**** VERSION 1.0 (manual.doc - 06/05/99) ****

Tools Page Laboratory Manual and Exercises

Kenneth J. Christensen, Ph.D., P.E.
Assistant Professor
Department of Computer Science and Engineering
4202 East Fowler Avenue, ENB 118
University of South Florida
Tampa, FL 33620
christen@csee.usf.edu

This short instructional manual describes how to obtain and use the tools on the Christensen "tools page". This tools page is located at *http://www.csee.usf.edu/~christen/tools/toolpage.html* and contains downloadable tools primarily related to performance evaluation of computer networks and TCP/IP sockets programming. Each tool is written in ANSI C and has been tested to compile with Borland C++ 5.02, (bcc32), Microsoft Visual C++ 5.0 (cl), and gcc v2.7 on Solaris. The format of all of the tools is the same with the program header describing tool purpose, general notes, sample input, sample output, build instructions, execution instructions, and author/contact information. All code is documented with extensive inline comments and header blocks for all functions. The goal of each tool is to serve as a teaching aid for the concept implemented by the tool (and as a model for good programming practice!). Thus, the emphasis is on simplicity and clarity. It is assumed the student will have access to a C compiler and have at least moderate experience in C programming. Below is a partial list of the tools:

- `iter.c` - Solves a sparse probability matrix using iterative methods
- `mva.c` - Mean value analysis for a closed queueing network
- `erlang.c` - Numerically stable implementation for computing Erlang-B and Erlang-C probabilities
- `mean.c` - Computes the mean value of a time series
- `var.c` - Computes the variance of a time series
- `autoc.c` - Computes the autocorrelation (for a given lag) of a time series
- `hurst.c` - Computes the R/S statistic for a time series and is used to estimate the Hurst parameter
- `block.c` - Aggregates a time series for a specified aggregation block size
- `shuffle.c` - Shuffles a time series for a specified block size
- `ttoc1.c` - Converts cumulative time values to counts
- `ttoc2.c` - Converts delta time values to counts
- `dtoclk.c` - Converts delta time values to cumulative time values
- `clktod.c` - Converts cumulative time values to delta time values
- `gendet.c` - Generates a time series with deterministically distributed values
- `genexp.c` - Generates a time series with exponentially distributed values
- `genhyp.c` - Generates a time series with hyperexponentially distributed values
- `genipp.c` - Generates a time series with Interrupted Poisson Process (IPP) distributed values
- `genpar1.c` - Generates a time series with unbounded Pareto distributed values
- `genpar2.c` - Generates a time series with bounded Pareto distributed values
- `merge.c` - Merges two files of time series
- `ttest.c` - Computes the 95% confidence interval of sample means using a Student's T test
- `lr.c` - Computes a linear regression line and coefficient of determination for a paired data series
- `rand.c` - Demonstrates a linear congruential generator to generate pseudorandom numbers (from [1])
- `mm1.c` - Simulates an M/M/1 queue using only C
- `h2d1.c` - Simulates an H2/D/1 queue using only C (H2 is hyperexponential)
- `mm1_smpl.c` - Simulates an M/M/1 queue using the SMPL function library (see [2] for SMPL)
- `mm1_csim.c` - Simulates an M/M/1 queue using the CSIM function library (see [3] for CSIM)
- `tracesim.c` - Simulates a "trace/D/1/K" queue (using a trace file for interarrival times)
- `dlc.c` - Computes utilization (*U*) for various Data Link Control (DLC) protocols
- `token.c` - Computes *U* for a Token Ring local area network
- `ether.c` - Computes *U* for an Ethernet local area network
- `server.c` / `client.c` - TCP/IP client/server implementation using BSD and Winsock sockets
- `server1.c` / `client1.c` - UDP/IP client server implementation using BSD and Winsock sockets
- `getaddr.c` - Gets the IP address for a given host name for BSD and Winsock sockets
- `local.c` / `remote.c` - Remote execution of DOS "exe" files for Windows only (uses Winsock)
- `timit.c` - Demonstrates `ftime()` function for timing program execution
- `thread.c` - Demonstrates Windows threads
- `process.c` - Demonstrates Window processes
- `system.c` - Demonstrates `system()` function for invoking executable programs
- `sleep.c` - Demonstrates Windows `Sleep()` function for delaying program execution

