Building Blocks of a Simulation Environment of the OSI Network Layer of Packet-Switching Networks

A. Gerisch
Dept. of Math. and Comp. Sc.
University of Halle
06099 Halle (Saale), Germany
gerisch@mathematik.uni-halle.de

A.T. Lawniczak
Dept. of Math. and Stat.
University of Guelph
Guelph, Ont N1G 2W1, Canada
alawnicz@uoguelph.ca

B. Di Stefano
Nuptek Systems, Ltd.
Toronto, Ont M5R 3M6
Canada
nuptek@sympatico.ca

Abstract

This paper describes the main building blocks of a simulation environment of the OSI Network Layer of packet-switching networks. The need for such a tool is presented and pitfalls of previous solutions are described. Remedies provided by the most recent solution are discussed. Architecture, organisation, and architectural decisions are explained.

Keywords: *Packet-switching networks; OSI Network Layer; simulation environment; Netzwerk-1.*

1. INTRODUCTION

The global Internet, wireless communication systems, ad-hoc networks, and sensors networks are some of the examples of data networks of packet-switching type. These packet-switching networks (PSNs) have experienced unprecedented growth that is going to continue in the foreseeable future.

The dynamics of PSNs is highly heterogeneous and unpredictable. For instance, the dynamics of flow and congestion in PSNs are influenced by many different independent variables. At infrastructure level some of the most important variables are connection topologies and routing strategies, while at business level the most significant independent variables include frequent and, often, sudden changes in quantity, quality, and type of service demanded by the user. The Internet is characterised by simultaneous coexistence of overloaded hotspots and underutilised resources.

Many small scale models and experiments tried by several researchers do not allow to scale up and do not accurately model the effect of the propagation of such phenomena as denial of service attacks or the spread of computer viruses on the global Internet. There is a need of simulation environments that can easily scale from small networks of the order of up to 10^3 nodes to large networks of the order of up to 10^5-10^6 nodes. Even if limitations imposed by available computing resources forces the simulation of small networks, the development of simulation tools that can scale up is warranted by the forecasted evolution of computer speeds, memory size, and by the availability of grid computers.

The evolution of PSNs and improvements in their design depend strongly on the ability to predict systems performance using analytical and simulation methods. For instance, the dynamics of large PSNs might significantly influence the performance of proposed new protocols and routing schemes. In general, protocol independent simulation tools are needed to better understand which characteristics the protocols should have.

One of the authors has had extensive experience in this research and has supervised the development of four different generations of simulation environments. Results obtained using the first generation are reported in [1, 2] and those using the second and third generation in [3, 4]. Some of the results obtained so far using the fourth generation software are reported in [4, 5, 6, 7].

This paper focuses on the description of software building blocks used to set up *digital laboratories* to perform large-scale simulation and to run *digital experiments* to test various hypothesis. Design decisions have been made in such a way as to allow for the use of existing public domain or readily available standard software as well as for portability across different operating systems and across different hardware platforms.

2. ARCHITECTURE OF THE SOFTWARE ENVIRONMENT

By *software environment* we mean the set of software modules and tools that make up the universe of the software used for conducting our research. Previously developed simulation environments [1, 2, 3], consisted of a MS Windows compatible simulator program with real-time GUIs and a set of Matlab scripts used to process the raw data produced during the simulation for the purpose of evaluation. However, the intrinsic slow response of computer display technology and the large amount of data to be generated, made it clear that to simulate relatively large size PSNs required to eliminate the real-time GUIs.

