# WEB TRAFFIC SIMULATION WITH SCALE-FREE NETWORK MODELS

RADU DOBRESCU, SEBASTIAN TARALUNGA, STEFAN MOCANU
"Politehnica" University of Bucharest, Faculty of Control and Computers,
313 Spl. Independenței, Bucharest
ROMANIA

*Abstract*. The paper proposes a scale-free model of an Internet and we want to see whether simulation of large-size scale-free networks is possible and if there are limitations in single-CPU simulation. We later simulate the same model using a distributed environment by dividing the task of running the simulation on a number of CPU's running in parallel in a cluster and note the differences in simulation time as well as some characteristics related to the efficiency of the simulation distribution.

*Keywords*: scale-free networks, simulation, parallel and distributed processing.

## 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. 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 $\gamma$ may vary approximately from 2 to 3 for most real networks. Many real networks have a scale-free degree distribution, including the Internet.

Simulation of scale-free networks is necessary in order to study their characteristics like fault-tolerance and resistance to random attacks. 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 large-scale network simulations, where a single-processor is not enough.

## 2 Scale-free topology

In order to simulate an Internet-like network we used an algorithm based on the concept of "preferential attachment". 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).

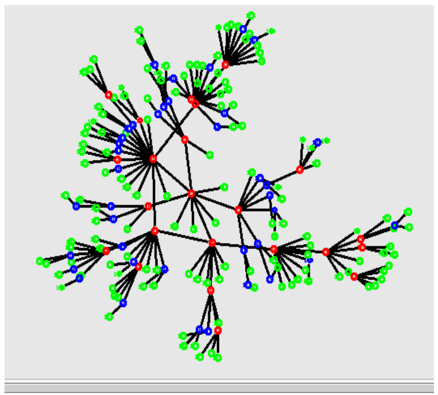

Fig. 1. Graphic representation of the generated network for 200 network nodes and $\lambda$=2.35

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).

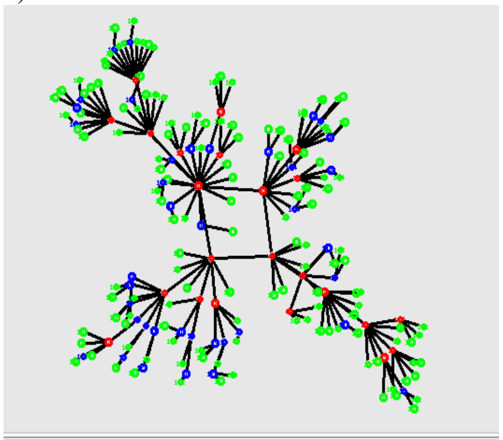

Fig. 2. Graphic representation of the generated network for 200 network nodes and λ=2.85

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 subtle restriction is that the generated network will also have redundant paths, multiple possible routes between nodes. In other words, the Internet model topology should not "look" like a tree, but should rather have numerous cycles.

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, so our algorithm makes sure such a model is not generated.

## 3 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.

### 3.1 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).

<u>Network generation algorithm:</u>

```
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 the 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.

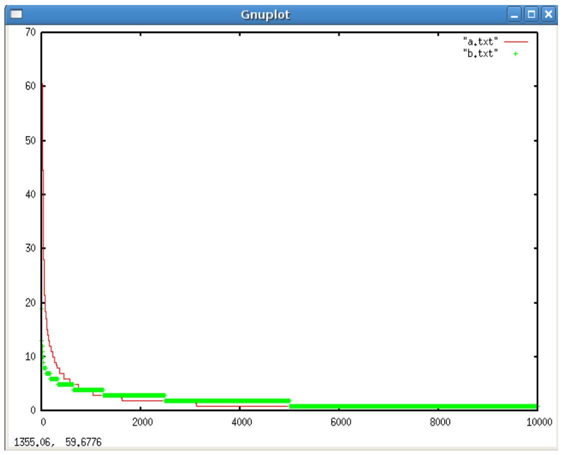

Fig. 3. Graphic representation of the distribution law for a scale-free network model and for a randomized network with 10000 nodes

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 an 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.

## 3.2 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.

## 3.3 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 [1]. 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.

### 3.4 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* [2], which uses a 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.

### 3.5 Cluster description

In order to create a parallel/distributed environment we have built a cluster using commodity hardware and running Linux as operating system [3]. The cluster can run applications using a parallelization environment.
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 [4]. Applications must be compiled using this specific library in order to permit communication. Another framework that can be used to run applications in a distributed manner is MPI (Message Passing Interface). MPI specifies a library for communication between tasks.
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. We were able to reduce this image to 16 megabytes by moving different parts of a running Debian Linux system over network file systems, leaving on the image only those components needed for booting and controlling the cluster nodes.
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.

### 3.6 Network Splitting

In order to use PDNS simulation, we needed to split the network into several quasi-independent sub-networks [5]. 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* [6], a simulation partitioning tool developed by Donghua Xu from Georgia Institute of Technology. This tool is based on the graph partitioning package called METIS [7]

*Autopart* takes an NS2 script and creates a number of pdns scripts that are ready to run in parallel on a number of machines, attempting to make the best trade-off between look-ahead, load balancing and communication overhead in the partitioning process, resulting in the best performance when being run by PDNS.

## 4 Simulation results
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).

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 |

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 |

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.
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 Conclusion
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.
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 Grants No. 1467A/2005 and No. 1357/2005-Postdoctoral.

*References:*
[1] J. Chung, *NS by example*, www.isi.edu/
[2] http://www.cc.gatech.edu/computing/ pdns/
[3] S. Mocanu and S. Taralunga, Cluster based simulations of Scale-Free Networks Immunization Strategies,, In: *WSEAS Transactions on Computers*, Issue 2, vol. 6, 2007, p. 268
[4] A. Grama, A. Gupta, G. Karpys and V. Kumar, *Introduction to Parallel Computing,* Prentice Hall, 2003
[5] B. Wilkinson, and M.A Pearson, *Parallel Programming,* Prentice Hall, 2005
[6] 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
[7] METIS - Serial Graph Partitioning and Fill-reducing Matrix, glaros.dtc.umn.edu/