Figure 1 shows the header for `mean.c` as an example of the common tools format. The program `mean.c` is a trivial tool used to compute the mean of a series of numbers. This is a representative example even for the other more complex tools. New tools are continuously added and bugs in existing tools fixed. To report a bug send email to *christen@csee.usf.edu*. For new tools see, *http://www.csee.usf.edu/~christen/tools/newtool.html.*

```
//=========================================================== file = mean.c =====
//=  Program to compute mean for a series X of size N                          =
//=============================================================================
//=  Notes:                                                                    =
//=    1) Input from input file "in.dat" to stdin (see example below)          =
//=        * Comments are bounded by "&" characters at the beginning and       =
//=          end of the comment block                                          =
//=    2) Output is to stdout                                                  =
//=-----------------------------------------------------------------------------=
//= Example "in.dat" file:                                                     =
//=                                                                            =
//=    & Sample series of data which can be integers or reals.                 =
//=      There are 11 values in this file. &                                   =
//=    50                                                                      =
//=    42                                                                      =
//=    48                                                                      =
//=    61                                                                      =
//=    60                                                                      =
//=    53                                                                      =
//=    39                                                                      =
//=    54                                                                      =
//=    42                                                                      =
//=    59                                                                      =
//=    53                                                                      =
//=-----------------------------------------------------------------------------=
//= Example output (for above "in.dat"):                                       =
//=                                                                            =
//=   ------------------------------------------------- mean.c -----           =
//=     Mean for 11 values = 51.000000                                         =
//=   ---------------------------------------------------------------          =
//=-----------------------------------------------------------------------------=
//=  Build: gcc mean.c, bcc32 mean.c, cl mean.c                                =
//=-----------------------------------------------------------------------------=
//=  Execute: mean < in.dat                                                    =
//=-----------------------------------------------------------------------------=
//=  Author: Kenneth J. Christensen                                            =
//=          University of South Florida                                       =
//=          WWW: http://www.csee.usf.edu/~christen                            =
//=          Email: christen@csee.usf.edu                                      =
//=-----------------------------------------------------------------------------=
//=  History: KJC (09/16/98) - Genesis                                         =
//=============================================================================
```

Figure 1 - Header for program `mean.c` to compute the mean of a series of numbers

The following is a step-by-step description of how to find, download, build, and use `mean.c`, including instructions on how to create an input file.

Step #1: Find `mean.c` at *http://www.csee.usf.edu/~christen/tools/toolpage.html.*

Step #2: Download `mean.c` by clicking on its hyperlink. The browser (Microsoft Internet Explorer or Netscape Navigator) will "pop-up" a dialog box asking to save or open the file as shown in Figure 2. Choose to save the file by clicking on the OK button. Next, select a directory or folder in which the tool should be saved (as shown in Figure 3 where the chosen directory is `d:\temp`.)

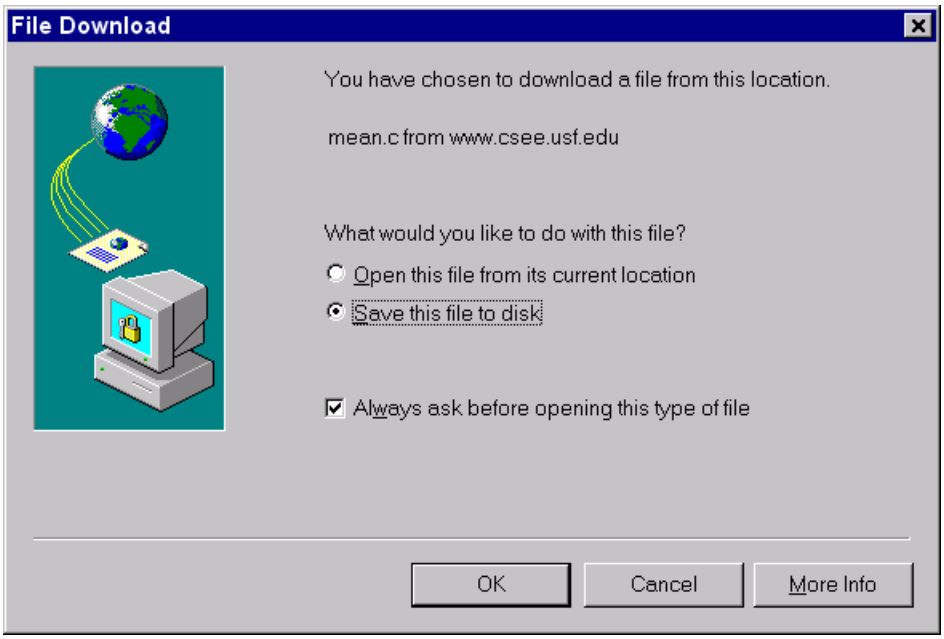

Figure 2- File Download dialog box for `mean.c` (Microsoft Internet Explorer is shown here)

