GENERATING SELF-SIMILAR TRAFFIC FOR OPNET™ SIMULATIONS

Arnold W. Bragg
Fujitsu Network Communications, Inc.
1000 Saint Albans Drive, Raleigh NC 27609 USA
arnold.bragg@fnc.fujitsu.com

ABSTRACT

Self-similar (s-s) arrival processes are realistic models for many types of network traffic. Unfortunately, generating synthetic s-s arrivals and interarrival times for simulations is rather involved. This paper discusses six issues related to s-s traffic generation for discrete-event simulations: **(i)** what s-s processes are, and why they are important to network modelers; **(ii)** where to find a fast s-s generator; **(iii)** how to install the generator and synthesize s-s arrivals; **(iv)** how to convert s-s arrival counts to interarrival times; **(v)** how to build a simple OPNET™ process model for generating interarrivals; and **(vi)** how traffic from simulated s-s arrival processes compares with traffic from simulated bursty and Poisson arrival processes.

1. SELF-SIMILAR PROCESSES

Network modelers used to routinely assume that arrival events were independent, or nearly so. The Poisson arrival process was convenient, was easy to model and analyze, and fit many types of network traffic reasonably well. More sophisticated models (*e.g.*, based on Markov-modulated Poisson or Bernoulli processes) were used to convey the autocorrelation or short-range dependence (SRD) found in bursty empirical traffic traces. In the early 1990s, researchers found evidence of long-range dependence (LRD) in empirical traffic traces. This raised serious doubts about modeling network arrivals using Poisson and Markov-modulated processes, as these (if stationary) cannot convey LRD [1, 2].

There is convincing evidence that LRD is present in many types of network traffic, including: Ethernet LAN traffic, WAN traffic, coded video traffic, frame relay traffic, connectionless DQDB traffic, common channel signaling traffic in LEC networks, ATM WAN traffic, narrowband ISDN traffic, telnet packet arrivals, the sizes of FTP bursts, and some types of WWW traffic [4, 5, 6, 7]. LRD greatly lengthens the tail of queue waiting time distributions, so ignoring LRD can lead to overly optimistic estimates of performance in buffer dimensioning. This can result in more frequent buffer overflow in finite queues, in delays an order of magnitude longer than expected in some queues, and/or other manifestations of poor performance [1, 3]. This phenomenon is illustrated in Section 6.

LRD is a characteristic of self-similar (s-s) processes, and so s-s processes are often used to generate synthetic arrivals in simulations when LRD is an important trait. Many s-s generators have been proposed. Most are based on fractional stochastic processes, fractals, chaotic maps, on/off models with heavy tails, superposed on/off sources, or wavelet transforms of s-s processes. We use Paxson's generator `fft_fgn` [3, 8] to synthesize a s-s fractional Gaussian noise (FGN) process. The generator uses a fast Fourier transform to estimate the power spectrum of an FGN process. The generator is scalable, fast, efficient, comparatively simple to implement, and provides reproducible traces that are statistically indistinguishable from FGN. The `fft_fgn` generator has switches for specifying the trace size, Hurst number (the degree of LRD in the trace), and scaling factors for linearly transforming the trace to any mean and variance.

SRD and LRD are both important in network simulations. *E.g.*, in finite buffers, SRD and LRD influence the delay distribution of packet arrivals, and LRD influences packet drops [3]. Carriers have begun using LRD models to engineer data networks, and are finding them to be more robust than their SRD counterparts [4]. ATM researchers have found that the s-s trait is robust and cannot be removed by shaping; at short time scales, SRD is reduced by shaping but LRD is not [5]. Finally, there is conflicting evidence about the impact of LRD at very short time scales. Some believe that SRD dominates, and that LRD has little influence on queue lengths at these time scales [6]. The simulation results summarized in Section 6 generate packet arrivals on the order of 0.01 seconds. The LRD phenomenon is clearly influential at this time scale. Peak queue lengths for s-s traffic are 4.5 times larger than for bursty traffic, and 35 times larger than for Poisson traffic.

