MOBILE COMPUTING AND COMMUNICATION

SOLUTIONS FOR A NOISY ENVIRONMENT

A Senior Thesis

By

Dominique Kilman

1996-97 University Undergraduate Research Fellow

Texas A&M University

Group: ELECTRICAL ENGINEERING/COMPUTER SCIENCE

Mobile Computing and Communication

Solutions for a Noisy Environment

Dominique Kilman

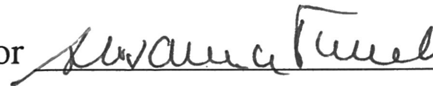
University Undergraduate Fellow, 1996-1997

Texas A&M University

Department of Computer Science

APPROVED

Fellows Advisor

Honors Director 

Mobile Computing and Communication Solutions for a Noisy Environment.

Dominique Marie Kilman (Nitin H. Vaidya), Computer Engineering, Texas A&M University.

Abstract

Business people today need an effective mobile computing system. This means that modern network systems must be comprised of both wired and wireless links Current transmission control protocols (TCP) implementations have been designed to work well in a wired environment in which error rates are very low and packet losses can be attributed to network congestion. In a wireless environment, there are higher error rates usually due to noise in the air. The loss due to congestion assumption is no longer correct in this environment. Current TCP implementations can not distinguish between a congestive loss and a wireless loss. Several strategies are currently being investigated to improve current TCP performance. One of the best candidates for common use is the Snoop Agent. The focus of my research has been to compare current TCP implementations with TCP combined with the Snoop Agent. Using a wireless test bed, the two implementations will be analyzed and tested to gauge their performance. Throughput and transmission times will be gathered and evaluated.

Mobile Computing and Communication

Solutions for a Noisy Environment

The computing industry today is moving quickly towards the need for reliable mobile communication techniques. Industry is shifting towards a mobile office in which business people move from customer to customer, rarely staying in a central office. Access to e-mail, messaging services, and the Internet are needed by business persons and industry while traveling and at home. The usual access to internet and network services is through a leased telephone line modem in the home or a LAN connection within an office. A LAN is defined as a local area network. This type of network traditionally covers a small area and is very fast. While traveling, individuals are unconnected and unproductive because of their inability to access network services. Much research is being done in the area of wireless mobile communication, but many problems must be overcome before these ideas can be marketed in the main stream.

A History of Wireless

The first documented use of wireless communication was in 1901. Guglielmo Marconi, an Italian physicist demonstrated a ship to shore communication method using Morse Code. Since Marconi, wireless communication has increased distances and improved performance.

One of the first forms of "modern" wireless communications was the cordless phone. The cordless phone has very limited mobility, within a few rooms, but it has excellent performance within this limited range. Paging is also a form of wireless communication. Because of the one-way nature of pagers, many people do not consider them to be a communication device. Several companies are investigating the idea of two way pagers. This would allow the receiver of a page to respond to a page without needing to find a phone. The advent of cellular phones brought a greater range to wireless communication. With a cellular phone, individuals could receive calls anywhere within the United States, if their service was nationally supported. Even cellular systems which worked only within the city were a great improvement in the mobility of business people. These phones allow individuals to travel while remaining in touch with their office.

Today, wireless LANs are available for computer users, but their performance as compared to wired LANs leaves much to be desired. Slow connection times and error prone systems leave most mobile computer users frustrated and wanting more.

Current Wireless Technology

There are many forms of wireless communication in use today. Cordless phones, pagers, and cellular phones are a type of wireless communication. Cellular technology has made great advances in allowing mobility in phone conversations. The problem with these technologies are their limitations. Cordless computer connections are limited to the

building in which their base resides. Cellular phones are more mobile, but voice and signal quality is lost. With both technologies, there is a loss of security and privacy.

Cellular Modems

Two forms of cellular communication are used today in computer communication : circuit-switched cellular transmission and cellular digital packet data (CDPD). Circuit-switched cellular communication works by connecting a cellular phone to a cellular modem. The modem is then connected to a notebook computer. Cellular modems are easy to use because they are (from the users standpoint) the same as any telephone modem. The charges are by-the-minute and can be cost effective for large data transfers. Small data files or e-mail attachments would be less cost effective because of the small amount of information sent. Cellular modems are versatile: they offer RJ-11 connectors to allow traditional land-line connections. Unfortunately, most cellular transmissions occur at the relatively slow rate of 7200 bps because faster transmissions result in more errors. These same modems can offer speeds of 14.4 kbps, but this speed is generally only available when a land-line is used. Cellular modems are also prone to static and interference which will introduce more error. This can cause a slow down in transmission time. Cellular modems, like cellular phones, are easy to intercept. Because of this, there is very little security when information is sent. It would be safer to send any important or confidential data on a traditional land-line. Cellular modems do not offer much compatibility;, costs are high and battery usage is poor. As a cellular user moves from one cell to another, the signal can weaken as the signal is transferred between the two cells.

CDPD

CDPD is a two way digital packet switching technology. CDPD uses traditional existing cellular phone structures. Data files are broken into packets and sent over the cellular network during the natural pauses that occur in human conversation. Because of the complexity of this system, it is hard to implement and may not work as well as expected. There have also been some problems with getting cellular services, application developers, and device manufacturers to work together to provide a seamless integrated package. With CDPD, transmission times are expected to be fast, possibly up to 19.2 kbps. Because of the packet technology, data transmission should be more reliable, cost effective, and secure than circuit-switching cellular. CDPD combines the data and voice capabilities into one device. This is a plus when considering cost effectiveness.

Radio Modems

Another way of implementing wireless is through radio transmission. Geographical areas are divided into cells. Each of these cells has a base station - a radio tower capable of sending and receiving data packets - which has a wired connection to every other base station, usually using telephone wire. To send information, the radio modem wirelessly sends a signal to the base station. The base station locates the intended receiver and sends the information to the base station located within the receiver's cell. The receiver's base station then wirelessly transmits the signal to the receiver if it is another wireless user, or sends the signal over wire if the receiver has a wired connection. As with cellular modems, the signal can weaken as the signal is handed from one cell to

another. Radio transmissions are digitally encoded; this allows for greater security than cellular communication. Transmission costs are based on packets, not time; therefore, it is less expensive to send e-mail attachments and other relatively small files. Radio waves are also less susceptible to interference than cellular connections, allowing for greater reliability. There are some drawbacks to radio transmissions. Since all users are sharing the same bandwidth, transmission times can be slow, usually less than 2400 bps. Radio transmission can only be used with e-mail applications. Batteries can last for up to 2 hours, but take 12 hours to recharge once they are used up. These modems are also expensive and usually, large making them a poor choice if the goal is mobility.

Paging

Paging is also a form of wireless communication. Paging provides one way communication, usually allowing only receipt of information. Pagers are lightweight and users can get news and information 24 hours a day by calling toll-free 800 numbers. Messages are transmitted quickly and have very few errors. New technology involving two-way paging has become available in the past year. This would allow the individual to send and receive messages. The pagers are capable of transmitting and receiving only small alphanumeric messages; however, so large amounts of data cannot be transmitted.

Wireless Local Area Networks

Wireless Local Area Networks (WLANs) are another form of wireless communication. A wireless LAN sends information over radio waves or uses infrared light. The obvious limitation of using infrared is the small distances needed to

communicate. The IEEE P802.11 Wireless LAN project is supposed to go to the standards board for approval in 1996[6]. When standards are set, greater advancements may be made in getting cooperation between different companies.

Network Protocol

The way in which computers communicate is called networking. Such communication is implemented using network protocol. This protocol is implemented using a layered approach as shown in figure 1. The approach shown below is the Open Systems Interconnection Model developed by the International Standards Organization for the network protocol. Some approaches use more or less layers depending on the interpretation of each layers' duties. Each layer, except for the first, only communicates with the layers directly above and below itself. An individual layer acts as if it is communicating with its peer layer at the other host. Each layer is essentially buffered from changes in other layers. The only thing that must remain stable in each layer is how the layers communicate with each other. The way each layer processes the information it receives or sends has no effect on the other layers. This layering method also allows different protocols to communicate. By the time the information is truly sent, the only thing being sent are pulses. The receiving host can process this information in any way appropriate to retrieve the information.

Layer	Examples
Application	FTP, Telnet
Presentation	XDR
Session	RPC
Transport	TCP, UDP
Network	IP
Data / Link	Ethernet, Token Ring
Physical	Wires

Figure 1 [13]

The first layer (starting from the bottom) is the physical layer. This layer involves the way in which an electrical, optical, or wireless pulse is transmitted from one host to the next. This layer can involve wire, fiber optic cable, or wireless signaling devices (satellite, radio or microwave). The physical layer, also, converts bits received from the upper layers into the signal at the generator and then converts the signal back to bits at the receiver.

The second layer shown is the link layer also called the data-link layer or network interface layer. This layer involves logically transmitting bits from one location to another. Addressing and error detection is performed at this layer.

The next layer up is the network layer. At this layer, the data are treated as packets. Information is also seen as end-to-end, meaning that the information is not sent and forgotten. Acknowledgments are sent and received to keep one host informed about the actions of another. The network layer is not concerned with ordering of packets or even if all packets arrived. The network layer only cares whether or not something arrived. The purpose of this layer is to hide the messiness of differing link and physical layer requirements from the upper layers. The most common protocol in the network layer is the Internet Protocol (IP). Other possible protocols are Internet Control Message Protocol (ICMP) and Internet Group Management Protocol (IGMP).

End to end messaging is performed in layer 4, the transport layer. This layer provides a flow of data between two hosts. The Transmission Control Protocol or TCP is often used here. User Datagram Protocol (UDP) is also used in the transport layer. The transport layer acknowledges the receipt of packets, accepts acknowledgments from the other end and distributes the packets to the appropriate applications. The transport layer determines when retransmission is needed. Flow control is also implemented at this layer.

The session layer and the presentation layer are usually not implemented in most networks. The purpose of the two layers is to control or manage the session and to change the format of the information being sent or received. A session is a single connection between two hosts for a specified duration of time. For example, one file transfer is a session. The format of one machine is often not compatible with the format on the one with which it is communicating. The presentation layer interprets the format

and changes it to one that is readable. Instead of having a program to change from every format to every other format, a network format has been implemented. Each machine has a program to change from its format to the network format. When messages are sent, they are changed to network format. The receiver gets the message, and translates the message from network format to its own format. The network format means that every machine needs only two programs: one to change to network format, and one to change from network format.

The last layer of the network protocol is the application layer. This layer handles the details of a particular application. If all other layers work properly, the application layer can ignore the details and simply provide user interface. Some examples of TCP/IP applications are Telnet, File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP) and Simple Network Management Protocol (SNMP).

TCP

Transmission Control Protocol is a layer four protocol. TCP is used to implement end-to-end messages, meaning that the entire message is sent from one host to another. TCP also performs error detection and correction. TCP also helps to determine the rate at which information is sent by employing time outs and round trip time estimates. These functions will be explained in the next section. TCP is one of the protocols which causes problems in wireless. The TCP protocol was designed to work with wired connections. Wired connections have high bandwidth and little error in the transmission. Wireless

connections, on the other hand have low bandwidth and a high amount of error in their transmissions. Efforts are being made to alter the TCP protocol so that it is more reliable when used in wireless data transmissions. There are many other factors involved in TCP transmissions.

How TCP Works

TCP works by accepting streams of data from local processes such as FTP. TCP then breaks the data into distinct. Each message is then sent as an IP datagram. An IP datagram is what the third layer calls the information that it sends. At the receiving end, TCP receives an IP datagram and reconstructs the original stream of data from this datagram. Both the sender and the receiver must agree on end points called sockets. To obtain TCP service, a connection must be explicitly established between a socket on the sender and a socket on the host using socket calls. Socket calls are listed in Appendix A. One socket can be used for multiple connections at one time.