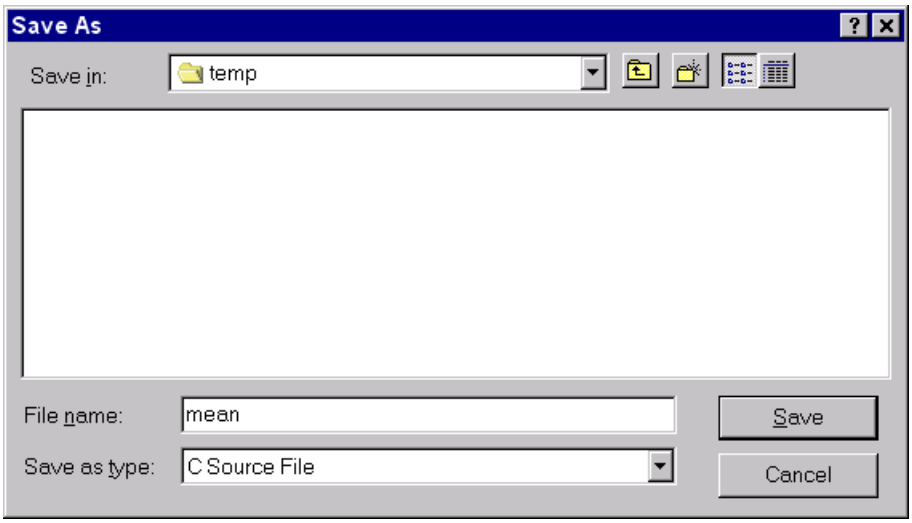

Figure 3 - Save As dialog box resulting from clicking OK on the File Download dialog box in Figure 2

Step #3: Build the tool using a C compiler. Using the command line, as opposed to the integrated development environment of some compilers, is the easiest way to build simple C programs. Figure 4 shows a WindowsNT console window where `mean.c` has been compiled using the Borland compiler, bcc32 (Windows9X would look exactly the same). On a Unix system, `gcc` is used in an identical manner. Note that the `gcc` compiler generates an executable file named `a.out` as the default.

Step #4: Create a test input file using a text editor (e.g., Windows `NotePad`, DOS `Edit`, or Unix `vi`). The test file should contain a series of numbers, all in a single column (see the example in the header of Figure 1).

Step #5: Execute the tool by redirecting the input file into `mean` at the command line. Figure 4 shows an example for WindowsNT executed in the `d:\temp` directory (and where `mean.c` has been compiled in this directory as shown). Execution is done in an identical manner in Unix. In the example shown in Figure 4, the file `in.dat` is output to the screen using the `type` command (use the `cat` command in Unix). The three values entered into `in.dat` with a text editor are displayed. If the output is to be written to a file, then the output can be redirected to a file using standard redirection mechanisms (e.g., for the output to be written to file `out.dat`, the command is, `mean < in.dat > out.dat`).