In the experiment to study flow and congestion of a PSN, the simulation model consists of a number of switching nodes which are interconnected by bidirectional communication links and its purpose is to transmit packets from nodes of origin to destination nodes. A packet is a capsule which carries the information payload and some additional information related to the internal structure of the network. One or more packets make up a message which a user wants to transmit over the network from a node of origin to a destination node. The simplifying hypothesis of messages consisting of one packet only allows to neglect issues such as message reconstruction at the destination node, an issue that does not contribute to the study of flow and congestion over the network. Each switching node can perform two functions: that of a host (source and destination of packets) and that of a router (message processor, that can store and forward packets). To simulate a wide variety of traffic situations, packets are created randomly at each node and independently from the other nodes, with a probability called *source load*. An incoming queue and outgoing queues are maintained by each node to store packets on this node. The outgoing queue can be just one per node or one per neighbouring node (that is per link connected to the node). With each delivered packet we associate the following numbers:

- the number of hops of the packet (the number of links traversed between source and destination),
- the delay time of the packet (time of delivery at destination minus time of creation at source), and
- the average speed of the packet (the number of hops divided by its delay time).

As described in [5], a number of sums and averages are calculated at every time step. Clearly, an attempt to display in real-time these sums and averages would result in a very slowly running program. Moreover, every experiment has to be repeated for a range of source loads. The modelling tool of the fourth generation, Netzwerk-1 [8], does not include any direct graphical user interface support. Rather it has a text based interface which can also be used to execute simple scripts. Selected simulation and statistical data can be saved in text files (MATLAB format) for further evaluation and processing. The choice of the text based interface in combination with script files has proven to be an efficient and very convenient way of running the large number of simulation experiments.

The abstraction used to model a PSN depends on which aspect of the system one wants to study. Research on the effects of network connection topology and routing algorithm on flow and congestion, as is currently our main interest, can be carried out if only the Network Layer of the OSI Reference Model is simulated. Research on the effects of the nature of data, e.g. studies on *mice* (small data bursts, e.g. short text messages) and *elephants* (large amount of data, e.g. streaming video files), on flow and congestion requires that also the Application Layer be modelled.

3. Netzwerk-1 — A PSN SIMULATION TOOL

The implementation of Netzwerk-1 follows the object-oriented programming methodology and uses the programming language C++ including its Standard Template Library[9]. Employing these standardised programming languages guarantees portability and maintainability.

The objects of a packet-switching network are obviously packets and switching nodes including their outgoing links to neighbouring switching nodes. These are implemented in the classes `Packet` and `SwitchingNode`, respectively. Further, the central object responsible for setting up a PSN and running as well as monitoring its operation is provided by the class `PacketSwitchingNetwork`.

3.1 Class Packet

The class `Packet` provides the functionality of a packet in a PSN. We assume that all packets have equal length and therefore ignore their payload. Each instance of the class has four data members: destination address, creation time, the number of hops already performed, and its state (either INTRANSIT, ARRIVED, or DISCARDED).

A packet is always in exactly one state. Each packet, created with the only constructor `Packet(Destination, CreationTime)` is initially in state INTRANSIT. From this state it can be switched to one of the other (final) states: ARRIVED, i.e. the packet is successfully delivered to its destination, or DISCARDED, i.e. the packet is going to be destroyed for some other reason. The public methods `switchToArrived(ArrivalTime)`, returning the packet's delay time, and `switchToDiscarded()`, respectively, must be used for these state changes.

These methods raise an error if applied to packets not in state INTRANSIT. Further, the destructor of the class raises an error if it destroys a packet in a non-final state. Those mechanisms prevent the unintended loss of packets and aid in debugging the code.

The number of hops of a packet is initialised with zero (constructor) and must be incremented whenever the packet has traversed a link. This is achieved by calling its method `incHopCounter()` which also raises an error if the packet's state is not INTRANSIT.

The class Packet provides a basic statistic by maintaining a set of static counters (stored as doubles to avoid overflow). These include the number of packets created, arrived and discarded, and sums of quantities associated with each delivered packet, see Sec. 2.

