinity of $10^{-3}$ [13].

- ***Mobility:*** One of the primary advantages of using wireless devices is the ease of movement. WLANs allow limited mobility ranging a few hundreds of meters, whereas WWANs allow coverage of a few kilometers [27].

- ***Varying bandwidth:*** Movement of mobile hosts from one base station's range to another's makes the number of mobile hosts associated with a base station variable at any given point in time. Since all the mobile hosts in a given range share the total bandwidth, the bandwidth available per host also varies with time.

- ***Limited power availability:*** Most of the mobile devices are battery operated, and hence, require prudent use of available power.

# Chapter 4. EFFICIENCY OF TCP IN WIRELESS NETWORKS

Typical wireless networks comprise of a mobile host connected wirelessly to a base station, which in turn is connected to the wired backbone (possibly the Internet). Hence, studying the performance of TCP in a network with both wired and wireless links has generated tremendous research interest of late. Since TCP was designed and fine-tuned for wired networks, wireless network characteristics, discussed in the previous section, can pose subtle and complex problems to their interoperability. Let us take a look at the effects of each of the wireless characteristics on TCP's operations. For the purpose of discussion, we use the term fixed host (FH) to refer to a device connected to the wired network, and mobile host (MH) to refer to a wireless device.

## *4.1 Effects due to high bit error rate*

As mentioned earlier, TCP has been tried and tested for various problems that can arise in wired networks. One of the primary problems countered by TCP is congestion. TCP detects congestion whenever it notices a loss of segment, either in the form of a timeout, or in the form of three dupACKs. In fact, any loss of segment is automatically assumed to be due to congestion. This assumption works well in wired networks. But, in wireless networks, segments could be lost due to high BER, which is a transient condition, unlike congestion in routers. TCP takes congestion control measures whenever a loss occurs in the wireless link, and reduces its congestion window, thereby reducing throughput unnecessarily. Since TCP ACKs cannot convey enough information about the type of loss, TCP treats congestion and random losses alike.

Average throughput of a connection can be expressed as

$$\text{Average throughput} = (1.22 * \text{MSS}) / (\text{RTT} * \sqrt{L})$$

where **L** is the loss rate of the connection [22]. Clearly, average throughput is inversely proportional to the square root of the loss rate. Wireless links with a higher loss rate reduce the overall average throughput of the end-to-end connection [33].

Retransmissions also have an important side effect of exponentially backing off the RTO at the sender, and a large RTO value can render TCP less capable of sensing error-free conditions. As a result, overall end-to-end connection time will be increased. Given the limited power availability to the mobile host (MH), unnecessary increase in connection time can prove to be expensive. This problem can arise irrespective of whether the fixed or the mobile host is the sender [36]. Increase in RTO will also increase the idle time in the network bringing down the end-to-end throughput.

## *4.2 Effects due to mobility*

A mobile host moving from one base station's range to another's may experience a short period of disconnection. The mobile host could also temporarily move out of range of its base station, thus losing connectivity. Connectivity can also be lost if the signals are interrupted by obstacles. In WLANs random access of the channel is used to achieve fairness. If there are several mobile hosts in a base station's range at a given point in time, a host may not get a chance to use the channel for a while, because of the other hosts hogging the channel (*capture syndrome* [33]). All of the aforementioned reasons could result in loss of segments during the period of inaccessibility to the channel. A fixed host (FH) sender in the wired network is oblivious of these reasons for the loss, and invokes congestion control algorithms erroneously, thus reducing effective throughput upon reconnection.

Consider the scenario where the FH transmits a segment, and the MH is disconnected. This results in the FH timing out and backing off the RTO after retransmission. If the mobile host gets disconnected again before receiving this retransmission, the FH will time out again, and back off the RTO further. This process can continue until the RTO value becomes considerable. Now even if the MH is reconnected, FH does not retransmit until this large value of RTO expires. The channel will remain idle wasting network resources, and decreasing the effective throughput. This scenario is caused by *serial timeouts* [14].

A different problem can arise if MH is the sender. MH TCP will not be aware of disconnections and reconnections (since it is handled by the lower layers), it might misinterpret a loss due to a disconnection as congestion, and repeatedly try to retransmit in vain. This will increase the power consumption for no reason [36].

## *4.3 Effects due to low and variable bandwidth*

The rate of increase in TCP's congestion window depends on the rate of arrival of the ACKs. This property of TCP is known as *ACK clocking* [29][20]. The rate of arrival of ACKs, in turn depends on the bandwidth in both the forward and reverse paths. Wireless links have lower channel capacity compared to their wired counterparts. This makes the end-to-end delay longer, or in other words, RTT is large. Research has shown that TCP is biased to connections that have shorter RTTs, as this would result in faster growth of congestion window. Connections with wireless links are disadvantaged by this aspect. Typical connections last for a very short duration, and most of the communication completes in the initial Slow start phase itself. If the exponential increase in *cwnd* is slowed in this phase, TCP throughput is affected considerably [6].

As a consequence of increase in RTT, RTO value at the sender becomes large as it is dependent on the RTT. Large RTO values make sender TCP less sensitive to actual congestion losses. Hence, if congestion occurs, the sender takes a while to reduce its congestion window, aggravating the congestion scenario. If RTO is large, TCP will also be unable to sense error-free conditions quickly. Longer RTT also has the inevitable problem of increasing the connection time which results in more power consumption at the MH [38].

# Chapter 5. SURVEY OF SCHEMES PROPOSED

More than a decade's extensive research has resulted in umpteen proposals to improve TCP's performance in networks with wireless links. Several surveys have been undertaken by researchers to weigh the pros and cons of some of these proposals [3][6][13][14][23][27][33][36]. Typically, the surveys classify the proposed schemes as *link layer enhancements (TCP-aware and TCP-unaware), split connection schemes*, and *end-to-end schemes*. Some of the other surveys categorize these schemes as *local enhancements* (involving only MHs and base stations) that hide wireless losses and *TCP level enhancements* (involving changes to the FHs as well) that handle wireless losses. For our discussion, we intend to take a different approach, and instead of classifying the schemes based on their semantics, we classify focusing on the wireless link problems, and how the schemes solve them. Consequently, the schemes are categorized as ones that handle high BER, ones that counter disconnection issues, and ones that handle varying bandwidth problems in wireless links.

Most of the schemes that we chose for our survey involve changes at either the base stations or MHs, as we consider schemes requiring changes to the FH-TCP impractical. Changing the FH-TCP involves altering the protocol stack of every single end host in the wired network. Although some researchers argue that redesigning TCP to account for wireless problems is the cleanest and a long term solution to the issues, we argue in favor of practicality and backward compatibility [23][4].

## *5.1 Schemes that address high BER issues*

A large percentage of the researched work is dedicated to solving TCP problems due to high BER in wireless links. Most of these concentrate on issues when FH is the sender. We will briefly describe the techniques used in some of these schemes.

### 5.1.1 Snoop

Snoop [4] is by far the most prominent proposal to solve high BER problems in wireless channels which has been proven to achieve a high throughput [3]. It is a TCP-aware Link layer scheme, i.e. a cross layer interaction between layer 2 and 4 is used for the design. The module is implemented at the base station which monitors and caches all the unacknowledged segments flowing across, between the FH and MH. The base station relies on dupACKs received from the MH, and timeouts to detect a loss in the wireless link. Upon detecting a loss, the base station locally retransmits the segment. Timeouts are finer than TCP timeout values, and hence, the base station tries to locally recover from the losses before a timeout is triggered at the FH sender. DupAcks from the MH are suppressed by the base station, thus preventing the FH from initiating the Fast retransmit procedure. Finally, any retransmission from the FH is also suppressed by the base station, in order to avoid competition to local retransmission in the wireless channel. For MH to FH transfer, negative acknowledgements are used by the base station to indicate to the MH about a loss in the wireless link.

Snoop has been tested and has exhibited good performance improvement over current TCP versions. Local recovery and implementation avoid the need to change the existing wired framework. TCP awareness of the link layer prevents retransmissions from TCP and Link layer from competing with each other, thus avoiding bandwidth wastage in the wireless channel. Nevertheless, Snoop also does not perform well when there are several losses in the same window [30]. In addition, when the percentage of wireless RTT is smaller compared to the end-to-end RTT (e.g. WWANs), Snoop's performance improvement is not significant, as it fails to prevent the FH from timing out [27]. Since the base station caches and processes segments, buffer requirements at the base station can prove to be a bottleneck when several MHs are associated with it. Also, a small delay is introduced into the end-to-end RTT due to additional processing. IPSec cannot be employed with Snoop, as the base station needs access to the TCP header, and IPSec encrypts it [37].

### 5.1.2 Explicit Loss Notification when MH is the sender

Explicit loss notification is a general technique used for several wireless applications. In this scheme, it is used to enhance TCP performance when the mobile host is the sender [5]. For traffic in the reverse direction, Snoop technique is employed in the experiments. So, the proposed technique can work in tandem with the Snoop protocol. Here, the base station keeps track of all the segments flowing from the MH to the FH. It maintains a list of sequence numbers. If any segment is lost in the wireless channel, the base station notes this as a hole in the list. When the first dupACK arrives from the FH indicating this loss, the base station sets the ELN (Explicit Loss Notification) bit in the ACK header. The MH, upon receiving this ACK, immediately retransmits the segment without changing its congestion window. This eliminates the need to wait for 2 other dupACKs to be sent, and the MH-TCP does not reduce its congestion window unnecessarily.

### 5.1.3 SNACK with New Snoop

Selective Negative Acknowledgement or SNACK with Snoop [34] is a combination of the Snoop scheme and TCP SACK. There are two components to the protocol, *SNACK-Snoop* which is implemented at the base station, *SNACK TCP* which is implemented at the MH. The scheme handles both FH to MH and MH to FH transmission scenarios. The base station plays a pivotal role in buffering segments, and monitoring the sequence numbers. SNACK blocks are added as TCP options, similar to TCP SACK. When FH is the sender, SNACK-TCP at the MH is responsible for detecting losses, and piggybacking the SNACK blocks with the DupACKs. SNACK-Snoop at the base station is responsible of processing the SNACK blocks arriving from the MH and locally retransmitting the lost segments. When MH is the sender, SNACK-Snoop adds SNACK blocks to the DupAcks arriving from the FH to indicate losses in the wireless link. SNACK-TCP at MH processes the SNACK blocks, and retransmits as soon as possible.

### 5.1.4 TULIP

Transport Unaware Link Improvement Protocol (TULIP) [24] is a TCP unaware protocol that can withstand high BER in the wireless link. It is designed for half duplex links, and requires service-type information (reliable or unreliable). TULIP provides reliable service for TCP data, and unreliable service for TCP ACKs. The receiver ensures in-order delivery of packets to higher layers, thus eliminating dupACK generation. TULIP module resides in between the

IP and MAC layers in the protocol stack at the base station. Data packets and link layer ACKs are interleaved based on the maximum propagation delay of the wireless link. TULIP receives packets from IP, and passes them down to the MAC layer for transmission. After the passing the first packet, TULIP waits for a predefined amount of time (computed using the maximum propagation delay) and allows the receiver to respond with an ACK. Once the wait period elapses, TULIP passes down the next packet, unless the MAC layer explicitly requests it to wait. Two signals, TRANS and WAIT are used for interaction between the TULIP module and MAC. TRANS signal is used by MAC to indicate to the TULIP module that the transmission of a packet has begun. TULIP waits for a specific amount of time for the Link layer ACK to arrive from the receiver, and initiates transmission of the next packet during the next time slot. The WAIT signal is used by MAC make the TULIP module wait for an additional amount of time (to allow transmission of variable length packets) before passing down the next packet for transmission. Flow control across the link is achieved using a window based scheme, and a bit vector is used by the receiver to indicate losses.

### 5.1.5 Delayed Duplicate Acknowledgements

In this proposal [37], modifications are restricted to the MH alone, and the base station requires no changes. The scheme was motivated by Snoop's inability to interoperate with IPSec. Link layer retransmissions are used to locally recover from wireless losses. In-order delivery of segments is not assured. But, if the MH receives out-of-order segments, it sends the first two dupACKs, and withholds the thirds and successive dupACKs for a predefined amount of time *d*. This delay postpones the triggering of Fast retransmit at the FH. If the loss was indeed in the wireless link, the base station must locally retransmit the lost packet within *d*. After *d*, if an in-order segment has not arrived at the MH yet, then all the withheld dupACKs are sent. The scheme works well if the losses are actually in the wireless channel. But, if the losses were due to congestion in the wired network, then the delaying of dupACKs, in fact, worsens the situation. In addition, an optimum value for *d* is hard to arrive at, given the varying RTT in the wireless link. The proposal is still open to further research.

### 5.1.6 Explicit Loss Notification with Link layer retransmission

The authors of the proposal [38] discuss the problem of finding the optimum number of retransmissions at the link layer at the base station to enable local recovery. Higher the number of retransmissions, better the chance to locally recover from a loss without affecting the FH sender. But, higher number of retransmissions also affect the end-to-end RTT, and hence, the RTO at the FH sender. This makes the sender incapable of acting quickly in the face of congestion. A new technique, wherein the base station ACKs every segment sent from the FH, is employed. Here, whenever the FH sends a segment, both the base station and the MH send ACKs. The ACK from the base station is called the *lhack* (last hop ACK). The arrival of a *lhack* indicates that any loss of the segment (if the ACK from MH fails to arrive) is in the wireless link, and hence, the sender does not alter its congestion window. FH also refrains from retransmitting, allowing local recovery. If the *lhack* fails to arrive and there is a loss of segment, it is perceived to be actual congestion, and the sender responds immediately without waiting for a timeout. Increasing the number of Link layer retransmissions at the base station will have no effect on the sender's sensitivity to react to congestion, as it bases its decision on the arrival of *lhacks*. The flip side of this technique is the doubling of reverse traffic, as 2 ACKs are being sent for a segment.

### 5.1.7 Wireless-TCP (W-TCP)

TCP operation is mimicked at the base station in this scheme [30]. The base station caches unacknowledged segments sent from the FH, and forwards it to the MH. Independent flow control is performed by the W-TCP agent at the base station. DupAcks and timeouts indicate losses in the wireless link. If a dupACK is received, then retransmission is performed, and the *wireless transmission window* is not altered at the base station. But, a timeout causes the wireless transmission window size to be set to 1, and unlike TCP, reception of a new ACK sets it back to the original value. Time spent in processing segments at the base station is hidden from the FH by using the TCP timestamp option whose value is modified at the base station to reflect the actual RTT. End-to-end semantics are maintained by the base station, as the only ACKs sent to the FH are the ones generated by the MH.

### 5.1.8 Indirect-TCP

I-TCP [2] is a split connection scheme, i.e. two separate connections are maintained between the FH and the MH. When the FH and MH establish a connection, FH actually connects to the MH's current Mobile Support Router (MSR). The FH is not aware of this split in connection, and sees an image of the MH at the MSR. A special transport layer protocol, I-TCP, is implemented between the MH and MSR. When the FH sends a segment, it is received by the MSR which immediately acknowledges it to the FH. The received segment is buffered at the MSR, and I-TCP is used to transmit it over the wireless link. Wireless channel's problems are completely hidden from the FH sender. End-to-end TCP semantics are lost in this scheme, as an ACK from the MSR does not mean the MH has received the segment. In fact, MH may never receive the segments if the MSR crashes. In addition, applications at the MH need to be modified to replace TCP systems calls with I-TCP system calls.

### 5.1.9 Combined use of link layer schemes

In this scheme [15], three link layer correction techniques, FEC (Forward Error Correction), ARQ (Automatic Repeat Request) and Snoop [4], are to achieve an enhanced TCP performance. Both ARQ and FEC are employed at the link layer, and each operates independent of the other. Experiments were used to arrive at the average number of damaged bits for the worst case MTU, and low Signal to Noise Ratio (SNR). It was found that for 70% of the frames in error, number of damaged bits were less than 30. Hence an FEC scheme that can correct a maximum of 15 symbols was used. FEC is selectively employed, i.e. a frame at the base station is encoded with an FEC only if the channel conditions are perceived to be bad. The channel condition is continuously monitored using the signal-to-noise ratio. Also, in order to reduce overhead, decoding of FEC at the MH is performed only if the CRC is in error.

## *5.2 Schemes that address long and frequent disconnection issues*

In this subsection, we will discuss some schemes that tackle TCP performance issues caused due to the movement of MHs that may result in disconnections. Problems arise not just during the disconnection, but also after the MH has reconnected to the same or a different base station. In order to achieve high TCP performance, sender TCP should not transmit, or alter its congestion window during the disconnection, and transmission has to immediately begin in full flow after reconnection. Disconnections can be long or frequent or both.

Frequent disconnections can cause serial timeout problems (see subsection 4.2). Most of the schemes discussed below that require changes to the MH-TCP assume the use of Mobile IP for movement across subnets [28], and cross layer signaling. Cross layer signaling is essential, because TCP is not aware of disconnections and reconnections, and requires this information to be provided from lower layers.

### 5.2.1 Triplicate Acknowledgements

Being one of the earliest schemes to address the TCP problems related to MH movement, this proposal [9] aims to reduce the delay in resuming transmission after the reconnection of the MH. If the disconnection time is long, FH sender will timeout serially, reducing its congestion window, and backing off its RTO value multiple times. After reconnection, transmission cannot resume immediately, as the FH has to wait for the exaggerated RTO to expire (refer to section 4.2). Meanwhile, network resources are being wasted. One of techniques proposed by the authors of this paper is to make the MH send 3 dupACKs immediately after reconnection, thus invoking Fast retransmission at the FH sender. The scheme assumes that MH TCP is aware of the reconnection event through cross layer interaction. Flip side of this approach is the reduction in the sender's congestion window on account of the Fast retransmission.

### 5.2.2 Freeze TCP

In our discussion about TCP flow control (subsection 2.4), we mentioned that the receiver can send an ACK with its window field set to zero, preventing the sender from transmitting. This process is called *freezing*. When the sender is "frozen," it transmits probes at exponentially increasing intervals (up to a maximum of 1 minute) to check if the receiver's buffer is still full. The probes have to be acknowledged by the receiver, but loss of probe-ACKs is ignored. Freeze-TCP [17] uses this concept to freeze the FH sender just before the disconnection, in order to avoid the decrease in congestion window, and unnecessary retransmissions. After reconnection, since a probe may not arrive in time to avoid idling, triplicate dupACKs are sent by the MH to enable immediate transmission (similar to the scheme used in [9]). For the MH-TCP to successfully send a Zero Window Advertisement (ZWA) before the disconnection, it must become aware of the impending disconnection 1 RTT prior to the event. If the ZWA is sent earlier than an RTT before disconnection, performance can degrade due to idling; if the ZWA is sent a little after, it may not reach the sender in time to freeze it successfully. In addition, predicting the disconnection in a network is a difficult feat to achieve [32]. Another problem noted by the authors is the determination of the optimum sending window size (sending rate) after reconnection. This becomes important if the new cell has a different available bandwidth compared to the old one.