2. APPROACH

Generating s-s traffic consists of five steps:
(a) Obtain and install the s-s traffic generator `fft_fgn`.
(b) Use `fft_fgn` to create a file of arrival counts, one

count per time slot (or "bin"). The bin width is constant (*e.g.*, 0.1 seconds). *E.g.*, `fft_fgn` can be configured to write a file containing 4,096 arrival counts. The first entry represents the number of arrivals between t = 0.0 and t = 0.1 seconds, the i^{th} entry between t = (i-1)×0.1 and t = i×0.1 seconds, and the 4096^{th} between t = 409.5 and t = 409.6 seconds.

(c) Convert each bin's arrival count to a set of event arrival times for that bin. The OPNET™ s-s module contains a simple five-state process model. The *Init* state reads the file of arrival counts and generates a file of event interarrival times. *E.g.*, if the average bin arrival count is 200 and there are 4,096 counts, then approximately 819,000 events can be scheduled.

(d) Schedule arrivals. The *Wait* state reads one record from the file of event interarrival times and schedules the next arrival.

(e) Process arrival events. The *Send* state creates and sends one packet and returns to *Wait*. The *Boundary* and *Endsim* states are stubs, and are accessed on bin boundaries and at the end of the simulation, respectively. No operations are performed in these states, but more sophisticated traffic models will undoubtedly use them.

3. OBTAINING AND INSTALLING THE SELF-SIMILAR TRAFFIC GENERATOR

Steps (a) and (b) of Section 2 rely on Schuler's implementation [8] of Paxson's s-s generator [3]. This section describes how to obtain and install the generator.

(a) Point to the Internet Traffic Archive at `http://ita.ee.lbl.gov/html/contrib/fft_fgn_c.html` and retrieve the .tar file (49K, compressed tar format). The file name is `fft_fgn_c-1_2_tar.Z`. (Contact this paper's author for a copy if the ITA is offline.)

(b) The .tar file contains programs and documentation written by Christian Schuler [8] (and based on [3]) for synthesizing a s-s process. It requires an ANSI C compiler.

(c) Follow the directions in the documentation. You must uncompress and untar the file, run a "make" script, and verify the generator's operation. (Detailed instructions are available from this paper's author.)

(d) The `fft_fgn` generator has twelve switches. Five are particularly important: Hurst number, mean, variance, sample size, and output file name.

(e) The `fft_fgn` generates about one million sample points per minute (Figure 1) on a Sun SPARC Ultra 1 workstation (170 MHz CPU, 512 MB RAM). Each sample point represents the number of arrival events in one time slot (bin), so the number of interarrival times one can generate is approximately the product of the mean and the sample size.

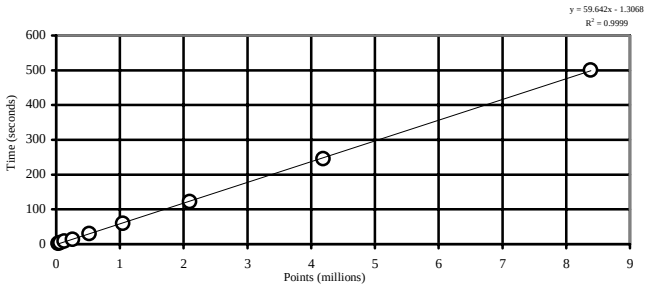

Figure 1. `fft_fgn` run time *vs*. points generated.

4. CONVERTING ARRIVAL COUNTS TO OPNET™ ARRIVAL TIMES

Simulations are driven by arrival times rather than by the number of arrivals per time slot (bin). A Poisson arrival process has exponentially distributed interarrival times, so it is simple to map Poisson arrivals to interarrival times. Unfortunately, there is no convenient way to map an s-s arrival process to interarrivals. The `fft_fgn` generates s-s arrival counts, but the counts must be filtered for non-positive values, converted to integers, mapped to bins, and scattered in some way over each bin. Arrivals can be equally spaced within the bin, or randomly scattered through the bin, or distributed through the bin in some way that introduces SRD (*e.g.*, via an ARMA process). More complicated methods of distributing arrivals within bins have also been proposed (*e.g.*, based on wavelets).