3.2 Class SwitchingNode

The class SwitchingNode sets up and runs a switching node in a PSN. For this purpose it maintains queues of packets and lists of neighbouring switching nodes. It has the capability to create and receive packets, and to store and forward (route) packets destined for other nodes of the network. In order to do the latter, it maintains a routing table which is used to determine the next switching node on an efficient path, according to some least-cost criterion, to a packets destination. The class SwitchingNode can also update its routing table values based on the routing table values of its neighbouring nodes. This capability is used to perform a distributed routing table update of the whole network. The class SwitchingNode is implemented as an abstract class and hence no objects of it can be instantiated. However, the class still provides all the functionality described by declaring pure virtual functions which must be implemented by derived classes of SwitchingNode. We derive two classes SwitchingNode1 and SwitchingNodeN. These classes maintain one outgoing queue in total per node and one outgoing queue for each neighbouring node, respectively. This implies that instances of class SwitchingNode1 can forward at most one packet per time step whereas instances of SwitchingNodeN can forward at most one packet to each neighbour.

3.3 Class PacketSwitchedNetwork

The class PacketSwitchedNetwork sets up a PSN by creating a network topology with switching nodes and interconnecting links. A variety of topologies (periodic and non-periodic; based on a square or a triangular lattice; with l additional, randomly generated links) can be created. The characteristic attributes (centralised or distributed routing table update, parameters for computing the costs of traversing links, the source load, etc.) of the network are initialised here.

After initialisation, this class operates as a supervisor of the created network: it can run the network for a specified amount of cycles of the network update algorithm, see [5], and in doing so it collects and stores statistical data as time series. This data can be saved for a more detailed evaluation with other tools. As the supervisor of the network, the class PacketSwitchedNetwork has also the capability to run a centralised update of the routing tables of all switching nodes.

3.4 Random Choices

An important part in the simulation process are random choices with certain probabilities, like creation of a packet or not, choice of an outgoing edge from a set of possible outgoing edges, etc. The user is free in its choice of the pseudo-random number generator as long as this generator is implemented in a class providing three public methods to serve the following purposes:

- `void SetSeed(unsigned long seed)` to set the seed of the generator,
- `double DrawDouble(void)` to draw a real valued pseudo-random number from the interval $[0, 1)$ with uniform probability, and
- `unsigned int DrawInteger(unsigned int n)` to select a value from the set $\{1, 2, \ldots, n\}$ with uniform probability. If $n = 0$ then 0 is returned.

Based on these methods, we can, e.g., decide on success or failure of a Bernoulli trial with probability of success $p \in [0, 1]$: if `DrawDouble()` $< p$ then the trial is successful, otherwise it failed.

The reason why the pseudo-random number generator should be implemented as a class is that the sequence of each instance used in the program must be independent of the others. We are not content with one sequence of random numbers in the simulation system because we use them for different purposes. For example we compare simulation results of PSNs with the same topology for various source load values and in this case we must fix the random number sequence used to select additional, randomly generated links.

Netzwerk-1 implements the pseudo-random number generator described in [10]. This generator combines a subtract-with-borrow generator with a Weyl generator. The combination generator returns `unsigned int` values in the inclusive range $0, 1, 2, ..., 2^{32} - 1$. The period of the combined generator is not known but in [10] it is stated that there is theoretical support for the empirical observation that combining two generators will produce better results, or at least no worse, than either of the component generators. Hence we expect a period of at least $\approx 10^{414}$—sufficiently long for large PSN models and long simulation times.

3.5 The Text Interface

We provide an easy-to-use text-based interface to the simulation system. The set of commands that can be executed is rather rich and can easily be expanded as needed by defining new command functions and by suitably designing and coding them. The current state of the system is saved in a structure which can be accessed and modified by all command functions. A rudimentary help system is provided, too.

The text interface allows for the following operations:

• setting of parameter values for the definition of the PSN model,

• creating and running a PSN model including monitoring of individual nodes and queue, and

• saving the gathered statistical data about the PSN network and the simulation run to text files.