### 5.2.3 Freeze TCP with timestamps

An extension to Freeze TCP [17], Freeze TCP with timestamps [11] considers the case where sender freezing fails before the disconnection. This can happen if the MH-TCP does not receive information about the impending disconnection in time. After reconnection, 3 dupACKs are sent by the MH to resume transmission. Since the ACK sent before the disconnection has not reached the FH, the first dupACK after reconnection is actually an original ACK from the FH's perspective. As such, FH uses this ACK's arrival time for the RTT, and hence, RTO estimation. This is erroneous, as the RTT estimate will now include

the disconnection time as well. To solve this issue, the researchers propose the use of TCP timestamp option, and to make the MH update the timestamp echo value after removing the disconnection time. Now, the timestamp value in the original ACK will depict the actual RTT (without the disconnection time), and the FH can make correct estimates.

### 5.2.4 M-TCP

M-TCP [8] was proposed to solve the problems of disconnections and varying bandwidth in the wireless links. In this section, we will look at the manner in which disconnection problems are solved by M-TCP. A three-tier architecture comprising of the MHs at the lower level, Mobile Support Stations (MSS) at the next higher level, and Supervisor Hosts (SH) at the highest level, is used for the design. Several MHs are controlled by an MSS, and several MSSs are controlled by an SH. The MH and the SH implement M-TCP as the Transport layer protocol. Along with the M-TCP client, an SH-TCP client resides at the SH which is responsible for interacting with the FH. When the FH sends a segment, it is received by the SH-TCP client at the SH, and passed on to the M-TCP client at the SH itself, which in turn sends it to the MH. The reverse path is followed by the ACKs. But, the SH withholds the latest ACK at all times. The M-TCP client at the MH is responsible for freezing all its timers upon disconnection, and sending a special ACK to the SH's M-TCP client upon reconnection. When the SH is aware of the disconnection, it uses the withheld ACK to "freeze" the FH sender. Similarly, when reconnection occurs, SH unfreezes the sender to resume transmission.

### 5.2.5 ATCP

Both FH to MH and MH to FH transfers are considered in this scheme [32]. Changes are proposed only to the MH. For FH to MH transfer, MH withholds the last two ACKs for a period less than TCP's typical timeout value (about 500ms). Upon reconnection, the first of these withheld ACKs is sent with a zero window advertisement by the MH, and second is sent with full window advertisement. This approach is suggested instead of the 3 dupACKs approach used in Freeze TCP [17][9], because the reduction in congestion window at the FH sender due to Fast retransmission is avoided. For MH to FH transfer, the following approach is proposed:

- If the send window is not full before disconnection, all timers are cancelled when disconnected, and the MH does not expect any ACKs. Instead, if the send window is full, MH waits for a timeout, even after disconnection.
- If a timeout occurs while disconnected, the MH reduces the *cwnd* to 1, but sets the *threshold* to the value just before disconnection, enabling a longer Slow start period (*cwnd* increases faster than normal TCP).
- After reconnection, if the send window is not full, MH resends the segments, and waits for a cumulative ACK. Else, if the send window is full, and a timeout occurs after reconnection, segment is retransmitted and *cwnd* is not altered.
- If there was no disconnection event, MH-TCP operates as normal TCP.

### 5.2.6 Indirect-TCP

We discussed I-TCP [2] semantics in section 5.1.8, and the way it hides the high BER from the FH. In this section, we will delve into I-TCP's technique to solve the disconnection problem. When the MH moves from one cell to another, the Mobile Support Router (MSR)

associated with the MH also changes. Since the end-points, MH and the FH, remain the same after reconnection, transferring the image of the MH from the old MSR to the new MSR will suffice. The FH is oblivious of this movement, and continues to communicate with the new MSR. Although this technique may seem simple in theory, handoffs and transfer of states can be complex in practice. In spite of the complexity, I-TCP cannot handle long disconnections efficiently, as it requires the MSR to buffer packets while the MH is disconnected. Buffer size at the MSR can pose a restriction on the duration of the disconnection.

## *5.3 Schemes to address varying bandwidth issues*

Wireless channel bandwidth is low compared to their wired counterparts. Split connection schemes like I-TCP [2] can hide this fact from the FH, because the ACKs are generated by the base station. Eventually, the difference in bandwidths of the two connections will place a burden on the buffer requirements at the base station. In this section, we will consider the problem of varying bandwidth in wireless links. Mobility of wireless devices makes it difficult to predict the amount of bandwidth available for individual MHs at all times. If the bandwidth per MH reduces drastically, data transmission effectively ceases for the MH. This scenario is similar to the case where the MH is disconnected. Let us look at a few schemes that try to avoid the problems to FH-TCP due to varying bandwidth available to the MH.

### 5.3.1 Adaptive-TCP

A-TCP [31] solves all the three problems: high BER, disconnection losses, and varying bandwidth. But, the solution to the varying bandwidth problem is the original contribution of this proposal. For high BER and disconnection problems, the authors use Snoop and M-TCP's solutions respectively. An A-TCP agent resides at the base station, and generates a *virtual host* object for every connection between an MH and an FH. Every virtual host object has a send and a receive buffer. A segment from the FH is first buffered in the virtual host's *retransmission buffer*, and sent to the MH. Segments are removed from the retransmission buffer when acknowledgements from the MH arrive. If the channel conditions are bad, or the available bandwidth in the channel is low, the rate of removal of segments from the retransmission buffer will be slower. Hence, the buffer occupancy is a clear indication of the channel conditions. Before forwarding the ACK from the MH to the FH, the virtual host object modifies the window field in the ACK to a value which is the minimum of receiver window size and available retransmission buffer size. This ensures that the FH does not transmit at a rate faster than what the current wireless channel conditions permit.

### 5.3.2 M-TCP

In section 5.2.4, we briefly discussed the way disconnections are handled by M-TCP [8]. In this section, we will look at how M-TCP handles the varying bandwidth problems. A bandwidth management module resides in the SH which allocates a fixed amount of bandwidth to each MH during connection establishment. This amount can vary during the lifetime of the connection depending on the number of users in the cell, and the status of the connection. The bandwidth available to each MH in a cell is continuously monitored by the SH. If this value goes below a threshold, SH uses the withheld ACK to send a zero window advertisement to the FH sender. The sender is allowed to transmit again when there is enough bandwidth available to the MH. This technique prevents the FH from timing out and reducing its congestion window.

# Chapter 6. TCP-ECN: A NEW SCHEME TO IMPROVE TCP PERFORMANCE

The extensive survey presented in the previous section helped us understand the major causes of TCP performance issues, and the requirements to overcome these. In [14], the author lists the desirable properties in a transport protocol as applied to wired-cum-wireless networks. Primarily, the protocol must be capable of differentiating congestion losses from random losses due to high BER, losses due to mobility, and wireless bandwidth related losses. The protocol must also retain end-to-end semantics, and not require changes to existing wired infrastructure. It is preferable that additional processing, and resource requirements are kept to a minimum, and the variation in end-to-end RTT is negligible. The protocol must provide good throughput for traffic in either direction. Satisfying all of these requirements is extremely difficult, as each requirement subtly affects the other. Hence, while designing a new scheme, we have to prioritize the requirements, satisfying the most important ones, and compromising on the others.

## *6.1 Motivation*

For our new scheme, we chose a local recovery approach (involving changes to the MH and base station) without requiring any changes to FH. We focus on FH to MH data transfer, and on solving problems due to high BER and long disconnections in the wireless links. We also intend to keep the memory and processing requirements at the base station to a minimum. Primarily, our experiments involve WLANs (802.11 networks), but the same principles can be applied to cellular networks as well.

We have based the central idea of our scheme on the approaches used in [5], [37] and [8] with several variations. Explicit Loss Notification (ELN) is a general idea that is used in several wireless applications. In ELN schemes, single bit information is used to intimate the type of the loss (congestion related or random). Most of the ELN based schemes, like those proposed in [5][38], provide information about a loss to the sender. In contrast, our scheme uses single bit information to intimate the type of loss to *the receiver (MH)*. We call it Explicit Congestion Notification.

## *6.2 TCP-ECN: Design details*

Our scheme, TCP-ECN, tries to solve both high BER and long disconnection problems in wireless links. In section 6.2.1, we present the solution to high BER, and in section 6.2.2, we will discuss the solution to the long disconnection problem.

### 6.2.1 Handling high BER

Local recovery based schemes rely on Link layer retransmissions at the base station. There are several Link layer schemes that do not use TCP awareness, like those in [37][24]. Researchers [3] have found TCP-unaware Link layer schemes lacking mainly because of the following reasons:

- ***Independent timers at TCP and Link layers:*** Since the Link layer timer at the base station, and the TCP timer at the fixed host do not depend on each other, competing retransmissions can occur. This happens if the Link layer is trying to locally retransmit, and the FH-TCP times out to initiate an unnecessary retransmission. Given the low bandwidth availability in the wireless channel, these redundant retransmissions can prove detrimental to the overall throughput.
- ***Out of order delivery of segments to the MH:*** The Link layer at the base station may not ensure in-order delivery of segments which can result in dupACKs being generated by the MH. If these dupACKs are received by the FH, Fast retransmission is triggered unnecessarily as the situation can be handled locally by the base station. Consequently, not only is the congestion window reduced, but local recovery is delayed due competing retransmission from FH-TCP.
- ***RTT and RTO exaggeration at the FH:*** Aggressive local retransmissions will increase the end-to-end RTT, and consequently the RTO value. This will make the FH less sensitive to congestion losses in the wired network, and also affect the FH's ability to sense error-free conditions quickly [36].

Experiments and extensive research have shown that in order to achieve an efficient solution to wireless-TCP problems, an interaction between layer 2 and layer 4 is essential [23][3]. Hence, we employ a TCP-aware (snoopy) scheme, and try to solve the aforementioned problems in local recovery schemes, focusing on reducing the overhead at the base station. Our scheme solves the second problem in local recovery schemes (problem due to out of order segment delivery).

#### 6.2.1.1 Problem due to independent timers

TCP uses coarse timers, typically around 500ms. Link layer timers are comparatively finer, usually much less than 100ms. For our scheme, we assume that the two timers collide rarely [38]. Hence, currently we choose not to handle this problem.

#### 6.2.1.2 Problem due to spurious dupACKs

Generation of dupACKs by the MH as a result of losses in the wireless channel is one of the fundamental reasons for TCP performance degradation. These dupACKs are not only unnecessary, but also consume precious bandwidth in the bandwidth-depleted wireless channel (e.g. 802.11 networks are half duplex in nature). They compete with the data flow in the other direction, and delay the retransmissions, and hence, local recovery. Consequently, for our scheme, we focus on avoiding these spurious dupACKs from being generated by the MH.

We use ECN sent by the base station to enable the MH to differentiate between a congestion loss and a random loss. The base station maintains a sorted list of unacknowledged TCP sequence numbers. Congestion losses are immediately evident at the base station, as it results in a jump in the sequence number list. The next out-of-order packet arriving at the base station is marked with the ECN bit, and forwarded to the MH. The MH, upon discovering the missing segment, checks to see if the ECN bit is set in the successive segments. If so, it understands that the loss was in the wired network. It then sends dupACKs (normal TCP operation). If the loss was in the wireless channel, the base station has seen segments in

order, and therefore does not set the ECN bit in the successive segments. When the MH notices a loss, and fails to see the ECN bit set in the next segment, it withholds dupACKs, and waits for local recovery. We will describe the scheme in detail with the aid of the following four cases:

- ***Case 1:***

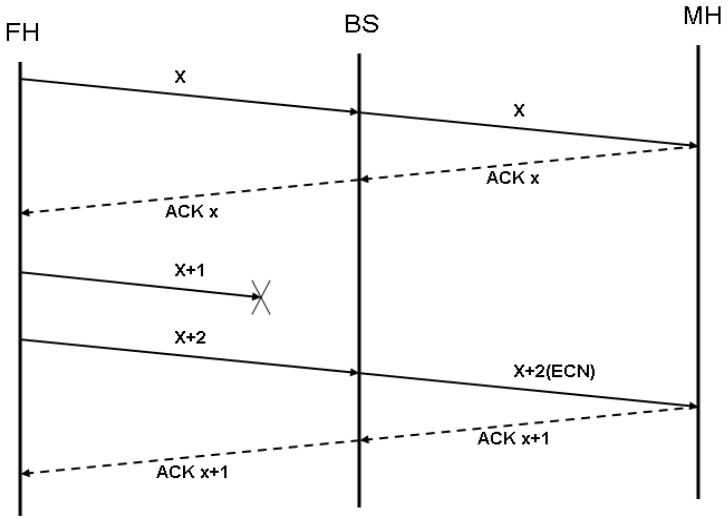


**Figure 1: Handling congestion loss**

Figure 1 depicts two scenarios: a normal error-free scenario and one where a congestion loss occurs. Let us assume that a segment numbered X arrives at the base station without errors. The sequence number X is added to the list at the base station, and the segment will be forwarded by the base station without any modifications. This is the normal case. Now, let us assume that segment numbered X+1 is lost due to congestion, and the next segment X+2 arrives at the base station. The base station notices the loss of X+1, marks the segment X+2 with the ECN bit, and forwards it to the MH. When the MH notices the missing segment X+1, and the ECN bit set in the next segment X+2, it deduces that the loss was due to congestion. Therefore, the MH sends a dupACK.

- ***Case 2:***

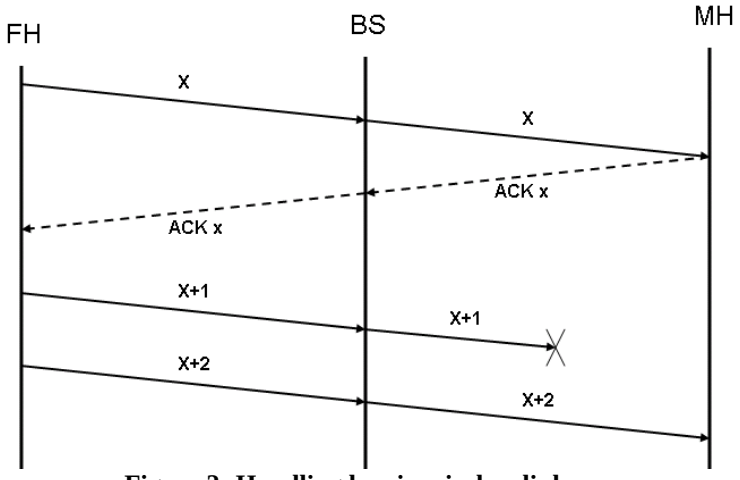


**Figure 2: Handling loss in wireless link**

We will now take a look at the scenario where the loss is in the wireless link. In Figure 2, segment X+1 is lost in the wireless link. When the MH receives segment X+2, it discovers the loss, but unlike the previous case, the ECN bit is not set for X+2. The ECN bit is not set, because the base station has seen all the segments in order. In this scenario, the MH avoids sending spurious dupACKs, waiting for local recovery.

- ***Case 3:***

In cases 3 and 4, we will consider both congestion and wireless losses occur. Firstly, in Figure 3, we look at a case where a congestion loss precedes a wireless loss. The base station notices the congestion loss of X, and sets the ECN bit for all the successive segments. When the MH receives X+2, it sends a dupACK for X, as the ECN bit is set in segment X+2. Once the segment X is retransmitted by the FH, the base station's list is complete, so it stops setting the ECN bit. Segment X is forwarded without the ECN bit set. Hence, the MH waits for local transmission of X+1, and does not send dupACKs for it.

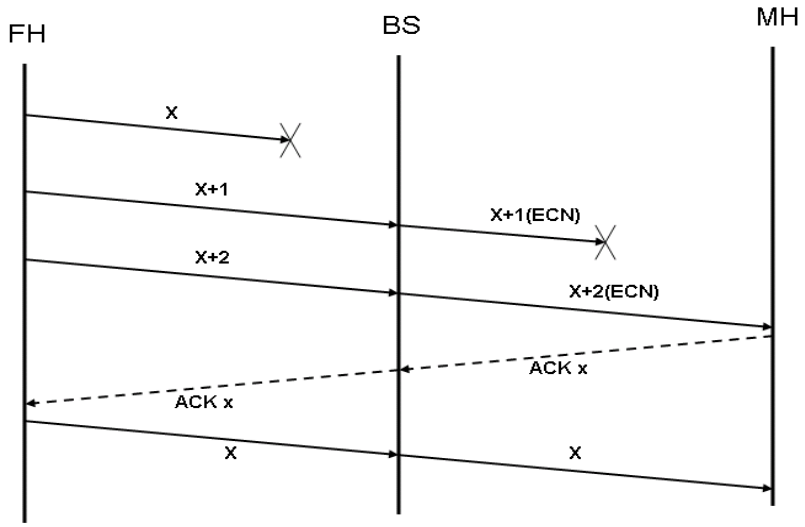


**Figure 3: Congestion loss followed by wireless loss**

- ***Case 4:***

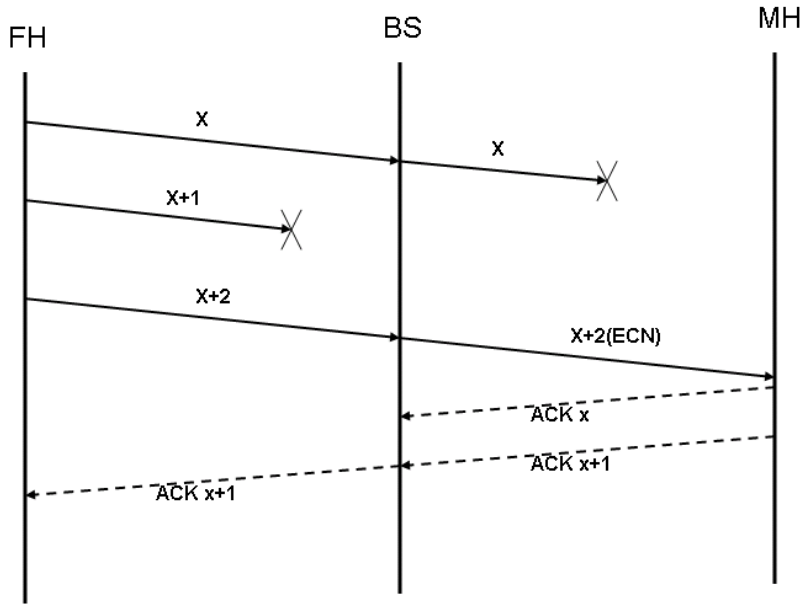


**Figure 4: Wireless loss followed by congestion loss**

If a wireless loss precedes a congestion loss, as shown in Figure 4, the ECN bit is set for X+2, and hence, the MH assumes that X was lost in congestion. This is a special scenario where the base station cannot convey enough information in a single bit to avoid spurious dupACKs. As such, any dupACKs generated for X have to be discarded at the base station.