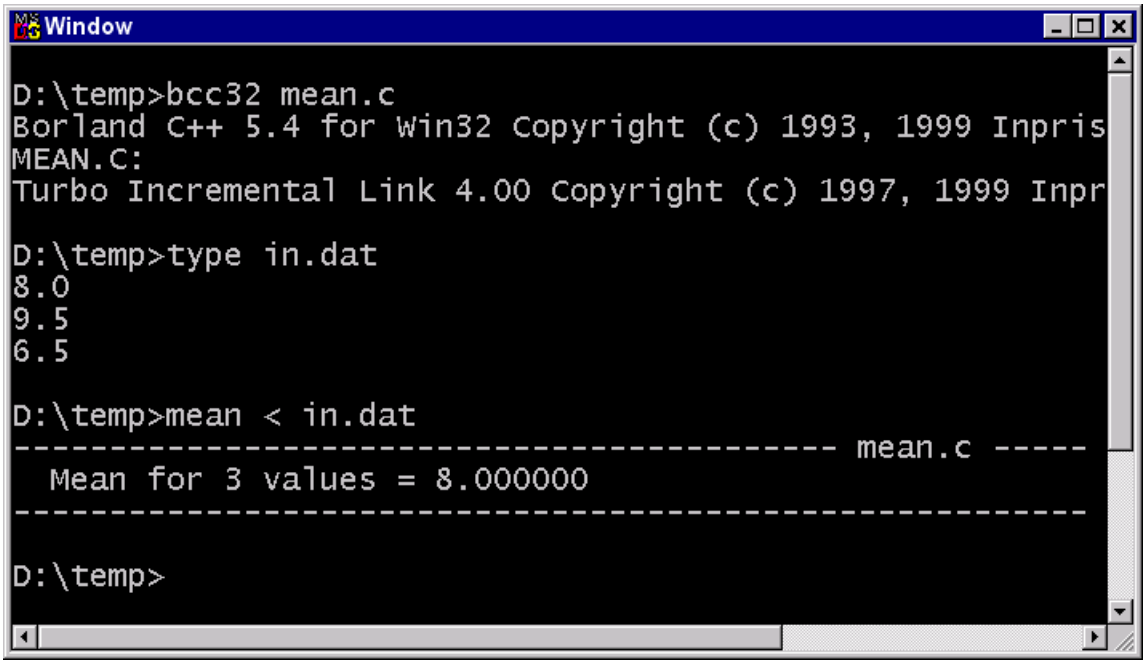

Figure 4 - Example build and execution of `mean.c`

Once a program has been downloaded (e.g., step #'s 1 through 3 above), read the program header for build and execution instructions. A text editor can be used to view the program header. Some programs, for example `dlc.c` for computing link utilization of DLC protocols, take user input from the console and then output results to the console (i.e., the screen). The program `dlc.c` is used in the first exercise.

The remainder of this manual contains exercises that utilize the tools available on the tools page. Several of the exercises partially repeat major research results in the field of computer networks. Especially significant are exercises #6, #7, and #8 that study self-similarity effects in network traffic. References are given at the end of each exercise for further information on the key concepts. The ten exercises and their topic of coverage include:

- Exercise #1 - Studies Data Link Control (DLC) flow and error control performance
- Exercise #2 - Studies the effects of frame size on Ethernet utilization
- Exercise #3 - Studies the effects of early token release (ETR) on Token Ring utilization
- Exercise #4 - Studies Markov chains and their solution as a set of equations
- Exercise #5 - Studies the effects of the second moment of an arrival process on queueing delay
- Exercise #6 - Reproduces the Bellcore "self similar" traffic study
- Exercise #7 - Studies the effects of traffic shifts on Long Range Dependent (LRD) statistics
- Exercise #8 - Studies the effects of LRD on queue length
- Exercise #9 - Analysis of collected round-trip delay "ping" data
- Exercise #10 - Studies sliding window performance in a UDP/IP client/server application

References

[1] R. Jain, The Art of Computer Systems Performance Analysis. John Wiley & Sons, Inc., New York, 1991.

[2] M. MacDougall, Simulating Computer Systems: Techniques and Tools. MIT Press, Cambridge, Massachusetts, 1987. An archive of SMPL source code can be found at the University of Florida at, *ftp://ftp.cis.ufl.edu/pub/simdigest/tools/*.

[3] H. Schwetman, "CSIM18 - The Simulation Engine," *Proceedings of the 1996 Winter Simulation Conference*, pp. 517 - 521, December 1996. URL: *http://www.mesquite.com*.

Exercise #1 - Studying DLC flow and error control performance

Situation

Bit Error Rate (BER) can significantly affect the achievable utilization of a link. For a 100-mile T1 link, what is the "tolerable" bit error rate for Stop-and-Wait (SAW) and Sliding Window (SW) DLC flow and error control given frame sizes of 64 bytes and 1500 bytes? For SW, consider both Go-Back-N (GBN) and Selective Reject (SR) Automatic Repeat Request (ARQ) methods.

Requirements

Compute the link utilization, *U*, for a range of BER for SAW, SW-GBN, and SW-SR for the link configuration described in the Situation above. An interesting range of BER is from 10^{-3} (a very high bit error rate) to 10^{-10} (a very low bit error rate). Gain some insights from the numerical results and make a recommendation on the tolerable BER for the link in question. A key assumption for the DLC utilization formulas derived in [1] is that bit errors are independent. It is also assumed that ACK and NACK frames are never in error. If one or both of these assumptions are invalid, how are the results (and the recommendation) affected?

Guidelines and hints

Using the `dlc.c` tool (from [2]) compute *U* values for the above described link for BER of 10^{-3}, 10^{-4}, ..., 10^{-10}. Consider a range of SW window sizes. Plot the results using a spreadsheet. Make a conclusion (i.e., a recommendation) based on the insights derived from the graph. Note that a T1 link date rate is 1.544-Mbps (full-duplex) and there are 1609 meters in a mile.

References

[1] W. Stallings, Data and Computer Communications, Sixth Edition. Prentice-Hall, Upper-Saddle River, New Jersey, 1999.

[2] Tools page for Kenneth J. Christensen, 1999. URL: *http://www.csee.usf.edu/~christen/tools/toolpage.html*.

Exercise #2 - Studying the effects of frame length in Ethernet

Situation

CSMA/CD Ethernet is widely believed to support a maximum throughput (throughput is effectively the same as link utilization, *U*) of no more than 30%. In other words, it is believed that an Ethernet is saturated at 30% utilization. In this exercise the origins of the 30% "myth" are studied.