For example, suppose `fft_fgn` generates a file containing arrival counts `211.053, 81.867, 4.211, -38.210, ...` . If the arbitrary bin width is 0.1 second, we can assign 211 arrivals to bin 0, 82 to bin 1, 4 to bin 2, 0 to bin 3, *etc*. The 211 arrivals in bin 0 can be scattered equally (with the first arrival at t = 0.0004739 (=1/211×0.1), the second at t = 0.0009479, ...), or randomly via calls to a uniform variate generator scaled to [0, 0.1), or by selecting 211 consecutive points from a stationary ARMA process and mapping their distribution to [0, 0.1), *etc*.

5. OPNET™ NODE AND PROCESS MODELS

Steps (c) through (e) of Section 2 are performed by the OPNET™ model. This section describes a simple model for generating and scheduling arrivals with LRD. In practice, one might use this model as a template to construct a more sophisticated scheduler for LRD arrivals.

The OPNET™ network model consists of the single node `traffic_generators`. The node model (Figure 2) consists of six traffic generator modules feeding six FIFO queue modules feeding a common sink module. It

is loosely based on the contributed model `Bursty Generator` distributed with OPNET™ v3.0b (`/…/models/contrib/bursty_gen`). That model compared ideal and simple bursty generators feeding FIFO queues feeding a common sink. The `traffic_generators` model contains process models for two Poisson, one bursty, and three s-s arrivals generators. The FIFO queues all use OPNET™'s `acp_fifo` process model, and the sink is OPNET™'s `sink` process model. The three s-s generators differ in the way they distribute arrival counts over bins (equally, randomly, and via a first-order autoregressive process).

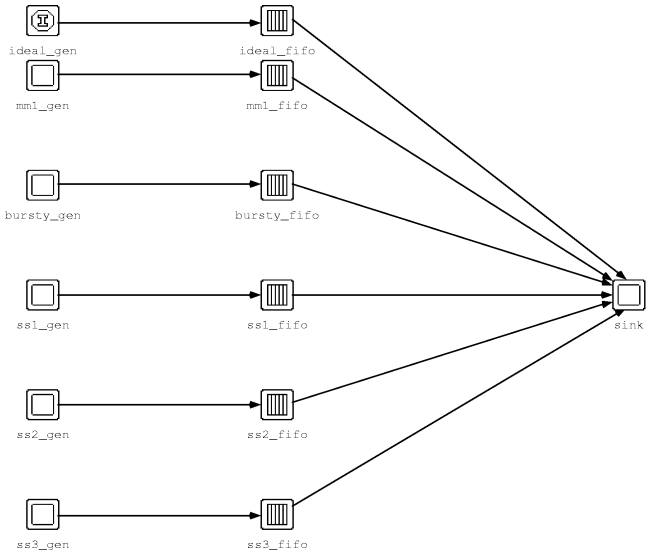

Figure 2. Node model. The ideal and bursty modules are from the contributed model `Bursty Generator`.

The model is used to compare the buffer requirements for the six traffic generators. Sources generate roughly the same number of packet arrivals, and FIFO queues are serviced at roughly the same rates. All six generators send packets to identical, separate FIFO queues at 100 packets per second on average. Each queue can service 150 packets per second on average. The ideal and Poisson-arrivals generators have exponentially distributed packet interarrival times. The bursty generator has an exponentially distributed burst interarrival time. When activated, it sends a burst of ten packets back-to-back. The three s-s-arrivals generators have interarrival times derived from the same s-s arrival process, but distributed within bins in different ways. Bin widths are 1 second. Each s-s process model has five states (Figure 3).

