Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 49(63), Fascicola 1, 2004

Network Simulation using OMNet++ environment

Petru Serafin[1]

Abstract – The article presents the implementation using the simulation environment OMNet++ for study of network congestion. This simulation environment with its topology description language offers a good choice of a simulation tool and is appropriate for the study of network congestion.
Keywords: OMNet++ simulation environment

I. INTRODUCTION

OMNet++ (Objective Modular Network Testbed in C++) is a general-purpose simulation environment with object-oriented design for discrete events. This simulation environment is appropriate to be used for traffic modeling of telecommunication networks, for queuing networks modeling, for protocol modeling or any other system for which the discrete events theory applies.

In the domain of network simulation there are other simulation environments that are commercial (ex. COMNET, OPNET) or academic (ex. NetSim++, SMURPH). We have chosen OMNet++ for being non-commercial and well supported on its community web site [1]. OMNet++ was developed at the Technical University of Budapest and it continues to be under regular development with improvements in new releases.

This article presents an implementation of a network in OMNet++ using the language for the description of network topologies with discrete events. The language contains an efficient way to create parameterized flexible topologies and is implemented as part of the OMNet++ simulator [2].

In most simulators, the support for defining the topology of the model is inadequate. For this task, simulators do not provide explicit support, or only fixed topologies are supported, or flexible topologies require programming.

The solution proposed and implemented in OMNet++ uses a description language with a powerful combination of simple constructs (multiple connections, conditional connections etc.) to allow parameterized description of regular structures. Parameterized structures that have been previously defined in the description language can also be created dynamically, during a simulation run. This feature eliminates the need for several independent runs if we are investigating how changes in model topology affect some performance measures.

OMNet++ provides some independent random numbers generators (RNG) that may be used as sources for random input with the simulations to avoid correlation. The simplest RNG is *intrand()* that outputs an integer between 1 and a maximal value that is parameterized in the options initial file. The RNG is a linear congruential generator (LCG) with a cycle length of 2^{31}, the method used is:

$$x = (x7^5) \bmod (2^{31} - 1) \qquad (1)$$

The RNG use an initial value (seed value) and perform deterministic calculations to produce a random number and the next seed. Choosing the right seeds for RNG is a difficult problem [3].

The simulation environment includes a complete distribution library [4].

The following functions for continuous distributions are available:

- *uniform(a,b)* uniform distribution range *[a,b)*
- *exponential(mean)* exponential distribution with the given *mean* distribution
- *normal(mean,stddev)* normal distribution with the given *mean* and standard deviation
- *truncnormal(mean,stddev)* normal distribution truncated to non-negative values
- *gamma_d(alpha,beta)* gamma distribution with parameters *alpha>0* and *beta>0*
- *cauchy(a,b)* Cauchy distribution with parameters *a* and *b*, where *b>0*
- *triang(a,b,c)* triangular distribution with parameters *a<=b<=c*, *a!=c*
- *lognormal(m,s)* lognormal distribution with mean *m* and variance *s>0*
- *weibull(a,b)* Weibull distribution with parameters *a>0* and *b>0*
- *pareto_shifted(a,b,c)* general Pareto distribution with parameters *a,b* and shift *c*

The following functions for discrete distributions are available:

- *intuniform (a,b)* uniform integer from *a..b*
- *bernoulli(p)* result of Bernoulli trial with probability *0<=p<=1*, *1* with probability *p* and *0* with probability *(1-p)*

[1] Alcatel– R&D Technical Center Romania, Timişoara
petru.serafin@alcatel.fr

- *binomial(n,p)* binomial distribution with parameters $n>=0$ and $0<=p<=1$
- *geometric (p)* geometric distribution with parameter $0<=p<=1$
- *negbinomial (n,p)* binomial distribution with parameter $n>0$ and $0<=p<=1$
- *poisson (lambda)* Poisson distribution with parameter *lambda*

Modules can have an arbitrary number of input/output gates to interconnect in the simulation network. The connection is determined from an out-gate to an in-gate. This principle of gates allows to create complex models and has the advantage of defining modules that are reusable.