Windowing is a way in which TCP can send more than one message at a time. When TCP sends a message, it must wait for an acknowledgment from the receiver before it can discard this message. Another TCP device is the time-out. Every time a message is sent, a time out timer is started. If the timer goes off, the message must be re-sent. This time out ensures that TCP does not wait for an indefinitely long period of time for an

acknowledgment. In the original TCP protocol, TCP would send a single message and wait for the acknowledgment before trying to send another. This wastes time because the sender must remain idle while waiting for an acknowledgment. The windowing implementation lets the network designer specify a certain window size that may be sent. The sender can then transmit the first message and start sending the next message without needing to wait for the acknowledgment of the first. Time outs are still used on each message that is sent. The sender can transmit up to the window size (n) number of packets before it has to wait for an acknowledgment. Once an acknowledgment has been received for the first packet, the window can be incremented by one. See figure 2.

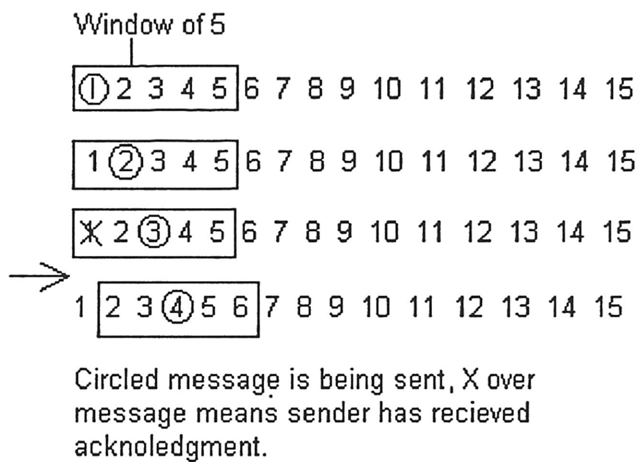


Figure 2

Windowing can cause several problems in the receipt of data. For example, the receiver has the problem of messages being received out of order. A message can be lost while in transit to the receiver. With windowing, the next message will be sent regardless of whether the first arrived. Messages can arrive out of order even without losing the first.

Because each message travels separately through the network, each one can take a different path. Some paths may take longer than others. Therefore, message two may arrive before message one because message two took a shorter, faster path.

Most systems are designed so that an acknowledgment will only be sent if all the previous packets have arrived. For example, if message one, two and five have arrived, an acknowledgment will be sent for message two, but not message five. Because of this feature, the sender may time out on messages three and up. The sender will re-send messages three through five. The receiver already has message five, therefore it now has a duplicate message. See figure 3 for clarification. Most receivers will discard the duplicate. Some researchers are working on a way to send selective acknowledgments. The goal is to send an acknowledgment telling the receiver exactly which messages have arrived as opposed to the cumulative acknowledgment currently sent. This would help to eliminate the problem of duplicate messages. Duplicate messages may also occur if the acknowledgment sent by the receiver gets lost in the network.

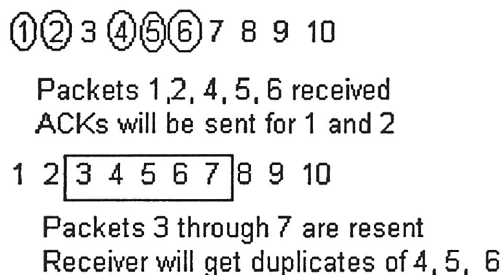


Figure 3

When messages get lost in a wired system, the usual cause is congestion. When a network system gets congested, some of the routing devices will discard random messages to lighted the load on the system. When this happens in a wired system, the best action the sender can take is to slow down the rate at which messages are being sent. How much the receiver should slow down is open to debate. TCP works off of the assumption that all losses are due to congestion and much research time has been spent determining how to slow down and speed up the send rates. TCP tries to avoid congestion initially by using a slow start algorithm. The sender and receiver negotiate a maximum segment size initially, for example 8K. If the sender knows that more than 4K will clog the network, it will set the maximum to 4K instead. This segment size is used to initialize a congestion window. The sender will then send a segment of the negotiated maximum size. If this segment is received and acknowledged before time out, the sender increases the congestion window size by one more segment. Now, as each segment is acknowledged, the congestion window is increased by one maximum length segment. If the congestion window is one, one packet is sent. When the acknowledgment is received, the congestion window is increased by one and is now two. Next, two segments are sent. When both of the acknowledgments arrive, the congestion window will be increased by two. The congestion window is now four. In effect, as each set of segments is sent and acknowledged on time, the congestion window is doubled. The Internet provides another parameter to alleviate congestion when it occurs called the threshold. The threshold is initially set to 64K. Whenever a time-out occurs, the threshold is divided by two and the congestion window is reset to one maximum size segment. The slow start algorithm is used to build up the congestion window, but the window can become no larger than the

threshold. From this point on, the congestion window grows linearly by one. See figure 4 for an illustration.

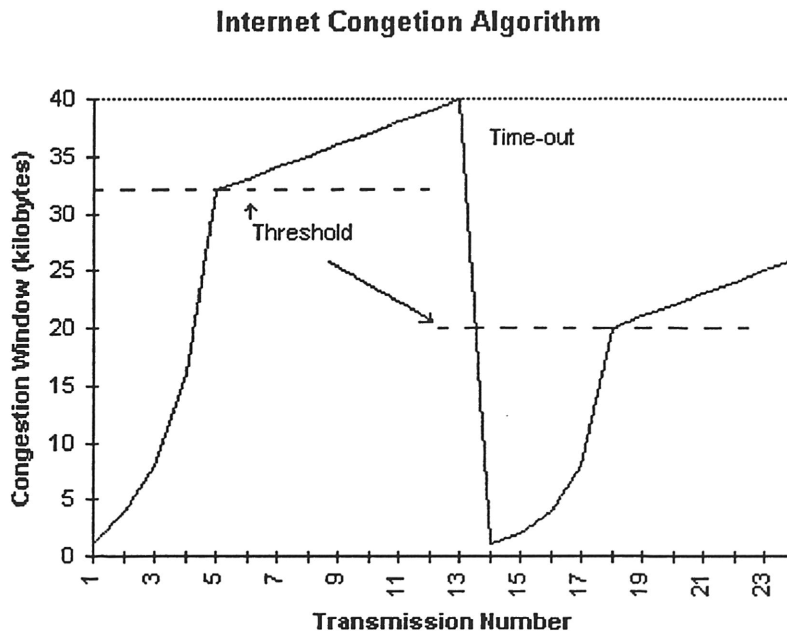


Figure 4 [13]

The estimate of the round trip time is also open to debate. The round trip time can be set to a constant value or dynamically calculated during connections. A dynamic calculation should yield better performance because the system will automatically react to changing network traffic and conditions. When making a dynamic estimate, should all packets sent out be used in the estimate? When packets are lost due to congestion, should their times be used in the estimate of the round trip time, or should these times be ignored? When congestion is high on a network, many packets may be discarded. If these values are used in the round trip time estimate, the time can be unnecessarily high. If these times are thrown out, will this cause the time to be too low? Because the round trip time is used

as the time-out value, unnaturally high or low values will cause degraded network performance. Values that are too high will reduce throughput and slow down the system. Throughput is the number of bytes, or kilobytes sent per second. Values that are too low will cause a large number of messages to be discarded. This will also decrease throughput and slow down the entire network. Phil Karn discovered this problem and offered a simple solution: do not use any times from messages that must be retransmitted in the round trip time calculation. Round trip times are measured starting with the time a message is sent and ending when the acknowledgment for that packet is received at the sender. After round trip times are calculated, time-outs can be determined dynamically using the round trip time (RTT). The most common function used today in TCP connections is Time-out = RTT + 4*D and $D = \alpha D + (1 - \alpha)$. In this formula, α is usually 7/8. These formulas can be attributed to Jacobsen [13].

TCP was designed to work well in a wired environment. In a wired environment, the error rate is low and the bandwidth is high. Under these conditions, a valid assumption is that all message loss is due to congestion. In a wireless environment, most message loss is due to errors and noise within the connection. Because there are many factors such as radio waves, weather, and other wireless devices around a wireless signal, errors are very common. When an error occurs, the message is corrupted and will be discarded by the receiver. For this reason, enhancements must be added to TCP to improve its performance in wireless connections.

Current Research and Methods

In the April 1995 issue of Personal Communication [8], Donald C. Cox discusses each of the current mobile devices in depth. He explains the advancements and limitations of all of these devices and the trend in mobile communication towards low-tier Personal Communication Services (PCS). Cox indicates that the current difficulty in organizing wireless computing is within the computer industry which views wireless as a plug-in interface card. The computing industry has not given the idea sufficient consideration or research for wireless to be a feasible option at this time.

The December 1995 issue of IEEE Personal Communications [4] has an article detailing a proposed architecture and future vision for a nomadic computing system. Some of the issues addressed are file caching and problems with changing connection methods. The nomadic computing environment must be adaptable. All modes of connection should be possible, and the computer should, of course, be portable. The change from connecting with a LAN, through a Modem, and possibly a wireless connection causes considerable design complications. The article also discusses several methods for testing the performance of the proposed nomadic systems. Examples of two of the primary components of the system - Instant Infrastructure Protocol and File Replication are also given.

Trial Systems

Several systems are currently being researched to improve the performance of TCP is a wireless network connection.

Split Connection

There are several ideas currently being researched to try and improve TCP for wireless networks. One of these ideas is to split the connection. Indirect-TCP [1] uses this approach. The network connection is divided into a wired and a wireless portion. In a split system, classic TCP can be used on the wired portion. A new version of TCP is used on the wireless connection. This will separate the wired portion from the problems associated with wireless networks.

Fast Retransmission

The fast retransmission approach deals only with the delays involved in handoffs from one base station to another. Fast retransmit works by having the mobile host send a threshold number of duplicate acknowledgments to the sender. TCP at the sender will immediately reduce the window size and start retransmitting from the first duplicate acknowledgment. Unfortunately, this approach only affects the problems occurring during handoffs, not the error characteristics inherent in the wireless link.

Link Level Retransmission

Another improvement being researched is the link-level retransmission approach. In this approach, the wireless link implements a retransmission protocol along with forward error correction at the link level. Studies have shown, however, that this and other independent retransmission protocols can actually degrade performance. The advantage of an independent transmission protocol is that it can improve the reliability of communication with out affecting any of the higher layers.

I-TCP

One system currently being tried at Rutgers University to improve TCP for wireless links is I-TCP (Indirect TCP).[1] I-TCP works by splitting the wireless connection into two separate interactions: one between the mobile host (MH) and its mobile support router (MSR) wirelessly and one from the MSR to the fixed host (FH) in a wired connection. All of the specialized TCP enhancements occur on the wireless link. The other connection can use traditional TCP with no problems. The MSR acts in the same manner as a cell station in a cellular phone connection. As the mobile host moves around, the mobile connection is transferred from one MSR to another. The fixed host is completely isolated from the handoff and is, therefore, unaffected by the switch. The throughput was measured under four different cell configurations: 1 No moves; 2 Moves between overlapped cells; 3 Moves between non-overlapped cells with 0 seconds between cells; 4 Moves between non-overlapped cells with 1 second between cells. Tables 1 and 2 show performance in local area networks and wide area networks.