The first s-s model (equal distribution) is described below. The *Init* state reads the s-s input data file `ss1.dat` (created by `fft_fgn`), extracts the arrival count from each line,

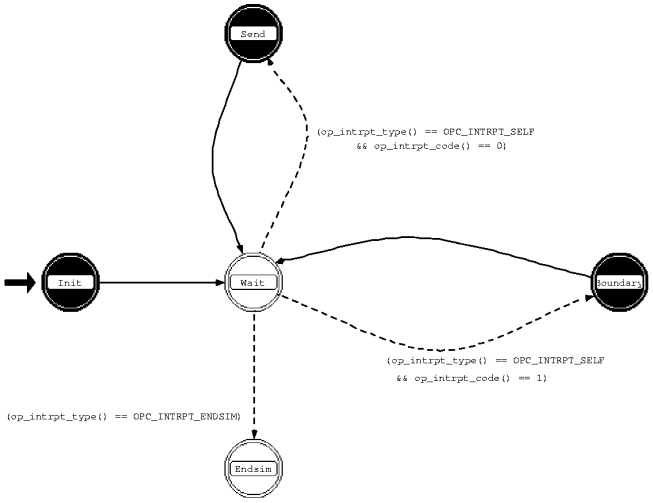

Figure 3. Process model for the three s-s generators.

rounds it to an integer, handles negative and zero counts, and generates interarrival times that are equally spaced across the 1-second bin. Synchronization points (bin boundaries) are inserted as negative integers. The interarrival times and bin boundaries are written to the output file `ss1.out` as follows:

Input file *ss1.dat*	**Output file** *ss1.out*
211.053 81.867 4.211 -38.210 ...	0.0047393364 (=1/211 × 1.0) <210 more identical lines> -1.0000000000 (bin boundary #1) 0.0121951219 (=1/82 × 1.0) < 81 more identical lines> -2.0000000000 (bin boundary #2) 0.2500000000 (=1/4 × 1.0) < 3 more identical lines> -3.0000000000 (bin boundary #3) 1.0000000000 -4.0000000000 (bin boundary #4) ...

Table 1. Input file to output file translation.

The *Wait* state reads the output file `ss1.out` (created by *Init*) and schedules the next arrival interrupt based on the interarrival time. Negative values are used to synchronize on bin boundaries. Packet arrival events and bin boundary events have self-interrupt codes of 0 and 1, respectively. Figure 4 illustrates the interaction among data files and programs/states.

The *Send* state creates and sends a packet of size 10 K bits. The *Boundary* state is accessed only at synchronization points. No operation is performed in this example, but a more realistic implementation might require them. The *Endsim* state is for housekeeping.

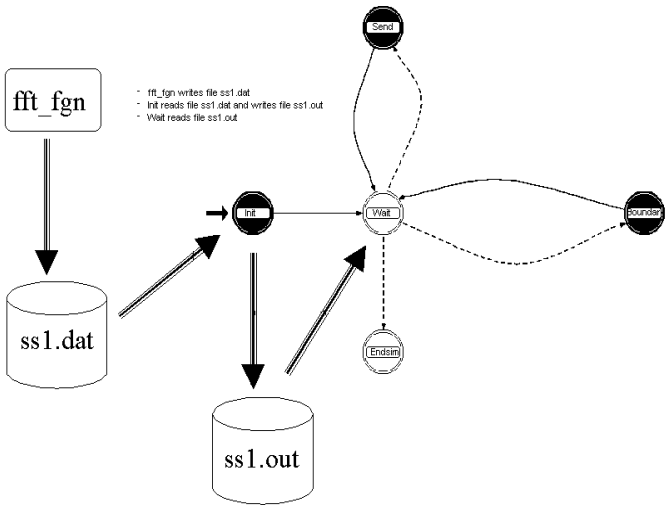

Figure 4. `fft_fgn`, model, and data file interaction.