Requirements

Compute the link utilization, *U*, for a range of Ethernet bus lengths, frame sizes, and number of stations. Fix the data rate to 10-Mbps with a range of frame lengths from 64 to 1500 bytes and a bus length of no greater than $2 \cdot T_{pr}$ bit times for a minimum 64 byte frame length (and a date rate of 10-Mbps and $2 \cdot 10^8$ meters/sec signal propagation). Frame preamble and interframe gap can be neglected to simplify the analysis. Determine, 1) why it is believed that an Ethernet saturates at 30% utilization, and 2) why it is probably more reasonable to believe that throughputs of higher than 30% can be achieved. It has been shown that the Binary Exponential Backoff (BEB) arbitration algorithm tends to allow a single station to transmit large numbers of consecutive frames without interference from other stations. This is called the "capture effect" and is studied in [1]. How does the capture effect change the *U* value?

Guidelines and hints

Using the `ether.c` tool (from [2]) compute *U* values for the a 10-Mbps CSMA/CD Ethernet. The formula implemented in `ether.c` (and also derived in [3]) comes from the original paper describing Ethernet (see [4]). In [5] a seminal experimental study showed that a 10-Mbps Ethernet can support sustained throughputs of much greater than 3-Mbps (or 30% utilization).

References

[2] M. Molle and K. Christensen, "The Effects of Controlling Capture on Multimedia Traffic for Shared Ethernet Systems," *Journal of Telecommunication Systems*, Vol. 9, No. 3-4, pp. 287 - 314, September 1998.

[2] Tools page for Kenneth J. Christensen, 1999. URL: *http://www.csee.usf.edu/~christen/tools/toolpage.html*.

[3] W. Stallings, Data and Computer Communications, Sixth Edition. Prentice-Hall, Upper-Saddle River, New Jersey, 1999.

[4] R. Metcalfe and D. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," *Communications of the ACM*, Vol. 19, No. 7, pp. 395 - 404, July 1976.

[5] D. Boggs, J. Mogul, and C. Kent, "Measured Capacity of an Ethernet: Myths and Reality," *Proceedings of ACM SIGCOMM '88*, pp. 222 - 234, August 1988.

Exercise #3 - Studying the effects of early token release (ETR) in Token Ring

Situation

Early Token Release (ETR) was first implemented in 16-Mbps IEEE 802.5 Token Ring; the original 4-Mbps standard only supported Normal Token Release (NTR). For what configurations (ring length, number of stations, and frame size) does ETR offer performance benefits over NTR?

Requirements

Quantify the performance benefits (i.e., improved utilization, *U*) of ETR over NTR for 16-Mbps Token Ring.

Guidelines and hints

Using the `token.c` tool (from [1]) study *U* values for ETR versus NTR over reasonable ranges of ring length, number of stations, and frame size. For example, fix the number of stations and then vary the ring length and frame size as dependent variables and solve for *U* as the independent variable. Plot the results using a spreadsheet. Make a conclusion on the value of ETR to Token Ring performance.

The tool `token.c` implements the utilization formulas derived in [2]. There is a key assumption to these formulas which must be understood when making conclusions on the performance of ETR versus NTR. Carefully, review the derivation of these formulas in [2]. Reference [3] contains a more advanced performance evaluation of the token ring protocol that removes this assumption.

References

[1] Tools page for Kenneth J. Christensen, 1999. URL: *http://www.csee.usf.edu/~christen/tools/toolpage.html*.

[2] W. Stallings, Data and Computer Communications, Sixth Edition. Prentice-Hall, Upper-Saddle River, New Jersey, 1999.

[3] A. Sethi and T. Saydam, "Performance Analysis of Token Ring Local Area Networks," *Computer Networks and ISDN Systems*, Vol. 9, No. 3, pp. 191 - 200, March 1985.

Exercise #4 - Studying Markov chains and their solution as a set of equations

Situation

Systems of state where the probability of transition to a next state at discrete time interval is dependent only on the current state (and thus independent of previous states) are called discrete Markovian systems. An example of such a system is a network with three states, “up”, “down”, and “under repair” where the transition probabilities between the states are defined, for example, at hourly time periods are known and are all independent. It is desirable to know the steady state (i.e., long term) probability for each of the states of such a system. These steady state probabilities, π , can be found by solving the probability matrix, *P*, for the system. An iterative method can be used to solve for a *P* matrix given an initial guess for π (see, for example, [1]).

Requirements

The requirement for this exercise is to solve for the steady state probability for each state for the system of figure 1 (figure 1 show the states and transition probabilities).

