Performance Modeling of TCP and UDP over Packet Radio Networks using the ns-2 Network Simulator

Paul D. Wiedemeier, Ph.D., KE5LKY
Clarke M. Williams, Jr. Endowed Professor of Computer Science
The University of Louisiana at Monroe
Computer Science and Computer Information Systems Department
College of Business Administration
700 University Avenue, Monroe, Louisiana 71209
318-342-1856 (Work) or 318-396-1857 (Fax)
wiedemeier@ulm.edu or KE5LKY@arrl.net

Abstract

Live network tests are often used to obtain performance metrics associated with the transmission of data using TCP and UDP over packet radio networks. However, only individuals who hold an amateur radio license issued by the United States Federal Communications Commission, as specified by Title 47, Code of Federal Regulations, Part 97, are permitted to operate an amateur station. For this reason, we advocate using the ns-2 network simulator in academic environments to allow unlicensed students to evaluate the performance of packet radio networks. In this paper, we discuss four topics concerning packet radio network performance evaluation. First, the models used to evaluate the performance of packet radio networks are presented. We next introduce the ns-2 network simulator and discuss the advantages and disadvantages associated with using this tool to evaluate the performance of TCP and UDP data transmissions. Third, we discuss the structure of two ns-2 Tcl scripts that can be used to simulate the transmission of data over packet radio networks. Lastly, we examine the output generated by an example ns-2 Tcl script.

Key Words

Performance Modeling, TCP, UDP, FTP, CBR, Packet Radio, ns-2, Network Simulator

Introduction

Evaluating the performance of the Transmission Control Protocol (TCP) (Postel, 1981) and the User Datagram Protocol (UDP) (Postel, 1980) can be accomplished through use of live network tests, test-beds, hardware emulation, software simulation, or mathematical models (Allman & Falk, 1999). Unfortunately, student researchers who have yet to obtain their Federal Communications Commission (FCC) amateur radio licenses cannot transmit live network data using packet radio. The ns-2 network (software) simulator allows such students to simulate the transmission of data over packet radio networks using TCP and UDP in preparation for live network tests. Throughout the remainder of this paper, the phrases ns-2 network simulator, ns-2 simulator, ns-2, and Ns-2 are synonymous.

We focus our discussion of packet radio network performance modeling on five main areas. In *Network Performance Modeling*, we present advantages and disadvantages associated with using five different models to evaluate the performance of data communication protocols. *The ns-2 Network Simulator* section of this paper presents and enumerates details about the ns-2 network simulator. We highlight the historical development of the ns-2 network simulator, the software that must be downloaded to build the

ns-2 simulator, where the software can be obtained, which computer platforms are supported, and what useful documentation exists. In *Writing Basic ns-2 Tcl Scripts* we present and discuss the basic Tcl instructions required to create scripts to simulate the transmission of data using TCP Reno and UDP over a 2-meter packet radio network. We illustrate packet radio network transmissions of files, using the File Transmission Protocol (FTP) (Postel, 1985) and TCP Reno, and data, using Constant Bit Rate (CBR) and UDP. In *Interpreting ns-2 Generated Trace Files* we discuss the results generated by the ns-2 network simulator, which are found within ns-2 trace files. In *Conclusions* we highlight how the ns-2 network simulator is used by those interested in packet radio performance modeling, specifically at the transport layer. We also discuss how the ns-2 simulation tool allows educators and students to evaluate the performance of 2-meter packet radio networks.

Network Performance Modeling

As mentioned in *Introduction*, data communication protocol performance is often evaluated using one or more of the following methods: live tests, test-beds, hardware emulation, software simulation, or mathematical models (Allman & Falk, 1999). Each method is discussed below.

Live Network Tests

Transmitting data over an actual communication network is the most comprehensive method for testing the performance of data communication protocols because real data is generated. However, live tests also exhibit two disadvantages. The inability to assess the impact a new protocol has on other network traffic sharing the same communication channel is one disadvantage. It is also often difficult to obtain access to private network environments for testing purposes.

Test-Beds

Small isolated test-bed networks are often created and used to test data communication protocols because large networks contain numerous uncontrollable variables that can adversely affect protocol tests. Evaluating the performance of data communication protocols using a test-bed has two advantages. First, test-beds use actual hardware, but without the complexity associated with larger communication networks. Second, since they lack complexity, the impact associated with using a new protocol is easier to access. Similar to live network tests, two disadvantages can be attributed to using a test-bed. First, they are limited, in their complexity and speed, by the equipment on hand. Second, if commercial operating systems are used, a researcher may be unable to modify proprietary protocol code.

Hardware Emulation

A network emulator is a hardware based protocol evaluation tool. Here, a test-bed is constructed and a computer emulates the function of a specific piece of the communication network. There exist two advantages to using a hardware emulator for performance evaluation. First, hardware emulators test real protocol implementations. Second, the non-emulated routers, hubs, and switch protocols are easy to modify. Although hardware emulators are beneficial, they exhibit three disadvantages. They simplify some of the modeled network's real behavior. They may not be able to represent complex or changing topologies. Lastly, a considerable amount of equipment may be required.

Software Simulation

A simulator is a software based network evaluation tool that permits researchers to test data communication protocols when an actual network is unavailable. Using a software simulator for performance evaluations is advantageous for six reasons. Each is discussed below (Allman & Falk, 1999).

1. Extensive computer or networking equipment is not required when using a software simulator. Software simulators are often installed on personal computers (PCs) or workstations running the Microsoft Windows operating system or a UNIX/Linux operating system.

2. Researchers are able to execute numerous simulations in a short time period using software simulators. The tremendous amounts of data generated can be used to evaluate quickly the performance of data communication protocols.

3. Software simulators allow researchers to simulate communication networks they cannot access. Likewise, software simulators can simulate networks that do not yet exist due to current limitations (i.e. physical or monetary).

4. Complex communication networks, which cannot be created within a test-bed, are easily constructed and evaluated using software simulators.

5. All data generated during a simulation are maintained in files created by a software simulator. This feature permits researchers to identify easily which simulation variables produce significant impact.

6. Actual communication networks are not affected by software simulation.

Software simulators exhibit three disadvantages. First, rather than using actual data communication protocol code found in real operating systems, some software simulators use abstract implementations. Second, software simulators often do not represent non-network events, such as operating system scheduler latency. Third, software simulators often make assumptions concerning real world events, such as competing traffic.

