PARALLEL INTERNET TRAFFIC SIMULATOR WITH SELF-SIMILAR SCALE-FREE NETWORK MODELS

RADU DOBRESCU, SEBASTIAN TARALUNGA, STEFAN MOCANU
"Politehnica" University of Bucharest, Faculty of Control and Computers,
313 Splaiul Independentei, Bucharest
ROMANIA
radud@isis.pub.ro
25-404.pdf

Abstract. The paper presents a platform, named Parallel Internet Traffic Simulator (PITS), which allows simulating Internet traffic and accurately replicating appropriate stochastic processes using scale-free models with self-similar topology. The experiments compare simulation of large-size scale-free networks when using different number of CPU's running in parallel in the platform cluster and note the differences in simulation time as well as some characteristics related to the efficiency of the simulation distribution is possible and if there are limitations in single-CPU simulation. We consider that PITS shows interesting properties when compared to other traffic generators and therefore it represents a real simulation tool for controlled traffic generation over real networks which allows to accurately analyze and evaluate the performance of network, transport, and application-level protocols.

Keywords: scale-free networks, simulation, parallel and distributed processing, self-similar traffic.

1 Introduction

To model a distributed network environment like the Internet, it is necessary to integrate data collected from multiple points in a network in order to get a complete picture of network-wide view of the traffic. Knowledge of dynamic characteristics is essential to network management (e.g., detection of failures/congestion, provisioning, and traffic engineering like QoS routing or server selections). However, because of a huge scale and access rights, it is expensive (sometime impossible) to measure such characteristics directly. To solve this, methods and tools for inferencing of unobservable network performance characteristics are used in large scale networking environment. The main previous results refer to the so called network tomography [1]. Network tomography can be regarded as a statistical inverse problem that includes two typical forms: inference of network-internal characteristics based on end-to-end path measurements [2], [3] and end-to-end flow behavior inference from aggregated flows [4], [5].
A model where inference based on self similarity and fractal behavior can be applied is the scale free network. Scale-free networks are complex networks in which some nodes are very well connected while most nodes have a very small number of connections. An important characteristic of scale-free networks is that they are size independent, that is they preserve the same characteristics regardless of the network size N. Scale-free networks have a degree distribution that follows a power relationship, $P(k) = k^{(-\lambda)}$, where the coefficient λ may vary approximately from 2 to 3 for most real networks. Many real networks have a scale-free degree distribution, including the Internet [6].
Simulation of scale-free networks is necessary in order to study their characteristics. However, large-scale networks are difficult to simulate due to the hefty requirements imposed on CPU and memory. Thus a distributed approach to simulation can be useful particularly for situations where a single-processor is not enough.

2 Scale-free topology

All real-life networks are finite so can be characterized by the degree of connectivity of the associated graph. But the degree distribution does not characterize the graph or ensemble in full; other quantities, such as degree-degree correlation and spatial correlations are also useful. Several models have been presented for the evolution of scale-free networks, each of which may lead to a different ensemble. The first suggestion was the *preferential attachment* model by Barabasi and Albert, which came to be known as the "Barabasi-Albert (BA)" model. Several variants have been suggested to this model. One of them known as the "Molloy-Reed

construction" [7], which ignores the evolution and assumes only the degree distribution and no correlations between nodes, will be considered in the following. Thus, the site reached by following a link is independent of the origin. This means that a new node will more probably attach to those nodes that are already very well connected, i.e. they have a large number of connections with other nodes from the network. Poor connected nodes, on the other hand, have smaller chances of getting new connections (see fig.1).