Since dupACKs are being withheld only for wireless losses, FH's reaction to congestion is not affected, unlike [37]. This provides us the flexibility of allowing the MH to wait for either local recovery, or in the worst case, for the FH to timeout. As we can quickly retransmit packets locally, the latter case occurs rarely. Failure to retransmit locally after multiple tries indicates extremely poor wireless channel conditions. In such cases, sender's timing out and reducing congestion window become imperative. Hence, the changes at MH are kept minimal: it just waits for local retransmission of a packet lost in the wireless link.

### 6.2.2 Handling disconnection and handoff problems

We make an assumption that Mobile IP [28] is being used to allow movement of the MH across subnets. Our scheme uses sender "freezing" as in the case of [17] (see section 5.2.2), but instead of the MH sending ZWA, here, the onus of freezing the FH is on the base station. Several schemes we surveyed including [17], have failed to address the case where the MH is idle for a while, and then immediately gets disconnected. Since TCP traffic is bursty in nature, we cannot assume that there will always be a segment transmitted by FH just before disconnection allowing the base station or the MH to freeze the sender. To overcome this problem, we allow the base station to buffer segments arriving during or after MH's disconnection, and send an ACK for the first segment with zero window advertisement to the sender. Once the MH is reconnected to another base station, Mobile IP [28] is used to transfer the buffered segment(s) to the MH. This technique accounts for the burstiness of TCP traffic.

A base station implementing TCP-ECN maintains a list of sequence numbers flowing across, but does not buffer the segments themselves. Only in the face of an impending disconnection will the base station buffer segments and ACK the first segment with ZWA. The base station can detect impending disconnections through 802.11 disassociation requests [25], or it can monitor the signal strength to detect fading. All the base stations interact with the home and foreign agents implementing Mobile IP. As soon as the MH reconnects to a new base station, the list of sequence numbers is transferred to the new base station, with the aid of home and foreign agents. The new base station unfreezes the FH, and takes over the responsibility of monitoring the sequence numbers. When transferring states between base stations only the list of sequence numbers need be transferred, and not the segments themselves. Therfore, in our new scheme, handoffs are less expensive and the complexity is minimal, unlike others schemes like Snoop [4].

## *6.3 High level design*

### 6.3.1 Flow chart for enhancement at the base station

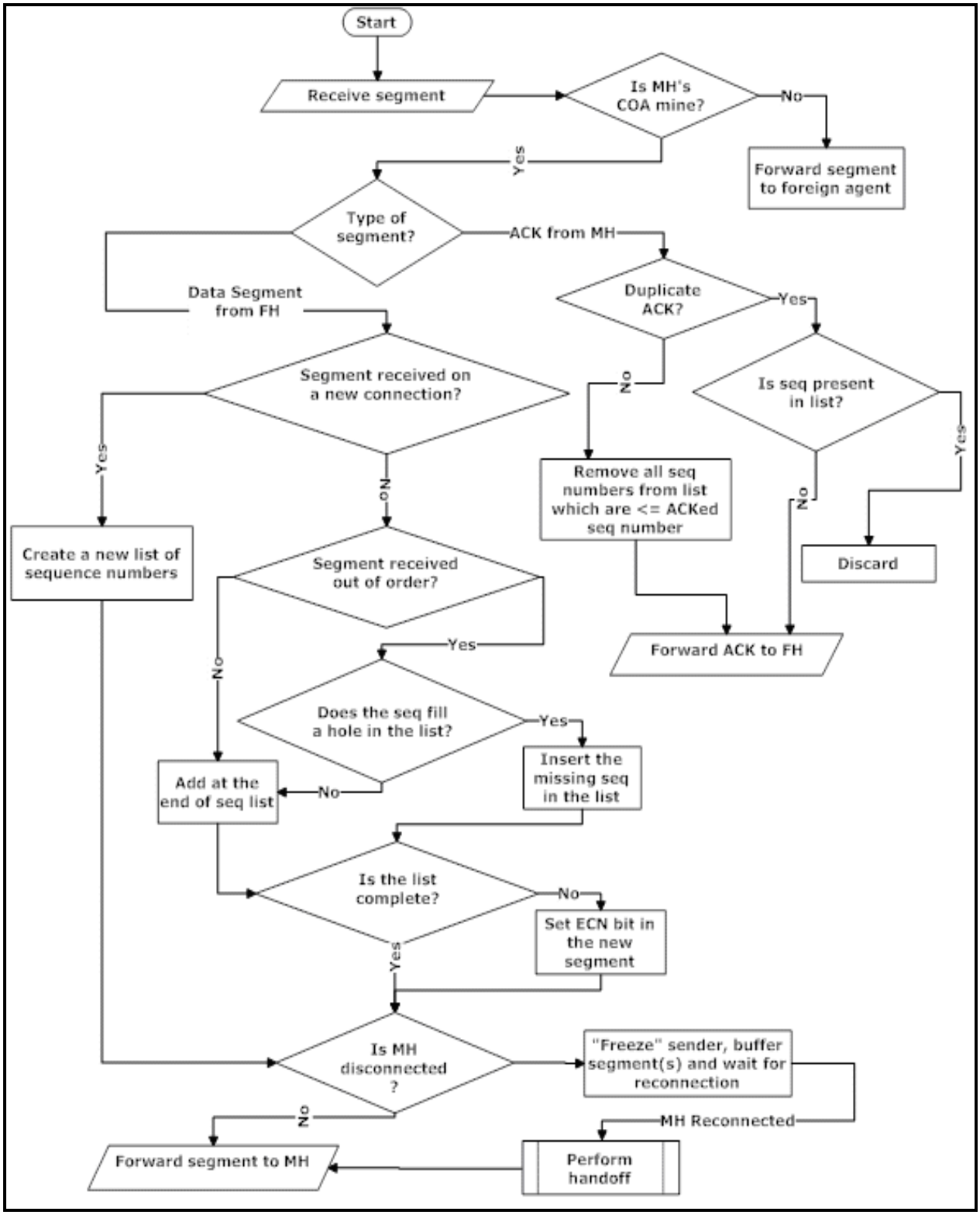


**Figure 5: High level design of the enhancement at the base station**

### 6.3.2 Pseudo code and description

```
1. Receive segment from FH or MH
2. If MH's COA = my COA then
3.    If segment received is a TCP-data segment then  //From FH
4.       If segment is a SYN segment then     //New TCP connection
5.          Create a new list of sequence numbers for the connection
6.       Else                              //Existing TCP connection
7.          If new seq no. >= next expected seq then
8.             Add seq no. at the end of the list for the connection
9.          Else                                    //fills a hole
10.            Insert seq no. at the correct position in the list
11.         End-if
12.      End-if
13.      If list has at least one hole then       //List is incomplete
14.         Set the ECN bit in the TCP header
15.         Re-compute TCP checksum
16.      End-if
17.      If MH is disconnected then
18.         Freeze the FH by sending Zero Window Acknowledgement
19.         Buffer the segment
20.         Wait for handoff and then forward segment to foreign agent
21.      Else
22.         Forward segment to MH
23.      End-if
24.   Else      //Segment received is a TCP-ACK from MH
25.      If new ACK no. = last ACK no. then        //Duplicate ACK
26.         If the ACK no. is present in the list then
27.            //Base station has seen the sequence number, and hence
28.            //can be locally retransmitted
29.            Discard the dupACK
30.         End-if
31.      End-if
32.      Forward ACK to FH
33.   End-if
32. Else       //MH is associated with some other base station
33.    Forward segment
34. End-if
```

Flowchart in Figure 5 and the above pseudo code depict the high level design of the enhancement at the base station. Now, we will describe the flowchart in detail. For our discussion, we assume that TCP data segments received at the base station are sent by the FH, destined to the MH. And, TCP-ACKs are sent by the MH to the FH which are snooped into by the base station. First, we will discuss TCP-data handling at the base station. Every base station implementing TCP-ECN enhancement maintains lists of TCP sequence numbers of all the TCP connections flowing through it. Evidently, only the base station to which an MH is currently associated (or the base station who is the owner of the Care-Of-Address of

the MH) should manage the TCP sequence lists for that MH. This is ensured by using the Care-Of-Address (COA) of the MH. A base station whose COA is not the same as the MH's COA simply forwards the segment to the foreign agent of the MH. Otherwise, sequence numbers of segments received on the connection are compared with the sequence numbers in the list by the base station.

If the data segment is received on a new connection, a new list of unacknowledged sequence numbers is created, and initialized. Otherwise, the sequence number of the new segment is checked to see if it is in order (the next expected sequence number). If the new sequence number is the next expected, it is added to the end of the sequence list. If not, the new sequence number could either fill a hole in the list, or is creating a new hole. For example, if the current list at the base station is as follows (segment size assumed to be 500 bytes): 500, 1000, 1500, 2500, and 3000. Let the newly arriving sequence number be 2000. Clearly, it is out of order, and is filling a hole in the list. In such cases, the new sequence is inserted at the correct position (list is kept sorted). Instead, if the new sequence number is 4000, it is out of order, but is creating a new hole in the list. Such sequences will be inserted at the end of the list.

Before forwarding the data segment to the MH, the list is checked for completeness. If the list is not complete, it is an indication of either congestion losses, or an abnormal forwarding delay in the wired network. The base station indicates this to the MH by setting the ECN bit in the out-of-order segment, and re-computing TCP checksum. In addition, MH's disconnection status is checked. If the MH is disconnected, the base station buffers the segment, and waits for the handoff to complete before forwarding the segment to the foreign agent. In addition, the base station "freezes" the FH sender by sending a zero window acknowledgement (ZWA). If there is no impending disconnection of the MH, the segment is forwarded to the MH.

We will now focus our attention on TCP-ACK handling at the base station. When a TCP ACK sent by the MH is received by the base station, it checks to see if the ACK is a duplicate (same as the last seen). If it is a new ACK, the base station removes all the sequence numbers from the list that are less than the ACK number (TCP ACKs are cumulative). Instead, if the received ACK is a dupACK, the base station first confirms that it has not seen the segment with the dupACK number, and then forwards the dupACK to FH. But, if it has seen the segment with the dupACK number, it simply discards the dupACK (loss was in the wireless link, and can be recovered locally, refer to case 4 shown in Figure 4).

### 6.3.3 Flowchart for enhancement at MH

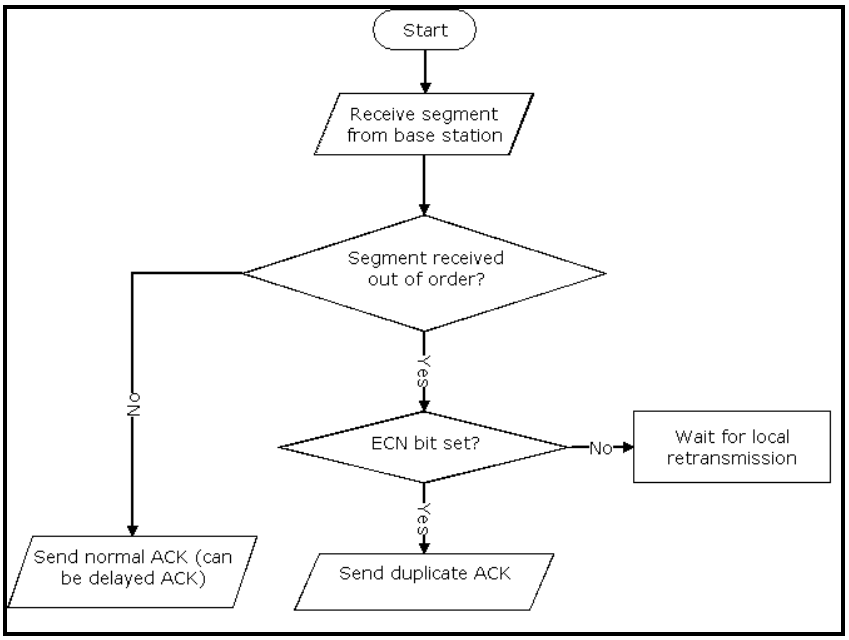


**Figure 6: High level design of the enhancement at the MH**

### 6.3.4 Pseudo code and description

```
1. Receive segment from base station
2. If new seq no. <> next expected seq no. then
3.    If ECN bit set is in the received segment then
4.       Withhold dupACK
5.       Wait for local retransmission of lost segment
6.    Else
7.       Send dupACK to FH
8.    End-if
9. Else
10.   Adhere to normal TCP operation
11. End-if
```

Design of the enhancement to be implemented at the MH is shown in Figure 6 and the pseudo code in this section. Under normal TCP operation, whenever a segment is received out of order, the TCP receiver (in our case, the MH) sends a dupACK to the TCP sender (FH). With TCP-ECN enhancement, every out-of-order segment received by the MH is checked to see if the ECN bit is set by the base station. If the ECN bit is set, MH sends dupACKs to the FH, as the loss was in the wired network. Otherwise, it withholds the dupACKs, and waits for local recovery. Under all other circumstances, we adhere to normal TCP behavior. Evidently, the amount of change required at the MH is minimal.

## *6.4 Innovation*

The main contribution of our scheme is to solve the problem of spurious dupACKs generated in TCP-unaware Link layer schemes that use local recovery. Our scheme uses TCP awareness, but is unlike other TCP-aware schemes like Snoop [4]. In Snoop, dupACKs sent

by the MH are used only to indicate a wireless loss, and are eventually discarded by the base station. We intend to avoid the generation of unnecessary dupACKs altogether, thus saving wireless bandwidth.

Another contribution of our scheme is to handle burstiness of TCP transmission in the face of long disconnections. Most schemes concentrate on solving the disconnection problem assuming continuous transmission of data. A more challenging scenario is when the MH is idle for a while before getting disconnected, and the FH transmits during the disconnection. Our scheme aims to solve this issue. Handoff complexity is highly reduced in the new scheme, as only the list of sequence numbers need to be transferred between base stations.

# Chapter 7. SIMULATION OF TCP-ECN

## *7.1 Simulation details*

We used ns2, version 2.29 [24], for simulating and testing TCP-ECN for wired-cum-wireless networks, and to verify its performance in high BER and long disconnection scenarios. For an introduction to ns-2, please refer to Appendix A. A new ns component was created, called *LLTcpEcn*. This component replaces the Link layer component at the base station. LLTcpEcn is responsible for processing only TCP packets and encapsulates TCP-ECN's base station functionality (refer to the flow chart in Figure 5). Minor modifications to *tcp-sink.cc* were made to refect the MH functionality of TCP-ECN (refer to flow chart in Figure 6).

The current version of ns2 does not support receiver driven flow control in TCP. As such, we had to modify *tcp-reno.cc* to handle sender freezing (ZWA). We achieved this by introducing a "freeze" bit in *hdr_flags* structure. We have also added an ECN bit to the same structure as required by our scheme. Both wireless and wired losses were introduced during testing by attaching error modules to the links (done in the TCL script). A new header structure called *hdr_tcpecn* was created to store per-packet information.

TCP-ECN enables the MH to distinguish between congestion and wireless losses. In consequence, the MH can withhold unnecessary dupACKs generated for wireless losses. An efficient mechanism is required at the base station to ensure timely local recovery to avoid triggering a timeout at the FH sender. In TCP-ECN we allow 802.11 MAC to return packets to the Link layer after it has given up retransmitting them. This *MAC callback* mechanism eliminates the need to buffer packets at the base station. The Link layer can place a ceiling on the number of retransmission tries allowed for a packet (for our simulation, three), and accordingly perform local recovery. The current number of Link layer retransmission tries for a packet is stored in the *hdr_tcpecn* part of its ns2 packet header.
*Note*: The Link layer retransmissions that we mention here are different from the 802.11 MAC retransmissions. For example, if the maximum number of MAC retransmissions is 7, and the maximum number of Link layer retransmissions is 3, in the worst case, the packet will be retransmitted 7*3=21 times.

In the next two sections, we will describe the algorithms used to deal with the high BER and disconnection scenarios in greater detail.

### 7.1.1 Details of algorithm used for handling high BER

A new component called LLTcpEcn is implemented in C++, and it resides at the base station. Figure 7 depicts the ns2 networking stacks at FH, base station and MH. The flowchart shown in Figure 5 is followed closely in the implementation of LLTcpEcn. TCP data from FH is processed in the function *TcpEcn::FHdata*, and ACKs from the MH are handled in the function *TcpEcn::MHack*. LLTcpEcn maintains one *TcpEcn* object for every connection between FH and MH. Each of these TcpEcn objects maintains a circular list of unacknowledged TCP sequence numbers for the connection. We do not store every sequence

number in the list. Instead, in order to reduce memory requirement, we store only the start and end sequence numbers of contiguous blocks of TCP sequence numbers. For example, let us assume that the base station sees segments with sequence numbers 500, 1000, 1500, 2500, and 3000, each of length 500 bytes. Instead of storing five sequence nodes in the list, the base station stores only two nodes. The first node has start sequence as 500, and its end sequence is 1500 (first contiguous block). The second node's start sequence is 2500, and the end sequence is 3000 (second contiguous block). A new node is created only when the newly received sequence number is not the next expected. Otherwise, the end sequence of the last node is updated.
Note: Ns2 sequence numbers are consecutive, and are independent of the length of segments. This allows us to use the aforementioned data structure as we need not store per-sequence information, like segment length, in the list.

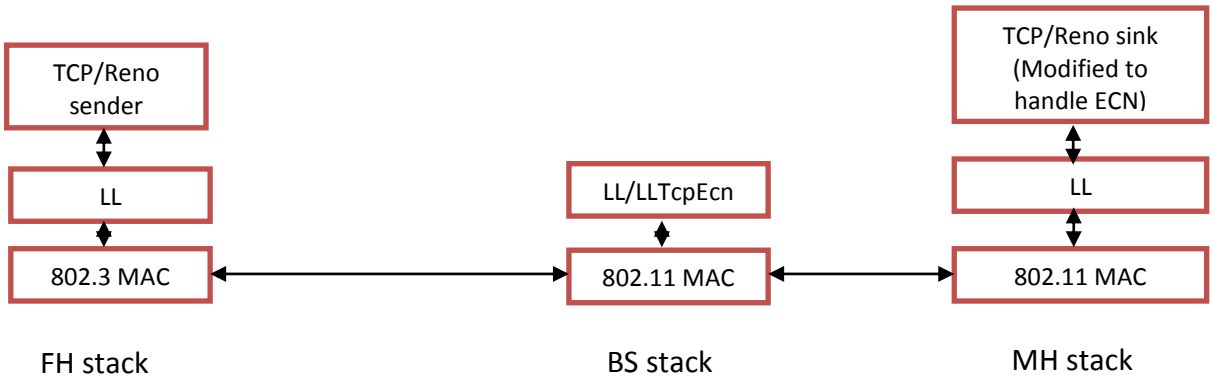


**Figure 7: Snapshot of the TCP/IP stack at FH, BS and MH**

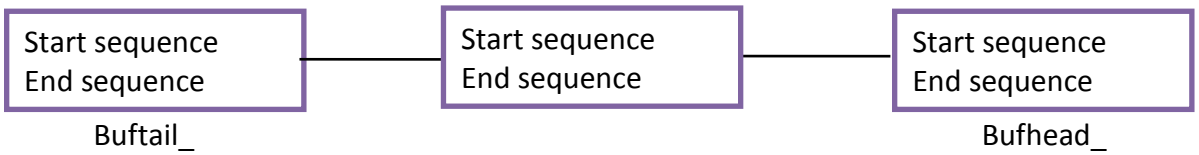


**Figure 8: Sequence list structure**

We use ns2's MAC layer callback functionality to return packets to LLTcpEcn, when their transmission repeatedly fails on the wireless link. The function *TcpEcn::retransmit* is invoked upon callback. This function determines if a packet can to be retransmitted again depending on the number of Link layer retries allowed. A packet to be retransmitted is simply passed to the MAC again, after incrementing the number of Link layer retries in the packet's header.