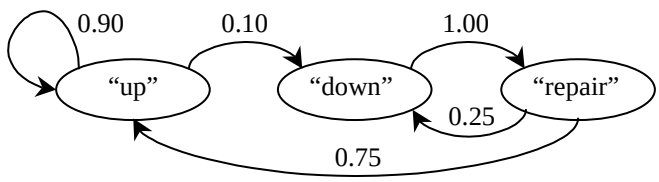

Figure 1 - Markov chain for an example system with three states

Guidelines and hints

For the system of Figure 1 a simple 3 x 3 probability matrix, *P*, can be constructed and solved (for the steady state probabilities π) where $\pi = \pi \cdot P$. The tool `iter.c` (available at [2]) can be used to solved for π . The true power of `iter.c` is in solving for large and sparse *P* matrices (i.e., *P* matrices where the majority of the elements are zero).

References

[1] L. Kleinrock, Queueing Theory: Theory, Volume 1. John Wiley & Sons, New York, 1975.

[2] Tools page for Kenneth J. Christensen, 1999. URL: *http://www.csee.usf.edu/~christen/tools/toolpage.html*.

Exercise #5 - Studying the effects of second moment on queueing delay

Situation

Queueing systems can be used to model computer networks. Queueing systems can be analyzed using analytical and simulation methods. Some knowledge of queueing theory is important for anyone doing serious research in the area of computer networks. The queueing delay, or residence time, of a single-server queue is the sum of the waiting time in the queue plus the service time. An important result is that the mean delay of a queue is highly dependent on the second moment (variance) of the interarrival time of arriving jobs (e.g., packets in a computer network). "Bursty" traffic has a high second moment, "smooth" traffic has a low second moment. In this exercise the mean delay of a single server queue with infinite buffer capacity and deterministic service time is compared given arrivals of increasing burstiness.

Requirements

Compare the mean queueing delay (or mean residence time) of an H2/D/1 queue with a service time of 1 second for a range of utilization values (e.g., from 10% to 90% in steps of 10%) and degrees of burstiness as measured by the Coefficient of Variation (CoV) (e.g., from 1 to 5 in steps of 1). The CoV of a time series is the standard deviation (e.g., of counts or interarrival times) divided by the mean. Make a conclusion on the relative and absolute effects of an increasing CoV on mean queueing delay.

Guidelines and hints

An H2/D/1 queue is a single server queue (the "1") with deterministic service times (the "D/") and hyperexponential interarrival times (the "H2/"). Reference [1] is the classic text for queueing theory. Reference [2] contains an excellent summary of key results from queueing theory. The `h2d1.c` tool from [3] (derived from a simulation program in [4]) is a simulation model of an H2/D/1 queue. Utilization is the ratio of service time to interarrival time (i.e., using the notation from `h2d1.c`, `U = Ts / Ta`). The utilization can be set by fixing the service time (`Ts`) to 1.0 and then varying the interarrival time (`Ta`) in `h2d1.c`. For a CoV of 1.0, the H2/D/1 is equivalent to an M/D/1 queue.

References

[1] L. Kleinrock, Queueing Theory: Theory, Volume 1. John Wiley & Sons, New York, 1975.

[2] R. Jain, The Art of Computer Systems Performance Analysis. John Wiley & Sons, Inc., New York, 1991.

[3] Tools page for Kenneth J. Christensen, 1999. URL: *http://www.csee.usf.edu/~christen/tools/toolpage.html*.

[4] M. MacDougall, Simulating Computer Systems: Techniques and Tools. MIT Press, Cambridge, Massachusetts, 1987.

Exercise #6 - Reproducing the Bellcore "self similar" traffic study

Situation

The early 1990's traffic study by Leland, Taqqu, Willinger, and Wilson at Bellcore discovered that network traffic is "self similar" (a self similar time series is Long Range Dependent (LRD)). This is considered by many researchers to be the most significant and far-reaching discovery in the field of networks in the past decade. Several papers were published describing this discovery, including [1]. In this exercise the key finding of [1] will be reproduced using the original Bellcore traffic trace file that is available on the Internet Traffic Archive (see [2]).

Requirements

The requirement for this exercise is to find the Hurst parameter, *H*, for the Bellcore data available at [2] (the file `pAugTL.z` was used in the solutions). Analyze packet counts only (i.e., the second column containing packet size data should be removed from the trace file). The Hurst parameter is a measure of self-similarity and ranges from 0.50 (totally independent - Poisson) to 1.0 (totally self-similar).

Guidelines and hints

The time stamps of the Bellcore raw data are cumulative. The first step is to convert this data into packet counts for some "small value" of time (0.010 seconds is recommended). The next step is to block the data by aggregations of 10 for at least three aggregations (i.e., M = 10, 100, and 1000). For each of the blocked files, compute the R/S value. The Hurst parameter is estimated from the slope of the line of a plot of Log(M) versus Log(R/S). Microsoft Excel can be used to fit a line to a plot of Log(M) versus Log(R/S).