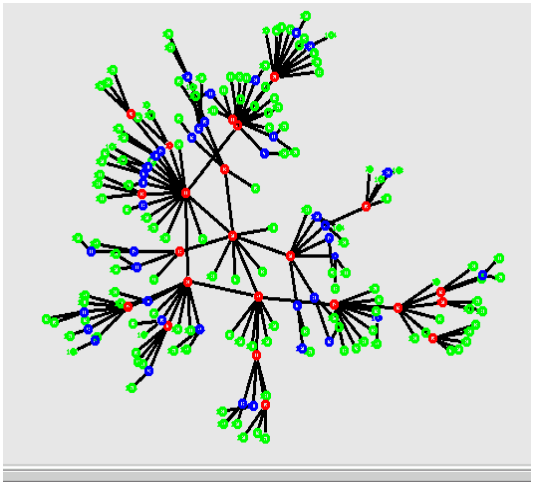

Fig. 1. Graphic representation of the generated network for 200 network nodes and λ=2.35

Besides following the repartition law mentioned above, some other restrictions (for example those related to cycles and long chains) had to be applied in order to make the generated model more realistic and similar to the Internet. Another obvious restriction is the lack of isolated components (see fig.2).

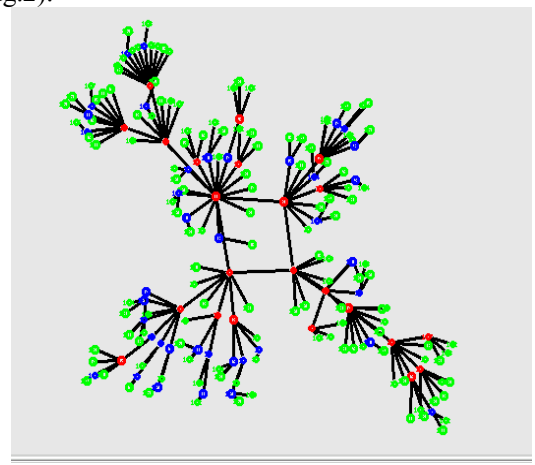

Fig. 2. Graphic representation of the generated network for 200 network nodes and λ=2.85

A more subtle restriction is related to the TTL (Time-to-living) which is a way to avoid routing loops in a real Internet. This translates in a restriction for our topology – there can be no more that 30 nodes to get from any node to any other node. Another restriction is that the generated network will also have redundant paths, multiple possible routes between nodes.
The algorithm used for the generation of the scale-free network topology is generating networks with a cyclical degree that can be controlled, in our case, approximately 4% of the added nodes form a cycle. One more restriction is that we try to avoid long-line type of scale-free networks – a succession of several interconnected nodes – structure that does not have a real-life Internet equivalent.

3. Self-similarity in traffic

3.1 Self-similar processes

In 1965, Benoit Mandelbrot introduces the *self-similar* (SS) processes (another wide-spread name is *fractal* processes). This is a model which finally captures the power-law behavior described above [8]. Referring directly to the increment process $X_{s,t} = X_t - X_s$, he defines stochastic self-similarity as:

$$X_{t0,t0+rt} = r^H X_{t0,\ t0+t}, \qquad \forall t_0,\ t,\ \forall r>0 \qquad (1)$$

Mandelbrot constructs his SS process (*fractional Brownian motion*, fBm) starting with two properties of the usual Brownian motion (Bm):

- Bm has independent increments.
- Bm is self-similar with Hurst parameter H= 0.5

Denoting Bm as B(t) and fBm as $B_H(t)$, here is a simplified version of Mandelbrot's definition of the fBm: $B_H(0) = 0$, $H \in [0,1]$ and

$$B_H(t) = \frac{1}{\Gamma(H+0.5)}\left\{\int_{-\infty}^{0}\left[(t-s)^{H-1/2}-(-s)^{H-1/2}\right]dB(s)+\int_{0}^{t}(t-s)^{H-1/2}dB(s)\right\} \qquad (2)$$

The integral in eq.(2) points to another important property of SS processes: *long-range dependence* (LRD). The SS process is called LRD if there are constants $\alpha \in (0,1)$ and $C>0$ such that

$$\frac{\rho(k)}{Ck^{-\alpha}} \xrightarrow{k\to\infty} 1 \qquad (3)$$

which we write simply $\rho(k) \sim k^{-\alpha}$ where $\rho(k)$ is the autocorrelation of lag k. When represented in logarithmic coordinates, eq.(3) is called the *correlogram* of the process, and has an asymptote of slope $-\alpha$. This property can be used to estimate α either graphically or through some analytic fitting procedure (e.g. least squares). It is to note that SS and LRD are not overlapping properties. There are SS processes which are not LRD (the simple Bm is

an example), and, conversely, there are LRD processes which are not SS. However, the fBm with H>0.5 is both SS and LRD.