Connection Type	No Moves	Overlapped Cells	Non-overlapped cells, 0 sec	Non-overlapped cells, 1 sec
Regular TCP	65.49 KB/s	62.59 KB/s	38.66 KB/s	23.73 KB/s
I-TCP	70.06 KB/s	65.37 KB/s	44.83 KB/s	36.31 KB/s

Table 1: LAN [1]

As can be seen in Table 1, overall I-TCP performed slightly better than regular TCP in a LAN situation. In the first two cases, regular TCP was expected to perform better because of the overhead needed to establish two connections. The reason I-TCP is thought to perform better is the FH gets more uniform round trip times using I-TCP.

Connection Type	No Moves	Overlapped Cells	Non-overlapped cells, 0 sec	Non-overlapped cells, 1 sec
Regular TCP	13.35 KB/s	13.26 KB/s	8.89 KB/s	5.19 KB/s
I-TCP	26.78 KB/s	27.97 KB/s	19.12 KB/s	16.01 KB/s

Table 2: WAN [1]

As shown in Table 2, I-TCP performed significantly better than regular TCP in a WAN situation. The differences between the two systems are more obvious because of the relatively long round trip delays inherent in a wide area network. Losses in the wireless link will have a significant effect on the throughput because of these long delays.

Snoop

The Snoop [3] protocol being researched at the University of California at Berkley monitors every packet that passes through the TCP connection in both directions. A cache is maintained with TCP segments that have been sent, but not acknowledged. Packet losses are detected by either a time-out occurring or a specified number of duplicate acknowledgments being received. The snoop agent will retransmit the packet if the packet resides in the cache. The snoop agent will then suppress the duplicate acknowledgments. This protocol is classified as a link layer protocol which takes advantage of the knowledge of the transport protocol. The advantage is that packets are retransmitted locally, avoiding congestion delays from the sender.

Selective Acknowledgments

Another idea being researched is the idea of selective acknowledgments, [3] SACK. In this system, acknowledgment would be sent per message with an identifier. An acknowledgment could be sent using the normal contiguous form, but it would include the packet that caused the SACK to be sent. Using this information, the sender can create a bitmask of packets that have been received. The Internet Draft proposes that each SACK contain information about three non-contiguous blocks of data which have been successfully received. Each block could be described by its starting and ending address.

SCPS-TP

The MITRE Corporation, Gemini Industries, NASA, and the Department of Defense have developed a set of extensions for TCP to be used in satellite

communications: the Space Communications Protocol Standards - Transport Protocol (SCPS-TP)[14]. Many of these enhancements can be applied to improve TCP performance on wireless links. The loss assumptions are set differently each time according to the network environment being used. Instead of a default assumption of congestion causing loss, error could be the default loss assumption in a wireless link. The assumptions could be adjusted when the sender communicates with a different client, or the mobile host uses a wired connection. SCPS-TP implements selective *negative* acknowledgments. This option serves the same purpose as the selective acknowledgments scheme discussed earlier, but it is implemented differently. This option provides the sender with more details of lost packets. SCPS-TP uses a different start up scheme sometimes referred to as TCP Vegas. This scheme increases the congestion window size more slowly, and tests the throughput gain after each increase to avoid any congestive losses. SCPS-TP performs well in a high delay system with high error rates. This design is good for satellite communication because of the large delays incurred when sending data to and from a satellite. SCPS-TP would also be a good candidate for a split network connection. The ability to use selective acknowledgments and distinguish between wireless and congestive losses could greatly enhance wireless TCP performance. Despite these advantages, SCPS-TP contains a significant number of changes to classic TCP; therefore, the majority of fixed hosts would be unable to benefit from it. According to its authors [9], SCPS-TP requires a great deal of tuning since it is very sensitive to round trip times.

My Work

The work that I did this semester involved TCP Reno and SNOOP TCP. These two schemes are implemented in the wireless lab in Bright. The lab contain a mobile host, a fixed host, and a base station, see Figure 5.

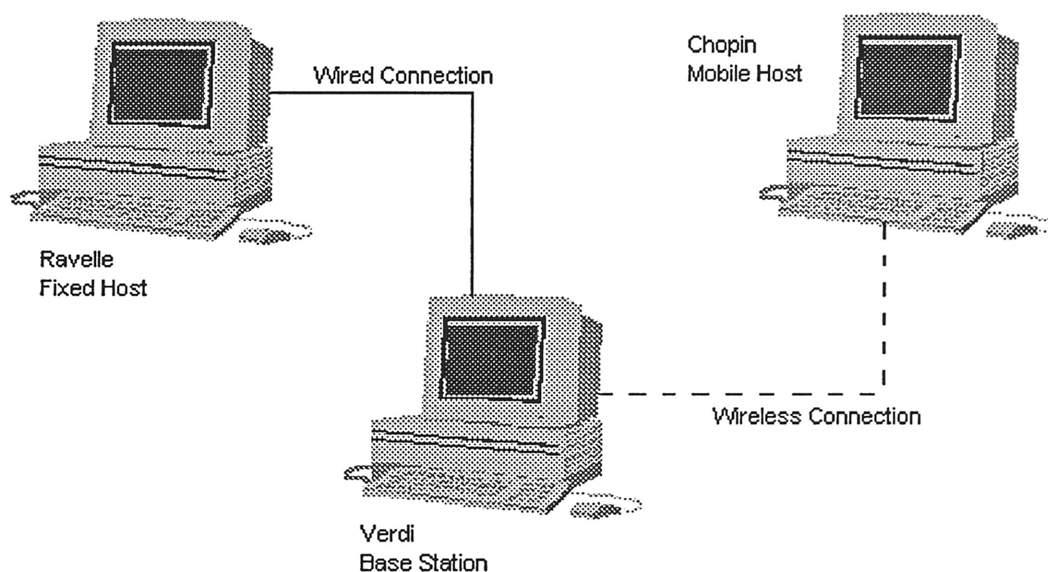


Figure 5

Error Injection

Because of the stability of the lab environment, a program was used to inject errors into data transfers between the three machines. This error injection was necessary to test the performance of TCP Reno versus snoop. The setup was tested using a bit error model implementation. The bit error model used was obtained from UC Berkley. The program

can use either a Poisson-distributed bit error or a Markov model. In my experiments, I always used the Poisson distribution.

The Poisson model can damage a single byte or create burst errors which will damage several packets in a row. One copy of the model runs on the mobile host to damage data packets, and another runs at the base station to damage the acknowledgments. Either the IP checksum or the TCP checksum is modified to introduce an error. Which checksum is modified depends on where the error occurs in the packet. This error in the checksums will force either the TCP or IP layer to discard the packet. The model then calculates the interval in bytes between injected error just injected and the place that the next error should occur. A table of 50,000 exponentially distributed integers is maintained to determine the interval. A number is randomly selected by the model and then scaled by the mean error rate. This scaled value is then used to determine the interval between errors. The mean error rate is a variable which can be changed by the experimenter.

The model does not exactly represent the way in which error would occur in a real situation, but the Poisson distribution is effective because it has a variance equal to its mean. As a result, errors are distributed unevenly.

Test Sender Program

The test sender application program allows users to send test data from the fixed host to the mobile host. The user can select a destination machine address and port

number. The default machine is Chopin, the mobile host; the default port number is 9, the discard server on a UNIX machine. The discard sever appears like a regular TCP receiver to the sender, but it throws away every packet it receives. This will let the user send data repeatedly without overloading the destination machine. The test sender program also lets the user to enable a debugging option which will collect information on all of the data sent. Enabling the debugger will create a record of incoming packets and the state of the TCP control block. This information can be examined later using trpt. The program also records the transfer time to the nearest millisecond. A copy of the test sender source code is included in Appendix B.

Configuration Program

The wireless configuration program allows the user to check and set the error model parameters and TCP enhancement features. This program allows the user to choose whether snoop and the error model are enabled or disabled. Other features such as the mean error rate and the burst error rate may be modified with this program. I used the program to change the mean error rate to compare the performance of TCP with and without snoop. The source code for the wireless configuration program is included in Appendix B.

Testing

In my tests, I used the mean error rate to compare TCP Reno and the snoop enhancement. Using the configuration program, I chose to enable the error model. Without running snoop, I varied the mean error rate. The mean error rate determines the

scaling factor which determines the interval between errors. A mean error rate of 20,000 implies there will be an error approximately every 20,000 bytes. The mean error rate was varied between 25,000 and 95,000. Figure six shows the transfer time versus the mean error rate for TCP Reno.

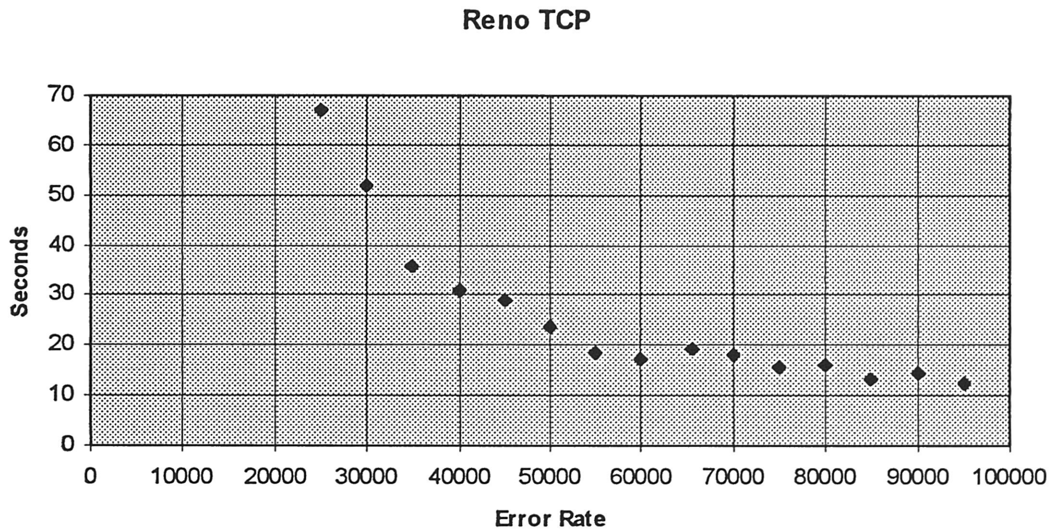


Figure 6

As can be seen in the figure, TCP transfer times become very large as the mean error rate becomes small. 2,000,000 bytes are being sent on each test. A mean error rate of 10,000 would yield a transfer time of five or more minutes which is unreasonable in a computing environment. Each mean error rate value was tested between ten and twenty times to get a good average value. The actual test data is included in Appendix A.

The snoop agent was then enabled, and the same test runs were performed. Figure seven shows the average transfer times for the snoop implementation versus the mean error rate.

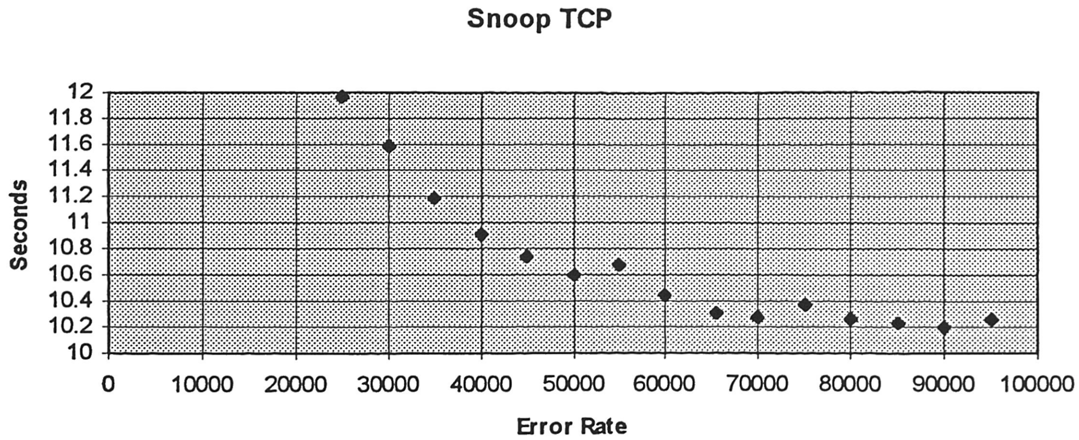


Figure 7

The times using the snoop agent were much better than TCP. The times varied less and remained lower. Figure eight shows a plot of the throughput of TCP with and without snoop.