Communication using gates is done by the principle that modules can find which in-gate has a message arrived and can send messages directly to a specified out-gate.

To specify the topology of simulation networks, OMNet++ uses a separate language called the NEtwork Description language (NED).

The NED files contain the simple modules (only declarations they are to be implemented as C++ class), compound modules and the connections between models.

The connections between models are channels and are described by the following parameters:

```
channel
        delay
        error
        datarate
endchannel
```

Simple modules are described by parameters and gates.

```
simple QServer
    parameters:
        serviceTime: numeric;
    gates:
        in: in[];
        out: out;
endsimple
```

OMNet++ programs can be run in graphical user interface which is preferred and also for repetitive launching the command line execution is offered by the simulation environment.

For debugging purpose OMNet++ simulation environment has all the facilities of an object-oriented environment: watches, snapshots, breakpoints, stack usage checker, library classes to collect results.

The watches can be declared over a variable that needs to be inspected and/or changed during the simulation. Watches are presented in the snapshots of the simulation.

The snapshot is a dump status of the entire network during the simulation that is redirected to a text file (default *omnetpp.sna*). The snapshot file contains modules, message queues and watches.

The breakpoints can be set by calling the function that suspends the execution of an *activity()*.

Stack usage is to be checked for co-routines as they use space on the stack.

The standard library provides several classes to collect results. For example *cStdDev class* can collect samples and compute standard deviation and means, the *cHistogram class* can store an approximated density.

The measurement extraction can be done at the end of the simulation or during a simulation run.

II. NETWORK TOPOLOGY

The network topology simulated for the study of network congestion is the following: we want to study the effect of a malfunction of a server in a 3 server configuration.

First we define the topology using the NED language, implementing 3 servers that use 3 entry populations of the same distribution (further tests are done for different distributions). The serving mechanism is implemented by the queues as FIFO.

We start with simulating 3 servers alone then add a backup server to obtain the mathematical model for the 3+1 server configuration.

The initial network topology for congestion study is constituted of 3 population generators, 3 classifiers to route messages, 3 delays to enter 3 queues that are served by all 3 servers. The topology is presented in Fig.1 and will be referred as configuration 1:

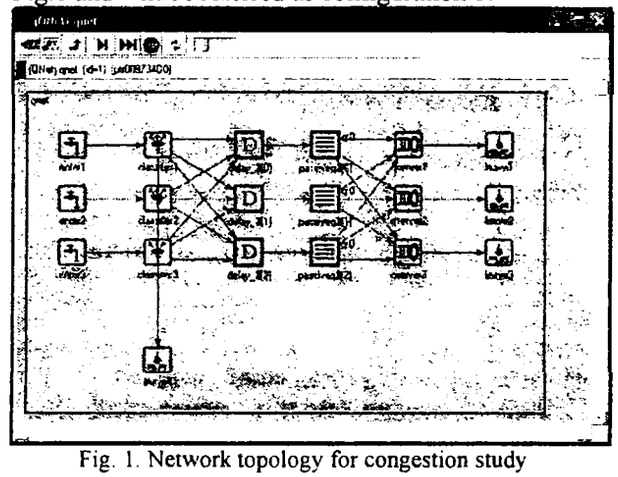

Fig. 1. Network topology for congestion study

During execution of normal traffic we will simulate the blocking of the 3rd server (qserver3) and observe the traffic being re-routed to the other functional servers. We will observe the leave-modules were implemented a counter for the queues.

The results observed in the queue corresponding to the blocked server (leave3) show that the counter increments until the time we simulated that server is repaired.

The result for the leave-modules is presented in Fig.2. Note that during server malfunction the queue corresponding to the server is increasing as the other functional server do not take all the traffic load.

To continue the congestion study we implement a backup server into the topology to obtain the configuration 2. This configuration has a first level of security: a backup server will take the traffic load from any of the other 3 servers if any of them is malfunctioning.