3.2 Self-similarity in network traffic

In a landmark paper [9] from 1993, Leland *et al*. report the discovery of self-similarity in local area network (LAN) traffic, more precisely Ethernet traffic. To be precise, they studied two separate processes: number of bytes arriving per time interval and number of IP packets. Since packets can vary widely in size (from 40 byte to 1500 byte in the case of Ethernet), it is in principle conceivable for the two processes to be quite different. However, the paper shows that they are both SS, with H estimated in the range (0.6 – 0.9) according to the changing network conditions. As a main critique of this approach, we note that all methods used in [9] (and in numerous papers that followed) detect and estimate LRD rather than SS. "Self-similarity" (actually LRD) has since been reported in various types of data traffic: LAN, WAN, Variable-Bit-Rate video, SS7 control, HTTP etc. (see [10] for an extensive bibliography). An overview of the above studies shows link speeds ranging from 10 Mbps (Ethernet) to 622 Mbps (OC-12), link type being "access" (typically connecting a university campus or research lab to an Internet Service Provider), average bandwidths between 1.4 and 42 Mbps, minimum time-scale of 1ms and at most 6 orders of magnitude for time-scales. Lack of access to high-speed, high-aggregation links, and lack of devices capable of measuring such links have until recently prevented similar studies from being performed on Internet *backbone* links. Actually, some researches have claimed that aggregating Internet traffic causes convergence to a Poisson limit [11]. For reasons presented in the next section, based on the remarks that on *shorter time scales* effects due to the network transport protocols are believed to dominate traffic correlations, although this property has not been definitively established and on *longer time scales*, effects such as diurnal traffic load patterns become significant, we disagree.

3.3 Relevance of self-similarity in network operation

In virtually all areas of communications networking it is of great importance to understand the characteristics of packet traffic. In a packet-switched environment, there are no resources reserved for a connection, which creates the possibility of packets arriving at a node at a rate higher than the processing rate. Packets arriving on any of a number of input ports can be destined for the same output port. Although there could be in general many operations to be performed, they are broadly divided into two classes: *serialization* the process of transmitting the bits out on the link) and *internal processing* (i.e. everything else that happens inside the node - classification, routing table and access list look-ups, policing, shaping etc). Serialization delays are deterministic and given by the speed of the output link. Internal processing delays are essentially random but advanced algorithms and data structures are doing in general a good job of maintaining overall deterministic performance. As a consequence, the only truly random part of the delay is the time a packet spends waiting for service. The internal workings of a node are relatively well understood (by either analysis or direct measurement) and controllable. In contrast, the (incoming) traffic flows are neither completely understood nor easily controllable. The focus is therefore on accurately modeling these flows and predicting their impact on network performance.

It is has been shown time and again that the high variability associated with LRD and SS processes can greatly deteriorate network performance, creating for instance queuing delays orders of magnitude higher than those predicted by traditional models. We make this point once again with the plot shown in fig.3: the average queue length created in a queuing simulation by the actual packet trace is widely divergent from the average predicted by a simple queueing model (M/D/1).