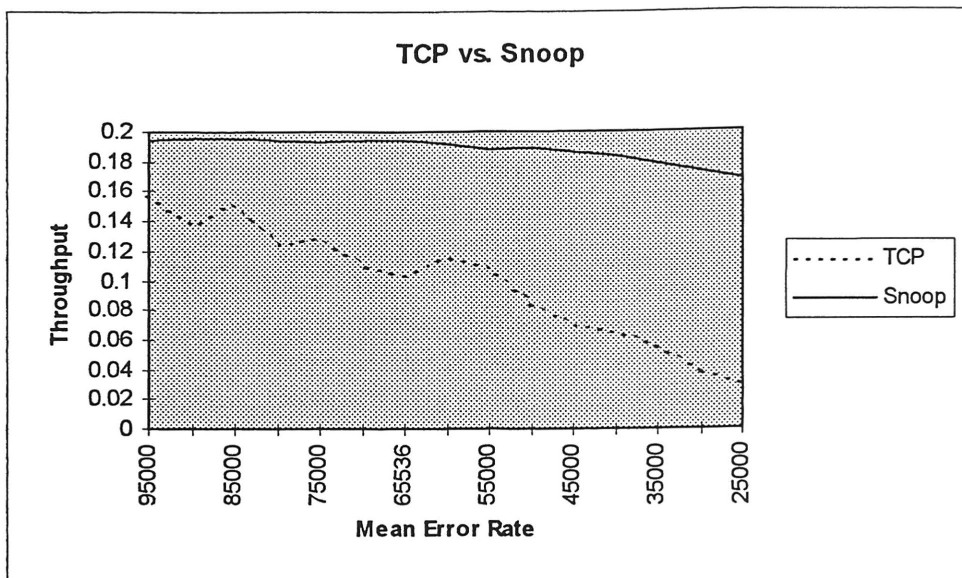


Figure 8

As shown in the graph, snoop TCP performs at a higher and more consistent rate. TCP's performance degrades rapidly as the interval between errors increases. Snoop's performance degrades a little, but in comparison, this degradation seems minimal.

Future Work

As time permits, I would like to adjust the duplicate acknowledgment values for the snoop protocol. In theory, a lower value for the duplicate acknowledgments should improve the throughput. The value currently used is three. If this value was decreased, I think that performance would be improved.

Future Ideas

An article in Byte magazine [5] shows how sometimes the entertainment industry can predict the future. In the TV series "Get Smart", main character Maxwell Smart is often show as having a telephone in his shoe. Research at MIT is involved with developing a PC small enough to fit into the heel of a shoe. This wearable PC is designed to help record anything the user might ever possibly need. A head mounted camera is used along with a microphone to record everything the user wants. Wireless options such as a global positioning system and cellular modem are included in the PC. Because of the large amount of storage capacity needed, a server is the ideal place to put the information gathered. This is the reason the wireless modem was included. Other technologies include using human skin to act as a transport medium for an Ethernet network. In an actual trial,

the information from one person's business card was transmitted to another person just through the act of shaking hands.

Conclusion

The ability to communicate without being tied down is a dream of society. Everyone wants to be able to work while reclining in bed, or at their favorite beach. Cellular technology seems to be the most promising avenue of expansion to meet this goal. Because of the already popular cellular phone, implementing CDPD seems to be the best way to begin wireless implementations. Before this system can become reliable enough to use everyday and by everyone, the TCP protocol must be adapted to handle wireless data transfers with an acceptable degree of reliability. The snoop agent has proven to work well in a laboratory environment. This enhancement may allow users more freedom in their movement. At this point, snoop seems to be the answer to many of TCP's performance problems. More tests should be run outside of the laboratory to determine the real performance of snoop.

Bibliography

[1] A. Bakre and B. R. Badrinath, "I-TCP : Indirect TCP for Mobile Hosts", Department of Computer Science, Rutgers University, October 1994.

[2] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving TCP/IP performance over Wireless Networks", Computer Science Division, University of California at Berkley, November 1995.

[3] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links", Computer Science Division, University of California at Berkley, August 1996.

[4] R. Bargrodia, W. Chu, L. Kleinrock and G. Popek, "Vision, Issues, and Architecture for Nomadic Computing," *IEEE Personal Communications*, vol. 2, no. 6, December 1995.

[5] N. Baron, "Get Smart - Wear a PC" *Byte*, v. 21, March 1996.

[6] T. E. Bell, J. A. Adam, and A. J Lowe, "Communications", *IEEE Spectrum*, January 1996, pp. 30-41.

[7] S. Biaz, M. Mehta, S. West, and N. H. Vaidya, "TCP over Wireless Networks Using Multiple Acknowledgments", Department of Computer Science, Texas A&M University, January 1997.

[8] D. Cox, "Wireless Personal Communications: What is it?," *IEEE Personal Communications*, vol. 2, no. 2, April 1995.

[9] R. C. Durst, G.J. Miller, and E. J. Travis, "TCP Extensions for Space Communications", *Proceeding of MOBICOM '96*, pp. 15-26, November 1996.

[10] J. A. Martin, "The Wireless Wanderer", *PC World*, v.12 August 1994.

[11] A. Seybold, "Wireless Mobility?", *Andrew Sybold's Outlook on Communications and Computing,* October 1995.

[12] W. R. Stevens, *TCP/IP Illustrated, Volume 1 The Protocols*, Addison-Wesley Publishing, New York, 1994, pp.2-3.

[13] A. S. Tannenbaum, *Computer Networks Third Edition,* Prentice Hall PTR, Upper Saddle River, New Jersey, 1996, pp. 28-36.

[14] S. M. West, "TCP Enhancements for Heterogeneous Networks", Department of Computer Science, Texas A&M University, March 21, 1997.

Appendix A

Data Files

Socket Calls

Primitive	Meaning
SOCKET	Create a new connection end point
BIND	Attach a local address to a socket
LISTEN	Announce a willingness to accept connections; giv queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

[13]

Reno TCP

Reno TCP								
Mean Error Rate	95000	90000	85000	80000	75000	70000	65536	60000
Transfer Time (s)	11.947	11.101	15.17	21.006	17.931	15.203	28.59	14.92
Transfer Time (s)	11.191	13.687	16.919	13.884	17.269	12.204	19.812	17.894
Transfer Time (s)	11.409	12.24	9.98	17.34	12.58	14.05	18.098	21.821
Transfer Time (s)	12.118	11.026	10.064	10.103	12.394	16.648	13.663	13.967
Transfer Time (s)	13.747	9.915	17.779	21.208	12.697	18.8	13.944	14.58
Transfer Time (s)	12.595	13.994	10.064	13.702	11.444	19.691	23.784	12.728
Transfer Time (s)	12.82	15.258	12.03	22.859	14.217	29.997	19.569	14.102
Transfer Time (s)	10.125	13.922	18.353	12.739	22.249	19.019	16.543	16.827
Transfer Time (s)	14.232	15.418	10.074	18.07	20.692	18.547	14.817	23.635
Transfer Time (s)	16.523	12.917	12.432	15.138	13.936	21.334	26.911	22.606
Transfer Time (s)		13.469		12.317		25.444	14.104	
Transfer Time (s)		12.424		16.595		12.076	14.733	
Transfer Time (s)		16.781		12.274		24.853	19.34	
Transfer Time (s)		16.564		23.454		32.649	18.276	
Transfer Time (s)		14.318		13.698		14.29	16.541	
Transfer Time (s)		34.519		16.737		16.382	24.9	
Transfer Time (s)		14.097		15.388		16.682	20.384	
Transfer Time (s)		12.232		14.331		9.975	31.071	
Transfer Time (s)		18.416		15.894		12.467	15.336	
Transfer Time (s)		9.923		15.126		12.91	17.561	
Mean Error Rate	95000	90000	85000	80000	75000	70000	65536	60000
Avg Transfer Time	12.6707	14.61105	13.2865	16.09315	15.5409	18.16105	19.39885	17.308

Reno TCP

55000	50000	45000	40000	35000	30000	25000
16.335	22.343	29.974	38.9	45.422	49.161	59.434
21.087	16.221	28.683	31.534	54.347	39.383	83.603
23.946	23.277	33.577	30.99	50.385	33.783	53.844
25.952	19.915	31.847	23.786	34.626	42.946	59.358
13.834	12.933	25.957	28.548	17.682	54.343	97.458
13.735	26.748	21.826	31.987	14.402	45.53	78.602
15.27	29.564	24.547	32.139	31.598	42.365	77.587
14.535	23.293	16.991	30.66	25.347	64.29	44.738
18.923	25.148	16.815	28.54	45.701	62.914	62.96
21.196	25.256	58.453	34.85	36.56	33.236	53.831
	24.281		17.402		75.453	
	27.891		34.919		63.606	
	23.361		42.35		69.747	
	27.562		27.977		52.317	
	38.266		27.511		37.899	
	16.868		29		36.799	
	19.275		23.189		63.366	
	29.317		33.765		49.501	
	22.158		32.395		50.851	
	20.94		37.702		67.581	
55000	50000	45000	40000	35000	30000	25000
18.4813	23.73085	28.867	30.9072	35.607	51.75355	67.1415

Snoop TCP

Snoop								
Mean Error Rate	95000	90000	85000	80000	75000	70000	65536	60000
Transfer Time (s)	10.215	10.08	11.158	10.254	10.116	10.167	10.327	10.484
Transfer Time (s)	10.106	10.204	10.101	10.583	11.932	10.252	10.302	10.542
Transfer Time (s)	10.623	10.845	10.053	9.976	11.333	10.524	10.643	11.061
Transfer Time (s)	10.019	10.093	10.455	10.565	10.222	10.042	10.06	10.296
Transfer Time (s)	10.03	10.091	10.234	10.053	10.628	10.556	10.66	10.248
Transfer Time (s)	10.418	10.086	10.377	10.512	10.267	10.3	10.311	10.158
Transfer Time (s)	10.255	10.163	10.033	10.285	10.001	10.269	10.262	10.398
Transfer Time (s)	10.058	10.007	10.258	10.152	10.084	10.752	10.203	10.252
Transfer Time (s)	10.742	10.213	10.062	10.091	10.403	10.095	10.198	10.448
Transfer Time (s)	10.188	10.382	10.064	10.499	10.165	10.27	10.786	10.193
Transfer Time (s)	10.726	9.9	10.133	10.057	10.061	10.371	10.128	10.412
Transfer Time (s)	10.041	10.063	10.77	10.141	10.171	10.017	10.581	10.595
Transfer Time (s)	10.291	10.543	10.04	10.104	10.397	10.235	10.342	10.879
Transfer Time (s)	10.074	10.042	10.03	10.304	10.386	10.157	10.404	10.33
Transfer Time (s)	10.11	10.014	10.057	10.318	10.025	10.181	10.158	10.576
Transfer Time (s)	10.271	10.307	10.139	10.036	10.316	10.052	10.077	10.442
Transfer Time (s)	10.623	10.661	10.063	10.067	10.145	10.219	10.46	10.136
Transfer Time (s)	10.196	10.028	10.044	10.703	10.336	10.229	10.188	10.632
Transfer Time (s)	10.078	10.18	10.2	10.395	10.204	10.161	10.128	10.772
Transfer Time (s)	10.271	10.034	10.314	10.241	10.325	10.582	10.089	10.107
Mean Error Rate	95000	90000	85000	80000	75000	70000	65536	60000
Avg. Transfer Time	10.26675	10.1968	10.22925	10.2668	10.37585	10.27155	10.31535	10.44805