A global variable *tcpecn_enabled* is used to control MH-TCP's behavior to the ECN bit in TCP headers. If this variable is set, MH-TCP sends dupACKs whenever it notices a missing segment and the ECN bit set in the following segments. Instead, if the ECN bit is not set in the following TCP segments, MH withholds the dupACKs. To disable TCP-ECN enhancement, *tcpecn_enabled* is turned off.

### 7.1.2 Details of algorithm used for handling disconnections

We use ns2's Mobile IP implementation for handling disconnection scenarios. In the following discussion, we will refer to the home agent as HA, and foreign agent as FA. The HA initially maintains the sequence lists for all the connections associated with an MH. When the MH moves away from HA and into the range of FA, all its corresponding sequence lists have to be transferred to FA, so that the FA can continue monitoring the sequence numbers. This is achieved by using Mobile IP's first *registration request* and *registration reply* messages. Upon receiving the first registration request from the MH after reconnection (forwarded by the FA), the HA attaches all the sequence lists associated with that MH in the corresponding registration reply packet. Subsequently, when the FA receives the first registration reply from the HA, it updates its list cache with the sequence lists sent by the HA in the reply packet. In this way the current states of sequence numbers in each of MH's connections are transferred to the FA.
*Note*: In ns2, the base stations are HA and FA themselves. There are no separate Mobile IP capable routers.

As we had mentioned earlier, receiver driven TCP flow control is not implemented in ns2. As such, we have introduced a new flag to enable the sender to freeze its congestion window upon intimation by the base station. When the base station (HA) determines that the MH is moving away from its range, it sets the freeze bit to 1 in the ACK sent from the MH to the FH. We have made a minor modification to the TCP sender at the FH to handle the freeze bit. When the sender notices the freeze bit set in an incoming ACK, it cancels its current retransmission timer, and does not alter its congestion window. In addition, it refrains from sending any more segments. This will continue until it receives an ACK with the freeze bit reset.

In ns2, the base station does not know the exact time at which the MH starts moving out of its range. Hence, we have used the "receive power" of a packet to simulate this scenario. Wireless PHY of the base station (HA) discards any packet whose receive power is less than a threshold called *RXThresh* (default value in ns2 is $3.652 \times 10^{-10}$W). At the link layer, TCP-ECN module at the base station checks the receive power of a successfully received packet. If this value is very close to RXThresh, then a disconnection is anticipated in the near future. A new threshold called *FZThresh* (default value set to $0.5 \times 10^{-10}$W) is used to decide whether freezing should be performed by the base station.

```
Packet received power < RXThresh + FZThresh
```

Whenever the above condition is true for an ACK packet received from the MH, the base station freezes the FH by setting the freeze bit in the ACK before forwarding it. The receive power value at which the base station freezes the FH has to be chosen very carefully. For a relatively large value of FZThresh early freezing can occur, thus reducing throughput. In contrast, for a relatively low value freezing may fail to occur. Upon reconnection, the last ACK segment is resent by the FA with the freeze bit reset. This enables the FH TCP to resume transmission.

Start of disconnection period is computed as the time when the wireless PHY at the HA discards a packet for the first time due to insufficient received energy (packet received power

< RXThresh). End of disconnection period is the time when the MH has successfully reconnected to the FA, and is ready to receive TCP packets again.

## *7.2 Testing with only high BER*

In order to better evaluate our new scheme, we have compared its performance with not only TCP Reno's performance, but also with that of Snoop [4]. Snoop has proven in the past to be better performing than most wireless TCP enhancements. In addition, we believe that TCP-ECN and Snoop are similar in terms of their scope: both operate at the base station, and both use local retransmissions to recover from wireless losses. Furthermore, both Snoop and TCP-ECN improve TCP performance for FH to MH data transfer scenario. Hence, we choose Snoop as a fair scheme to compare TCP-ECN with.

We have used the implementation of Snoop available with ns2 distribution for the purpose of testing. Implementation of Snoop in ns-2 has not been designed to work with 802.11 networks. We had to make minor modifications to Snoop code (both TCL and C++) in order to satisfy WLAN requirements. Firstly, we used the wireless error model, instead of wired error model (originally used) for simulation. Secondly, Snoop code has been modified to bypass processing of any packets other than TCP and ACK packets. This allows other packet types, like ARP requests and responses, to be handled normally without the intervention of Snoop module.

### 7.2.1 Simulation scenario: Single source and single receiver

#### 7.2.1.1 Simulation setup

For the first set of tests, we use a simple topology with a single source (FH), and a single receiver (MH), connected through a base station (access point). Figure 9 depicts the topology we used to run our simulation scripts. Disconnections were not introduced in this set of tests.

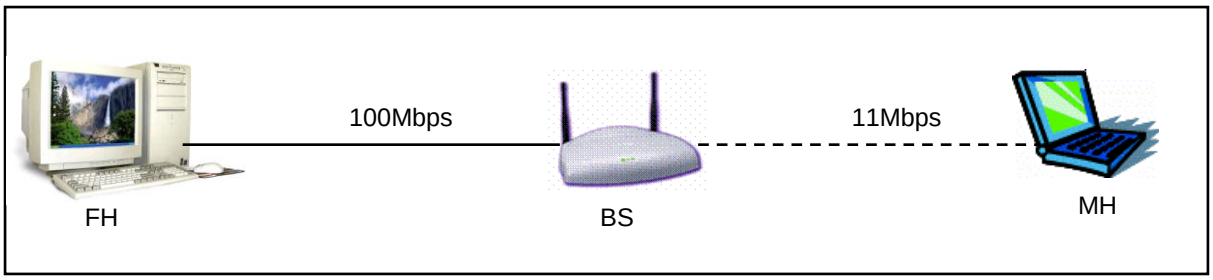


**Figure 9: Topology used for simulation: 1 source and 1 receiver**

#### 7.2.1.2 Results and analyses

- **TCP sequence number progression**
  - The graph is presented in Figure 10. The simulation parameters for this test case were
    - ***Wireless error rate:*** 0.2
    - ***Wireless bandwidth:*** 11Mbps
    - ***Wired bandwidth:*** 100Mbps
    - ***Wired network delay:*** 500ms
    - ***Simulation time:*** 250 seconds
    - ***No wired network errors***

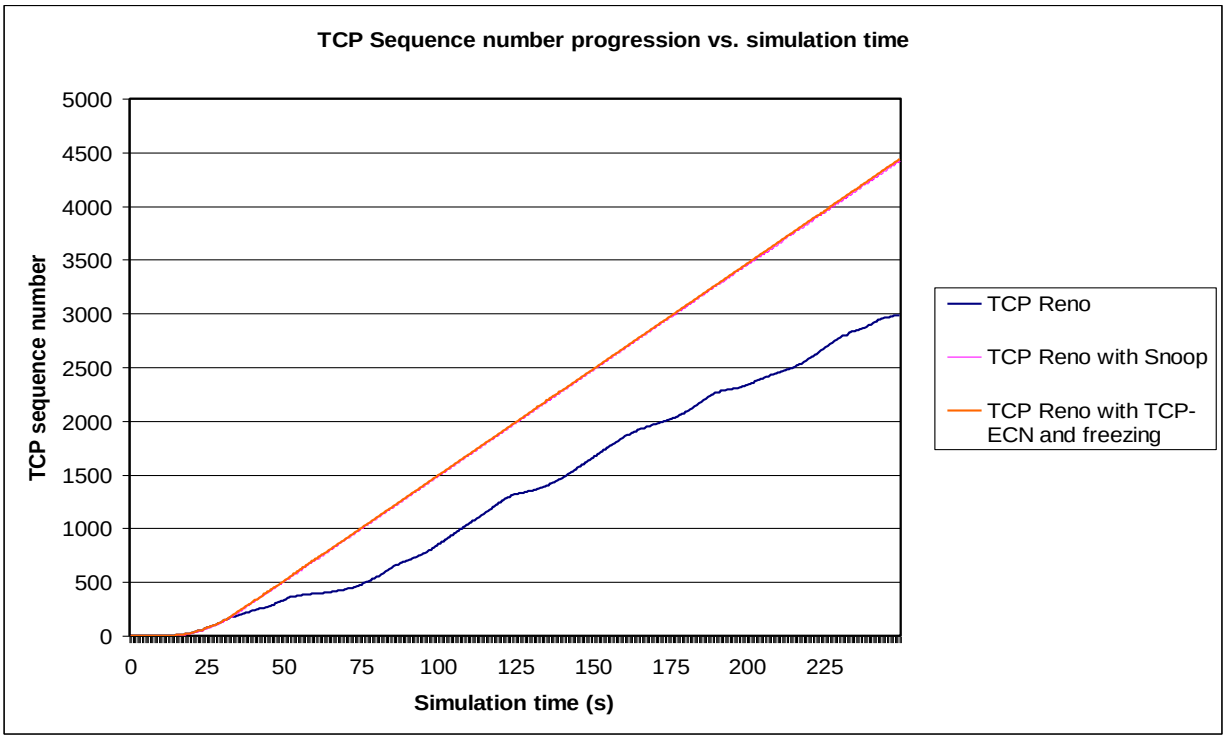


**Figure 10: TCP sequence number progression (1 source, 1 receiver)**

Figure 10 shows the TCP sequence number progression as measured at the TCP receiver (MH). Steady increase in sequence numbers is an indication of better insulation of FH-TCP from wireless losses. The curves for TCP-ECN and Snoop coincide with each other. Both these schemes show marked performance improvement over TCP Reno.

- **TCP throughput vs. wireless error rate**
  - The simulation parameters for this test case were
    - ***Wireless bandwidth:*** 11Mbps
    - ***Wired bandwidth:*** 100Mbps
    - ***Wired network delay:*** 500ms
    - ***Simulation time:*** 250 seconds
    - ***No wired network errors***

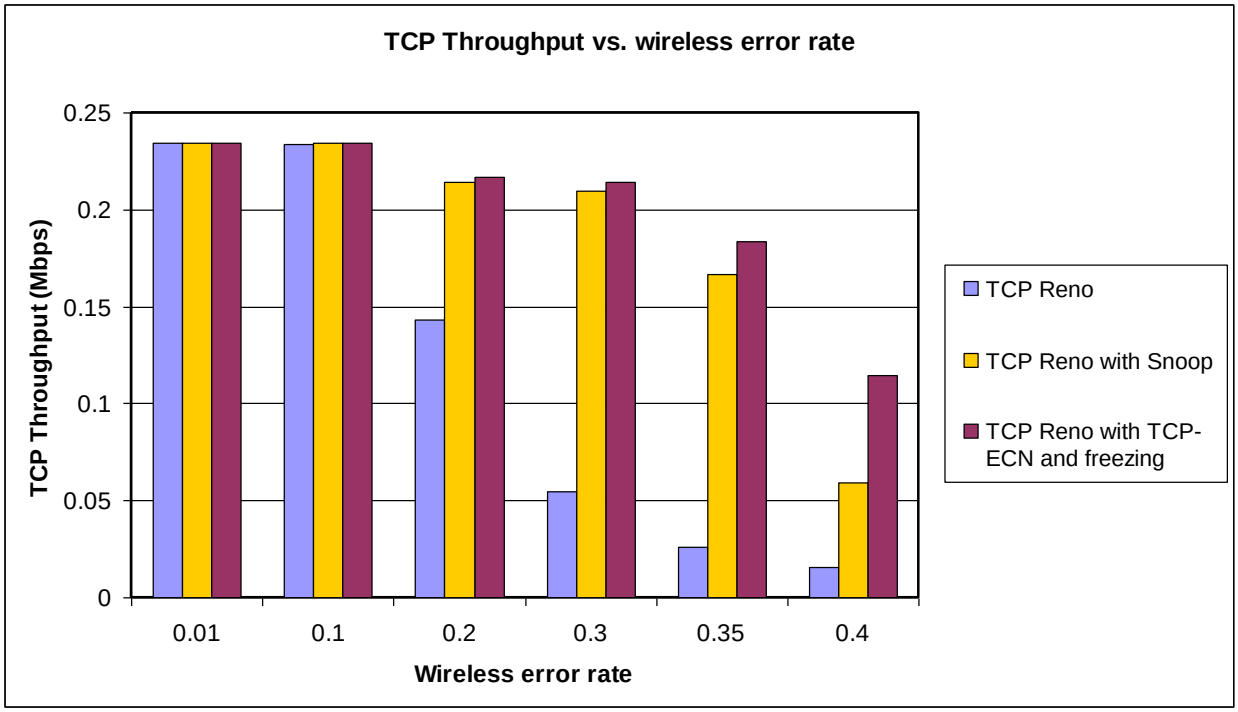


**Figure 11: TCP throughput vs. wireless error rate (1 source, 1 receiver)**

When we varied the wireless error rate, and measured TCP throughput at MH, we obtained the graph shown in Figure 11. Both TCP-ECN and Snoop perform better than TCP Reno for all error rates. But, as the error rate increases, base station's buffering capacity becomes a bottleneck for Snoop, as the number of packets to be buffered increases proportionally. In addition, the number of dupACKs sent by the MH increases with an increase in wireless error rate. All these dupACKs compete with local retransmissions (802.11 links are half duplex) for wireless bandwidth, thus reducing effective throughput. TCP-ECN eliminates unnecessary dupACKs allowing faster local recovery. Consequently, TCP-ECN performs better than Snoop for higher error rates.

- **TCP throughput vs. wired network delay**
  - The simulation parameters for this test case were
    - ***Wireless error rate:*** 0.2
    - ***Wireless bandwidth:*** 11Mbps
    - ***Wired bandwidth:*** 100Mbps
    - ***Simulation time:*** 250 seconds
    - ***No wired network errors***

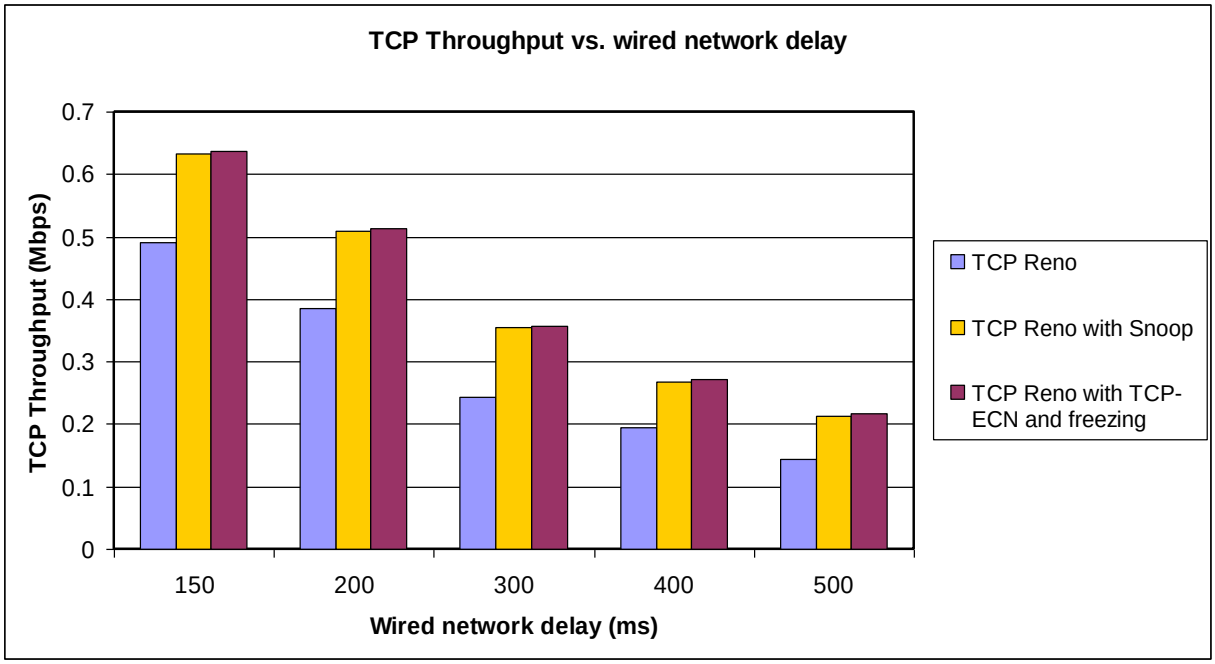


**Figure 12: TCP throughput vs. wired network delay (1 source, 1 receiver)**

Wired network can involve several hops, and increase in wired network delay has a direct consequence on TCP's RTT and RTO calculations. Figure 12 shows the performance of the three schemes with varying wired network delay. As expected, an increase in wired network delay causes a proportional decrease in throughput with all three schemes. But, both TCP-ECN and Snoop perform consistently better than TCP Reno, in spite of the variation in RTO at the FH sender.
*Note*: Decrease in RTO can cause early retransmission from sender TCP hindering local retransmissions.