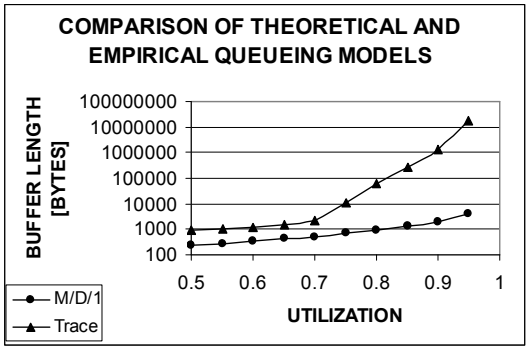

Fig. 3 Comparison between the queue lengths

The notation is easily explained: "M" stands for the (unique) input process, which is assumed Markovian, i.e. a Poisson process; "D" stands for the service, assumed Deterministic; and "1" means that only one facility is providing service (no parallel service). In view of our simple model of a node discussed above, hypotheses "D" and "1" seem reasonable, so we would like to identify "M"

as the cause of the massive discrepancy. The question then is: Is there an "fBm/D/1" queuing model? The answer "almost" will be argued in the next section.

4 Proposed Internet model

The generated topology consists of three types of nodes: Routers, defined as nodes with one or several links. Routers do not initiate traffic and do not accept connections. Routers can be one of the following types: routers that connect primarily customers, routers that connect primarily servers and routers that connect primarily other routers. Routers that connect primarily customers have hundreds or thousands of type one connections (leaf nodes) and a reduced number of connections to other routers.

Routers that connect primarily servers have a reduced number of connections to servers in the order of tenths and reduced number (2 or 3) connections to other routers. Routers that connect primarily routers have a number in the order of tenths of connections to other routers and do not have connections to neither servers nor customers.

Servers are defined as nodes with one connection but sometimes could have two or even three connections. Servers only accept traffic connections but do not initiate traffic.

Customers (end-users) defined as nodes that have only one connection, very seldom two connections. Customers initiate traffic connections towards servers at random moments but usually in a time succession. For our proposed model, we chose a 20:80 customers to servers ratio.

4.1 Scale-free network design algorithm

We designed and implemented an algorithm that generates those subsets of the scale-free networks that are close to a real computer network such as the Internet. Our application is able to handle very large collections of nodes, to control the generation of network cycles, and the number of isolated nodes. The application was written in Python being, as such, portable. It runs very fast on a decent machine (less than 5 minutes for 100.000 nodes model).

Network generation algorithm:

1. set node_count and λ
2. compute the optimal number of nodes per degree
3. create manually a small network of 3 nodes
4. for each node from 4 to node_count
    4.1. call add_node procedure
    4.2. while adding was not successful
    4.2.1. call recompute procedure
    4.2.2. call add_node procedure
5. save network description file

add_node procedure

1. according to the preferential attachment, compute the degree of the parent node
2. if degree could be chosen then exit procedure
3. compute the number of links that the new node shall establish with descendants of its future parent, according to copy model
4. chose randomly a parent from the nodes having the degree as computed above
5. compute the descendant_list, the list of descendants of the newly chosen parent
6. create the new node and links
7. for each descendant of the descendant_list
    7.1. create the corresponding links
8. exit procedure with success code

recompute procedure

1. for each degree category
    1.1. calculate the factor needed to increase the optimal count of nodes per degree
    1.2. if necessary increase the optimal number of nodes per degree
2. exit procedure

The algorithm starts with a manually created network of several nodes, then using preferential attachment and growth algorithms, new nodes are added. We introduced an original component, the computation in advance of the number of nodes on each degree-level. The preferential attachment rule is followed by obeying to the restriction of having the optimal number of nodes per degree.

We noticed that the power law is difficult to follow while the network size is growing, as a result we calculate again the optimal number of nodes per degree-level at given points in the algorithm. This is necessary because the bigger the network the higher the chance that a new node will be attached only to some specific very-connected nodes. In a real network such as Internet this will not happen.

If only the preferential and growth algorithms are followed, then the graph will have no cycles, which is not realistic, therefore we introduced a component from the "copy model" for graph generation in order to make the network graph include cyclical components.

This component ensures that each new node is also attached to some of its parent-node descendants using a calibration method. The calibration method

computes the number of additional links that a new node must have with the descendants of its parent. This number depends on how well-connected is the parent and it also includes a random component.
The output of the application is a network description file that can be used by several tools like for instance a tool to display the power law. This file is stored using a special format needed in order to reduce the amount of disk writes.