Snoop TCP

55000	50000	45000	40000	35000	30000	25000
10.375	10.898	11.014	10.751	11.261	11.127	11.839
11.022	10.264	10.655	10.328	10.791	11.888	11.813
10.606	10.547	10.568	11.896	11.117	11.915	11.145
10.943	10.826	10.612	10.772	10.825	11.673	12.367
10.476	10.6	10.324	11.266	12.433	11.232	13.275
10.48	10.609	11.074	10.818	11.477	11.219	12.144
11.567	10.76	10.452	11.033	10.882	11.35	12.522
10.385	10.668	11.112	11.12	11.581	10.864	11.669
11.13	10.973	11.135	11.449	11.052	12.115	12.044
10.79	10.806	10.758	10.739	11.331	11.109	12.02
10.664	10.187	10.629	10.593	11.632	12.259	11.383
10.809	10.678	11.026	10.689	10.806	11.666	11.19
10.173	10.428	10.283	10.856	11.837	12.223	12.235
10.437	10.22	10.366	10.775	10.705	10.963	11.674
10.139	10.458	11.071	11.555	10.893	11.55	11.901
10.696	10.676	10.642	10.82	11.102	11.518	12.043
10.492	10.639	10.82	10.378	10.968	10.952	11.591
10.363	10.704	11.011	10.279	10.807	12.572	12.061
10.887	10.381	10.584	11.546	11.165	10.928	11.854
11.124	10.597	10.725	10.625	11.094	12.614	12.645
55000	50000	45000	40000	35000	30000	25000
10.6779	10.59595	10.74305	10.9144	11.18795	11.58685	11.97075