Mathematical Models

File transmission time, channel throughput, and channel utilization are easily computed mathematically. File transmission times specify the time required by a communication protocol to move all bits in a file from source to destination. Mathematically, the time required to transmit a file (i.e. file transmission time) across a communication channel is the sum of the communication channel's transmission delay and the quotient of file size and the communication channel's transmission rate (Kruse, 1995). See Figure 1. Overall, a smaller transmission time results in a faster transmission of the file.

Channel throughput defines the amount of data transmitted in a given period of time (e.g. bits per second), or the channel's effective transmission rate. Mathematically, channel throughput is the quotient of file size and file transmission time (Allman & Glover & Sanchez, 1999). See Figure 1. Generally, the larger the channel throughput, the closer the data transmission protocol is to achieving the channel's transmission rate while transmitting the file. The best throughput is the channel's defined transmission rate. Channel throughput close to 0.0 is considered poor.

Channel utilization also defines the performance of data transmission protocols. Channel utilization is throughput normalized to unity using the channel's defined transmission rate. Mathematically, channel utilization is the quotient of channel's throughput and channel's transmission rate (Henderson & Katz, 1999). See Figure 1. The best channel utilization is 1.0, which means the channel's throughput matches the channel's defined transmission rate. Channel utilization close to 0.0, again, is poor.

Hosts that transmit data using connection-oriented transport layer protocols, such as TCP, retransmit packets when an acknowledgement has not arrived within a specified time. The primary disadvantage associated with using mathematical models is that they do not account for channel congestion and dropped packets. However, computed file transmission times, channel throughputs, and channel utilizations are fairly close to actual data transmissions times generated by connectionless transport layer protocols, like UDP.

$$\text{File Transmission Time} = \frac{\text{File Size}}{\text{Channel Transmission Rate}} + \text{Channel Transmission Delay}$$

$$\text{Channel Throughput} = \frac{\text{File Size}}{\text{File Transmission Time}}$$

$$\text{Channel Utilization} = \frac{\text{Channel Throughput}}{\text{Channel Transmission Rate}}$$

Figure 1: Mathematical Equations used to Model the Performance of Communication Channels.

The ns-2 Network Simulator

Many network simulators exist and are used in business, government, and academia. OpNet, M5, J-sim, and ns-2 represent only a small number of available network simulators (Allman & Falk, 1999). A larger list of network simulators can be found on the World Wide Web (Floyd, 2007). We employ the discrete event network simulator ns-2, version ns-2.31[1] (Breslau et. al., 2000), because it was developed to provide network researchers with a software tool to support simulation of TCP, routing, and multicast protocols over wired and wireless (i.e. local and satellite) networks.

Ns-2 History

The ns-2 simulator project began in 1989 as a variant of the REAL network simulator. During the 1990's, ns-2 was developed through support provided by the Defense Advanced Research Projects Agency, the Lawrence Berkeley National Laboratory's Virtual InterNetwork Testbed project, Xerox PARC, the University of California at Berkeley, and the University of Southern California's Information Sciences Institute. Today, ns-2 development is supported by DARPA, the University of Southern California's Information Sciences Institute, and the University of California at Berkeley's International Computer Science Institute's Center for Internet Research.

[1] Ns-2.31 was released March 10, 2007.

Ns-2 Software

Functionally, ns-2 uses two programming languages, C++ and Otcl, to support simulation of data transmissions over communication channels. Thus, ns-2 is considered a "split-language" program. The C++ programming language implements the simulator's Internet protocols. Specifically, C++ is used for processing at the packet level because compilation generates a fast, detailed, and complete binary. The OTcl programming language implements the simulator's interface. OTcl is used to setup and control the simulation because it an interpreted language. As such, instructions are easy to write and change.

The advantages associated with using two languages to construct ns-2 are found in the speed of execution at the low level using C++ and the flexibility of OTcl when writing script to setup and control the simulation. Unfortunately, to modify existing ns-2 supplied protocols or create new protocols, a researcher must be able to write in both languages.

Ns-2 Advantages

We use the ns-2 network simulator to obtain the times (i.e. transmission times) associated with transmitting files across a theoretical 2-meter packet radio network. See Figure 3. While the network simulators mentioned previously in this paper are all capable of simulating data transmissions and modeling the performance of data communication protocols, we selected the ns-2 simulator for the following five reasons.

1. The ns-2 simulator can easily define complex network topologies for simulation purposes. This includes topologies consisting of packet radio and terrestrial channels.

2. Many researchers use the ns-2 simulator to evaluate the performance of both TCP and UDP (Wiedemeier & Tyrer, 2007, Ishac & Allman, 2001, Henderson & Katz, 2000, Breslau et. al., 2000, Mathis & Mahdavi, 2000, Fall & Floyd, 1996).

3. The ns-2 simulator can easily be configured to simulate the transmission of files using standard and user defined transport layer data transmission protocols.

4. The ns-2 simulator is freeware. The simulator code is readily available for downloading and is quickly installed. Simulation can then begin almost immediately.

5. The ns-2 simulator allows small research groups, such as ours, to evaluate the performance of communication protocol without much effort and without incurring significant expense. As stated in *Introduction*, students can evaluate the performance of transport layer protocols over packet radio networks without ever touching a transceiver.

Ns-2 Software Distribution, Hardware Requirements, and Associated Documentation

The official source for the ns-2 network simulator distribution and associated documentation is the ns-2 web site (Ns-2, 2007). The ns-2 distribution can also be obtained from the SourceForge web site (SourceForge, 2007). The full ns-2 distribution is actually comprised of several software packages and Table 1 lists the required and optional packages included in the full distribution. The Microsoft Windows and UNIX/Linux ns-2 distributions can be obtained via a single package and installed en masse. Alternatively, the packages listed in Table 1 can be downloaded individually and installed separately.

Two main tools are required to build and install the ns-2 network simulator: a computer and a C++ compiler. Ns-2 was developed to run on top of a wide range of hardware, including PCs and workstations, and a variety of operating systems, including FreeBSD, Linux, Microsoft Windows, and Sun Microsystems SunOS/Solaris. To install ns-2 on Microsoft Windows the Cygwin or MinGW C++ compilers are required. We currently have ns-2 installed on a Dell Precision M90 laptop running Fedora Linux, version 2.6.20-1.2963.fc6.