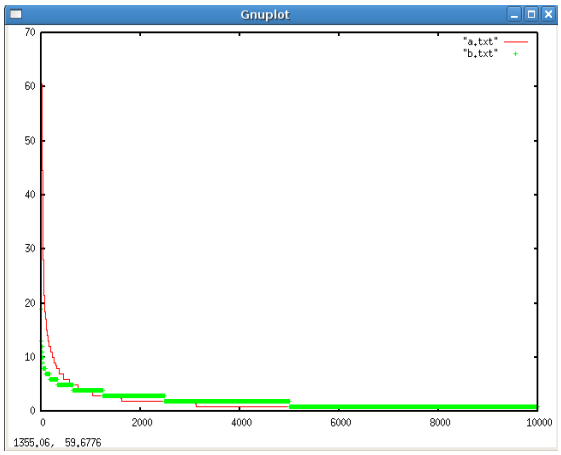

Fig. 4. Graphic representation of the distribution law for a scale-free network model and for a randomized network with 10000 nodes

In Fig. 3 we compare an almost random network distribution law and a free-scale distribution law. On the Y axis we represent the number of connections and on the X axis the number of nodes having this number of connections. It was impossible to obtain a completely random network given the limitations imposed by the Internet model. In this paper we further describe only the scale-free network model since we think that such a model can lead to a better balancing based on the preferential attachment mechanism.

4.2 Traffic generation

Traffic generation is an essential part of the simulation as such, we decided to initiate randomly between 1 and 3 simultaneous traffic connections from "customer" nodes and for the sake of simplicity we used ftp sessions to randomly chosen destination servers. We also decided that the links connecting routers should have higher speeds than lines connecting customers to routers, for example - server-router 1 Gbps, client-router 10 Mbps, router-router 10, 100 Mbps or 1Gbps depending on the type of router. The code generated respecting these two conditions is added to the network description file, being ready to be processed by the simulator.

4.3 Single-CPU simulation

We used a modular approach that allows us to later reuse components for different parts of the simulation. For example, the same network model generated by the initial script can be used for both single-CPU and distributed simulations, allowing a comparison between the two types of simulation.
Standalone simulations were run under University of California Berkeley's NS2 network simulator. NS2 (The Network Simulator) is a very complex open source discrete event simulator targeted at networking research [12]. The simulator is actually an OTcl interpreter, which also makes it quite easy to use.
We noticed that on a single machine, as network size increases, very soon we hit the limit of the network sizes that can be simulated due to resources limitations mostly memory but also high CPU load. In case of small size network models, such as with a few nodes, simulations can be run on a single machine. One of the models generated with a number of 10000 nodes and a lot of traffic connections could not be simulated on an AMD Athlon(tm) 64 Processor 3200+ with only 512 Megabytes of RAM available.
The results provided by NS2 were visualised using the nam (network animator) software package. The topology generator gives different colours to different type of nodes: server, client, router. Details about the networking traffic through each network node are parsed from the simulator output.

4.4 Multi-CPU simulations

Unfortunately NS was not designed to run on parallel machines. Only in the NS version 3, now under alpha development, there are discussions about distributed processing. The main obstacle in running ns in a distributed/parallel environment is related to the description of objects in the simulation.
As such we ran our distributed simulations under Georgia Tech's extension to NS2, *pdns* [13], which uses syntax close to that of NS2, the main differences being a number of extensions needed for the parallelization so that different instances of pdns can communicate with each other and create the global image of the network to be simulated.
Each simulator running on different nodes needs to know the status of other simulators. Furthermore, if we try to split the network description file into separate files and run each of these in separate simulation contexts, we need to find a way to communicate parameters between the simulation nodes.

The simulation process consists of a number of steps, of which, defining network nodes links, queue and topology must take into consideration the fact that other nodes may not reside under the same simulator. All simulations are running 40 seconds of simulated traffic scenarios.