The OPNET™ code for the first s-s model is:

```
/* ---header block--- */
#include <math.h>
FILE  *fp_in_1;
FILE  *fp_out_1;
/* ---state variables--- */
int       \sent_1;
double    \sim_time_1, \last_boundary;
/* ---temporary variables--- */
char     *c, line [128];
double   bin_count, interarrival_time;
int      read_1, write_1, i;
/* ---Init enter--- */
/* generate interarrival times from bin counts   */
/* equally spaced across the bin, and write the  */
/* IATs to a new file; insert synchronization    */
/* points (bin boundaries) via negative integers */
fp_in_1  = fopen("ss1.dat", "r");
fp_out_1 = fopen("ss1.out", "w");
read_1=0;
write_1=0;
sim_time_1 = 0.0;
do {
  c = fgets(line, 128, fp_in_1);
  if (c != NULL) {
    sscanf(line, "%lf", &bin_count);
    bin_count = rint(bin_count);
    if (bin_count < 1.0) bin_count = 1.0;
    read_1++;
    for (i = 0; i < (int) bin_count; i++) {
      fprintf(fp_out_1, "%30.15f\n",
        (double) (1.0/bin_count));
      write_1++;
    }
    sim_time_1 = sim_time_1 + 1.0;
    fprintf(fp_out_1, "%30.15f\n",
      (double) (-sim_time_1));
    write_1++;
  }
} while (c != NULL);
fclose(fp_in_1);
fclose(fp_out_1);
fp_in_1  = fopen("ss1.out", "r");
printf("ss gen 1 read  %d\n", read_1);
printf("ss gen 1 wrote %d\n", write_1);
sent_1 = 0;
sim_time_1 = 0.0;
/* ---Wait enter--- */
/* schedule next interrupt to send packet       */
/* read next, get interarrival time if not null */
/* negative values used to synchronize the bins */
c = fgets(line, 128, fp_in_1);
if (c != NULL) {
  sscanf(line, "%lf", &interarrival_time);
  if (interarrival_time < 0.0) {
    sim_time_1 = -interarrival_time;
    op_intrpt_schedule_self (sim_time_1, 1);
    last_boundary = sim_time_1;
  }
  else {
    sim_time_1 = sim_time_1 + interarrival_time;
    if (sim_time_1 > last_boundary + 1.0)
     sim_time_1 = last_boundary + 1.0;
    op_intrpt_schedule_self (sim_time_1, 0);
  }
}
/* ---Send enter--- */
op_pk_send (op_pk_create (10000), 0);
sent_1++;
/* ---Boundary enter--- */
/* no-op */
/* ---Endsim enter--- */
fclose(fp_in_1);
printf("ss gen 1 sent %d\n", sent_1);
```

The second and third s-s process models also have five states. The second model's *Init* state distributes packet arrivals randomly across each bin. The third model's *Init* state introduces SRD by using a first-order autoregressive process (ARMA(1,0) = AR(1)) to distribute packet arrivals across each bin. The other four states mimic their counterparts in the first s-s process.

6. RESULTS

Figures 5 through 11 compare the Poisson ("mm1"), bursty, and three s-s generators. The simulated run time is 500 seconds, s-s bin widths are 1 second, packet generation rates are approximately 100 packets/second, and queue service rates are approximately 150 packets/second. The Poisson, bursty, and s-s generators have nearly the same number of arrivals (49,852, 49,990 and 52,230). Figure 5 is an overlay plot of queue length *vs.* run time for the five generators. Poisson traffic is buried in the noise floor, as the Poisson queue length never exceeds 13. Most of the very narrow, sharp spikes just above the noise floor are due to the bursty traffic. Half of the bursty traffic never exceeds a queue length of 11, and the remainder never exceeds 100. Figure 6 is an individual plot for the first s-s generator. Note the low level of activity between the large peaks.

The s-s traffic in Figures 5 and 6 is noteworthy in that half of it never exceeds a queue length of 1 (like Poisson), 70% never exceeds 20 (like bursty), and 90% never exceeds 100 (the bursty peak). The remainder of the s-s traffic is distributed between 100 and 454. Five "incidents" peak well above 100, and the broad incident

is actually a set of 4-5 incidents. Their juxtaposition in time contributes to the queue peak of 454. In summary, the s-s traffic in this example is generally indistinguishable from a mix of Poisson and bursty traffic 90% of the time. It ultimately peaks at a queue length that is 4.5 times larger than the bursty peak and 35 times larger than the Poisson peak. Peaks are broad.