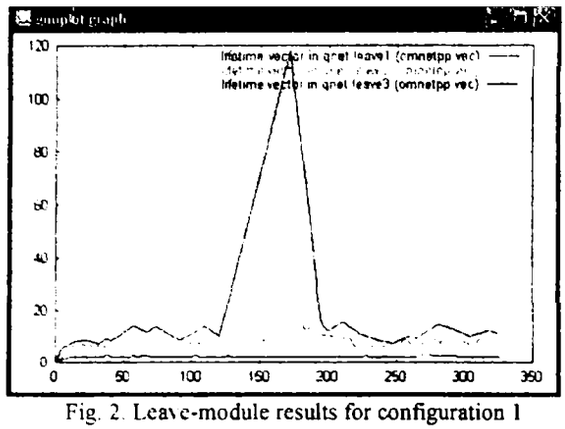

Fig. 2. Leave-module results for configuration 1

Fig.3 presents the configuration 2 for implementing the backup server:

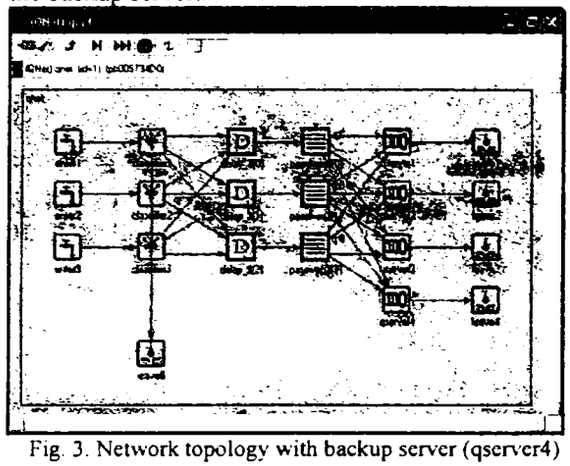

Fig. 3. Network topology with backup server (qserver4)

Now having a backup server we can secure one malfunctioning server, all its traffic being executed by the backup server. Fig.4 represents a comparison of the queue of the malfunctioning server in configuration 1 with the queue being executed by the backup server in configuration 2. Evidently the backup server is assumed functional to be able to execute all traffic that was previously supported by the malfunctioning server.

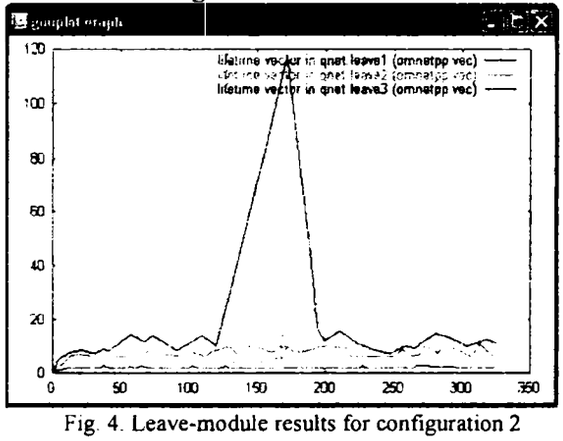

Fig. 4. Leave-module results for configuration 2

Now we can imagine an even worse situation: what happens if the backup server is already occupied, for example if it is already supporting the traffic load of qserver3 that is malfunctioning and at a later moment another server is detected malfunctioning. We analyze what happens if the other server becomes faulty while backup server is still occupied.

For this purpose we will determine the mathematical model for the blocking server situation, corresponding to the possible states of congestion. We will determine the liability for this network with 3 servers and one backup server, as this configuration will be used in further study.