Appendix B

Program Files

```
/****************************************************************************
 *                                                                          *
 *                              Stephen West                                *
 *                           swest@cs.tamu.edu                              *
 *                          Test Sender Program                             *
 *                                                                          *
 *    Directions:  This file must be compiled as follows:                   *
 *                                                                          *
 *                 > cc -o test_sender test_sender.c                        *
 *                   or                                                     *
 *                 > cc -o test_sender test_sender.c -lsocket -lnsl         *
 *                                                                          *
 *    Reference:      Internetworking with TCP/IP, Douglas E.               *
 *                    Comer David L. Stevens Vol. I,III                     *
 *                    UNIX Networking Programing, Stevens                   *
 *                                                                          *
 *    Affiliation:    Department of Computer Science                        *
 *                    Texas A&M University                                  *
 *                    College Station, TX 77843-3112                        *
 *                                                                          *
 ****************************************************************************/

#include <sys/types.h>
#include <sys/param.h>
#include <sys/time.h>
#ifdef  _AIX
#include <sys/select.h>
#endif
#include <sys/socket.h>
#include <sys/file.h>
#include <sys/ioctl.h>

#include <netinet/in_systm.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_var.h>
#include <netinet/ip_icmp.h>
#include <netinet/udp.h>
#include <netinet/tcp.h>
#include <netdb.h>
#include <ctype.h>

#include <stdio.h>
#include <errno.h>
#include <string.h>

#define TCP_TAMU_STATS

#define INPUT_LINE          80
#define TEST_ARRAY_SIZE 2000000


/* -------------------------- Function Prototypes ------------------------- */

int main_menu(void);
int tcp_stats_menu(void);
void send_test_data(void);
void sys_err(char *str);
void sys_msg(char *str);
void test_array_init(void);
long timediff(struct timeval lasttime);
int write_n(int fd);


/****************************************************************************
 *
 *  This program is used to send a stream of data from the local machine to a
 *  remote machine using a TCP connection.  The performance obtained during the
 *  transfer can then be measured both at the sender and the receiver.
 *
 ****************************************************************************/
```

```
char Test_Array[TEST_ARRAY_SIZE];
char dest_machine[INPUT_LINE] = "chopin-wv";
int dest_port=9;
int enable_stats=0;

main()
{

    int response=0, sub_menu_selection=0;

/*
 * Initialize the array structure used for sending transfers.
 */

    test_array_init();

    while(response != 1){
        response = main_menu();
        switch(response){
            case 1:
                break;
            case 2:
                sys_msg("FEATURE NOT IMPLEMENTED IN SENDER PROGRAM");
                break;
            case 3:
                send_test_data();
                break;
            case 4:
                sub_menu_selection = tcp_stats_menu();
                switch(sub_menu_selection){
                    case 1:
                        break;
                    case 2:
                        enable_stats = 1;
                        sys_msg("\n\n\t TCP stats collection enabled \n");
                        break;
                    case 3:
                        enable_stats = 0;
                        sys_msg("\n\n\t TCP stats collection disabled \n");
                        break;
                    default:
                        sys_err("Invalid Sub-Menu Response Returned");
                        break;
                }
                break;
            default:
                sys_err("Invalid Main Menu Response Returned");
                break;
        }
    }
    system("clear");

}


/***************************************************************************
 *
 *    The function main_menu displays the main menu for the wireless
 *    configuration program.
 *        Input:    User Selections
 *        Output:   Menu Number Related To The User's Selection
 *
 ***************************************************************************/
int main_menu(void)
{

    int response=0;

    while((response < 1) || (response > 4)){
        system("clear");
```

```
        printf("\n\n\n\t\t TCP NETWORK TEST PROGRAM \n");
        printf("\t\t\t Main Menu \n\n\n");
        printf("\t 1) Exit the Program\n");
        printf("\t 2) Set/Check TCP Enhancements - Not Needed at Sender\n");
        printf("\t 3) Send Test Data\n");
        printf("\t 4) Set/Check TCP Statistics\n");
        printf("\n\n\t Enter Selection (1 - 4) --> ");

        scanf("%d", &response);
    }

    return response;
}


/*****************************************************************************
 *
 *    The function tcp_stats_menu allows the user to enable and disable tcp
 *    statistical collection via tcp_trace().  Once stats collection is enabled,
 *    it will be used for each transfer until it is disabled.  The user must
 *    be careful not to overfill the stats buffer within the kernel during a
 *    transfer.  The buffer is circular and has a size of 2000 packets currently.
 *    After each run in which stats are collected, trpt should be run to store
 *    the stats to a file.
 *        Input:    User's Choice
 *        Output:   Menu Item Number Selected By The User
 *
 *****************************************************************************/
int tcp_stats_menu(void)
{

    int response=0;

    while((response < 1) || (response > 3)){
        system("clear");
        printf("\n\n\n\t\t TCP NETWORK TEST PROGRAM \n");
        printf("\t\t   TCP Statistics Menu \n\n");
        if(enable_stats == 1)
            printf("\t      TCP stats collection:  Enabled\n\n\n");
        if(enable_stats == 0)
            printf("\t    TCP stats collection:  Not Enabled\n\n\n");
        printf("\t 1) Exit to the main menu\n");
        printf("\t 2) Enable tcp stats collection \n");
        printf("\t 3) Disable tcp stats collection \n");
        printf("\n\n\t Enter Selection (1 - 3) --> ");

        scanf("%d", &response);
    }

    return response;
}


/*****************************************************************************
 *
 *  The function send_test_data is the heart of the program.  It sends a set of
 *  test data to the remote machine and records the transfer time and amount of
 *  data sent.  A menu is displayed which askes the user if they want to use
 *  the default settings, or enter new ones.  Any new settings entered by the
 *  user will become the new default.  If the enable_stats flag is set, socket
 *  debugging is also turned on within the kernel so that tcp_trace() will
 *  capture sender tcp statistics during the transfer.  This data can then be
 *  examined with the trpt program.  Socket debugging is on a per connection
 *  basis.  Therefore, it must be re-enabled for each transfer.  When the
 *  socket is closed, the debugging will automatically be turned off for that
 *  socket.
 *      Input:    User's Menu Choices and Destination Machine Name & Port Number
 *      Output:   Test Data to Remote Machine and Transfer Time & Size
 *
 *****************************************************************************/
void send_test_data(void)
```

```
{
    struct sockaddr_in dest_addr;
    char hostname[100], hostaddr[100];
    register struct hostent *hp;
    int debug_enable, debug_len;
    int option, option_len;
    int sockfd;

    int response1=0, response2=0;
    struct timeval start_time;
    long transfer_time_usec;
    double transfer_time_sec;
    int bytes_written;

    system("clear");
    printf("\t\t TCP NETWORK TEST PROGRAM\n");
    printf("\t    Destination Machine Selection Menu\n\n");

    printf("\t     Default dest. machine:  %s \n", dest_machine);
    printf("\t     Default dest. port number:  %d \n", dest_port);
    if(enable_stats == 1)
        printf("\t      TCP stats collection:  Enabled\n\n\n");
    if(enable_stats == 0)
        printf("\t    TCP stats collection:  Not Enabled\n\n\n");


/*
 * Get the destination machine from the user and try and convert it using
 * gethostbyname() function.
 */

    while((response1 < 1) || (response1 > 3)){
        printf("\t 1) Exit the Program \n");
        printf("\t 2) Use the default settings \n");
        printf("\t 3) Change the default settings.\n");
        printf("\n\t Enter Selection (1 - 3) --> ");
        scanf("%d", &response1);
    }

    if(response1 == 1)
        return;
    else if(response1 == 3){
        printf("\n\t Enter destination machine name  --> ", dest_machine);
        scanf("%s", dest_machine);
    }

    if((hp = gethostbyname(dest_machine)) == NULL){
        printf("\n\n\t Unable to find destination machine \"%s\"\n",
            dest_machine);
        printf("\t 1) Re-enter the destination machine name\n");
        printf("\t 2) Abort the transfer\n");
        printf("\t Enter Selection (1 - 2) --> ");
        scanf("%d", &response2);
    }

    while((hp == NULL) && (response2 !=2)){
        printf("\n\t Enter destination machine name  --> ", dest_machine);
        scanf("%s", dest_machine);
        if((hp = gethostbyname(dest_machine)) == NULL){
            printf("\n\n\t Unable to find destination machine \"%s\"\n",
                dest_machine);
            printf("\t 1) Re-enter the destination machine name\n");
            printf("\t 2) Abort the transfer\n");
            printf("\t Enter Selection (1 - 2) --> ");
            scanf("%d", &response2);
        }

    }

    if(response2 == 2)
        return;
```

```
    strcpy(hostname,hp->h_name);
    strcpy(hostaddr,(char *)inet_ntoa(*(struct in_addr *)hp->h_addr));


/*
 * Get the destination port from the user.
 */

    if(response1 == 3){
        printf("\t Enter the dest port number --> ");
        scanf("%d", &dest_port);
    }

    if((sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0){
        sys_msg("Socket Error - Aborting Transfer");
        return;
    }

    dest_addr.sin_addr.s_addr = inet_addr(hostaddr);
    dest_addr.sin_port = htons(dest_port);
    dest_addr.sin_family = AF_INET;
/*
 * Set socket option to monitor tcp variables and record them
 * in the buffer.
 */

    if(enable_stats == 1){
        debug_enable = 1;
        if(setsockopt(sockfd, SOL_SOCKET,SO_DEBUG,(char *) &debug_enable,
                sizeof(debug_enable)) < 0)
            printf("\t Unable to Set Socket to Debug Mode!\n");

        debug_enable = 0;
        debug_len = sizeof(debug_enable);
        if(getsockopt(sockfd,SOL_SOCKET,SO_DEBUG,(char *) &debug_enable,
                &debug_len) < 0)
            printf("\t Unable to Get Socket Debug Status!\n");

        if(debug_enable == 0)
            printf("\t SO_DEBUG Not Set to 1!\n");
        else
            printf("\n\t SO_DEBUG Successfully Turned On.\n");
    }


/*
 * Start the timer and open the connection.
 */

    printf("\t Starting transfer ... ");
    fflush(stdout);

    gettimeofday(&start_time, NULL);

    if(connect(sockfd,(struct sockaddr *)&dest_addr, sizeof(dest_addr)) < 0){
        sys_msg("\t Unable to Connect to Destination - Aborting Transfer");
        return;
    }

    bytes_written = write_n(sockfd);

    close(sockfd);


/*
 *    Finished with transfer.  Calculate amount of data sent and transfer time.
 */

    transfer_time_usec = timediff(start_time);
    transfer_time_sec = ((double) transfer_time_usec)/1000000;
```

```
    printf("%d Bytes Sent \n", bytes_written);
    printf("\t Total Transfer Time = %6.3f Seconds\n", transfer_time_sec);


    sys_msg("");

}


/*************************************************************************
 *
 *  The function sys_err takes a string and displays it for the user.
 *  Then it terminates the program with the error.  It is intended to be used
 *  with fatal errors which must terminate the program, not for program
 *  warnings.
 *      Inputs:    Error Message
 *      Outputs:   Message to The Screen
 *
 *************************************************************************/
void sys_err(char *str)
{
    printf("%s\n",str);
    exit(1);
}


/*************************************************************************
 *
 *  The function sys_msg takes a string and displays it for the user.  It then
 *  pauses until the user presses the return key.  Unlike sys_err, it is not
 *  intended to be used with fatal errors.  It displays non-fatal errors and
 *  other general information.
 *      Inputs:    Error Message
 *      Outputs:   Message to The Screen
 *
 *************************************************************************/
void sys_msg(char *str)
{
    int response;

    printf("%s\n",str);
    printf("\t Press the return key to continue\n");
    response = getchar();
    response = getchar();
}


/*************************************************************************
 *
 *  The function test_array_init fills the array of characters called
 *  Test_Array with character values in the range of 48-122 (decimal).  This
 *  is 0 - z in terms of characters.  The idea was to have easily recognized
 *  printable characters in case they are ever printed to the screen.
 *      Input:    Empty Array
 *      Output:   Initialized Array
 *
 *************************************************************************/
void test_array_init(void)
{
    int i;
    char j=48;

    for(i=0; i < TEST_ARRAY_SIZE; i++){
        Test_Array[i] = j;
        j++;
        if(j > 122)
            j = 48;
    }
}
```

```c
/***************************************************************************
 *
 *  The timediff function checks the time that has elapsed during the transfer.
 *      Input:    None
 *      Output:   The time interval in microseconds
 *
 ***************************************************************************/
long timediff(struct timeval lasttime)
{
    struct timeval curtime;
    long cur=0, last=0;

    gettimeofday(&curtime, NULL);
    cur=(curtime.tv_sec%10000)*1000000+curtime.tv_usec;
    last=(lasttime.tv_sec%10000)*1000000+lasttime.tv_usec;

    return (cur-last);
}


/***************************************************************************
 *
 *  The function write_n is a more robust method of writing a message to a
 *  socket.  It takes into account the fact that the write system call will
 *  not necessarily write the entire length of the buffer.  Therefore, it
 *  loops until the entire message has actually been written.
 *      Inputs:    Socket Descriptor
 *      Outputs:   It Returns the Number of Bytes Actually Written
 *
 ***************************************************************************/
int write_n(int fd)
{
    int nleft, nwritten=0;
    char *output_ptr = &Test_Array[0];

    nleft = TEST_ARRAY_SIZE;
    while(nleft > 0){
        nwritten = write(fd, output_ptr, nleft);
        nleft -= nwritten;
        output_ptr += nwritten;
    }
    return(TEST_ARRAY_SIZE - nleft);            /* return >= 0 */
}
```

```
/****************************************************************************
 *                                                                          *
 *                              Stephen West                                *
 *                            swest@cs.tamu.edu                             *
 *                       Wireless Configuration Program                     *
 *                                                                          *
 *     Directions:  This file should be compiled as follows:               *
 *                                                                          *
 *                  > cc -o config_wireless config_wireless.c               *
 *                                                                          *
 *     Reference:      Internetworking with TCP/IP, Douglas E.             *
 *                     Comer David L. Stevens Vol. I,III                   *
 *                     UNIX Networking Programing, Stevens                 *
 *                                                                          *
 *     Affiliation:    Department of Computer Science                      *
 *                     Texas A&M University                                *
 *                     College Station, TX 77843-3112                      *
 *                                                                          *
 ****************************************************************************/

#include <stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include <netdb.h>
#include <errno.h>
#include <sys/time.h>
#include <netinet/tcp.h>

/*
 * Removing this definition causes only those operations which may be