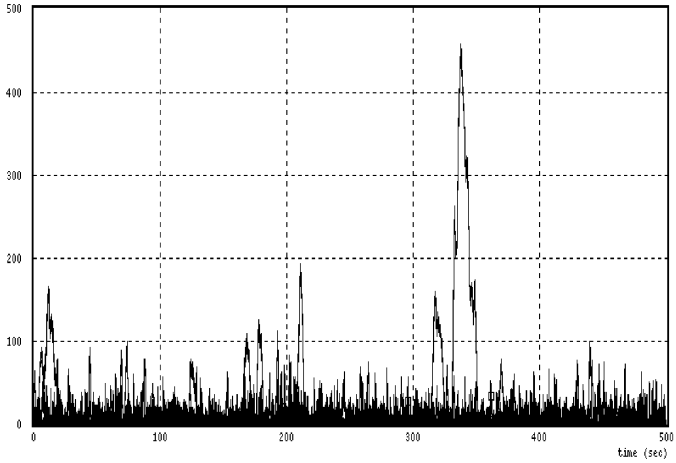

Figure 5. Queue length *vs.* time from a 500 second simulation with all generators.

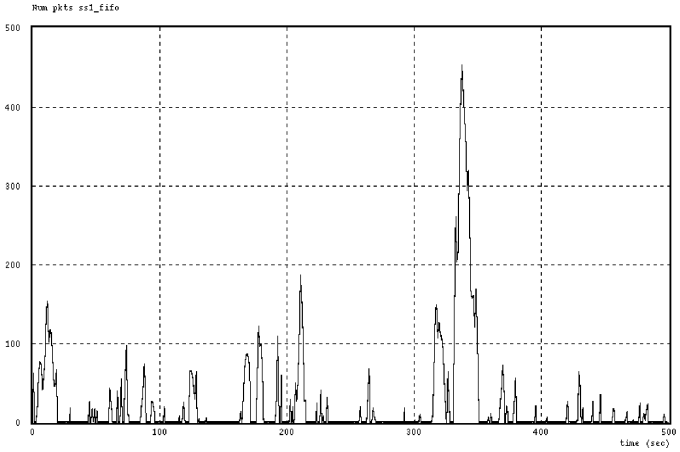

Figure 6. Queue length *vs.* time from a 500 second simulation for the first s-s generator.

Figure 7 is an 80 second subset of Figure 5. Most of the sharp, very narrow spikes just above the noise floor are due to the bursty traffic. The much taller and wider incidents are due to the three s-s traffic generators. Figure 8 is a 9 second subset of Figure 4. Here, the bursty traffic is composed of sets of sharp spikes just above the noise floor, corresponding to queue lengths between 10 and 35. Some burst sets consist of 5-9 contiguous spikes. The much taller three-plot incident rising from the noise at t = 123 and extending to about t = 130 are the three s-s variants. The heavy line between

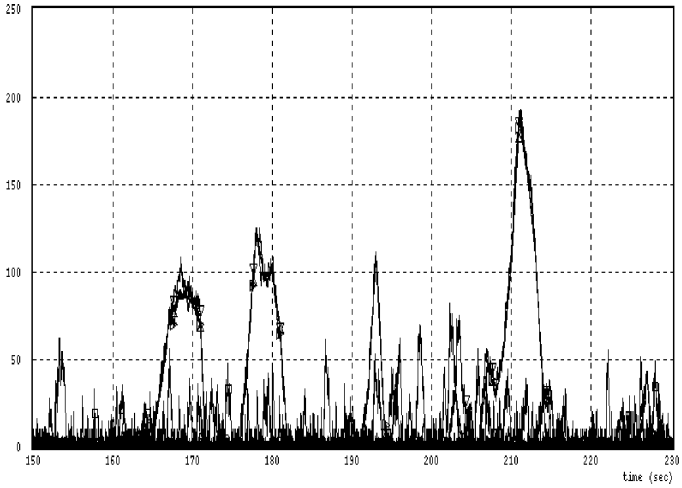

Figure 7. Queue length *vs.* time for an 80 second subset of the 500 second simulation with all generators.