For this exercise the following tools (from [3]) are needed: `ttoc1.c`, `block.c`, and `hurst.c`. The tools, `mean.c`, `var.c`, `cov.c` and `autoc.c` can also be useful in studying the Bellcore traffic trace. The tool `genexp.c` can be used to generate Poisson data for a comparison analysis. The linear regression tool, `lr.c`, can be used instead of Microsoft Excel to fit a line to the R/S data.

References

[1] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the Self-Similar Nature of Ethernet Traffic (Extended Version)," *IEEE/ACM Transactions on Networking*, Vol. 2, No. 1, pp. 1 - 15, February 1994.

[2] BC - 4 million-packet traces of LAN and WAN traffic seen on an Ethernet. URL: *http://ita.ee.lbl.gov/html/contrib/BC.html*.

[3] Tools page for Kenneth J. Christensen, 1999. URL: *http://www.csee.usf.edu/~christen/tools/toolpage.html*.

Exercise #7 - Studying the effects of traffic shifts on LRD statistics

Situation

Exercise #6 reproduced the famous "self similar" Bellcore traffic study. Following the publication of [1], many researchers questioned if network traffic is truly self-similar or if underlying non-stochastic (e.g., daily trend) phenomena may be affecting analysis results. A very mathematical treatment of this issue can be found in [2]. In this exercise, a synthetic traffic file will be analyzed for self-similarity where the file will contain a predictable shift in traffic intensity.

Requirements

The requirement for this exercise is to find the Hurst parameter, H, for a synthetically generated traffic file containing a shift in intensity. Generate a traffic file with about 50 seconds of exponentially distributed values with arrival rate (λ) of 100.0 and another file with about 50 seconds of exponentially distributed values with arrival rate of 200.0. Append the second file to the end of the first file (e.g., by using a text editor) simulating a traffic intensity shift occurring at about time = 50 seconds. Determine the Hurst parameter for the resulting combined file. Analyze packet counts (i.e., the same as in exercise #6).

Guidelines and hints

For this exercise the following tools (from [3]) are needed: `genexp.c`, `ttoc2.c`, `block.c`, and `hurst.c`. The tools, `mean.c`, `var.c`, `cov.c`, and `autoc.c` can also be useful in studying the traffic files. See Exercise #6, Guidelines and hints, for a discussion of how the Hurst parameter can be estimated.

References

[1] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the Self-Similar Nature of Ethernet Traffic (Extended Version)," *IEEE/ACM Transactions on Networking*, Vol. 2, No. 1, pp. 1 - 15, February 1994.

[2] N. Duffield, J. Lewis, N. O'Connell, R. Russell, and F. Toomey, "Statistical Issues Raised by the Bellcore Data," *Proceedings of 11th IEE UK Teletraffic Symposium*, Cambridge, 1994.

[3] Tools page for Kenneth J. Christensen, 1999. URL: *http://www.csee.usf.edu/~christen/tools/toolpage.html*.

Exercise #8 - Studying the effects of "self similarity" on queueing behavior

Situation

Exercise #6 reproduced the famous "self similar" Bellcore traffic study. Following the publication of [1], an important next step was to study the effects of traffic with Long Range Dependence (LRD) on queueing behavior. A fairly simple proof of the effects of LRD on mean queue delay is found in [2]. This exercise proves that LRD in an input process has significant effects on the mean delay of a single server queue by reproducing the "shuffle experiment" in [2].

Requirements

The requirement for this exercise is to compare the mean delay of a single server queue given an input (arrival process) that is LRD, Short Range Dependent (SRD) only, and shuffled LRD. By shuffling an LRD traffic trace, LRD properties can be removed while keeping SRD properties. For the LRD input the Bellcore traffic trace from [3] can be used (the file `pAugTL.z` was used in the solution). From the results of the experiments, make conclusions on the effects of input traffic with LRD and SRD on queueing behavior.

Guidelines and hints

The time stamps for the Bellcore data are cumulative. The first step is to convert this data into "delta" time stamps. Then, the converted trace file can be used directly as input to `tracesim.c` where the mean service time (`Ts`) and buffer size need to be set. The `Ts` value should be set so that `Ts` / `Ta` is a value less than one (e.g., 0.75). This value (the ratio of `Ts` / `Ta`) is the queue utilization. The value `Ta` is determined as the mean value of the converted trace file. Synthetic traces files containing exponentially and hyperexponentially distributed interarrival times can be generated to study SRD traffic processes. The converted Bellcore trace file can be shuffled with various block sizes to study what happens when long range correlations are broken (but short-range correlations within a block size are maintained).