- **TCP Throughput vs. wired loss rate**
  - The simulation parameters for this test case were
    - ***Wireless error rate:*** 0.2
    - ***Wireless bandwidth:*** 11Mbps
    - ***Wired bandwidth:*** 100Mbps
    - ***Wired network delay:*** 500ms
    - ***Simulation time:*** 250 seconds

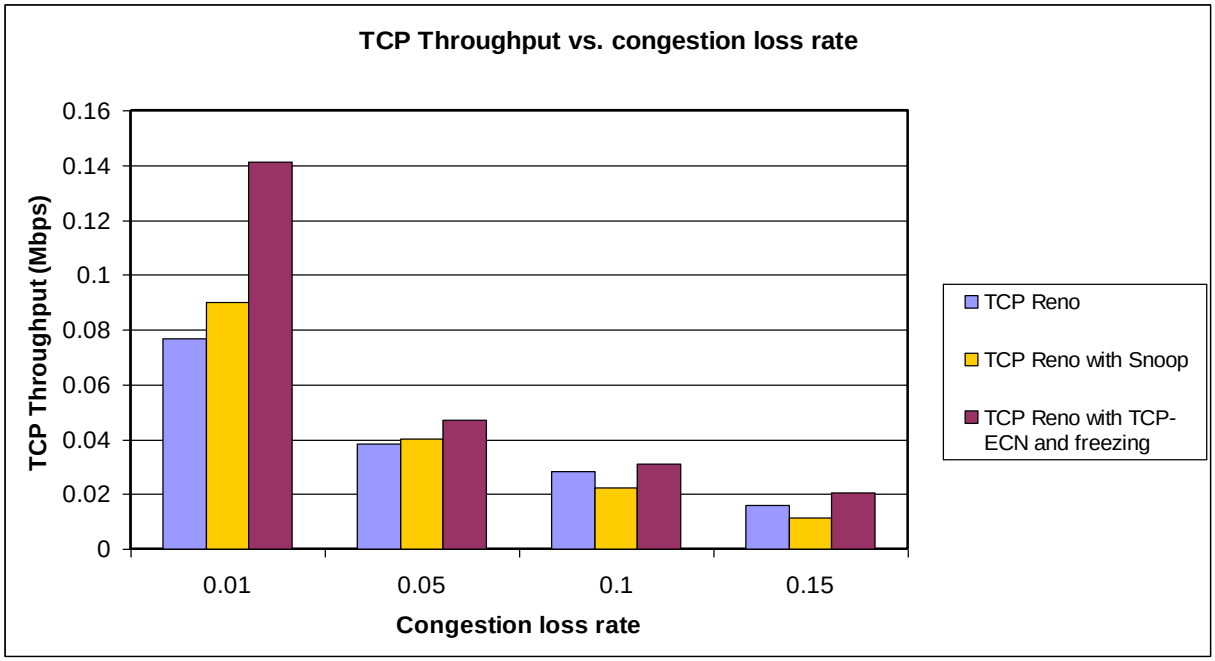


**Figure 13: TCP throughput vs. congestion loss rate (1 source, 1 receiver)**

Figure 13 presents the graph for the simulation scenario with both wired and wireless losses. We keep the wireless error rate as a constant, and vary wired error rate (congestion loss rate). Higher congestion deteriorates TCP performance with all three schemes. However, TCP-ECN performs the best among the three schemes. Snoop suffers from high buffer occupation for higher congestion losses. When a packet is lost in the wired network, all the successive packets have to be buffered at the base station until the FH retransmits the lost packet. If this situation is coupled with wireless losses, buffer requirement at the base station becomes very high for Snoop. In addition, wireless losses cause unnecessary dupACKs in wireless link. These reasons cause Snoop to perform slightly worse than TCP-ECN and TCP Reno for high wired error rates.

For both TCP-ECN and Snoop, an increase in congestion loss rate decreases the advantage of local recovery. The base station does not have packets to locally retransmit and has to wait for the FH to retransmit. As such, performances of these schemes are comparable to that of normal TCP for higher congestion loss rates.

### 7.2.2 Simulation scenario: Two sources and two receivers

#### 7.2.2.1 Simulation setup

We added a source and a receiver to the topology shown in Figure 9 to create the topology shown in Figure 14. Two TCP connections were maintained (FH1-to-MH1 and FH2-to-MH2), and packets from both connections were processed by a single base station. All three schemes: TCP-ECN, Snoop and TCP Reno, were evaluated for this setup.

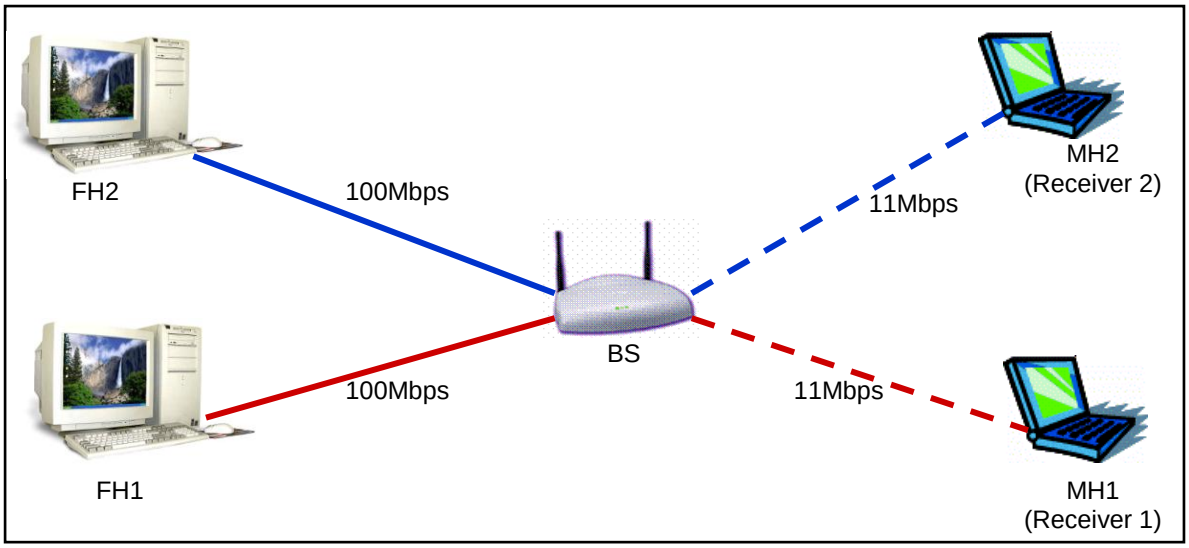


**Figure 14: Topology for simulation (2 sources and 2 receivers)**

### 7.2.2.2 Results and analyses

- **TCP sequence number progression**
  - The simulation parameters for the test case were:
    - ***Wireless bandwidth:*** 11Mbps
    - ***Wired bandwidth on both links:*** 100Mbps
    - ***Wired link delay:*** 500ms
    - ***Simulation time:*** 250 seconds
    - ***Wireless error rate:*** 0.2
    - ***No wired network loss***

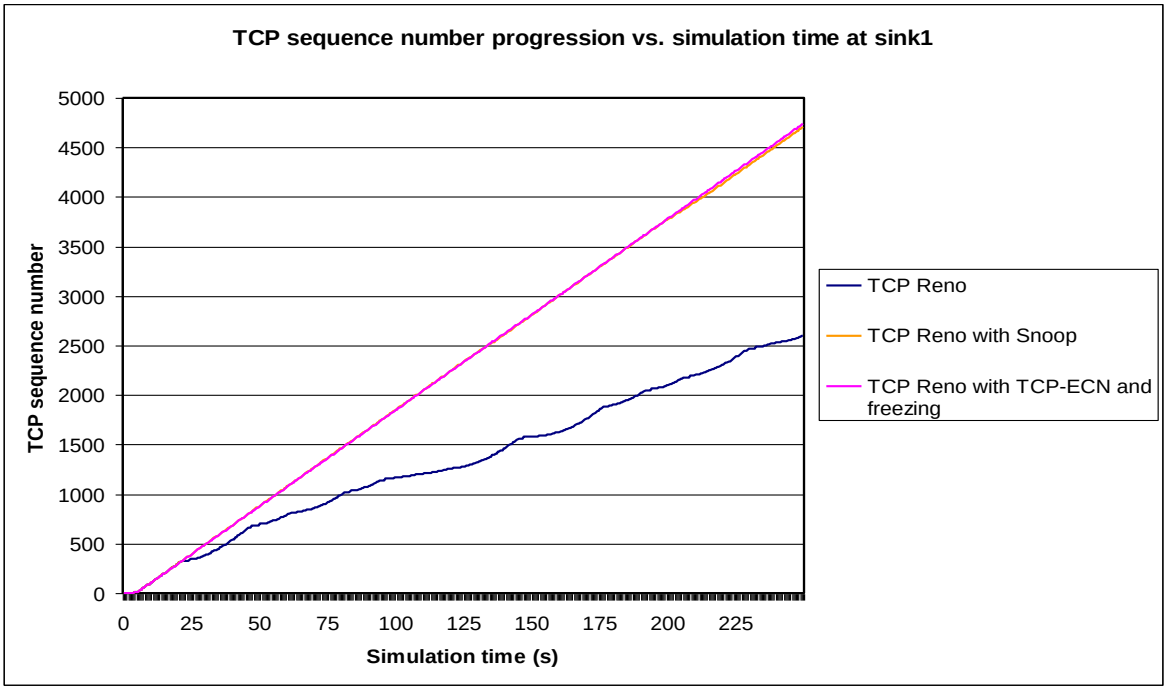


**Figure 15: TCP sequence number progression at sink1 (2 sources, 2 receivers)**

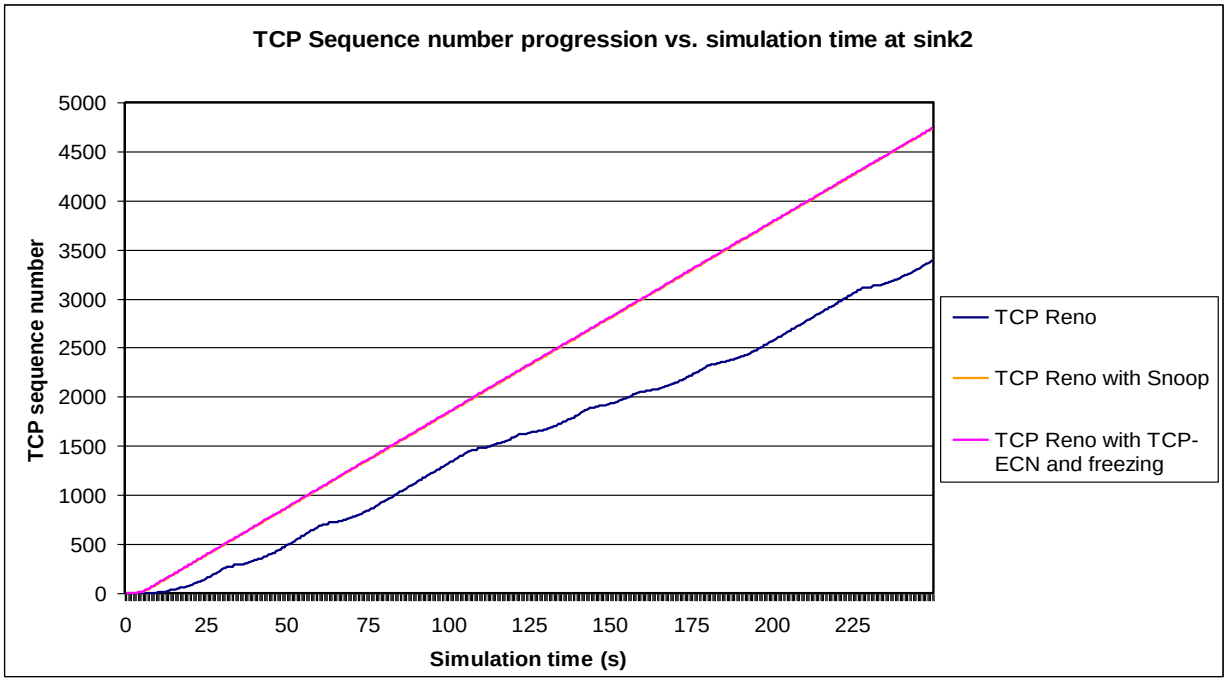


**Figure 16: TCP sequence number progression at sink2 (2 sources, 2 receivers)**

Figure 15 and Figure 16 show the sequence number progression measured at MH1 and MH2 respectively. Curves for TCP-ECN and Snoop coincide with each other in both cases. When compared with Figure 10, there is little variation in TCP-ECN and Snoop's performances. Similar to the results from the previous setup, both TCP-ECN and Snoop perform better than TCP Reno.

- **TCP throughput vs. wireless error rate**
  - The simulation parameters for the test case were:
    - ***Wireless bandwidth:*** 11Mbps
    - ***Wired bandwidth on both links:*** 100Mbps
    - ***Wired link delay:*** 500ms
    - ***Simulation time:*** 250 seconds
    - ***No wired network errors***

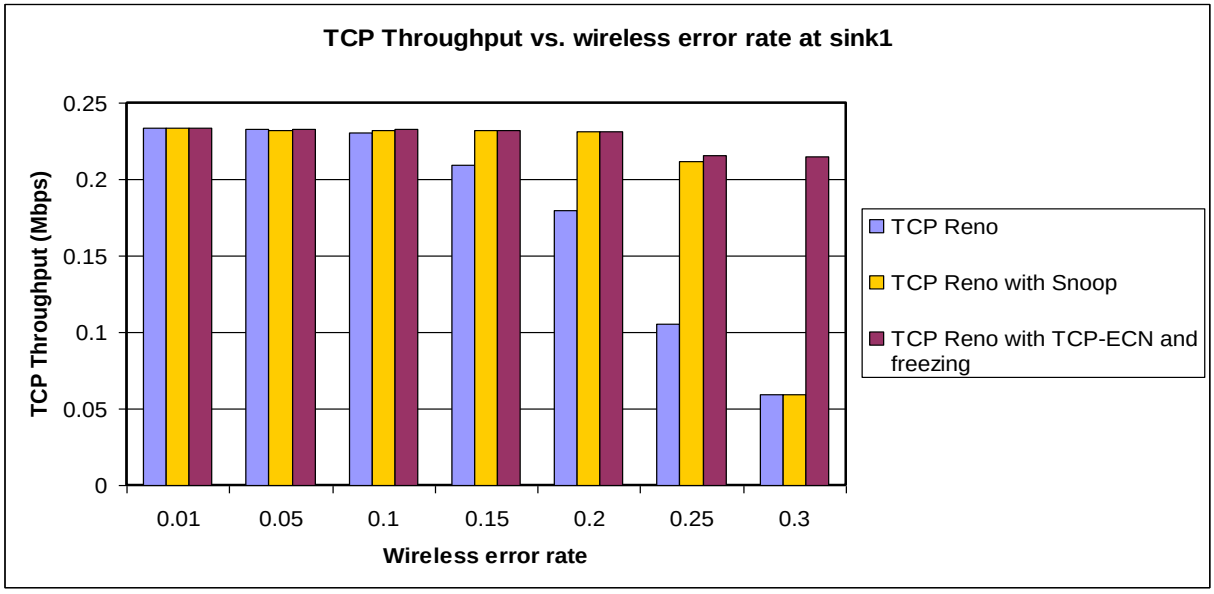


**Figure 17: TCP throughput vs. wireless error rate at sink1 (2 sources, 2 receivers)**

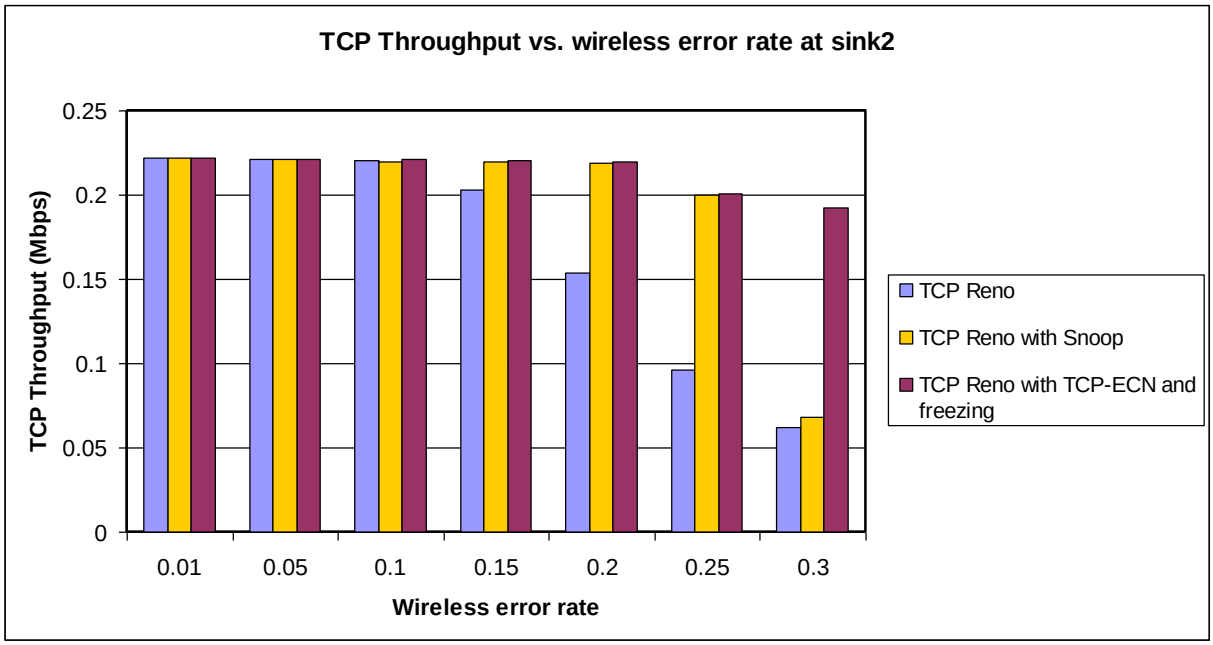


**Figure 18: TCP throughput vs. wireless error rate at sink2 (2 sources, 2 receivers)**

TCP throughput measurements for varying wireless error rates were measured at MH1 and MH2, and are shown in Figure 17 and Figure 18 respectively. Since the base station's memory requirement is doubled compared to the single-source-single-receiver scenario, Snoop's performance reduces drastically for higher error rates. Yet another problem with Snoop that compounds the previous problem is due to spurious dupACKs consuming wireless bandwidth. TCP-ECN performs consistently better than both TCP Reno and Snoop for higher error rates.

## *7.3 Testing with only disconnections*

### 7.3.1 Simulation setup

We introduced another base station to the setup shown in Figure 9. The topology used for this set of tests is shown in Figure 19. MH moves from BS1's range to BS2's range. Both the base stations implement Mobile IP. BS1 is the HA, and BS2 is set up as the FA. BS1 is placed at co-ordinates (1, 2, 0), and BS2 is placed at (650, 600, 0), in order to have the worst case distance (about 8 km) between the two. No wireless losses or wired network losses were introduced. Disconnection time was varied by altering the speed of the MH. We measure disconnection time as explained in section 7.1.2.