t = 124 and t = 125 in Figure 8 is the first s-s variant (arrivals equally distributed across the 1-second bin); the line slightly above the solid line in this bin is the second s-s variant (arrivals randomly distributed across the bin); the line slightly below the solid line in this bin is the third s-s variant (arrivals distributed across the bin according to an AR(1) process). It is not clear in Figure 8, but the second and third variants alternate being above/below the solid line in bins between t = 123 and t = 129 in this example. From these plots, it is obvious that s-s traffic requires buffer sizes an order of magnitude larger than Poisson traffic. Figures 9 and 10 show the empirical cumulative distribution functions of queue length for the Poisson and bursty generators. Figure 11 shows the long-tailed empirical probability mass function of queue length for the first s-s generator.

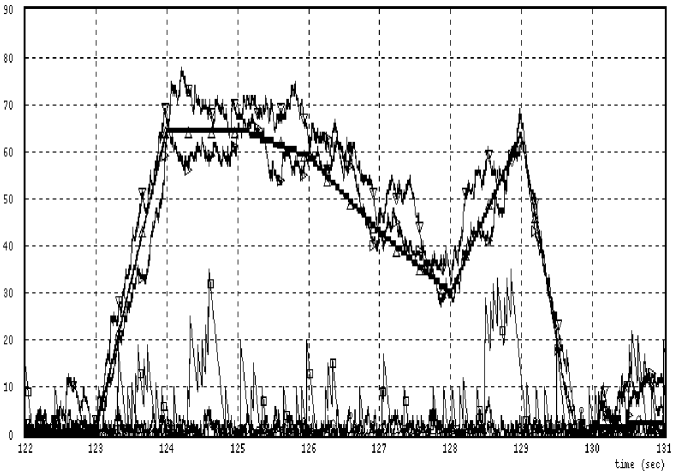

Figure 8. Queue length *vs.* time for a 9 second subset of the 500 second simulation with all generators.

As noted, Figure 9 is the empirical cumulative distribution function (cdf) of queue length for Poisson traffic. It has a short tail (13), 50% of its mass is 1 or less, and 95% of its mass is 4 or less.

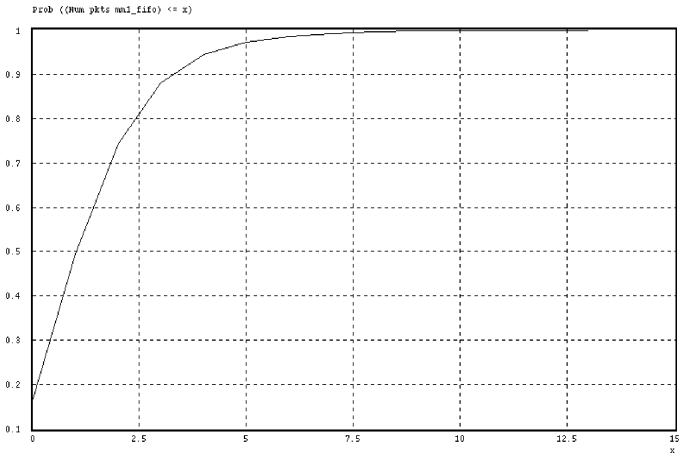

Figure 9. Empirical cdf for the Poisson generator.

As noted, Figure 10 is the empirical cdf of queue length for bursty traffic. It has a longer tail (100), 50% of its mass is 11 or less, and 95% of its mass is 43 or less.

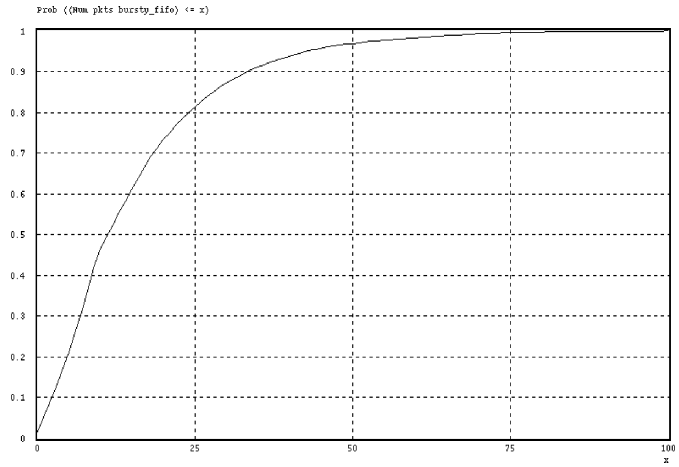

Figure 10. Empirical cdf for the bursty generator.