The text interface allows control of the simulation system by script files. A simple script file for running a single simulation is the following:

```
set LatticeSize = 20
set Geometry = SQUARE
set NumberOfRandomLinks = 1
set FinalSimulationTime = 200
set UpdateRoutingTable = DISTRIBUTED
set SwitchingNodeType = SN1
set OutputFileName = network.out
set UPRSeed = 1
createPSN
runPSN
savePSN
destroyPSN
quit
```

4. CONCLUDING REMARKS

We have presented main design decisions of our packet-switching network simulation tool Netzwerk-1. We placed emphasis on a portable code using standardised programming tools. The proposed structure is extendable to include other routing mechanisms, such as partial table routing [1], or more heterogeneous network connection topologies. Currently the packet traffic generation at each node uses a Bernoulli distribution specified by a given source load value. In order to incorporate the observed long-range dependence of real packet traffic in the PSN model, it is straightforward to replace the packet creation method at each node by implementing other distributions or the chaotic map described in [11]. The software system Netzwerk-1 is completed by a set of Matlab scripts, described elsewhere, which read the saved statistical data and evaluate and compare simulation runs.

Acknowledgements: A.T. Lawniczak and A. Gerisch acknowledge partial support from the University of Guelph and The Fields Institute for Research in Mathematical Sciences. A.T. Lawniczak acknowledges additionally partial support from the Natural Science and Engineering Research Council (NSERC) of Canada. B. Di Stefano acknowledges total financial support from Nuptek Systems Ltd. The authors acknowledge the use of the SHARCNET computational resources at the University of Guelph.

References

[1] H. Fukś, A.T. Lawniczak, “Performance of data networks with random links”, *Mathematics and Computers in Simulation*, vol. 51, 101-17, 1999.

[2] H. Fukś, A.T. Lawniczak, and S. Volkov, “Packet Delay in Models of Data Networks”, *ACM Transactions on Modeling and Computer Simulation*, vol. 11, 1-18, 2001.

[3] A.T. Lawniczak, P.Zhao, A. Gerisch, and B. Di Stefano, “Modeling flow and congestion in packets switching networks”, IEEE Canadian Review, 23-27, Winter 2002.

[4] A.T. Lawniczak, A. Gerisch, P.Zhao, and B. Di Stefano, “Effects of Randomly Added Links on Average Delay and Number of Packets in Transit in Data Network Traffic Models”, To appear in the Proceedings of DCDIS’2003, Guelph , Ontario, Canada, May 15-18, 2003.

[5] A.T. Lawniczak, A. Gerisch, B. Di Stefano, “OSI Network-layer Abstraction: Analysis of Simulation Dynamics and Performance Indicators” (submitted for publication), 2003.

[6] A.T. Lawniczak, A. Gerisch, B. Di Stefano, “Development and Performance of Cellular Automaton Model of OSI Network Layer of Packet Switching Networks”, To appear in the Proceedings of CCECE’2003, Montreal Quebec, Canada May 4-7, 2003.

[7] A.T.Lawniczak, A. Gerisch, and K.Maxie, “Effects of randomly added links on a phase transition in data network traffic models”, To appear in the Proceedings of DCDIS’2003, Guelph , Ontario, Canada, May 15-18, 2003.

[8] A. Gerisch, A.T. Lawniczak, B. Di Stefano, “Netzwerk- a packet-switching network simulation enviroment”, (in preparation), 2003.

[9] D. R. Musser, G. J. Derge, A. Saini, “STL Tutorial and Reference Guide. C++ Programming with the Standard Template Library”, 2nd Edition, Boston: Addison-Wesley, 2001.

[10] G. Marsaglia, B. Narasimhan, A. Zaman, “A random number generator for PC’s”, *Comp. Phys. Comm.*, vol. 60, 345-49, 1990.

[11] M. Woolf, D. K. Arrowsmith, R. J. Mondragón-C and J. M. Pitts, “Optimization and phase transitions in a chaotic model of data traffic”, *Phys. Rev. E*, vol. 66, 046106, 2002.