 * performed at the mobile host to be displayed on the main menu.
 */
#define BASE_STATION


/* -------------------------- Function Prototypes ------------------------- */

int ackp_enable_disable(int choice);
void error_enable_disable(int choice);
void error_model_disp_params(void);
int error_model_menu(void);
void error_set_burst_size(void);
void error_set_mean_rate(void);
void error_set_model_type(void);
void error_set_timer_gran(void);
void error_set_TRANS0(void);
void error_set_TRANS1(void);
int main_menu(void);
int snoop_enable_disable(int choice);
void sys_err(char *str);
void sys_msg(char *str);
int tcp_model_menu(void);


/****************************************************************************
 *
 *   This program is used to set parameters at the base station and the mobile
 *   host.  It creates a dummy socket and then uses that socket to make
 *   getsockopt() and setsockopt() system calls.  These calls read and write
 *   values from/to global variables defined in the kernel which control the
 *   error injection model and tcp performance improvement options.
 *
 ****************************************************************************/

int Dummy_Fd;         /* The dummy socket required to get/set sock options */

main()
{
    int response=0, sub_menu_selection=0;
```

```
/*
 * Assign a dummy TCP socket so that we are able to use setsockopt() and
 * getsockopt() to change error model and tcp model parameters and then
 * verify that the parameter was correctly set.
 */

    if((Dummy_Fd = socket(AF_INET,SOCK_STREAM,0)) < 0)
        sys_err("\t Dummy Socket Call Failed");


/*
 * This is the main program loop.
 */

    while(response != 1){
        response = main_menu();
        switch(response){
            case 1:
                break;
            case 2:
                sub_menu_selection = error_model_menu();
                switch(sub_menu_selection){
                    case 1:
                        break;
                    case 2:
                        error_model_disp_params();
                        break;
                    case 3:
                        error_enable_disable(1);
                        break;
                    case 4:
                        error_enable_disable(0);
                        break;
                    case 5:
                        error_set_mean_rate();
                        break;
                    case 6:
                        error_set_model_type();
                        break;
                    case 7:
                        error_set_TRANS0();
                        break;
                    case 8:
                        error_set_TRANS1();
                        break;
                    case 9:
                        error_set_timer_gran();
                        break;
                    case 10:
                        error_set_burst_size();
                        break;
                    default:
                        sys_err("Invalid Sub-Menu Response Returned");
                        break;
                }
                break;
            case 3:
                sub_menu_selection = tcp_model_menu();
                switch(sub_menu_selection){
                    case 1:
                        break;
                    case 2:
                        snoop_enable_disable(1);
                        break;
                    case 3:
                        snoop_enable_disable(0);
                        break;
                    case 4:
                        ackp_enable_disable(1);
                        break;
```

```
                    case 5:
                        ackp_enable_disable(0);
                        break;
                    default:
                        sys_err("Invalid Sub-Menu Response Returned");
                        break;
                }
                break;
            default:
                sys_err("Invalid Main Menu Response Returned");
                break;
        }
    }
    system("clear");
    close(Dummy_Fd);

}


/***************************************************************************
 *
 *  The function ackp_enable_disable looks at the parameter choice and if it
 *  is 1, the ackp code is enabled for incoming packets.  If choice is 0, the
 *  ackp code is disabled for incoming packets.  The option is called
 *  TCP_SNOOP_ACKP_ENABLE within the kernel and it sets the global variable
 *  snoop_ackp_enable defined in snoop.c
 *      Input:    User's Choice (Enabled/Disabled)
 *      Output:   Updated Kernel Value for Enabling/Disabling Ackp
 *
 ***************************************************************************/
int ackp_enable_disable(int choice)
{
    int ackp_enable;
    int option_len;

    if(choice == 1){
        ackp_enable = 1;
        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_ACKP_ENABLE,
                (char *) &ackp_enable, sizeof(ackp_enable)) < 0){
            sys_msg("\n\t Unable to Enable Ackp Option at Basestation!");
            return;
        }
        ackp_enable = 0;
        option_len = sizeof(ackp_enable);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_ACKP_ENABLE,
                (char *) &ackp_enable, &option_len) < 0){
            sys_msg("\n\t Unable to Get Ackp Enabled/Disabled Status!");
            return;
        }

        if(ackp_enable == 1)
            sys_msg("\n\t Ackp Option Has Been Enabled at Basestation!");
        else
            sys_msg("\n\t Error:  Ackp Was Not Turned On Within The Kernel");
    }
    else{
        ackp_enable = 0;
        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_ACKP_ENABLE,
                (char *) &ackp_enable, sizeof(ackp_enable)) < 0){
            sys_msg("\n\t Unable to Disable Ackp Option at Basestation!");
            return;
        }
        ackp_enable = 1;
        option_len = sizeof(ackp_enable);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_ACKP_ENABLE,
                (char *) &ackp_enable, &option_len) < 0){
            sys_msg("\n\t Unable to Get Ackp Enabled/Disabled Status!");
            return;
        }

        if(ackp_enable == 0)
```

```
            sys_msg("\n\t Ackp Option Has Been Disabled at Basestation!");
        else
            sys_msg("\n\t Error:  Ackp Was Not Turned Off Within The Kernel");
    }
}


/*******************************************************************************
 *
 *  The function error_enable_disable looks at the parameter choice and if it
 *  is 1, the bit error model is enabled for incoming packets on the wireless
 *  link.  If choice is 0, the bit error model is disabled for incoming packets
 *  on the wireless link.  The option is called TCP_SNOOP_BER_DISABLE within
 *  the kernel and it sets the global variable ber.disable defined in ber.c and
 *  ber.h
 *      Input:    User's Choice (Enabled/Disabled)
 *      Output:   Updated Kernel Value for Enabling/Disabling Error Injection
 *
 *******************************************************************************/
void error_enable_disable(int choice)
{
    int error_disable;
    int option_len;

    if(choice == 1){
        error_disable = 0;
        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_BER_DISABLE,
                (char *) &error_disable, sizeof(error_disable)) < 0){
            sys_msg("\n\t Unable to Set Error Model to The On State!");
            return;
        }
        error_disable = 1;
        option_len = sizeof(error_disable);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_BER_DISABLE,
                (char *) &error_disable, &option_len) < 0){
            sys_msg("\n\t Unable to Get Error Model Enabled/Disabled Status!");
            return;
        }

        if(error_disable == 1)
            sys_msg("\n\t Set Option Error - Error Model Was Not Turned On");
        else
            sys_msg("\n\t Error Model Has Been Enabled!");
    }
    else{
        error_disable = 1;
        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_BER_DISABLE,
                (char *) &error_disable, sizeof(error_disable)) < 0){
            sys_msg("\n\t Unable to Set Error Model to The Off State!");
            return;
        }
        error_disable = 0;
        option_len = sizeof(error_disable);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_BER_DISABLE,
                (char *) &error_disable, &option_len) < 0){
            sys_msg("\n\t Unable to Get Error Model Enabled/Disabled Status!");
            return;
        }

        if(error_disable == 0)
            sys_msg("\n\t Set Option Error - Error Model Was Not Turned Off");
        else
            sys_msg("\n\t Error Model Has Been Disabled!");
    }
}


/*******************************************************************************
 *
 *  The function error_model_disp_params creates a display screen showing the
 *  current settings for each of the bit error model parameters.  It is a read
```

```
 *  only screen.
 *      Input:    None
 *      Output:   Prints Values of All Kernel Bit Error Models to Screen
 *
 ***************************************************************************/
void error_model_disp_params(void)
{
    int err_option, option_len;

    system("clear");
    printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
    printf("\t\t   Current Error Model Values \n\n\n");

    option_len = sizeof(err_option);
    if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_ERRPROB0,(char *)
            &err_option, &option_len) < 0)
        sys_msg("\n\t Unable to Get ERRPROB0!\n");
    printf("\t ERRPROB0 \t = \t %d\n", err_option);

    option_len = sizeof(err_option);
    if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_ERRPROB1,(char *)
            &err_option, &option_len) < 0)
        sys_msg("\n\t Unable to Get ERRPROB1!\n");
    printf("\t ERRPROB1 \t = \t %d\n", err_option);

    option_len = sizeof(err_option);
    if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_BER_MODEL,(char *)
            &err_option, &option_len) < 0)
        sys_msg("\n\t Unable to Get BER_MODEL!\n");
    if(err_option == 0)
        printf("\t BER_MODEL \t = \t Poisson\n");
    else
        printf("\t BER_MODEL \t = \t Markov\n");

    option_len = sizeof(err_option);
    if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_TRANS0,(char *)
            &err_option, &option_len) < 0)
        sys_msg("\n\t Unable to Get TRANS0!\n");
    printf("\t TRANS0 \t = \t %d\n", err_option);

    option_len = sizeof(err_option);
    if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_TRANS1,(char *)
            &err_option, &option_len) < 0)
        sys_msg("\n\t Unable to Get TRANS1!\n");
    printf("\t TRANS1 \t = \t %d\n", err_option);

    option_len = sizeof(err_option);
    if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_TIMERGRAN,(char *)
            &err_option, &option_len) < 0)
        sys_msg("\n\t Unable to Get TIMERGRAN!\n");
    printf("\t TIMERGRAN \t = \t %d\n", err_option);

    option_len = sizeof(err_option);
    if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_BURSTRATE,(char *)
            &err_option, &option_len) < 0)
        sys_msg("\n\t Unable to Get BURSTRATE!\n");
    printf("\t BURSTRATE \t = \t %d\n", err_option);

    option_len = sizeof(err_option);
    if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_BER_DISABLE,(char *)
            &err_option, &option_len) < 0)
        sys_msg("\n\t Unable to Get BER_DISABLE!\n");
    if(err_option == 0)
        printf("\t BER_DISABLE \t = \t False (Injecting Errors)\n");
    else
        printf("\t BER_DISABLE \t = \t True (Not Injecting Errors)\n");

    sys_msg("");
}
```

```
/***************************************************************************
 *
 *  The function error_model_menu creates a menu screen which allows the user
 *  to choose an error model parameter to change.  The user may also choose to
 *  view all of the current settings or to exit without doing anything.
 *      Input:    None
 *      Output:   The Menu Option Selected By The User
 *
 ***************************************************************************/
int error_model_menu(void)
{
    int response=0;

    while((response < 1) || (response > 10)){
        system("clear");
        printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
        printf("\t\t        Error Model Menu \n\n\n");
        printf("\t 1)  Exit to Main Menu\n");
        printf("\t 2)  Check Status of Bit Error Parameters \n");
        printf("\t 3)  Enable Error Injection \n");
        printf("\t 4)  Disable Error Injection \n");
        printf("\t 5)  Change Mean Bit Error Rate (ERRPROB0) \n");
        printf("\t 6)  Change Bit Error Model (Markov or Poisson) \n");
        printf("\t 7)  Change Markov TRANS0 \n");
        printf("\t 8)  Change Markov TRANS1 \n");
        printf("\t 9)  Change Error Model Timer Granularity \n");
        printf("\t 10) Change Error Burst Size \n");
        printf("\n\n\t Enter Selection (1 - 10) --> ");

        scanf("%d", &response);
    }

    return response;
}


/***************************************************************************
 *
 *  The function error_set_burst_size allows the user to choose the number of
 *  packets in a row which will be injected with errors.  The default is a
 *  single packet.  The option is called TCP_SNOOP_BURSTRATE within the kernel,
 *  and it sets the global variable ber.burst_rate defined in ber.c and ber.h.
 *      Input:    User Entered Burst Size
 *      Output:   Updated Kernel Value for Burst Size
 *
 ***************************************************************************/
void error_set_burst_size(void)
{
    int response=0;
    int err_option=0;
    int option_len;

    while((response < 1) || (response > 2)){
        system("clear");
        printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
        printf("\t      Set Error Model's Packet Burst Size \n");
        printf("\t\t Default Burst Size is 1 packet \n\n\n");

        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_BURSTRATE,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Get Burst Size! \n");
            return;
        }
        printf("\t Current Burst Size = %d packets \n\n", err_option);

        printf("\t 1) Exit to Main Menu \n");
        printf("\t 2) Change Burst Size \n");
        printf("\n\n\t Enter Selection (1 - 2) --> ");

        scanf("%d", &response);
```

```
    }

    if(response == 1)
        return;
    else{
        printf("\n\n\t Enter New Burst Size --> ");
        scanf("%d", &err_option);

        if(err_option < 1){
            sys_msg("\n\t Burst Size Must Be Greater Than or Equal to 1");
            return;
        }

        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_BURSTRATE,
                (char *) &err_option, sizeof(err_option)) < 0){
            sys_msg("\n\t Unable to Change Burst Size!");
            return;
        }

        err_option = 0;
        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_BURSTRATE,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Read New Burst Size!\n");
            return;
        }
        if(err_option == 0)
            sys_msg("\n\t Set Option Error - Burst Size Was Not Changed");
        else{
            printf("\t New Burst Size = %d packets\n\n", err_option);
            sys_msg("");
        }
    }
}


/**************************************************************************
 *
 *  The function error_set_mean_rate allows the user to choose the mean number
 *  of bytes between errors.  The default value of 65,536 means that one out
 *  every 65,536 bytes will have an error injected or roughtly one out of
 *  every 45 packets.  The option is called TCP_SNOOP_ERRPROB1 within the
 *  kernel, and it sets the global variable ber.error_prob[1] defined in ber.c
 *  and ber.h.
 *      Input:    User Entered Mean Error Rate
 *      Output:   Updated Kernel Value for Mean Error Rate
 *
 **************************************************************************/
void error_set_mean_rate(void)
{
    int response=0;
    int err_option=0;
    int option_len;

    while((response < 1) || (response > 2)){
        system("clear");
        printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
        printf("\t       Set Error Model's Mean Error Rate \n");
        printf("\t       Default Rate is 1 Per 65536 Bytes \n\n\n");

        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_ERRPROB0,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Get Mean Error Rate (ERRPROB0)!\n");
            return;
        }
        printf("\t Current Mean Error Rate = 1 error per %d bytes\n\n",
                err_option);

        printf("\t 1) Exit to Main Menu\n");
        printf("\t 2) Change Mean Error Rate\n");
```

```
        printf("\n\n\t Enter Selection (1 - 2) --> ");

        scanf("%d", &response);
    }

    if(response == 1)
        return;
    else{
        printf("\n\n\t Enter New Mean Error Rate --> ");
        scanf("%d", &err_option);

        if(err_option < 1){
            sys_msg("\n\t Mean Error Rate Must Be Greater Than or Equal to 1");
            return;
        }

        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_ERRPROB0,