As noted, Figure 11 is the empirical probability mass function (pmf) of queue length for the first s-s generator. It has an extremely long tail (454), 50% of its mass is 1 or less (like Poisson), 95% of its mass is 152 or less, and 99% of its mass is 384 or less. Less than 1% of the mass falls between 385 and 454.

Simulation results (for this example) confirm that peak queue lengths for s-s traffic are 4.5 times larger than for bursty traffic, and 35 times larger than for Poisson traffic. S-s traffic has important implications for data and telecommunications traffic engineering.

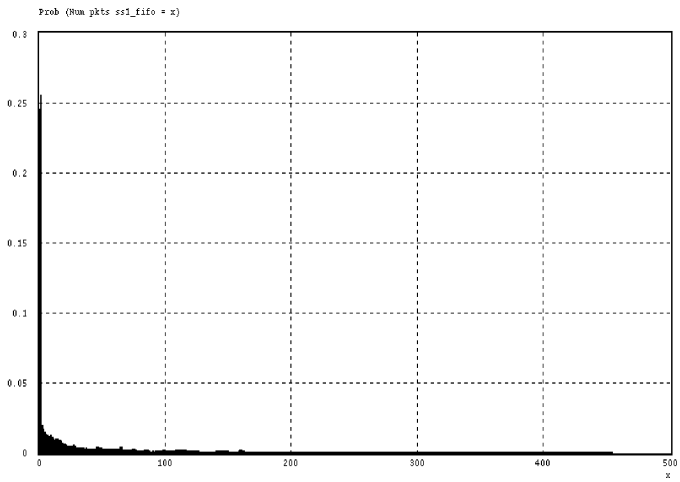

Figure 11. Empirical pmf for the first s-s generator.

In this example, buffers engineered to levels two times larger than predicted by a Poisson arrival process would overflow about 20% of the time with s-s traffic. Buffers engineered to levels two times larger than predicted by an aggressive bursty arrival process would overflow about 5% of the time. Buffers engineered to levels four times larger than predicted by the bursty arrival process would still overflow nearly 1% of the time.

7. REFERENCES

[1] Leland, W. *et al*. "On the self-similar nature of Ethernet traffic." *IEEE/ACM Trans. Networking.* Vol. 2. 1994.

[2] Paxson V. and S. Floyd. "Wide-area traffic: the failure of Poisson modeling". *Proc. SIGCOMM '94.* September 1994.

[3] Paxson, V. "Fast approximation of self-similar network traffic." *Lawrence Berkeley Laboratory Report LBL-36750.* April 1995.

[4] Ramaswami, V. and P. Wirth (*eds*.). *Proc. 15th Intl. Teletraffic Congress (ITC 15).* Washington DC. June 1997.

[5] Molnar, S. and A. Vidacs. "On modeling and shaping self-similar ATM traffic." *Proc. 15th Intl. Teletraffic Congress (ITC 15).* Washington DC. June 1997.

[6] Michiel, H. and L. Koen. "Teletraffic engineering in a broad-band era". *Proc. IEEE.* Vol. 85. No. 12. December 1997.

[7] Sahinoglu, Z. and S. Tekinay. "On multimedia networks: self-similar traffic and network performance". *IEEE Communic. Magazine.* January 1999.

[8] Schuler, C. "`fft_fgn`". Research Institute for Open Communication Systems, GMD FOKUS, Hardenbergplatz 2, D-10623 Berlin, Germany.

Name: Arnold Bragg

Address: Fujitsu Network Communications, Inc.
1000 Saint Albans Drive
Raleigh, North Carolina 27609

Phone: 919 790 2069

FAX: 919 790 3330

Email: arnold.bragg@fnc.fujitsu.com