For this exercise the following tools (from [4]) are needed: `mean.c`, `var.c`, `cov.c`, `genexp.c`, `genhyp.c`, `shuffle.c`, and `tracesim.c`. See also the "Guidelines and hints" from Exercise #5 for a discussion of the simulation of queues.

References

[1] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the Self-Similar Nature of Ethernet Traffic (Extended Version)," *IEEE/ACM Transactions on Networking*, Vol. 2, No. 1, pp. 1 - 15, February 1994.

[2] A. Erramilli, O. Narayan, and W. Willinger, "Experimental Queueing Analysis with Long-Range Dependent Packet Traffic," *IEEE/ACM Transactions on Networking*, Vol. 4, No. 2, pp. 209 - 223, April 1996.

[3] BC - 4 million-packet traces of LAN and WAN traffic seen on an Ethernet. URL: *http://ita.ee.lbl.gov/html/contrib/BC.html*.

[4] Tools page for Kenneth J. Christensen, 1999. URL: *http://www.csee.usf.edu/~christen/tools/toolpage.html*.

Exercise #9 - Analysis of collected round-trip delay "ping" data

Situation

Network round-trip times can easily be measured using the ping program found on Windows and Unix systems. The ping program sends a packet to a remote host which then returns the packet (via an echo service) to the sending host. The round-trip time is measured. Are network delays self-similar? A study reported in [1] suggests that they may be.

Requirements

The requirement for this exercise is to collect network round-trip delay data and determine its characteristics including mean, variance, autocorrelation, and Hurst parameter.

Guidelines and hints

The ping utility can be used to collect the round trip data. A "large number" of values must be collected for a good analysis. For example, 10,000 round trip delay with one second between each measurement may be a sufficient number of values. It may also be interesting to compare the delay characteristics of a long (e.g., overseas) versus short network path. The programs `mean.c`, `var.c`, `autoc.c`, `block.c`, and `hurst.c` from [2] can be used to characterize the collected data. See Exercise #6, Guidelines and hints, for a discussion of how the Hurst parameter is estimated.

References

[1] M. Orella, S. Uldag, G. Brewster, and I. Sidhu, "Self-Similarity of Internet Packet Delays," *Proceedings of the IEEE International Conference on Communications*, pp. 513 - 517, 1997.

[2] Tools page for Kenneth J. Christensen, 1999. URL: *http://www.csee.usf.edu/~christen/tools/toolpage.html*.

Exercise #10 - Studying sliding window in a UDP/IP client/server application

Situation

In exercise #1 the performance of Data Link Control (DLC) flow control methods was studied. In particular, it was seen that Sliding Window (SW) results in higher link utilization than Stop-and-Wait (SAW) for long/slow links with small packets. When using UDP/IP to transfer data packets, there is no "built in" error or flow control as there is with TCP/IP. The situation to be studied is a UDP/IP client/server configuration where the server sends data packets to the client and the client acknowledges each packet. It is of interest to see what performance improvements are possible with a SW versus SAW implementation of the acknowledgements.

Requirements

The requirement for this exercise is to build a client and server UDP/IP application that sends data packets from a sender (server) to a receiver (client) and measure the packet transfer rates given SAW and SW implementations. The receiver will ACK each packet and keep a count of the total number of packets received. The goal is to determine the improvement possible in the packet transfer rate over both high-speed (e.g., 10-Mbps Ethernet) and low-speed (e.g., 28.8-kbps modem) links. Additionally, the effects of packet size can also be studied.

Guidelines and hints

For this exercise the programs `server1.c` and `client1.c` from [1] are easily modified to include window flow control. In addition, `timeit.c`, also from [1], can be used to see how timing of program execution can be implemented in a Windows and Unix portable manner. The server program should have a loop that sends data packets and receives ACK packets. For example,

```
while(true)
  send a data packet
  receive an ACK packet
endwhile
```

The receive of a packet from the client is the ACK from the client. The client program should have a loop that receives and then sends a specified number of packets within a start and end timer For example,

```
start timer
count = NUM_PACKETS
while(count > 0)
  receive a data packet
  send an ACK packet
  count = count - 1
endwhile
end timer and determine elapsed time
compute packet transfer rate
```

To implement SW, a "window size minus one" number of packets can be sent from the server before entering the loop shown above. This number of packets corresponds to the outstanding packets in a SW window. Note that if a packet is lost, the above outlined implementations will not be able to recover. Packet loss is very rare in most networks and should not present a problem for this exercise. However, a time-out mechanism could easily be implemented in the client.

References

[1] Tools page for Kenneth J. Christensen, 1999. URL: *http://www.csee.usf.edu/~christen/tools/toolpage.html*.