4.5 Cluster description

In order to create a parallel/distributed environment we have built a cluster using commodity hardware and running Linux as operating system [14]. We have written and tested applications using PVM (Parallel Virtual Machine) which is a framework consisting of a number of software packages that accomplish the task of creating a single machine that spans across multiple CPU's, by using the network inter-connection and a specific library [15]. Another framework that can be used to run applications in a distributed manner is MPI - Message Passing Interface [16].

Our cluster consists of a "head" machine and a number of six cluster nodes. The "head" provides all services for the cluster nodes – IP allocation, booting services, File System (NFS) for storage of data, facilities for updating, managing and controlling the images used by the cluster nodes as well as access to the cluster. The "head" computer provides an image for the operating system that is loaded by each of the cluster nodes since the cluster nodes do not have their own storage media. As this image resides in the memory of each cluster node, we took special steps to reduce the size of this image and to make most of the memory available to the running processes.

The application partition is mounted read-only while the partition where data is stored is mounted read-write and accessible to the users on all machines in a similar manner providing transparent access to user data. In order to access the cluster, users must connect to a virtual server located on a head machine. This virtual server can also act as a node in the cluster when extra computation power is needed.

4.6 Network Splitting

In order to use PDNS simulation, we needed to split the network into several quasi-independent sub-networks [17]. Each instance of PDNS handles a specific sub-network, thus the dependencies between them need to be minimal, i.e. there shall be as few as possible links between nodes located in different sub-networks.

We chose to have a federated simulation approach. We designed and implemented a federalization algorithm in order to split the original generated network into several small ones. The algorithm that generates n federative components chooses the most n linked nodes, assigned them to an empty federation and starts a procedure similar to the breadth-first search algorithm. Each node is marked as being owned by a federation.

The pdns script generator takes as input the generated network description and the generated federations, respectively. Depending on the connectivity of nodes, they are assigned the role of routers, servers, end-users and corresponding traffic scenario are associated with them. We also used a different approach to partitioning a ns script into several *pdns* scripts by using *autopart* [18], a simulation partitioning tool based on the graph partitioning package called METIS [19]

5 Simulation results

5.1. Testing SFN models parallel processing

We have decided to run simulations for 40 seconds of traffic for a scale-free network model with 10000 nodes. At such a scale, a one-node processing is impossible because the cluster node runs out of memory. Still, to get valid results we had run the simulation on a much more powerful machine with plenty of memory and virtual memory.

We chose two different scenarios, one with a moderate network traffic and another scenario with a heavy network traffic. Each scenario was simulated five times under similar load conditions, using two to six CPU's and we noted the time used for the actual simulation (in seconds, see Table 1).

Table 1. Scale-free network model with 10000 nodes and moderate network traffic (40 seconds)

	Number of cluster nodes used					
	1	2	3	4	5	6
Run 1	failed	68	46	32	29	40
Run 2	failed	68	41	31	30	37
Run 3	failed	67	43	32	33	31
Run 4	failed	68	45	30	29	43
Run 5	failed	67	45	32	31	40
Average	135	67.6	44	31.4	30.4	38.2

For the first scenario we noted that there is a point where adding more nodes in the simulation does not help but rather increases the simulation time. In this scenario, the optimum number of nodes is 5.

The second scenario requires much more resources as can be seen from the single-processor simulation which again failed on the cluster nodes but was

successful on a more powerful machine, although it takes a longer time. Also in this simulation we see that adding more nodes (in our case more than 4) the simulation process is slower (see Table 2).

Table 2 Scale-free network model with 10000 nodes and heavy network traffic (40 seconds)

	Number of cluster nodes used					
	1	2	3	4	5	6
Run 1	failed	319	338	135	173	165
Run 2	failed	343	357	140	176	171
Run 3	failed	347	351	134	177	166
Run 4	failed	316	347	139	177	165
Run 5	failed	308	320	138	178	163
Average	1139	326.6	342.6	137.2	176.2	166

Another observation is that the 2-CPU simulation is actually faster than the 3-CPU simulation, although the optimal number of nodes is not 2.