                (char *) &err_option, sizeof(err_option)) < 0){
            sys_msg("\n\t Unable to Change Mean Error Rate!");
            return;
        }

        err_option = 0;
        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_ERRPROB0,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Read New Mean Error Rate!\n");
            return;
        }

        if(err_option == 0)
            sys_msg("\n\t Set Option Error - Mean Error Rate Was Not Changed");
        else{
            printf("\t New Mean Error Rate = 1 error per %d bytes\n\n",
                    err_option);
            sys_msg("");
        }
    }
}


/************************************************************************
 *
 *  The function error_set_model allows the user to choose between a straight
-*  Poisson error model or a Markov error model with both good and bad states.
 *  The default is a Poisson error model.  The option is called
 *  TCP_SNOOP_BER_MODEL within the kernel, and it sets the global variable
 *  ber.model defined in ber.c and ber.h.
 *      Input:    User Entered Model Type
 *      Output:   Updated Kernel Value for Model Type
 *
 ************************************************************************/
void error_set_model_type(void)
{
    int response=0;
    int err_option=0;
    int option_len;

    while((response < 1) || (response > 3)){
        system("clear");
        printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
        printf("\t\t Choose The Type of Error Model \n");
        printf("\t\t    Default Model is Poisson \n\n\n");

        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_BER_MODEL,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Get Model Type! \n");
            return;
        }
```

```
        if(err_option == 0)
            printf("\t Current Model Type = Poisson \n\n");
        else
            printf("\t Current Model Type = Markov \n\n");

        printf("\t 1) Exit to Main Menu \n");
        printf("\t 2) Set Model Type to Poisson \n");
        printf("\t 3) Set Model Type to Markov \n");
        printf("\n\n\t Enter Selection (1 - 3) --> ");

        scanf("%d", &response);
    }

    if(response == 1)
        return;
    else if(response == 2){
        err_option = 0;
        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_BER_MODEL,
                (char *) &err_option, sizeof(err_option)) < 0){
            sys_msg("\n\t Unable to Change Model Type!");
            return;
        }

        err_option = 1;
        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_ERRPROB0,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Read New Model Type!\n");
            return;
        }
        if(err_option == 1)
            sys_msg("\n\t Set Option Error - Model Type Not Set to Poisson");
        else
            sys_msg("\n\t New Model Type = Poisson");
    }
    else{
        err_option = 1;
        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_BER_MODEL,
                (char *) &err_option, sizeof(err_option)) < 0){
            sys_msg("\n\t Unable to Change Model Type!");
            return;
        }

        err_option = 0;
        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_ERRPROB0,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Read New Model Type!\n");
            return;
        }
        if(err_option == 0)
            sys_msg("\n\t Set Option Error - Model Type Not Set to Markov");
        else
            sys_msg("\n\t New Model Type = Markov");
    }
}


/******************************************************************************
 *
 *  The function error_set_timer_gran allows the user to choose the timer
 *  granularity used by the error model in microseconds.  The default is
 *  100,000 microseconds or 100 milliseconds.  This parameter is used only by
 *  the Markov model.  The option is called TCP_SNOOP_TIMERGRAN within the
 *  kernel, and it sets the global variable ber.time_granularity defined in
 *  ber.c and ber.h.
 *      Input:    User Entered Timer Granularity
 *      Output:   Updated Kernel Value for Timer Granularity
 *
 ******************************************************************************/
void error_set_timer_gran(void)
```

```
{
    int response=0;
    int err_option=0;
    int option_len;

    while((response < 1) || (response > 2)){
        system("clear");
        printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
        printf("\t      Set Markov Model's Timer Granularity \n");
        printf("\t      Default Gran. is 100000 microseconds \n\n\n");

        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_TIMERGRAN,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Get Mean Timer Granularity! \n");
            return;
        }
        printf("\t Current Timer Granularity = %d microseconds \n\n",
                err_option);

        printf("\t 1) Exit to Main Menu\n");
        printf("\t 2) Change Timer Granularity\n");
        printf("\n\n\t Enter Selection (1 - 2) --> ");

        scanf("%d", &response);
    }

    if(response == 1)
        return;
    else{
        printf("\n\n\t Enter New Timer Granularity --> ");
        scanf("%d", &err_option);

        if(err_option < 1){
            sys_msg("\n\t Timer Gran. Must Be Greater Than or Equal to 1");
            return;
        }

        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_TIMERGRAN,
                (char *) &err_option, sizeof(err_option)) < 0){
            sys_msg("\n\t Unable to Change Timer Granularity!");
            return;
        }

        err_option = 0;
        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_TIMERGRAN,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Read New Timer Granularity!\n");
            return;
        }

        if(err_option == 0)
            sys_msg("\n\t Set Option Error - Timer Gran. Was Not Changed");
        else{
            printf("\t New Timer Granularity = %d microseconds \n\n",
                    err_option);
            sys_msg("");
        }
    }
}


/****************************************************************************
 *
 *  The function error_set_TRANS0 allows the user to set the Markov model's
 *  TRANS0 parameter.  The default is 30.  The option is called
 *  TCP_SNOOP_TRANS0 within the kernel, and it sets the global variable
 *  ber.trans_perc[0] defined in ber.c and ber.h.
 *      Input:    User Entered TRANS0
 *      Output:   Updated Kernel Value for TRANS0
```

```
 *
 ******************************************************************************/
void error_set_TRANS0(void)
{
    int response=0;
    int err_option=0;
    int option_len;

    while((response < 1) || (response > 2)){
        system("clear");
        printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
        printf("\t\t Set Markov Model's TRANS0 Value \n");
        printf("\t\t      Default Value is 30 \n\n\n");

        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_TRANS0,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to TRANS0! \n");
            return;
        }
        printf("\t Current TRANS0 Value = %d \n\n", err_option);

        printf("\t 1) Exit to Main Menu\n");
        printf("\t 2) Change TRANS0\n");
        printf("\n\n\t Enter Selection (1 - 2) --> ");

        scanf("%d", &response);
    }

    if(response == 1)
        return;
    else{
        printf("\n\n\t Enter New TRANS0 Value --> ");
        scanf("%d", &err_option);

        if(err_option < 1){
            sys_msg("\n\t TRANS0 Must Be Greater Than or Equal to 1");
            return;
        }

        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_TRANS0,
                (char *) &err_option, sizeof(err_option)) < 0){
            sys_msg("\n\t Unable to Change TRANS0 Value!");
            return;
        }

        err_option = 0;
        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_TRANS0,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Read New TRANS0 Value!\n");
            return;
        }

        if(err_option == 0)
            sys_msg("\n\t Set Option Error - TRANS0 Value Was Not Changed");
        else{
            printf("\t New TRANS0 Value = %d \n\n", err_option);
            sys_msg("");
        }
    }
}


/******************************************************************************
 *
 *  The function error_set_TRANS1 allows the user to set the Markov model's
 *  TRANS1 parameter.  The default is 30.  The option is called
 *  TCP_SNOOP_TRANS1 within the kernel, and it sets the global variable
 *  ber.trans_perc[1] defined in ber.c and ber.h.
 *      Input:    User Entered TRANS1
```

```
 *      Output:   Updated Kernel Value for TRANS1
 *
 ****************************************************************************/
void error_set_TRANS1(void)
{
    int response=0;
    int err_option=0;
    int option_len;

    while((response < 1) || (response > 2)){
        system("clear");
        printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
        printf("\t\t Set Markov Model's TRANS1 Value \n");
        printf("\t\t      Default Value is 70 \n\n\n");

        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_TRANS1,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to TRANS1! \n");
            return;
        }
        printf("\t Current TRANS1 Value = %d \n\n", err_option);

        printf("\t 1) Exit to Main Menu\n");
        printf("\t 2) Change TRANS1\n");
        printf("\n\n\t Enter Selection (1 - 2) --> ");

        scanf("%d", &response);
    }

    if(response == 1)
        return;
    else{
        printf("\n\n\t Enter New TRANS1 Value --> ");
        scanf("%d", &err_option);

        if(err_option < 1){
            sys_msg("\n\t TRANS1 Must Be Greater Than or Equal to 1");
            return;
        }

        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_TRANS1,
                (char *) &err_option, sizeof(err_option)) < 0){
            sys_msg("\n\t Unable to Change TRANS1 Value!");
            return;
        }

        err_option = 0;
        option_len = sizeof(err_option);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_TRANS1,(char *)
                &err_option, &option_len) < 0){
            sys_msg("\n\t Unable to Read New TRANS1 Value!\n");
            return;
        }

        if(err_option == 0)
            sys_msg("\n\t Set Option Error - TRANS1 Value Was Not Changed");
        else{
            printf("\t New TRANS1 Value = %d \n\n", err_option);
            sys_msg("");
        }
    }
}


/****************************************************************************
 *
 *  The function main_menu displays the main menu for the wireless
 *  configuration program
 *      Input:    User Selections
 *      Output:   Menu Number Related To The User's Selection
```

```
 ****************************************************************************/
int main_menu(void)
{

    int response=0;

#ifdef BASE_STATION
    while((response < 1) || (response > 3)){
#else
    while((response < 1) || (response > 2)){
#endif
        system("clear");
        printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
        printf("\t\t\t   Main Menu \n\n\n");
        printf("\t 1) Exit the Program\n");
        printf("\t 2) Modify Error Model\n");
#ifdef BASE_STATION
        printf("\t 3) Set/Check TCP Enhancement Options\n");
        printf("\n\n\t Enter Selection (1 - 3) --> ");
#else
        printf("\n\n\t Enter Selection (1 - 2) --> ");
#endif

        scanf("%d", &response);
    }

    return response;
}


/****************************************************************************
 *
 *  The function snoop_enable_disable looks at the parameter choice and if it
 *  is 1, the snoop code is enabled for packets traversing the wireless link.
 *  If choice is 0, the snoop code is disabled for packets traversing the
 *  wireless link.  The option is called TCP_SNOOP_DISABLE within the
 *  kernel and it sets the global variable snoopstate->disable defined in
 *  snoop.c and snoop.h
 *      Input:    User's Choice (Enabled/Disabled)
 *      Output:   Updated Kernel Value for Enabling/Disabling Snoop
 *
 ****************************************************************************/
int snoop_enable_disable(int choice)
{
    int snoop_disable;
    int option_len;

    if(choice == 1){
        snoop_disable = 0;
        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_DISABLE,
                (char *) &snoop_disable, sizeof(snoop_disable)) < 0){
            sys_msg("\n\t Unable to Enable Snoop Option at Basestation!");
            return;
        }
        snoop_disable = 1;
        option_len = sizeof(snoop_disable);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_DISABLE,
                (char *) &snoop_disable, &option_len) < 0){
            sys_msg("\n\t Unable to Get Snoop Enabled/Disabled Status!");
            return;
        }

        if(snoop_disable == 1)
            sys_msg("\n\t Error:  Snoop Was Not Turned On Within The Kernel");
        else
            sys_msg("\n\t Snoop Option Has Been Enabled at Basestation!");
    }
    else{
        snoop_disable = 1;
        if(setsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_DISABLE,
                (char *) &snoop_disable, sizeof(snoop_disable)) < 0){
```

```
            sys_msg("\n\t Unable to Disable Snoop Option at Basestation!");
            return;
        }
        snoop_disable = 0;
        option_len = sizeof(snoop_disable);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP, TCP_SNOOP_DISABLE,
                (char *) &snoop_disable, &option_len) < 0){
            sys_msg("\n\t Unable to Get Snoop Enabled/Disabled Status!");
            return;
        }

        if(snoop_disable == 0)
            sys_msg("\n\t Error:  Snoop Was Not Turned Off Within The Kernel");
        else
            sys_msg("\n\t Snoop Option Has Been Disabled at Basestation!");
    }
}


/*****************************************************************************
 *
 *  The function sys_err takes a string and displays it for the user.
 *  Then it terminates the program with the error.  It is intended to be used
 *  with fatal errors which must terminate the program, not for program
 *  warnings.
 *      Inputs:    Error Message
 *      Outputs:   Message to The Screen
 *
 *****************************************************************************/
void sys_err(char *str)
{
    printf("%s\n",str);
    exit(1);
}


/*****************************************************************************
 *
 *  The function sys_msg takes a string and displays it for the user.  It then
 *  pauses until the user presses the return key.  Unlike sys_err, it is not
 *  intended to be used with fatal errors.  It displays non-fatal errors and
 *  other general information.
 *      Inputs:    Error Message
 *      Outputs:   Message to The Screen
 *
 *****************************************************************************/
void sys_msg(char *str)
{
    int response;

    printf("%s\n",str);
    printf("\t Press the return key to continue\n");
    response = getchar();
    response = getchar();
}


/*****************************************************************************
 *
 *  The function tcp_model_menu allows the user to choose what enhancements if
 *  any to add to the tcp/ip protocol suite.
 *      Input:    User's Choice
 *      Output:   Menu Item Number Selected By The User
 *
 *****************************************************************************/
int tcp_model_menu(void)
{

    int response=0;
    int snoop_disable;
    int ackp_enable;
```

```
    int option_len;

    while((response < 1) || (response > 5)){
        system("clear");
        printf("\n\n\n\t\t WIRELESS CONFIGURATION PROGRAM \n");
        printf("\t\t      Enhancement Options \n\n\n");

        snoop_disable=0;
        option_len = sizeof(snoop_disable);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_DISABLE,(char *)
                &snoop_disable, &option_len) < 0)
            sys_msg("\n\t Unable to Get Snoop Enabled/Disabled Status!\n");

        if(snoop_disable == 0)
            printf("\t Snoop Is Currently Enabled at The Basestation\n\n");
        else
            printf("\t Snoop Is Currently Disabled at The Basestation\n\n");

        ackp_enable=0;
        option_len = sizeof(ackp_enable);
        if(getsockopt(Dummy_Fd, IPPROTO_TCP,TCP_SNOOP_ACKP_ENABLE,(char *)
                &ackp_enable, &option_len) < 0)
            sys_msg("\n\t Unable to Get Ackp Enabled/Disabled Status!\n");

        if(ackp_enable == 0)
            printf("\t Ackp Is Currently Disabled at The Basestation\n\n");
        else
            printf("\t Ackp Is Currently Enabled at The Basestation\n\n");

        printf("\t Note:  Snoop must be enabled for Ackp to work\n\n");

        printf("\t 1) Exit to Main Menu\n");
        printf("\t 2) Enable Snoop Option at Basestation\n");
        printf("\t 3) Disable Snoop Option at Basestation\n");
        printf("\t 4) Enable Ackp Option at Basestation\n");
        printf("\t 5) Disable Ackp Option at Basestation\n");
        printf("\n\n\t Enter Selection (1 - 5) --> ");

        scanf("%d", &response);
    }

    return response;
}
```