Table 1: Required and Optional ns-2 Software Packages.

Tcl/Tk (Required)
Otcl (Required)
TclCL (Required)
ns-2 (Required)
nam-1 (Optional)
xgraph (Optional)
perl (Optional)
tcl-debug (Optional)
dmalloc (Optional)
sgb2ns conversion program (Optional)
tiers2ns conversion program (Optional)
Cweb and sgb source code (Optional)

Documentation discussing the operation of the ns-2 network simulator is also available from the ns-2 web site (Ns-2, 2007). Of particular interest to the new ns-2 user would be "The ns Manual" (Fall & Varadhan, 2007), "Tutorial for the Network Simulator ns" (Greis, 1998), and "NS Simulator for Beginners" (Altman & Jimenez, 2003).

Writing Basic ns-2 Tcl Scripts

To simulate the transmission of data communication protocols using ns-2, one must write an ns-2 Tcl script. In the subsections that follow, we discuss the basic ns-2 Tcl script instructions required to simulate connection-oriented and connectionless data transmissions. In the first subsection, *File Transmissions using FTP and TCP Reno*, we present the structure of an ns-2 Tcl script that simulates file transmissions using FTP and TCP Reno over a 2-meter packet radio network. In the *Data Transmissions using CBR and UDP* subsection, we present the structure of an ns-2 Tcl script that simulates CBR data transmissions using UDP over a 2-meter packet radio network. For those readers who are unfamiliar with Tcl script programming, the author suggests (Trantor & Raines, 1999) as a reference.

The 2-meter packet radio network we simulated using ns-2 exhibits a transmission rate of 1200 bits per second (bps) (Jones, Page 11, 1996) and a 25 microsecond (μs) transmission delay. We chose a 25 μs transmission delay for our simulations because, in the future, we intend to conduct live tests between transceivers located at the author's home and work. The physical distance between these two locales is approximately 4.6 miles, which corresponds to a 25 μs transmission delay. Figure 2 shows the physical topology of this network, while Figure 3 shows the logical topology of this network. It is the logical network shown in Figure 3 that we will simulate using ns-2.

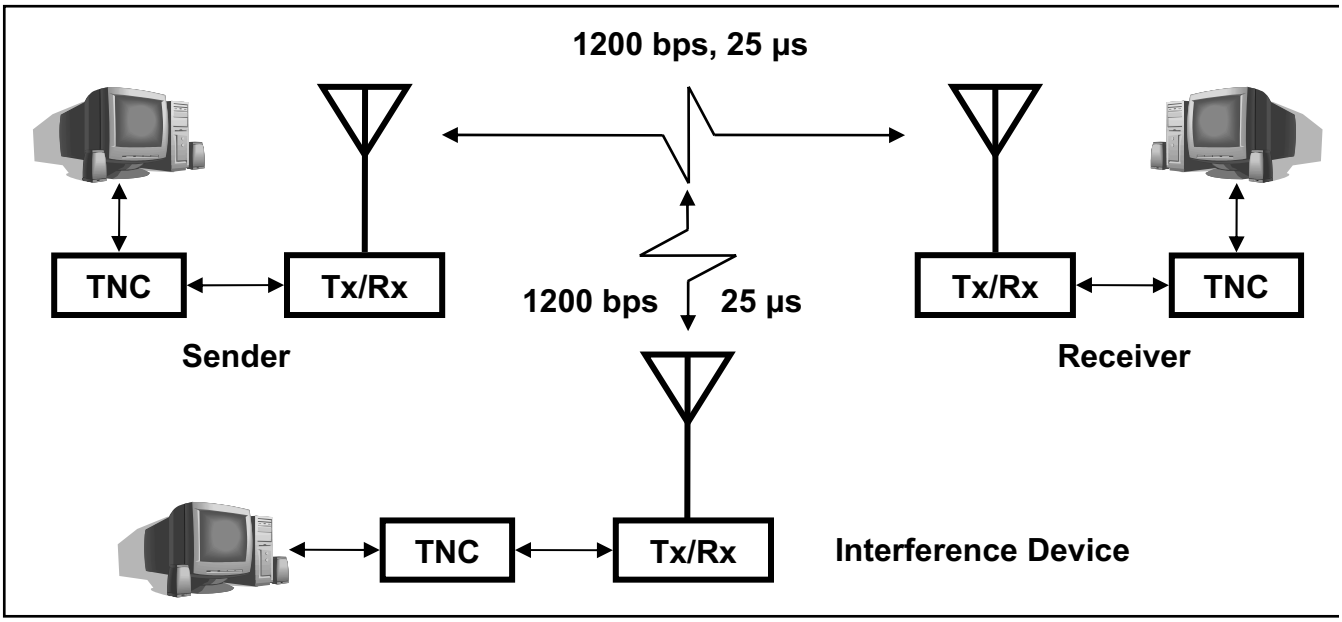


Figure 2: An Example 3 Node, 1 Link Packet Radio Network.

Notice that the physical topology shown in Figure 2 includes three nodes: a sender, a receiver, and an interference device. The logical topology shown in Figure 3 only includes two nodes because the interference device, at least for those simulations involving TCP Reno, will be represented by a bit error module that is attached to the link between the sender and receiver in Figure 3. No error module is required for UDP transmissions because this protocol is connection-less.