5.2. Testing self-similarity in traffic

The method used for self-similarity testing is named *Rescaled Range Statistics* [20], and it is based on long range dependencies. The analyzed data describes the investigated traffic as time series. To find long range dependencies, measured data in the simulation model has to be evaluated in different time-scales. The same sample of cell numbers versus time has to be cut into lags. Cuts have to be done several times, varying the number and length of lags. The length of the lags ***n*** is the time base for further investigations and also it is the number of observations in the lag. For each **n**, a number of lags are selected randomly. This number of lags must be the same for all values of **n**. For selected lags, two parameters are calculated:

- $R(n) = max(0, W_1, W_2,..., W_n) - min(0, W_1, W_2,..., W_n)$ with $W_i = (X_1 + X_2 + ... + X_i) - i\overline{X}(n)$ is the *sample range* of the lag.
- $S(n)$ is the *variance* of the set $\{X_1 + X_2 +... + Xn\}$ of one lag.

For short range dependent sets of observations the expected value $E[R(n)/S(n)]$ is about $c_0 n^{1/2}$. Contrary to that, for long range dependent sets of observations $E[R(n)/S(n)]$ is about $c_0 n^H$ with $0.5 < H < 1$. H is the Hurst parameter and c_0 is a constant of minor importance [21].

The tests were executed on simulated video traffic: two different video encoders (H.261, JPEG) were used to encode a video stream from a camera. For all experiments, the predefined frame rate and transmission rate have been selected for the highest quality transmission that is 30 fps frame rate and 3072 kbps transmission rate. The lag size ranges from 5 seconds to a size where at least 7 non overlapping lags could be built from the measured data. The streams were multiplexed in order to be processed in parallel on the cluster. In theory, when sequences of the same *H* are multiplexed, the same *H* should result for the multiplexed sequence. We applied this as follows: each data set of 8 hours length was cut into 10 pieces first and then these 10 streams were multiplexed. The cut of the original stream results in streams of approximately the same *H*. On the multiplexed stream, the same procedure for self-similarity testing as for the original data was applied. After that, the same has been done cutting the original data into 100 pieces, simulating the multiplex of 100 cell streams. Table 3 summarizes all experiment statistics and results. One can observe that the *Hurst* parameters of the multiplexed streams are more or less in the same range as for the single streams.

Table 3 Simulation results for self-similarity tests

Encoder	Measure time (s)	*H* values		
		single stream	10 mux. streams	100 mux. streams
H261	50094	0.685	0.717	0.705
JPEG	28782	0.842	0.857	0.862

6 Conclusions

Although the traffic processes in high-speed Internet links exhibit *asymptotic* self-similarity, their correlation structure at short time-scales makes their modeling as *exact* self-similar processes (like the fractional Brownian motion) inaccurate. This causes existing queueing results to be poor predictors of network performance; nevertheless, based on statistical and queueing analysis of data traces from high-speed Internet links, we conclude that Internet traffic retains its self-similar properties even under high aggregation. Running *pdns* is more efficient than running NS2 especially on large size network models where sometimes *pdns* is the only solution. However, there are limitations in the number of cluster nodes that could process a given network model since more nodes are used, more traffic links between different cluster nodes are to be simulated and therefore more time is spent on inter-processor communication.

It is very important to split the network model correctly into smaller sub networks (federations) since there is a trade-off between the degree of

separation and federation balancing - the more separated the sub networks are, the more unbalanced they become.

We assume that the results observed in scenario number two where the 2-CPU simulation is actually faster than the 3-CPU simulation although not being the optimal number of cluster nodes, is related to the federalization algorithm which failed to reach an optimal solution for the 3-CPU scenario thus the processing times higher than 2-CPU.

The results of the tests on the traffic obtained by Hurst parameter examination shown that self-similarity manifests in slowly decaying variances and a slowly decaying autocorrelation function. The most serious consequence of self-similar traffic concerns the size of bursts: the burst lengths on different time scales become predictable.

Further work is necessary to confirm the results observed, processing on more than six processors and the study of other federalization algorithms. We are currently developing a program that can be used to study the efficiency of the parallel processing and help us understand the interlocking mechanisms and further help us improve the efficiency of the simulation.