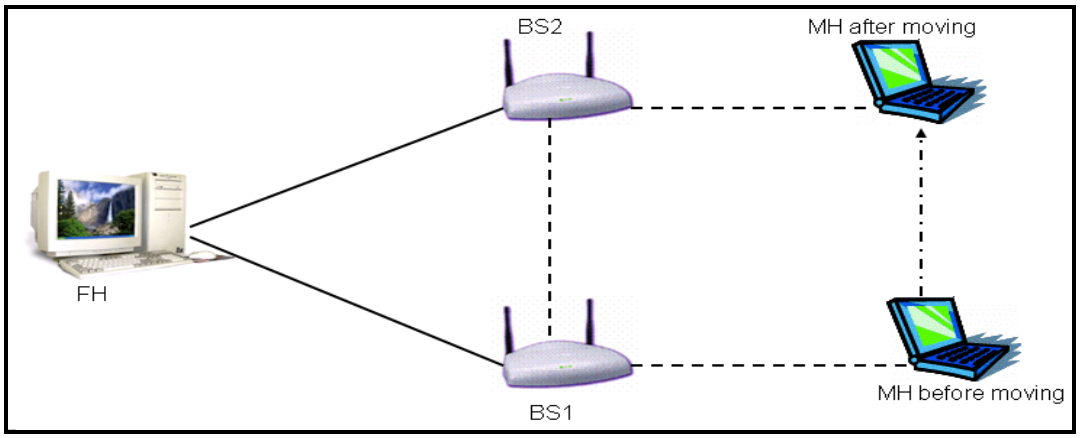


**Figure 19: Topology used for simulation of disconnection scenarios**

### 7.3.2 Results and analyses

- **TCP sequence number progress**
  - The simulation parameters were set up as:
    - ***Wireless bandwidth:*** 11Mbps
    - ***Wired bandwidth:*** 100Mbps
    - ***Wired network delay:*** 500ms
    - ***Simulation time:*** 150 seconds
    - ***No wired network errors***
    - ***Disconnection time:*** 31.4 sec
    - ***Wireless error rate:*** 0

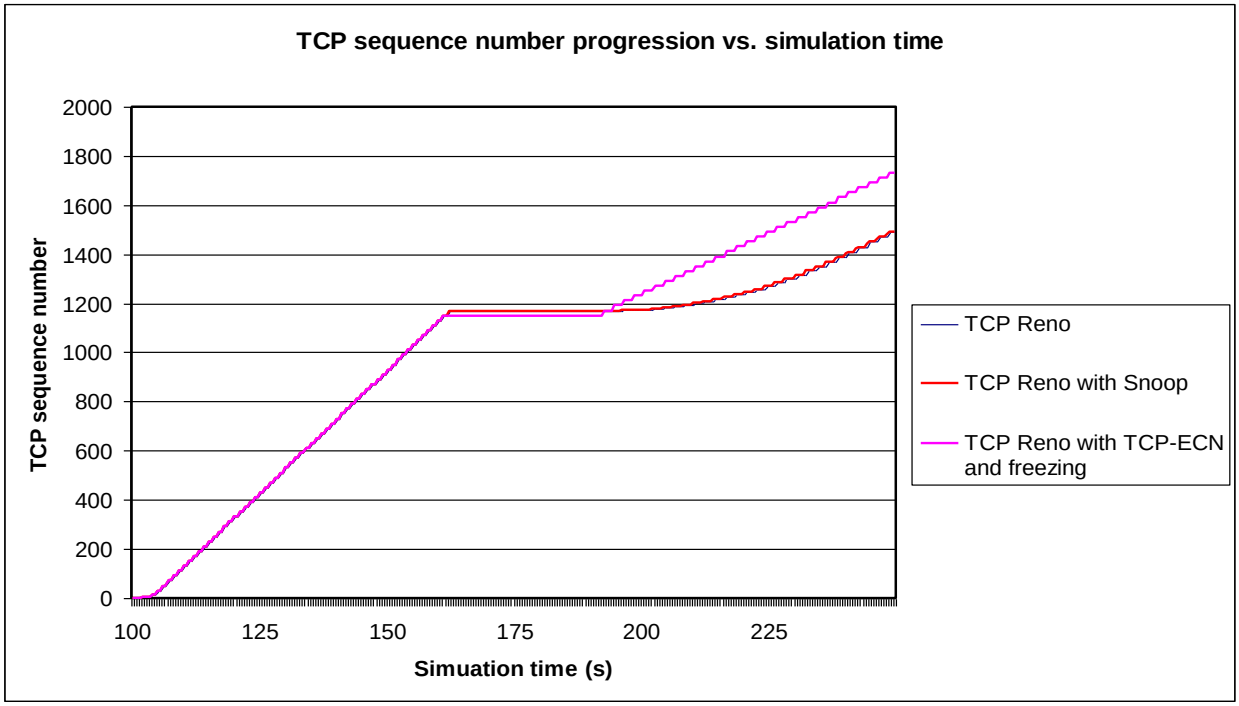


**Figure 20: TCP sequence number progression with disconnections**

Figure 20 depicts the sequence number progression as measured at the MH for a disconnection period of 31.4s. The curves for TCP Reno and Snoop coincide with each other. Snoop's implementation in ns2 does not handle disconnections. Hence, it behaves like TCP Reno when there are no wireless losses. Sequence numbers increase linearly after reconnection when TCP-ECN is used, whereas they increase slowly for TCP Reno and Snoop. This is because the FH's congestion window is reduced when packets are lost during the disconnection period. Since TCP-ECN freezes the FH just before the disconnection, FH's congestion window is not affected during disconnection. FH resumes from where it left off, after reconnection. Although, note that disconnection happens slightly earlier for TCP-ECN. The reason for this behavior is that the base station freezes FH when the receive power of the incoming ACK from MH is marginally greater than RXThresh (refer to section 7.1.2). This causes the FH to stop sending segments slightly earlier than the actual disconnection.

- **Throughput vs. disconnection time**
  - The simulation parameters were setup as:
    - ***Wireless bandwidth:*** 11Mbps
    - ***Wired bandwidth:*** 100Mbps
    - ***Wired network delay:*** 500ms
    - ***Simulation time:*** 150 seconds
    - ***No wired network errors/ wireless error rate: 0***

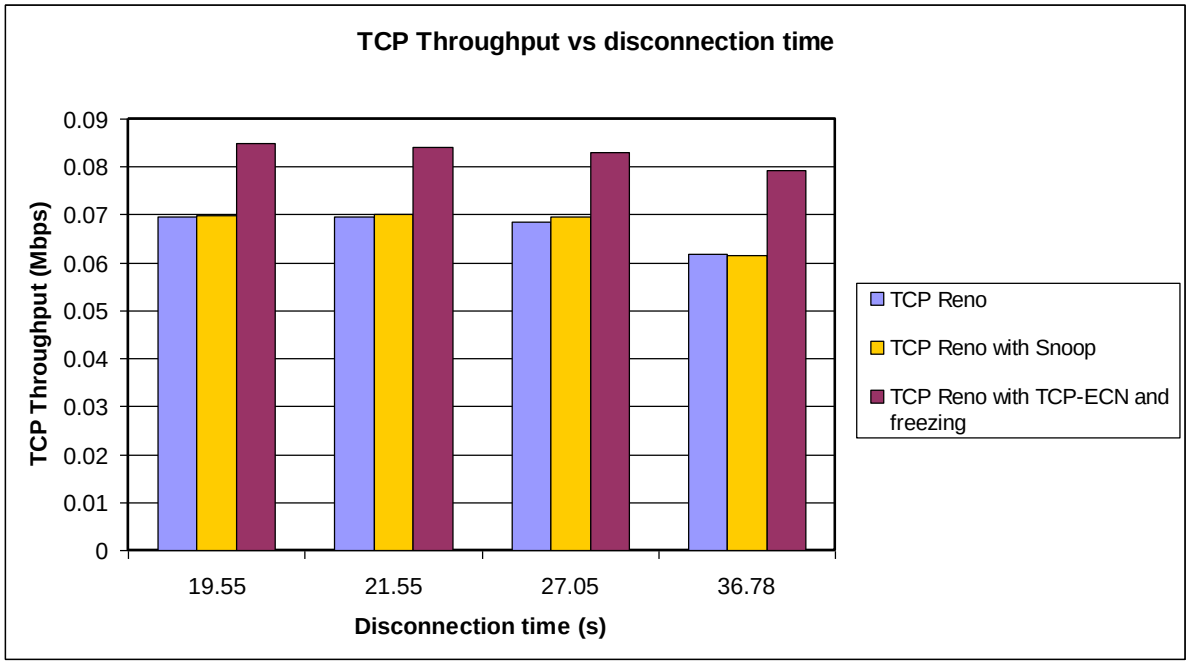


**Figure 21: TCP Throughput vs. disconnection time**

TCP throughput with TCP-ECN is better than TCP Reno and Snoop for varying disconnection periods as shown in Figure 21. Snoop's performance is similar to TCP Reno's for reasons stated earlier.

## *7.4 Testing with both high BER and disconnections*

### 7.4.1 Simulation setup

Topology shown in Figure 19 was reused for this set of tests. However, now we introduce wireless errors in addition to disconnections.

### 7.4.2 Results and analyses

- **TCP Sequence number progression vs. time**
  - The simulation parameters were setup as follows:
    - ***Wireless bandwidth:*** 11Mbps
    - ***Wired bandwidth:*** 100Mbps
    - ***Wired network delay:*** 500ms
    - ***Simulation time:*** 150 seconds
    - ***No wired network errors***
    - ***Disconnection time:*** 21.5 sec
    - ***Wireless error rate:*** 0.15

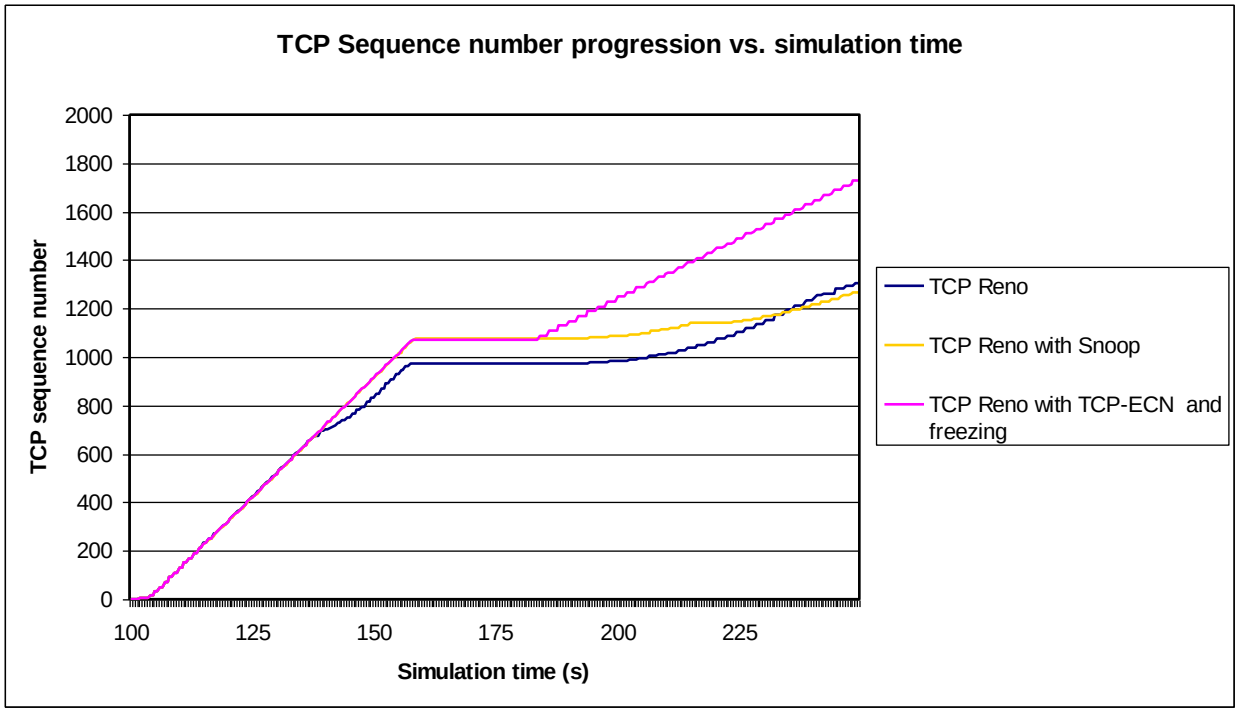


**Figure 22: TCP sequence number progression with disconnections and wireless losses**

Figure 22 shows the TCP sequence number progression with all three schemes. Although Snoop's performance is better than TCP Reno's and similar to that of TCP-ECN until the disconnection, it deteriorates after the reconnection. TCP Reno's performance dips heavily due to both wireless losses and the disconnection.

- **TCP Throughput vs. wireless error rate**
  - The simulation parameters were setup as follows:
    - ***Wireless bandwidth:*** 11Mbps
    - ***Wired bandwidth:*** 100Mbps
    - ***Wired network delay:*** 500ms
    - ***Simulation time:*** 150 seconds
    - ***No wired network errors***
    - ***Disconnection time:*** 33.56 sec

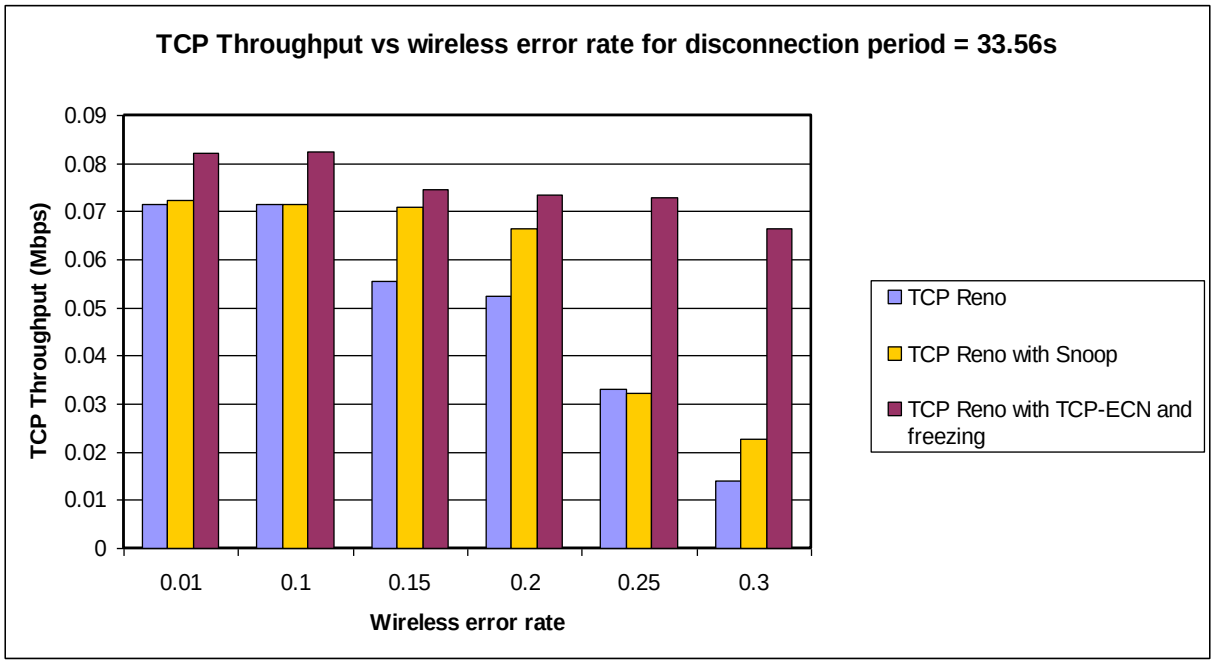


**Figure 23: TCP Throughput vs. wireless loss rate for a fixed disconnection period**

Higher wireless error rates affect both TCP Reno and Snoop's TCP throughput, as shown in Figure 22. TCP-ECN maintains almost uniform throughput in spite of varying error rate and the disconnection.

## *7.5 Maximum memory requirement at the base station*

We compared the maximum memory requirement at the base station for TCP-ECN and Snoop, as both maintain state at the base station. Figure 24 depicts the values with both these schemes for varying wireless error rate. The buffer requirement for TCP-ECN is very minimal as it maintains only TCP sequence numbers at the base station, and does not buffer packets themselves. Snoop, on the other hand, shows varying and comparatively high buffer requirement, as it needs to buffer packets at the base station. This can prove to be a non-negligible bottleneck when a very large number of connections are handled by the base station. The simulation parameters for this test were:

- ***Wireless bandwidth:*** 11Mbps
- ***Wired bandwidth:*** 100Mbps
- ***Wired network delay:*** 500ms
- ***Simulation time:*** 250 seconds
- ***No wired network errors***

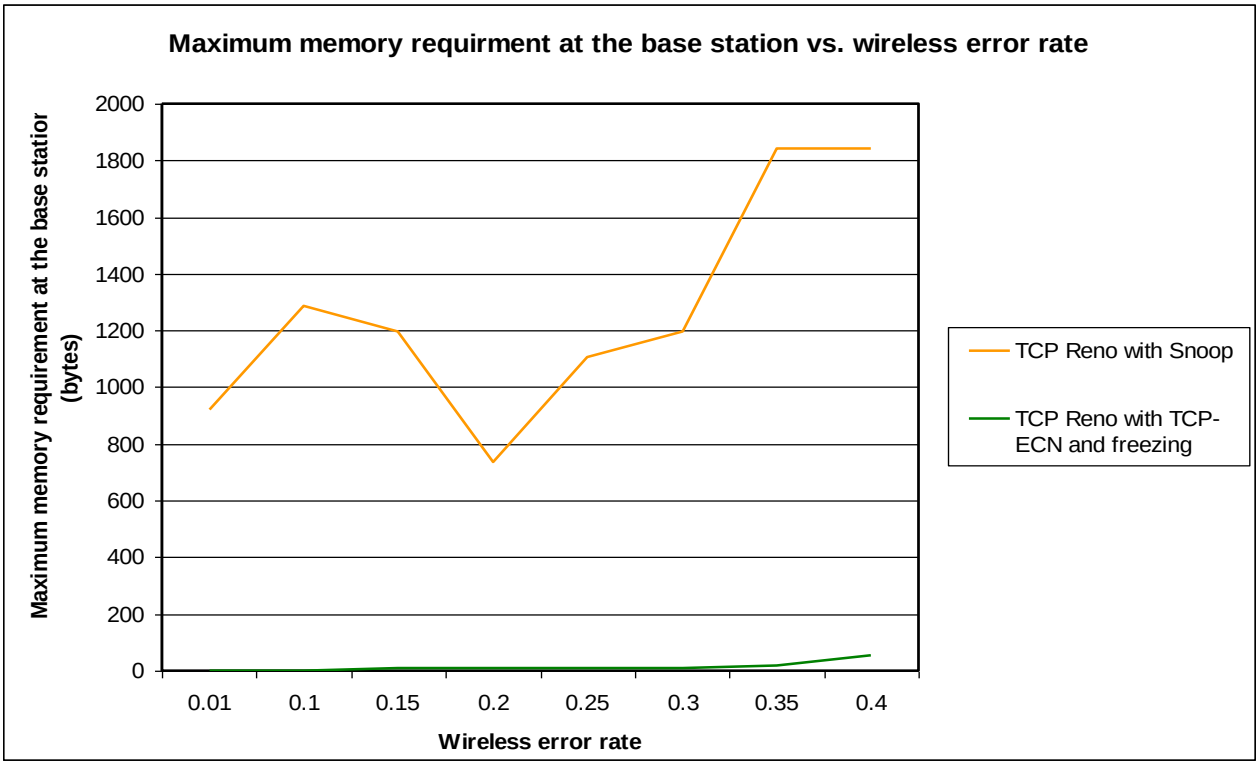


**Figure 24: Maximum memory requirement at the base station vs. wireless error rate**

# Chapter 8. IMPLEMENTATION OF TCP-ECN

We have implemented TCP-ECN in the Linux kernel, version 2.6.12. Although, simulation environment allowed us to create complex topologies, we were restricted to use a simple topology for implementation due to hardware constraints. In addition, we have implemented and tested only the high BER handling module of TCP-ECN. Due to time and resource restrictions, we could not implement the disconnection handling module. In the next few sections, we will discuss the details of implementation and testing.