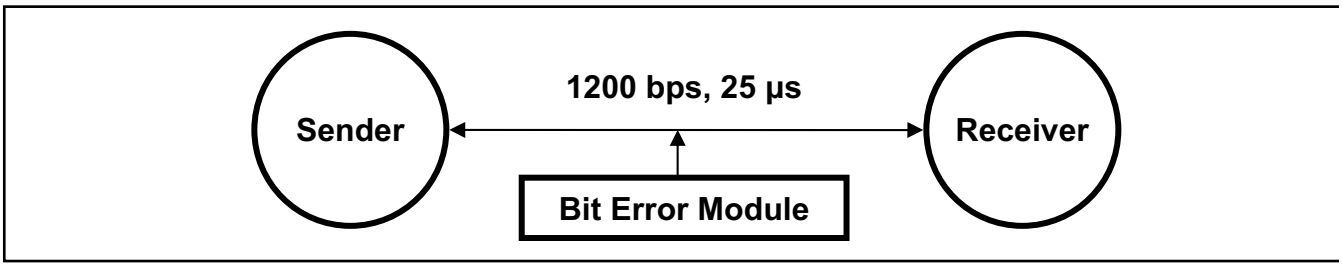
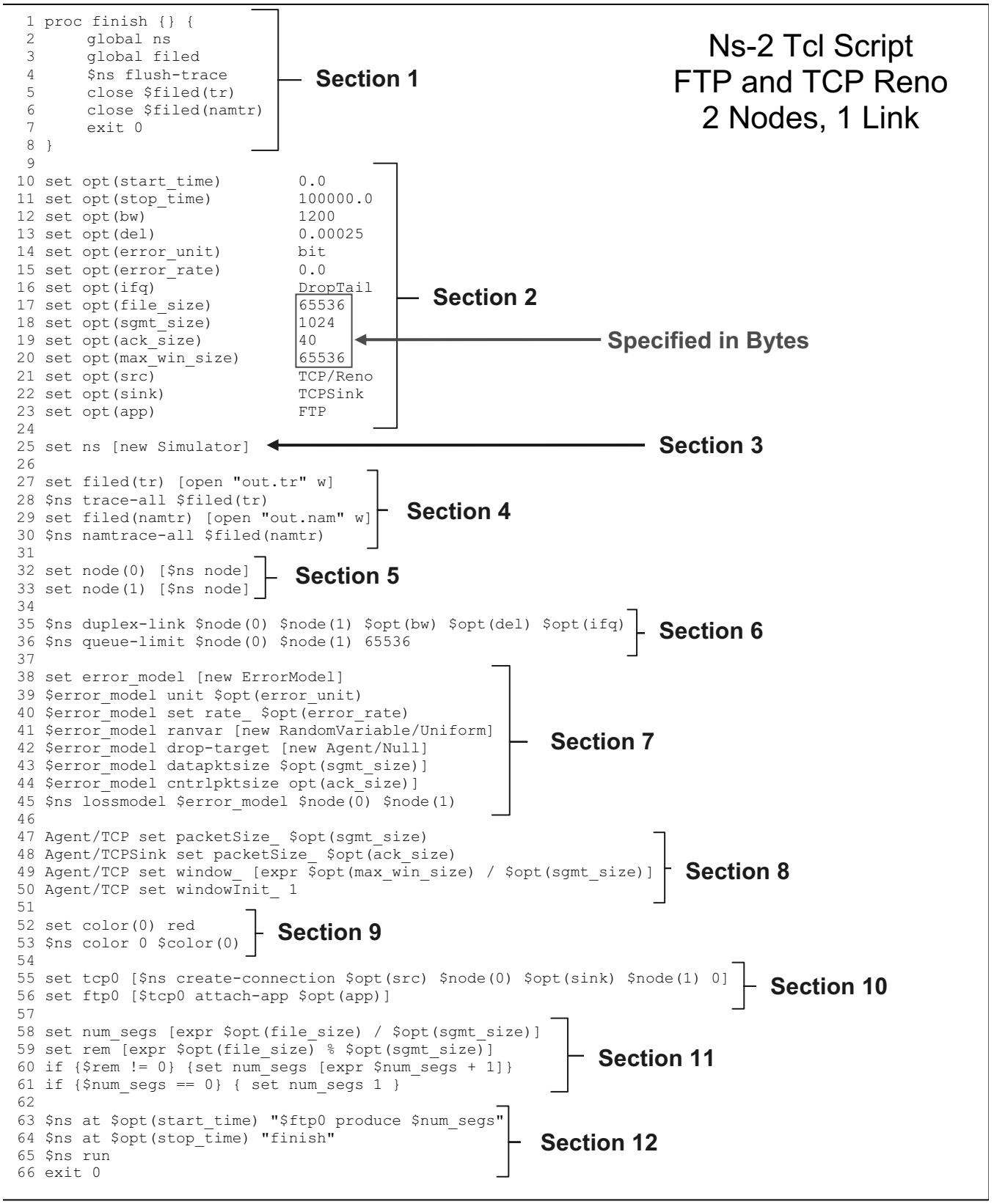


Figure 3: An Example 2 Node, 1 Link ns-2 Network Topology.

File Transmissions using FTP and TCP Reno

Figure 4 shows an ns-2 Tcl script that simulates the transmission of a file using the topology shown in Figure 3. Again, the transmission rate and transmission delay associated with the link are 1200 bps and 25 µs respectively. The numbers at the beginning of each line in Figure 4 are line numbers. They are included here for illustration purposes and are not actual Tcl instructions. The ns-2 Tcl script shown in Figure 4 is divided into 12 sections, each labeled 1-12. All are discussed is the paragraphs that follow.

The finish procedure, specified by lines 1-8 in Section 1, is called by ns-2 upon termination of the simulation. The ns-2 simulator terminates on line 64 in Section 12. The finish procedure flushes data




```
 1 proc finish {} {
 2      global ns
 3      global filed
 4      $ns flush-trace
 5      close $filed(tr)
 6      close $filed(namtr)
 7      exit 0
 8 }
 9
10 set opt(start_time)        0.0
11 set opt(stop_time)         100000.0
12 set opt(bw)                1200
13 set opt(del)               0.00025
14 set opt(error_unit)        bit
15 set opt(error_rate)        0.0
16 set opt(ifq)               DropTail
17 set opt(file_size)         65536
18 set opt(sgmt_size)         1024
19 set opt(ack_size)          40
20 set opt(max_win_size)      65536
21 set opt(src)               TCP/Reno
22 set opt(sink)              TCPSink
23 set opt(app)               FTP
24
25 set ns [new Simulator]
26
27 set filed(tr) [open "out.tr" w]
28 $ns trace-all $filed(tr)
29 set filed(namtr) [open "out.nam" w]
30 $ns namtrace-all $filed(namtr)
31
32 set node(0) [$ns node]
33 set node(1) [$ns node]
34
35 $ns duplex-link $node(0) $node(1) $opt(bw) $opt(del) $opt(ifq)
36 $ns queue-limit $node(0) $node(1) 65536
37
38 set error_model [new ErrorModel]
39 $error_model unit $opt(error_unit)
40 $error_model set rate_ $opt(error_rate)
41 $error_model ranvar [new RandomVariable/Uniform]
42 $error_model drop-target [new Agent/Null]
43 $error_model datapktsize $opt(sgmt_size)]
44 $error_model cntrlpktsize opt(ack_size)]
45 $ns lossmodel $error_model $node(0) $node(1)
46
47 Agent/TCP set packetSize_ $opt(sgmt_size)
48 Agent/TCPSink set packetSize_ $opt(ack_size)
49 Agent/TCP set window_ [expr $opt(max_win_size) / $opt(sgmt_size)]
50 Agent/TCP set windowInit_ 1
51
52 set color(0) red
53 $ns color 0 $color(0)
54
55 set tcp0 [$ns create-connection $opt(src) $node(0) $opt(sink) $node(1) 0]
56 set ftp0 [$tcp0 attach-app $opt(app)]
57
58 set num_segs [expr $opt(file_size) / $opt(sgmt_size)]
59 set rem [expr $opt(file_size) % $opt(sgmt_size)]
60 if {$rem != 0} {set num_segs [expr $num_segs + 1]}
61 if {$num_segs == 0} { set num_segs 1 }
62
63 $ns at $opt(start_time) "$ftp0 produce $num_segs"
64 $ns at $opt(stop_time) "finish"
65 $ns run
66 exit 0
```