ACKNOWLEDGEMENTS

This work was partially supported by the Romanian Ministry of Education and Research under Grant CNCSIS No. I 121/2007

References:

[1] Y. Vardi, Network tomography: estimating source-destination traffic intensities from link data. *J. Amer. Stat. Assoc*., 1996, p.365-377

[2] M. Coates and R. Nowak, Network delay distribution inference from end-to-end unicast measurement, *Proc. IEEE Int. Conf. Acoust., Speech and Signal Proc*., May 2001.

[3] A. Bestavros, K. Harfoush and J. Byers, Robust identification of shared losses using end-to-end unicast probes, *Proc. IEEE Int. Conf. Network Protocols*, Osaka, Japan, Nov.2000.

[4] J. Cao, S. Van der Wiel, B. Yu and Z. Zhu, A scalable method for estimating network traffic matrices, *Bell Labs Tech. Report*, 2000**.**

[5] J. Cao, D. Davis, S. Wiel and B. Yu, Time-varying network tomography : Router link data, *The Journal of American Statistics Association*, **95**(452), 2000, p. 1063–1075

[6] S. da Silva and P. R. Guardieiro, Studying Traffic Engineering in Next Generation Internet, *WSEAS Transactions on Systems*, Issue 1, Volume 2, 2003, p.107-112

[7] M, Molloy and B. Reed, "The size of the giant component of a random graph with a given degree sequence", *Combin. Probab. Comput.* **7**, 1998, p. 295–305

[8] B. B. Mandelbrot and J. W. Van Ness, "Fractional Brownian Motions, Fractional Noises and Application", *SIAM Review*, Vol.10, No.4, October 1968

[9] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, On the self-similar nature of Ethernet Traffic, *Proceedings of ACM SIGCOMM'93*, 1993, p.183-193

[10] R. Dobrescu, M. Dobrescu and St. Mocanu, Using Self Similarity To Model Network Traffic, *WSEAS Transactions on Computers*, Issue 6, Vol3, dec.2004, p. 1752-1757

[11] B. H. He, H.L. Zhang, W.Zhang and M.Hu, Designing a Framework for Worm Detection Based on Similarity, *WSEAS Transactions on Comm.,* Issue 8, Volume 4, 2005, p.570-578

[12] J. Chung, *NS by example*, www.isi.edu/

[13] http://www.cc.gatech.edu/computing/ pdns

[14] S. Mocanu and S. Taralunga, Cluster based simulations of Scale-Free Networks Immunization Strategies,, *WSEAS Transactions on Computers*, Issue 2, vol. 6, 2007, p. 268-275

[15] A. Grama, A. Gupta, G. Karpys and V. Kumar, *Introduction to Parallel Computing,* Prentice Hall, 2003

[16] M. M. Balas, V. E. Balas and L. R. Szantho, The Fuzzy-Interpolative Concept Applied In Soft Computing,. *Proc. of the 7th WSEAS Int. Conf. on Automation and Information,* 2006, pag.95-100

[17] B. Wilkinson and M.A Pearson, *Parallel Programming,* Prentice Hall, 2005

[18] G. Riley, M. Ammar, R.Fujimoto, A. Park, K. Perumalla and D. Xu, A Federated Approach to Distributed Network Simulation, *ACM Trans. on Modeling and Computer Simulation* (TOMACS), Vol. 14(2), April 2004

[19] METIS - Serial Graph Partitioning and Fill-reducing Matrix, glaros.dtc.umn.edu/

[20] M. Dobrescu and S. Mocanu, Resource management for real time parallel processing in a distributed system. *WSEAS Transactions on Computers*, Issue 3, vol.2, 2004, p.732-737

[21] R. Dobrescu, S. Taralunga and S. Mocanu, Web Traffic Simulation With Scale-Free Network Models, Proceedings of the 7 th WSEAS Int. Conference on Applied Informatics and Communications, AIC 07, pp.277-282