## *8.1 Topology and setup used for implementation*

The topology used for implementation and its testing is shown in Figure 25. Two laptops, an IBM Thinkpad and a Dell Inspiron 6000, each with an Intel® Pentium® M 1.5GHz processor were used. The IBM Thinkpad represented BS, or the access point, and Dell Inspiron 6000 was used as the MH. For the FH, we used a Dell 400SC server. A 100MBps Ethernet cable was used to connect the FH and the BS. BS and MH communicated wirelessly. The MH used an Intel® PRO/Wireless 2200BG network adapter with the ipw2200 wireless driver, version 1.0.2. An Atheros® 108 Mbps 802.11b/g WiFi PCI Wireless Card with the MADWiFi wireless driver, version 0.9.2 [21] was used in the BS.

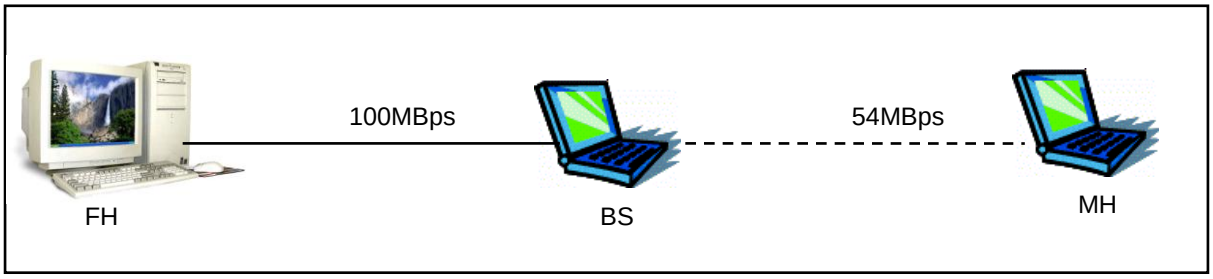


**Figure 25: Topology used for implementation of TCP-ECN**

## *8.2 Implementation details*

We have modified the MADWiFi wireless driver [21] on the BS to incorporate the functionality depicted in flowchart of Figure 5. Similarly, we have modified the TCP implementation of Linux on MH to include the functionality shown in Figure 6.

*linux/include/linux/tcp.h* was modified to convert one of the reserved bits in the TCP header to ECN flag. *linux/net/ipv4/tcp_input.c* was modified on the MH to withhold dupACKs whenever the ECN bit was set in an incoming TCP segment. Two global variables were introduced to turn on/off TCP-ECN capability: *tcp_ecn_enabled* and *sysctl_te_max_retries. Tcp_ecn_enabled* was used in the MH, and *sysctl_te_max_retries* was used in the BS to set the maximum number of Link layer retransmissions.

On the base station, we have modified *madwifi/ath/if_athvar.h* and *madwifi/ath/if_ath.c* to invoke FH data, MH ack and MAC callback processing. Incoming FH TCP data is processed in the *ath_hardstart* function, and ACKs from the MH are handled in the *ath_rx_tasklet*

function in file *madwifi/ath/if_ath.c*. MAC callback functionality is handled in *ath_tx_processq* function. The data structure to maintain sequence lists for different connections is implemented as part of the private area in the *net_device* structure. This data structure is implemented as a hash table of linked lists of connection entities. Each of these connection entities holds a linked list of sequence numbers. Hash value is computed using source address, destination address, source port number and destination port number of an incoming packet. Figure 26 depicts this structure more succinctly.

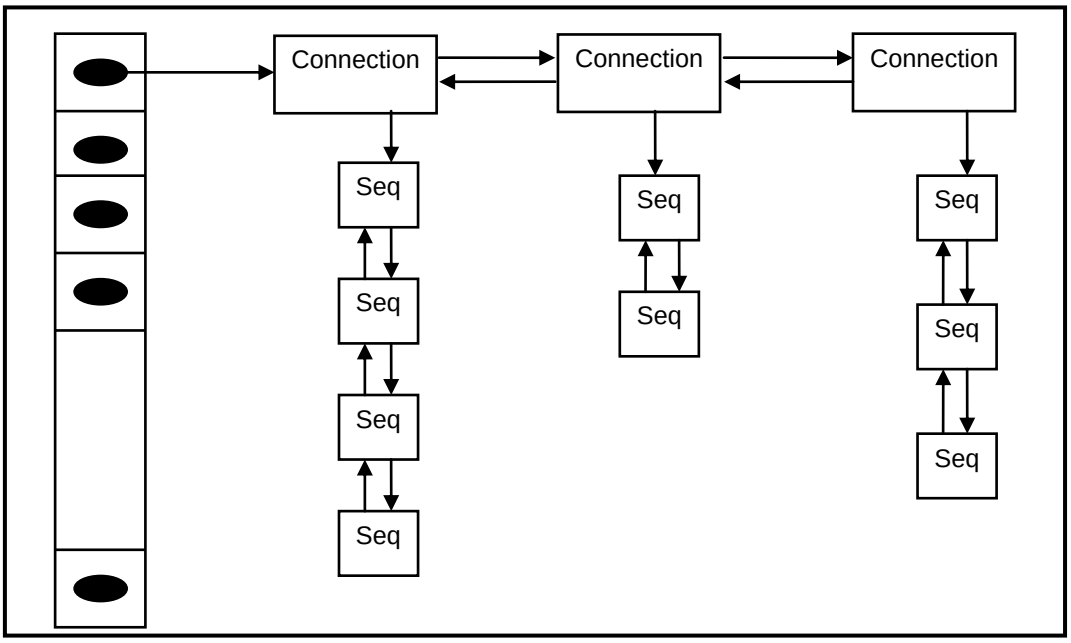


**Figure 26: Data structure to maintain sequence lists for all the connections**

The structure of the connection and sequence entities shown in Figure 26 are as follows:

- Seq node
  - Seq number
  - Length of segment
  - Retries
  - Next seq node pointer
  - Prev seq node pointer
- Connection node
  - Source port
  - Dest port
  - Src IP
  - Dest IP
  - Seq node pointer
  - Next connection node pointer
  - Prev connection node pointer

A new connection entity is created when a TCP-SYN segment is seen at the access point. An existing connection entity is deleted when a TCP-FIN or a TCP-RST segment is received by the base station.

## *8.3 Test Results and analyses*

We tested our implementation in low and high wireless error conditions. We could not repeat all the tests we performed during simulation, since it is very difficult to quantify wireless error rate in real world scenarios. In order to create low wireless error conditions, we ran our implementation with the BS and the MH next to each other, without any obstructions between them. Also, we allowed the network interface to choose the best rate to operate in. In contrast, to create high wireless error conditions, we fixed the rate to the maximum possible: 54Mbps (higher data rates suffer from higher loss rates). Furthermore, we kept the BS and MH as far apart as possible with a lot of obstructions in between. For both the scenarios, we have turned off TCP SACK, TCP FACK and TCP timestamp options.

We used iperf [12] to generate TCP traffic between FH and MH. We ran iperf for 10 seconds. Tcptrace [35] was used to obtain sequence plots. Finally, Gnuplot [16] was used to convert Tcptrace plots to the graphs shown in Figure 27, Figure 28, Figure 29, and Figure 30.

### 8.3.1 Time-sequence plots for low wireless error conditions

Figure 27 and Figure 28 are the plots obtained by running our implementation in low wireless error conditions. Clearly, there is very little difference in TCP sequence number progression between the two plots.

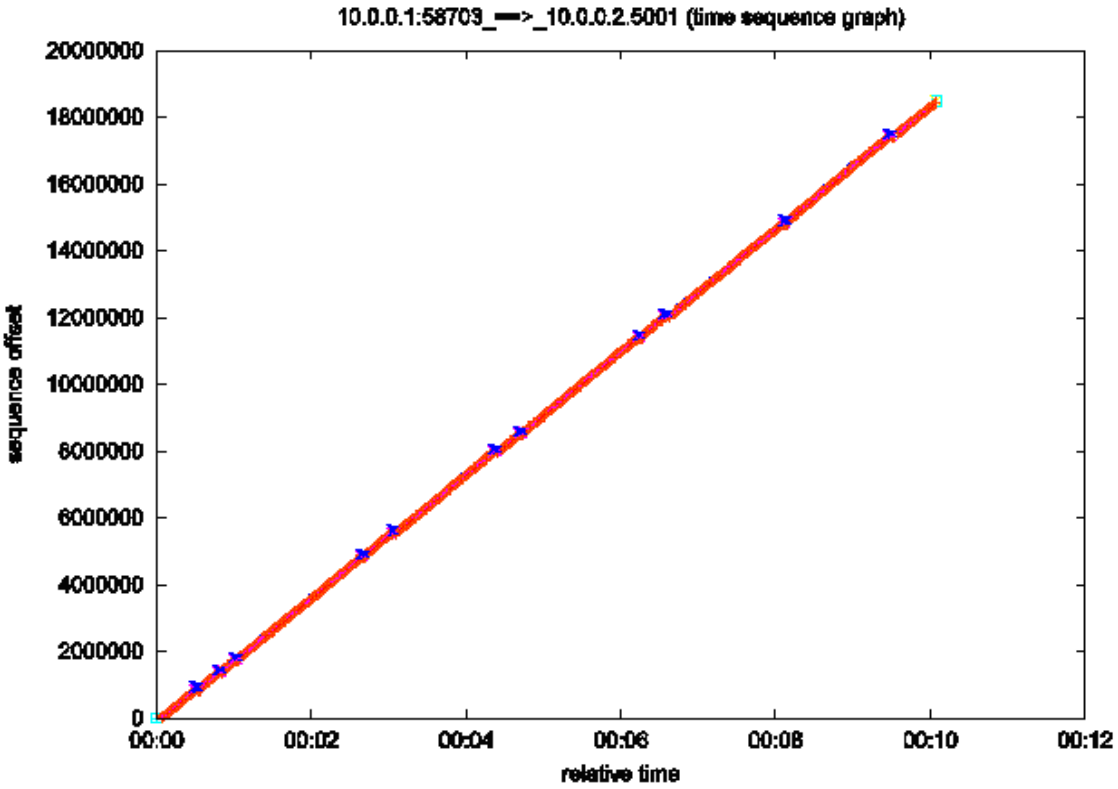


**Figure 27: TCP sequence number progression for TCP without ECN enhancement**

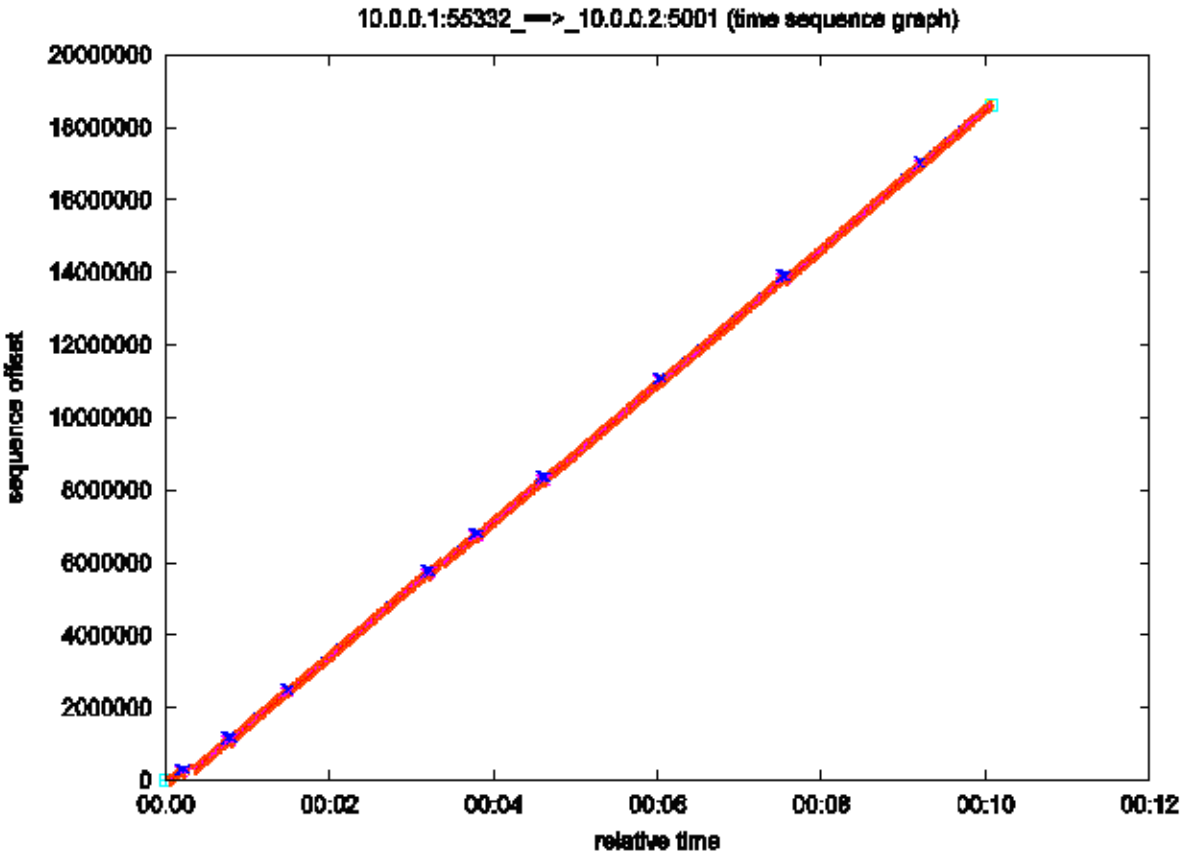


**Figure 28: TCP sequence number progression for TCP with TCP-ECN enhancement**

### 8.3.2 Time-sequence plots for high wireless error conditions

Plots shown in Figure 29 and Figure 30 were obtained for high wireless error conditions without and with TCP-ECN enhancement, respectively. Clearly, TCP without enhancement suffers from low throughput due to wireless losses. On the contrary, TCP-ECN enhancement maintains almost the same throughput as the low error rate scenario (Figure 28).

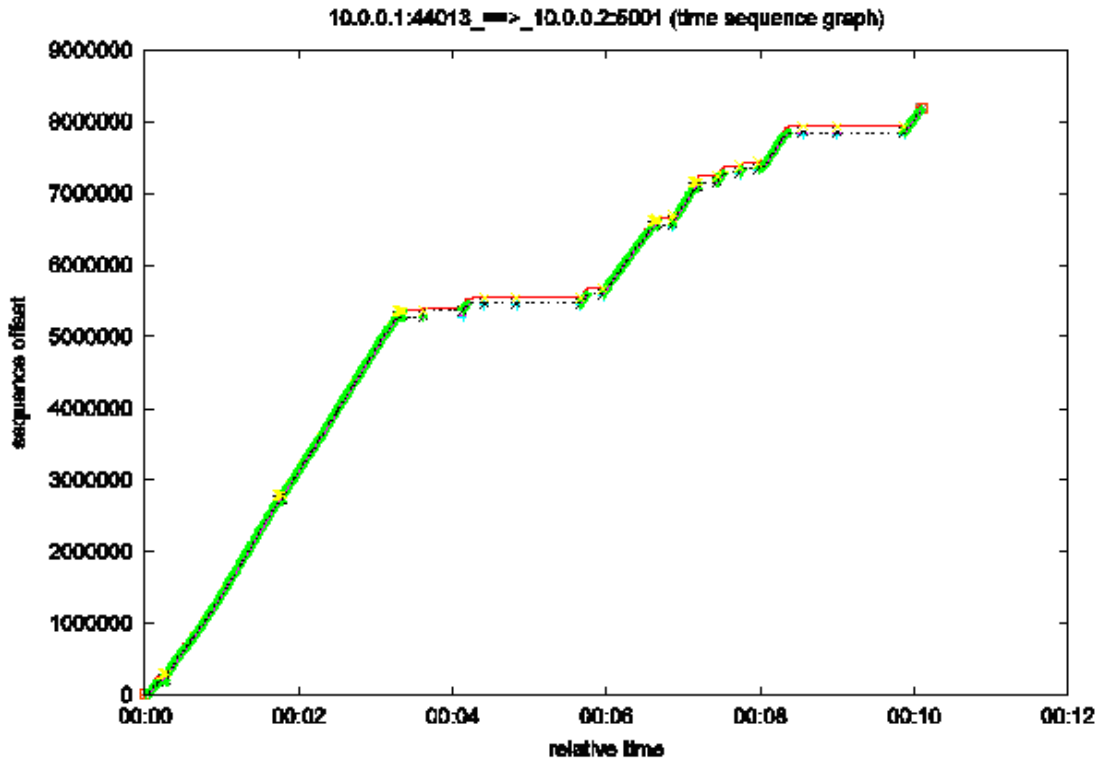


**Figure 29: TCP sequence number progression for TCP without TCP-ECN enhancement**

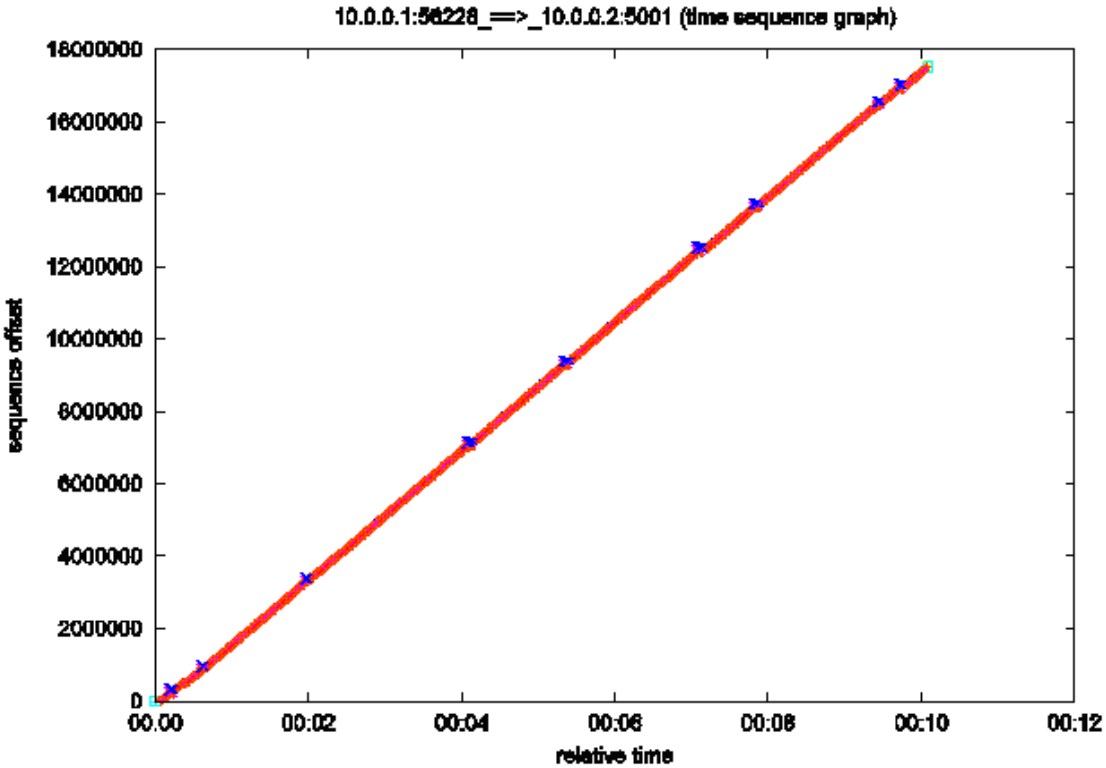


**Figure 30: TCP sequence number progression for TCP with TCP-ECN enhancement**

### 8.3.3 Throughput plots

Figure 31 shows TCP throughput with and without TCP-ECN enhancement. For high wireless error conditions, TCP-ECN improves TCP performance by about 119 percent.