Figure 4: An Example ns-2 Tcl Script for File Transmissions using FTP and TCP Reno.

not yet written to the trace files, as specified by filed(tr) and filed(namtr), closes both trace files, and then exits. On lines 10-23 in Section 2 an associative array named opt() is defined. The values are placed into opt() will be used later in the Tcl script. Note that the values for opt(file_size), opt(sgmt_size), opt(ack_size), and opt(max_win_size) are specified in Bytes. See lines 17-20 in Section 2.

Line 21 in Section 2 defines opt(src) as the TCP agent to use during simulation. The value we use for simulation purposes is TCP/Reno (Fall & Floyd, 1996) because it is the minimum benchmark for TCP (Glover & Kruse, 1998) and it is the de-facto implementation of TCP (Mathis & Mahdavi, 1996). The ns-2 simulator permits the simulation of several other TCP agents, including TCP (i.e. regular TCP), TCP/Newreno, TCP/Vegas, TCP/Sack1, and TCP/Fack. On line 25 in Section 3 an instance of the ns-2 simulator is defined. Since C++ and OTcl are object-oriented languages, this means that an object of the ns-2 Simulator class has been created.

Lines 27-30 in Section 4 open the trace file and nam trace file to accept trace data generated by the ns-2 simulator. The ns-2 trace facility is also initialized. On lines 32-33 in Section 5 the sender and receiver nodes are defined. Here, node(0) is the sender and node(1) is the receiver. The actual link characteristics between the sender and the receiver are defined on lines 35-36 in Section 6. The bandwidth, opt(bw), the transmission delay, opt(del), and the interface queue type, opt(ifq) are defined and the queue length is set to 64 Kilobytes.

Lines 38-45 in Section 7 create an error model for data that flows between the sender and receiver. The error model will act as the interference device shown in Figure 2. The bit error rate is held by the variable opt(error_rate) and is defined on line 15 in Section 2. An error rate of 0.0 implies that no data is corrupt during transmission. An error rate of 0.0001 (i.e. 1.0e-04) specifies that 1 bit in 10,000 will experience corruption with a uniform distribution.

Lines 47-50 in Section 8 define the TCP agent's packetSize_, window_ (i.e. window size), and windowInit_ (i.e. initial window size) variables. The variables opt(sgmt_size), opt(ack_size), and opt(max_win_size) are required to defined these values. The default values for packetSize_, window_, and windowInit_ are 1000, 20, and 1 respectively (Fall & Varadhan, Chapter 34.1.4, Page 290, 2007). These values are also defined in the file ns-2/tcl/lib/ns-default.tcl from the UNIX/Linux ns-2 distribution.

Definition of the color of data transmissions, as shown by the nam visualization program, occurs on lines 52-53 in Section 9. Many colors can be defined, including red (as shown), blue, purple, orange, or black. In general, "the color name should be one of the names listed in color database in X11 (/usr/X11/lib/rgb.txt)" (Fall & Varadhan, Chapter 46.1.1, Page 397, 2007). These two lines can be removed if you do not intend to use the nam visualization program. If so, also omit lines 29-30 in Section 4 and line 6 in Section 1.

The TCP agent is defined on line 55 in Section 10. Specifically, a TCP/Reno agent is attached to node(0), the sender, and a TCPSink is attached to node(1), the receiver. On line 56 in Section 10 an FTP application is defined and is attached to the TCP agent defined on line 55 in Section 10.

By default, an ns-2 FTP application continuously generates and transmits a new packet each time it receives an acknowledgement for any packet in its congestion window. Packet transmission continues until the stop time occurs, as defined by opt(stop_time) on line 64 in Section 12. On lines 58-61 in

Section 11 the maximum number of packets the FTP application must transmit is defined and is based on the file size and the segment size. Here, segment size and packet size are synonymous. At a minimum, one segment will be transmitted by the FTP application. See line 61 in Section 11.

Line 63 in Section 12 defines the time at which the FTP application will start transmitting segments. As stated earlier, the time at which the ns-2 simulation will end is specified on line 64 in Section 12. This value is stored in the variable opt(stop_time). Mathematically, the equation for transmission time can be used to determine the value for opt(stop_time). Line 65 in Section 12 starts the ns-2 simulation and line 66 in Section 12 terminates the Tcl script.

The Tcl script shown in Figure 4 has numerous variables; each can be assigned several variables. An execution of the ns-2 simulator against the Tcl script shown in Figure 4 represents a "run" using those variables. To change the value of a variable the Tcl script file must be opened, the specific change is made, the file is saved and closed, and the ns-2 simulator is rerun against the Tcl script file. Within a UNIX shell, the command to execute an ns-2 Tcl script is $ns TclScript.tcl, where $ is the UNIX shell prompt, ns is the binary, and TclScript.tcl is the name of the file containing your ns-2 Tcl script.

Data Transmissions using CBR and UDP

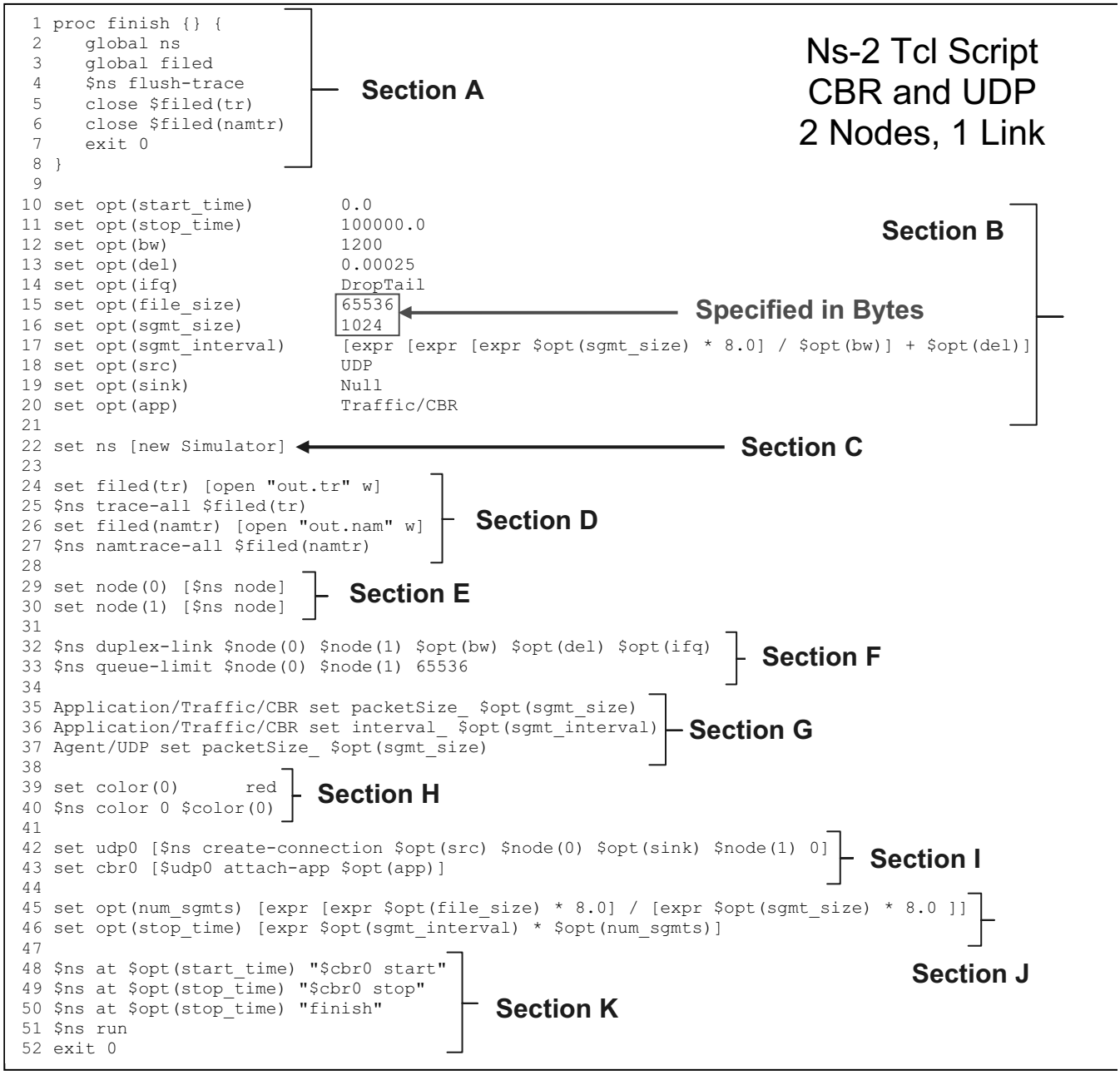
Figure 5 shows an ns-2 Tcl script that transmits data using CBR and UDP. Similar to the ns-2 Tcl script shown in Figure 4, the one shown in Figure 5 transmits data over the theoretical topology shown in Figure 3. The same 1200 bps transmission rate and 25 μs transmission delay apply here. We again use line numbers to illustrate specific Tcl instructions; they do not actually exist in the code itself. The ns-2 Tcl script shown in Figure 5 is divided into 11 sections, each labeled A-K. The sections are discussed in the paragraphs that follow.

Sections A, C, D, E, F, and H in Figure 5 are similar to Sections 1, 3, 4, 5, 6, and 9 in Figure 4, and will not be covered here. Figure 5 Section B differs from Figure 4 Section 2 only in the number and type of opt() values defined; opt(file_size) and opt(sgmt_size) remain and are again defined in Bytes. The values for opt(src), opt(sink), and opt(app) have changed between Figure 4 and Figure 5 and are UDP, Null, and Traffic/CBR respectively. See lines 18-20 in Section B.

In Section G, lines 35-37, packetSize_ and interval_ variables for Application/Traffic/CBR are defined, as well as the packetSize_ for Agent/UDP. Again, opt(sgmt_size) is used to define both values for packetSize_. The interval_ variable defines the interval at which the CBR traffic generated will transmit segments. This value is defined using opt(sgmt_interval) from line 17 in Section B. The default value for Agent/UDP packetSize_ is 1000 (Fall & Varadhan, Chapter 33.1, Page 286, 2007). The default value for Application/Traffic/CBR packetSize is 210 and is defined in the file ns-2/tcl/lib/ns-default.tcl. A default value for Application/Traffic/CBR interval_ is not explicitly defined. However, the Application/Traffic/CBR rate_ is set to 448 KB, which corresponds to an interval_ of 3.75 milliseconds, or 0.00375 seconds. Again, see file ns-2/tcl/lib/ns-default.tcl from the UNIX/Linux ns-2 distribution.

The UDP agent is defined on line 42 in Section I. The source is node(0) and the destination is node(1), which essentially acts as a sink. On line 43 in Section I the CBR application is attached to the UDP agent. On line 45 in Section J the value opt(num_sgmts), which is the number of segment to be transmitted based on the size of the file to be transmitted, is defined. On line 46 in Section J the value opt(stop_time) is redefined to be the product of the segment interval and the number of segments to transmit.