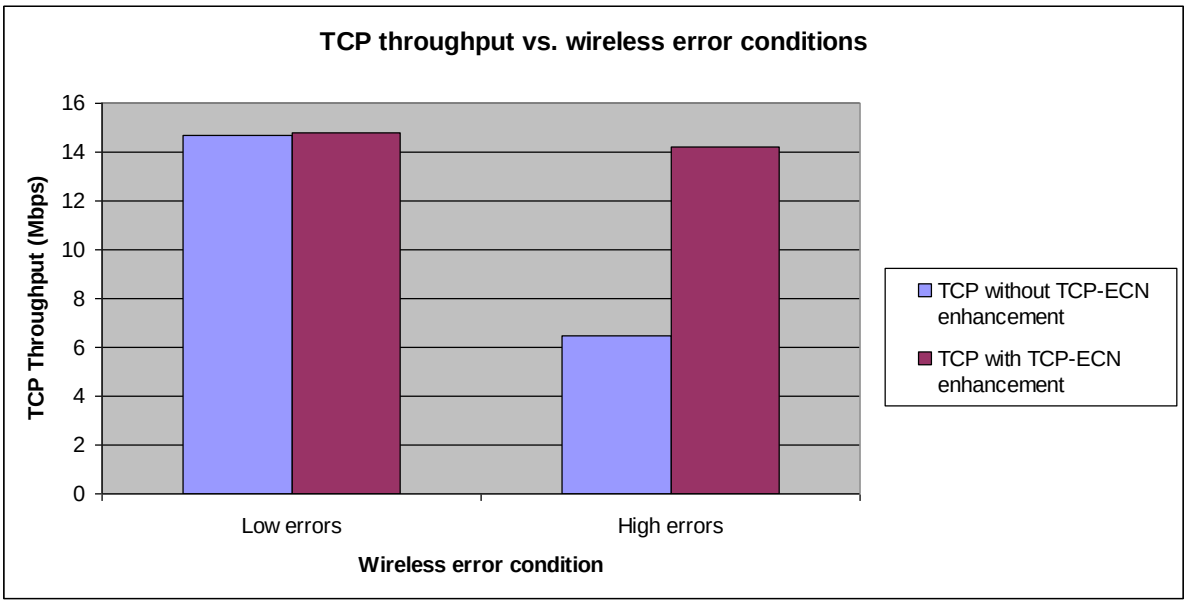


**Figure 31: TCP throughput plot for two different wireless error conditions**

# Chapter 9. CONCLUSION AND FUTURE WORK

The report provided a succinct introduction to TCP's operation, highlighted important characteristics of wireless networks, and analyzed the effects of these characteristics on TCP's performance in wired-cum-wireless networks. An extensive survey of schemes proposed over the past decade was presented. These proposals aim to solve the three principal problems in wireless networks: high BER, long and frequent disconnections, and variable bandwidth. We introduced a new scheme called TCP-ECN which focused on solving TCP problems due to high BER and long disconnections.

The essence of the new scheme is to use Explicit Congestion Notification to enable the MH to distinguish between wired and wireless losses. In addition, we also allow the base station to perform sender freezing in the face of an impending disconnection of the MH. Both these techniques insulate the FH sender from the idiosyncrasies of the wireless channel. We simulated TCP-ECN using ns2, and implemented the solution to high BER in the Linux kernel. The results of our tests were analyzed, and the performance of TCP-ECN was compared with the existing version of TCP, and Snoop. TCP-ECN markedly improved TCP throughput, and performed better than Snoop for higher wireless error rates.

For the future, we intend to implement TCP-ECN's solution to long disconnections in the Linux kernel. In addition, we would like to evaluate its performance with newer enhancements to TCP, like TCP SACK, and modify its semantics to accommodate these enhancements.

# REFERENCES


**[1]** M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," RFC 2581, Apr. 1999.

**[2]** A. Bakre and B. Badrinath, "I-TCP: Indirect TCP for mobile hosts," *Technical Report DCS-TR-314, Rutgers University*, Oct. 1994. [Online]. Available: http://citeseer.ist.psu.edu/bakre95itcp.html

**[3]** H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links," in *Proceedings of ACM SIGCOMM '96*, Stanford, CA, Aug. 1996.

**[4]** H. Balakrishnan, S. Seshan, and R. H. Katz, "Improving reliable transport and handoff performance in cellular wireless networks," in *ACM Wireless Networks,* vol. 1, Dec. 1995.

**[5]** H. Balakrishnan and R. Katz, "Explicit Loss Notification and Wireless Web Performance," in *Proc. IEEE Globecom Internet Mini-Conference*. IEEE, Nov. 1998.

**[6]** C. Barakat, E. Altman, and W. Dabbous, "On TCP performance in a heterogeneous network: A survey," *IEEE Communication Magazine*, vol. 38, no. 1, pp. 40–46, Jan. 2000.

**[7]** E. R. Braden, "Requirements for Internet Hosts – Communication Layers," RFC 1122, Oct. 1989.

**[8]** K. Brown and S. Singh, "M-TCP: TCP for Mobile Cellular Networks," in *ACM Computer Communications Review (CCR)*, vol. 27, no. 5, 1997.

**[9]** R. Caceres and L. Iftode, "Improving the performance of reliable transport protocols in mobile computing environments," in *IEEE JSAC Special Issue on Mobile Computing Network,* 1994.

**[10]** J. Chung and M. Claypool, "NS by Example," http://nile.wpi.edu/NS/

**[11]** G. M. T. Da Costa and H. R. Sirisena, "Freeze TCP with timestamps for fast packet loss recovery after disconnections," in *Computer Communications*, vol. 26, pp. 1792-1799, 2003.

**[12]** IPerf, http://www.noc.ucf.edu/Tools/Iperf/default.htm

**[13]** L. Daniel, "Introduction to TCP and its Adaptation to Networks with Wired-cum-Wireless Links," in *Seminar on Transport of Multimedia Streams in Wireless Internet, Department of Computer Science, University of Helsinki, Finland,* Sep. 2003. [Online]. Available: http://www.cs.helsinki.fi/u/jmanner/Courses/seminar_papers/tcp_wireless.pdf

**[14]** H. Elaarag, "Improving TCP Performance over Mobile Networks," in *ACM Computing Surveys*, vol. 34, no. 3, Sep. 2002, pp. 357-374.

**[15]** M. Garcia, R. Aguero, L. Munoz, and J. A. Irastorza, "Smart enhancement of TCP performance over lossy WLAN channels through the combined use of link layer techniques," in *Wireless Personal Communications*, vol. 37, pp. 91-103, 2006.

**[16]** Gnuplot, http://www.gnuplot.info/

**[17]** T. Goff , J. Moronski, D. S. Phatak, and V. Gupta, "Freeze-TCP: A true end-to-end enhancement mechanism for mobile environments," in INFOCOM, (Israel), 2000.

**[18]** M. Greis, "Tutorial for the network simulator ns," http://www.isi.edu/nsnam/ns/tutorial/

**[19]** V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," RFC 1323, May 1992.

**[20]** J. Kurose and K. Ross, *Computer Networks: a top-down approach featuring the Internet*, 2nd ed. Addison Wesley, 1996.

**[21]** Madwifi, a multiband Atheros® driver for wifi, http://madwifi.org/

**[22]** M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgement Options," RFC 2018, Oct. 1996.

**[23]** G. Montenegro and S. Dawkins, "Wireless Networking for the MNCRS, Internet Draft," *IETF draft*, Aug. 1998. [Online]. Available: http://ietfreport.isoc.org/idref/draft-montenegro-mncrs/

**[24]** Network Simulator – ns-2, http://www.isi.edu/nsnam/ns/

**[25]** B. O'Hara and A. Petrick, *The IEEE 802.11 Handbook: A Designer's Companion*, 2nd ed. IEEE, 2005.

**[26]** C. Parsa and J. J. Garcia-Luna-Aceves, "Improving TCP performance over wireless networks at the link layer," in *ACM Mobile Networks and Applications*, vol. 5, no. 1, pp. 57–71, Mar. 2000. [Online]. Available: citeseer.nj.nec.com/article/parsa99improving.html

**[27]** K. Pentikousis, "TCP in wired-cum-wireless environments," in *IEEE Communications Surveys & Tutorials*, vol. 3, no. 4, Fourth Quarter 2000.

**[28]** C. E. Perkins, "Mobile IP," in *International Journal of Communication Systems*, vol. 11, 1998, pp. 3-20.

**[29]** J. Postel, "Transmission Control Protocol," RFC 793, Sept.1981.

**[30]** K. Ratnam and I. Matta, "WTCP: An Efficient Mechanism for Improving TCP Performance over Wireless Links," in *Proceedings of the Third IEEE Symposium on Computer and Communications (ISCC '98)*, Jun. 1998.

**[31]** S. Seok, S. Youm, S. Kim, and C. Kang, "A modification of TCP flow control for improving end-to-end TCP performance over networks with wireless links," in *Computer Communications*, vol. 26, pp. 1998-2010, 2003.

**[32]** A.K. Singh and S. Iyer, "ATCP: Improving TCP Performance over Mobile Wireless Environments," in *Fourth IEEE Conference on Mobile and Wireless Communications Networks, Stockholm, Sweden*, Sep. 2002. [Online]. Available: http://citeseer.ist.psu.edu/kr02atcp.html

**[33]** I. Stojmenovic, *Handbook of Wireless Networks*, Wiley Interscience, 2002.

**[34]** F. Sun, V. O. K. Li, and S. C. Liew, "Design of SNACK mechanism for wireless TCP with New Snoop," in *IEEE Wireless Communications and Network Conference (WCNC)*, Mar. 2004.

**[35]** Tcptrace, http://jarok.cs.ohiou.edu/software/tcptrace/

**[36]** V. Tsaoussidis and I. Matta, "Open issues on TCP for mobile computing," in *The Journal of Wireless Communications and Mobile Computing, John Wiley & Sons*, vol. 2, no. 1, Feb. 2002. [Online]. Available: http://citeseer.ist.psu.edu/tsaoussidis02open.html

**[37]** N. Vaidya, M. Mehta, C. Perkins, and G. Montenegro, "Delayed duplicate acknowledgements: A TCP-unaware approach to improve performance of TCP over

wireless," *Technical Report, Computer Science Dept., Texas A&M University*, 1999. [Online]. Available: http://citeseer.ist.psu.edu/289955.html

**[38]** Z. Zhizhao and H. Zhenzhi, "Improving TCP performance over wireless links using link layer retransmission and explicit loss notification," in *Journal of Systems Engineering and Electronics*, vol. 13, no. 4, pp. 17-23, 2002.

# APPENDIX A

## Basics of ns2

Ns2 is a discrete event simulator that is developed using two languages: C++ and OTcl. All the network components are pre-compiled into the ns2 library, and are written in C++. User simulation scripts are executed by an OTcl interpreter. C++ components execute faster, but are cumbersome for making minor changes as they require re-compilation. Hence time-sensitive simulation components are implemented in C++, whereas OTcl is used for writing the simulation scripts. Since OTcl scripts are interpreted, making changes to the simulation setup becomes easier. An independent discrete event scheduler is implemented which is accessible to all the network components.

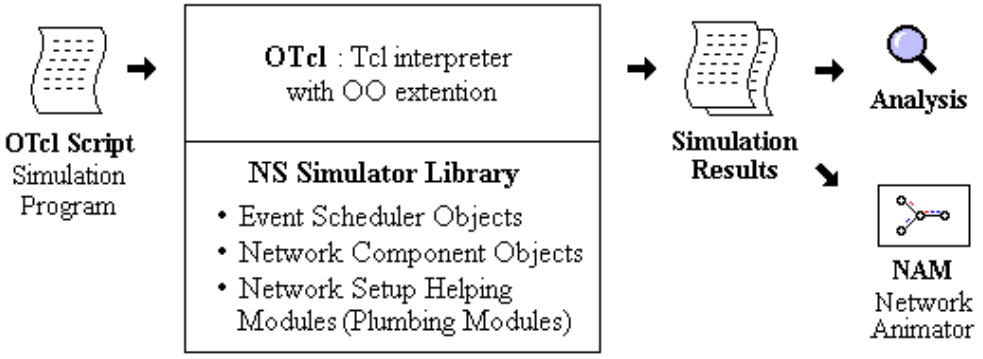


**Figure 32: Control and data flow in ns2 [10]**

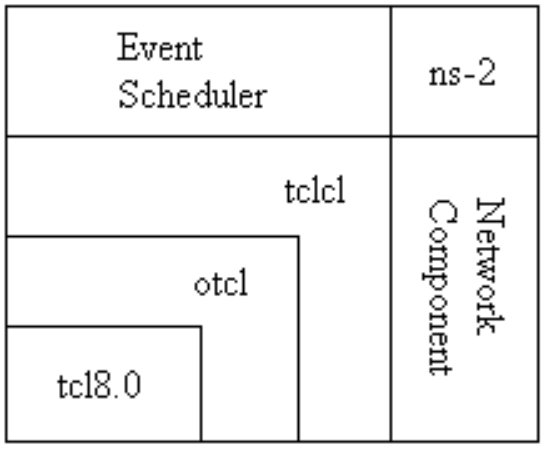


**Figure 33: User view of the ns2 system [10]**

Figure 32 depicts the control and data flow in ns2. Figure 33 shows the user's view of the system when viewed from the from the bottom left corner. Tclcl is used to interface between the OTcl and C++ domains.

The following code snippet [10] in Figure 34 demonstrates a C++ class that establishes the *shadow object* relationship between the OTcl class Agent/MyAgentOtcl and the C++ class MyAgent. Whenever the user creates an object of Agent/MyAgentOtcl, ns2 automatically creates the shadow object MyAgent in the C++ domain. Hence the OTcl hierarchy is reflected in the C++ hierarchy as shown in Figure 35. There could be a few classes in C++ or OTcl without any shadow counterparts.

```
class MyAgent : public Agent {
public:
        MyAgent();
protected:
        int command(int argc, const char*const* argv);
private:
        int    my_var1;
        double my_var2;
        void   MyPrivFunc(void);
};
```

```
static class MyAgentClass : public TclClass {
public:
        MyAgentClass() : TclClass("Agent/MyAgentOtcl") {}
        TclObject* create(int, const char*const*) {
                return(new MyAgent());
        }
} class_my_agent;
```

**Figure 34: Code snippet to create shadow object association [10]**

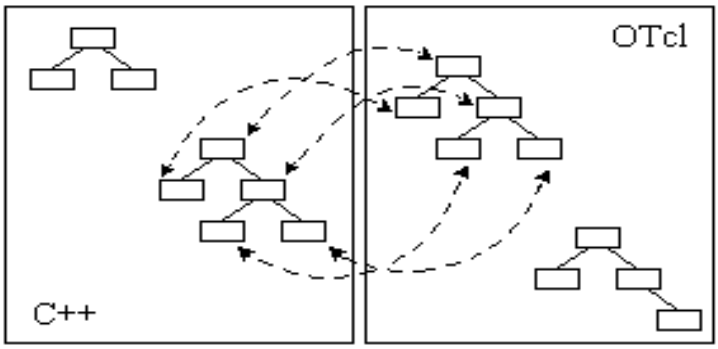


**Figure 35: Reflection of class hierarchies in C++ and Otcl domains [10]**

The code snippet in Figure 36 shows how a C++ variable can be bound to an OTcl variable, so that a user can have direct access to the C++ variable from the simulator. The binding is done in the C++ class constructor.

```
MyAgent::MyAgent() : Agent(PT_UDP) {
        bind("my_var1_otcl", &my_var1);
        bind("my_var2_otcl", &my_var2);
}
```

**Figure 36: Binding variables [10]**

```
int MyAgent::command(int argc, const char*const* argv) {
      if(argc == 2) {
            if(strcmp(argv[1], "call-my-priv-func") == 0) {
                  MyPrivFunc();
                  return(TCL_OK);
            }
      }
      return(Agent::command(argc, argv));
}
```

**Figure 37: Accessing C++ commands in the simulator [10]**

```
void MyAgent::MyPrivFunc(void) {
      Tcl& tcl = Tcl::instance();
      tcl.eval("puts \"Message From MyPrivFunc\"");
      tcl.evalf("puts \"      my_var1 = %d\"", my_var1);
      tcl.evalf("puts \"      my_var2 = %f\"", my_var2);
}
```

**Figure 38: Executing OTcl commands in a C++ program [10]**

Figure 37 shows how a C++ "command" can be issued in the simulator. Here *call-my-priv-func* can be invoked as an OTcl function by the user in a simulation script, and the control is transferred to the C++ domain during execution. In Figure 38, OTcl instructions are being executed in a C++ function by using *tcl.eval* and *tcl.evalf* functions. The string passed to these functions is actually executed in the OTcl interpreter, and the result of this execution can be accessed by the C++ function.

The above techniques allow an ns2 developer to switch between C++ and OTcl domains with ease. This section aimed to provide a brief introduction to the architecture and mechanics of ns2. For further details, please refer to ns documentation at [24], or tutorials at [10][18]