Line 48 in Section K sets the time at which the CBR application will start (i.e. at time 0.0). On line 49 in Section K the time at which the CBR application will stop (i.e. at time opt(stop_time)) is set. Line 50 in Section K defines the time when the finish procedure will be executed (i.e. again at time opt(stop_time)). The ns-2 simulation begins on line 51 in Section K, whereas line 52 in Section K terminates the Tcl script.




```
 1 proc finish {} {
 2    global ns
 3    global filed
 4    $ns flush-trace
 5    close $filed(tr)
 6    close $filed(namtr)
 7    exit 0
 8 }
 9
10 set opt(start_time)        0.0
11 set opt(stop_time)         100000.0
12 set opt(bw)                1200
13 set opt(del)               0.00025
14 set opt(ifq)               DropTail
15 set opt(file_size)         65536
16 set opt(sgmt_size)         1024
17 set opt(sgmt_interval)     [expr [expr [expr $opt(sgmt_size) * 8.0] / $opt(bw)] + $opt(del)]
18 set opt(src)               UDP
19 set opt(sink)              Null
20 set opt(app)               Traffic/CBR
21
22 set ns [new Simulator]
23
24 set filed(tr) [open "out.tr" w]
25 $ns trace-all $filed(tr)
26 set filed(namtr) [open "out.nam" w]
27 $ns namtrace-all $filed(namtr)
28
29 set node(0) [$ns node]
30 set node(1) [$ns node]
31
32 $ns duplex-link $node(0) $node(1) $opt(bw) $opt(del) $opt(ifq)
33 $ns queue-limit $node(0) $node(1) 65536
34
35 Application/Traffic/CBR set packetSize_ $opt(sgmt_size)
36 Application/Traffic/CBR set interval_ $opt(sgmt_interval)
37 Agent/UDP set packetSize_ $opt(sgmt_size)
38
39 set color(0)      red
40 $ns color 0 $color(0)
41
42 set udp0 [$ns create-connection $opt(src) $node(0) $opt(sink) $node(1) 0]
43 set cbr0 [$udp0 attach-app $opt(app)]
44
45 set opt(num_sgmts) [expr [expr $opt(file_size) * 8.0] / [expr $opt(sgmt_size) * 8.0 ]]
46 set opt(stop_time) [expr $opt(sgmt_interval) * $opt(num_sgmts)]
47
48 $ns at $opt(start_time) "$cbr0 start"
49 $ns at $opt(stop_time) "$cbr0 stop"
50 $ns at $opt(stop_time) "finish"
51 $ns run
52 exit 0
```

Figure 5: An Example ns-2 Tcl Script for Data Transmissions using CBR and UDP.

Interpreting ns-2 Generated Trace Files

The ns-2 trace file shown in Figure 6 was generated by the Tcl script shown in Figure 4 (i.e. FTP and TCP Reno). An ns-2 trace file receives data generated by the ns-2 simulator during execution. On lines

27 and 28 in Section 4 of Figure 4 an ns-2 trace file named “out.tr” is opened. The ns-2 trace facility is initialized to write output to this file. On lines 29 and 30 in Section 4 an ns-2 nam trace file is opened and the ns-2 trace facility is initialized to generate ns-2 nam trace output.

As was mentioned earlier in this paper, nam an optional ns-2 software package that permits researchers to view their ns-2 simulation output visually. That is, nam interprets the data in a nam trace file (i.e. out.nam) and displays the output graphically. While the output contained in nam trace files can help researchers debug ns-2 Tcl scripts, our discussion focuses on regular trace files.

Similar to Figures 4 and 5, Figure 6 includes left oriented number to help identify specific trace file lines. All entries (i.e. lines) in an ns-2 trace file are comprised of 12 fields. The first field (i.e. the beginning of a line) is either a character “+”, “-“, or “r”. The “+” specifies a segment enqueue operation and the ”–“ specifics a segment dequeue operation. The enqueue and dequeue operations are performed at the source. The “r” specifies a segment receive operation, which is performed at the destination.

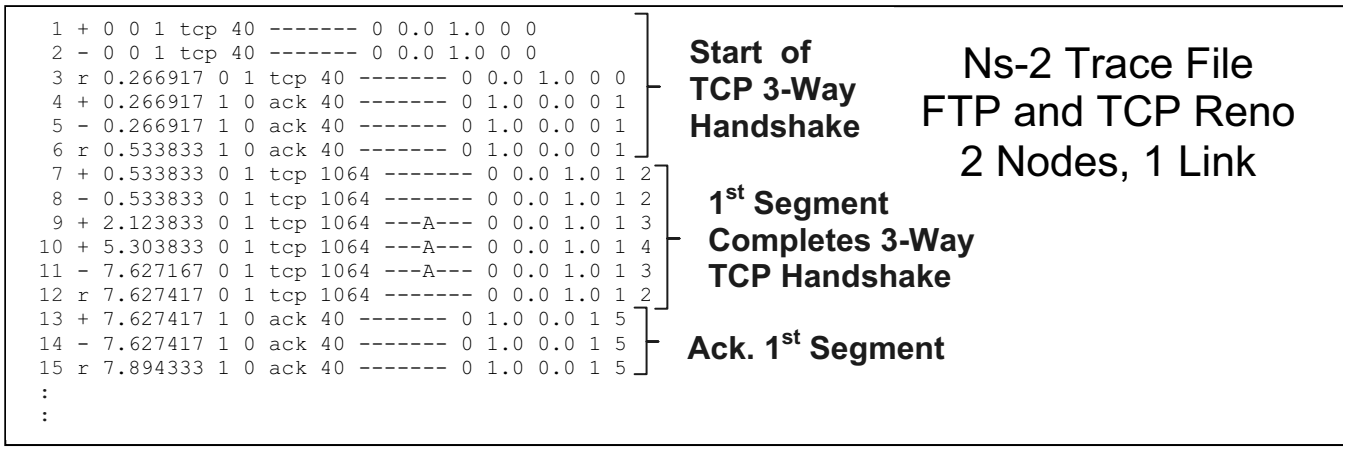


Figure 6: An Example ns-2 Trace File.

The second field specifies the time at which the operation occurred within the ns-2 simulator. The third and fourth fields numerically identify which node is the transmitter (i.e. the first of the two numbers) and which is the transmittee (i.e. the second of the two numbers). The fifth field shows a descriptive name that represents the type of packet transmitted (i.e. tcp, ack, or udp.). The sixth field shows the size of the segment in Bytes as encoded in the Internet Protocol (IP) header.

The seventh field represents a list of seven flags (at most) that describes a specific line. The four following flags are used to define ECN (Explicit Congestion Notification) in TCP/IP. The “E” flag is used to specify “Congestion Experienced”, the “N” flag specifies “ENC-Capable Transport Indications in the IP Header”, the “C” flags specify “ECN-Echo”, and the “A” flag is used to specify “Congestion Window Reduced in the TCP Header”. Two additional flags can be defined: “P” for “Priority” and “F” for “TCP Fast Start”.

The eighth field defines the IP flow identifier as specified by IP version 6. The ninth and tenth fields specify the numerical source and destination node addresses, while the eleventh field indicates a sequence number. Starting with ns-2, version 2, only agents wishing to generate sequence numbers do so. All ns-2, version 1, agents did so. The twelfth, and last, field represents a unique packet identification number.

Fifteen trace lines (i.e. entries) are shown in Figure 6. Lines 1-6 show the transmission of the first two segments associated with a TCP 3-way handshake. The completion of the "handshake" occurs when the first 1064 Byte segment is transmitted from node 0 to node 1 in lines 7-12. Note that a 1064 Byte segment is comprised of 1024 Bytes of data and a 40 Byte header. Lines 13-15 show the acknowledgement sent by node 1 to node 0 in response to receiving the segment transmitted by node 0 on line 12. For detailed information regarding the interpretation of ns-2 generated trace files, the reader should refer to "The ns Manual", Chapter 26, Section 4 (Fall & Varadhan, 2007).

Conclusions

This paper introduces the ns-2 network simulator for student research and shows that the freeware software that comprises the ns-2 network simulator can be installed on PCs and workstations running the Microsoft Windows or UNIX/Linux operating systems. This paper also demonstrates how write to basic ns-2 Tcl scripts to simulate the transmission of data (i.e. files) over a 2-meter packet radio network using FTP/TCP and CBR/UDP. While some programming background is advantageous, this paper provides example ns-2 Tcl scripts and discusses the functionality provided by each line. Trace file output generated by an example ns-2 Tcl script run is also presented and discussed.

In summary, the ns-2 network simulator provides a means for packet radio enthusiasts to evaluate the performance of data transmission using TCP and UDP before creating an actual communication network. From an educational perspective, the ns-2 simulator permits students to simulate data transmission over packet radio networks before obtaining their FCC amateur radio license.

Acknowledgements

The author graciously thanks the following two individuals for their important contributions to this paper. First, thanks to my Elmer, E. Benson Scott II, M.D., AE5V, for his generous dispensation of packet radio knowledge. Thanks also to my wife, Allison M.D. Wiedemeier, Ph.D., for reading several drafts of this paper. The author's research is supported through funds provided by the Clarke M. Williams, Jr. Endowed Professorship in Computer Science at The University of Louisiana at Monroe.

References

Allman, Mark & Falk, Aaron. (1999, October). On the Effective Evaluation of TCP. *ACM Computer Communication Review*. vol 5. no 29.

Allman, M. & Glover, D. & Sanchez, L.. (1999, January). Enhancing TCP Over Satellite Channels using Standard Mechanism. *Request For Comments 2488*.

Altman, Eitan & Jimenez, Tania. (2003, December). Ns Simulator for Beginners. University de Los Andes, Merida, Venezuela. ESSI, Sophia-Antipolis, France. Retrieved July 17, 2007, from: http://www-sop.inria.fr/maestro/personnel/Eitan.Altman/COURS-NS/n3.pdf.

Breslau & Estrin & Fall & Floyd & Heidemann & Helmy & Huang & McCanne & Varadhan & Xu & Yu. (2000, May). Advances in Network Simulation. *IEEE Computer*. vol 33. no 5. pg 59-67.

Fall, Kevin & Floyd, Sally. (1996, July) Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. *ACM Computer Communications Review*. vol 26. no 3.

Fall, Kevin (Ed.) & Varadhan, Kannan (Ed.). (2007, July). *The ns Manual*. Retrieved July 17, 2007, from: http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf.

Floyd, Sally. (2007, July). Network Simulators. Retrieved July 17, 2007, from: http://www.icir.org/models/simulators.html.

Glover, Daniel R. & Kruse, Hans. (1998, April). TCP Performance in a Geostationary Satellite Environment. *Annual Review of Communications*.

Greis, Marc. (1998). Tutorial for the Network Simulator ns. Retrieved July 17, 2007, from: http://www.isi.edu/nsnam/ns/tutorial/.

Henderson, Thomas R. & Katz, Randy H.. (2000). Network Simulation for LEO Satellite Networks. *American Institute of Aeronautics and Astronautics*. vol 37. American Institute of Aeronautics and Astronautics.

Henderson, Thomas R. & Katz, Randy H.. (1999, December). TCP Performance over Satellite Channels. *University of California at Berkeley Computer Science*.

Ishac, Joseph & Allman, Mark. (2001). On the Performance of TCP Spoofing in Satellite Networks. *IEEE Milcom 2001*. Institute of Electronic and Electrical Engineers.

Jones, Greg (Ed.). (1996). *Packet Radio: What? Why? How? Articles and Information on General Packet Radio Topics*. Tucson Amateur Packet Radio Corporation Publisher. ISBN: 0-9644707-0-5.

Kruse, Hans. (1995). Performance of Common Data Communication Protocols Over Long Delay Links: An Experimental Examination. *3rd International Conference on Telecommunication Systems Modeling and Design*.

Mathis, Matthew & Mahdavi, Jamshid. (1996). Forward Acknowledgment: Refining TCP Congestion Control. *SIGCOMM*. pg 281-291. Association for Computing Machinery.

Ns-2. (2007, July). Retrieved July 17, 2007, from: http://www.isi.edu/nsnam/.

Postel, John. (1980, August). User Datagram Protocol. *Request For Comments 768*.

Postel, John. (1981, September). Transmission Control Protocol. *Request For Comments 793*.

Postel, John. (1985, October). File Transfer Protocol. *Request For Comments 959*.

SourceForge. (2007, July). Retrieved July 17, 2007, from: http://sourceforge.net/projects/nsnam/.

Trantor, Jeff & Raines, Paul. (1999, March). *Tcl/Tk in a Nutshell*. O'Reilly Publisher. ISBN 13: 9781565924338.

Wiedemeier, Paul D. & Tyrer, Harry W.. (2007, March). Improved Near-Earth Internet Data Transmission Using New Multi-Layer OSI Protocol Designs. *Proceedings of the 2007 IEEE Aerospace Conference*. Institute of Electrical and Electronics Engineers. Big Sky, Montana. ISBN: 1-4244-0525-4.