III. MATHEMATICAL MODEL

The proposed mathematical model is based on defining the following states of the network:

State 0 : All servers functional in nominal regime

State 1 : Occupation of backup server is detected by a periodical test

State 2 : Occupation of backup server is detected by an internal test

State 3 : Blocking of a server is detected by a periodical test

State 4 : Blocking of a server is detected by a periodical test and backup server is free

State 5 : Blocking of a server is detected by an internal test but backup server is already occupied

State 6 : Blocking of a second server is detected by a periodical test but backup server is already occupied

State 7 : Blocking of a second server is detected by an internal test but backup server is already occupied

The status automata are presented in the following figure:

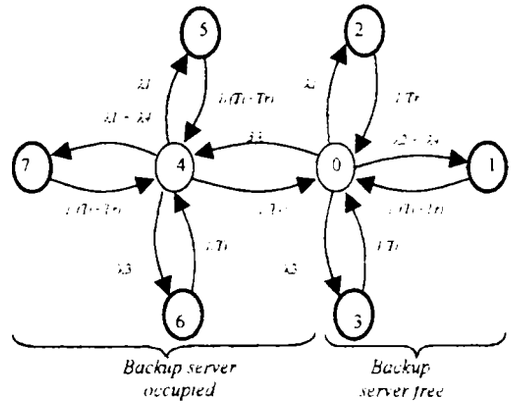

Fig. 5 Status automata for backup server

Liability calculation can be done using the definitions of unavailability for the states in Fig.5.

The total unavailability of the network can be expressed as the sum of individual unavailability in the modeled states:

$$U = P1 + P2 + P3 + P5 + P6 + P7 \quad (2)$$

As we seek unavailability during congestion it is normal not to include the probabilities in states 0 and 4, because these are functional states with no restrictions: 0 is the default state where all servers are functional, 4 is the working state where one server has been detected faulty and all its traffic is being rerouted to the backup server.

For example, we may calculate the probability P4 of the backup server currently working for the traffic of another server that is faulty :

$$P4 = \frac{3\lambda Tw}{1 + 3\lambda Tw} \qquad (3)$$

With the data from the simulation we can compute this probability and we obtain the value $5{,}13 \cdot 10^{-3}$. Note that state 4 is stable, it has the backup server occupied so transition is possible to unstable states 5, 6 or 7, or transition back to state 0 if the faulty server is repaired and in consequence backup server is liberated.

We base the expression on the following variables that represent probability of malfunction and intervention delays:

- $\lambda 1$: malfunction rate of one server
- $\lambda 2$: malfunction rate of one common component for two servers
- $\lambda 3$: malfunction rate of one common component for three servers
- $\lambda 4$: malfunction rate of backup server
- λ : malfunction rate of simultaneous two servers
- Tp : mean delay of a periodical test
- Ti : delay of immediate intervention
- Tw : delay of waiting for intervention

Replacing the terms in expression (2) with the formula for probabilities for backup server free or occupied, we obtain:

$$U = \left(\frac{1}{1 + 3\lambda Tw}\right)\left[\lambda 1 \frac{Tp}{2} + (\lambda 2 + \lambda 3)\left(\frac{Tp}{2} + Ti\right) + \lambda 4 Ti\right] + \left(\frac{3\lambda Tw}{1 + 3\lambda Tw}\right)\left[\lambda 1\left(\frac{Tp}{2} + Ti\right) + (\lambda 2 + \lambda 3)\left(\frac{Tp}{2} + Ti\right) + \lambda 4 Ti\right] \qquad (4)$$

When we calculate this expression using the standard simulation data we obtain the results for each unavailability in the network topology of configuration 2 as presented in Table 1:

Table 1. Unavailability results for configuration 2

Level	*Unavailability* (10^{-5})
P1	0,125
P2	0,092
P3	0,153
P5	negligible
P6	negligible
P7	negligible
Total	*0,37*

The total unavailability is then the sum of individual unavailability. Corresponding to the chosen parameters for simulation we obtained the total unavailability for the presented topology of configuration 2 (3 servers + 1 backup server) to be $0{,}37 \cdot 10^{-5}$, that represents if expressed by rapport of time unavailability of 1,92 seconds per year.

The ITU standard indication for telecommunication equipment unavailability is $2{,}88 \cdot 10^{-5}$ which is equivalent to unavailability of 15 minutes per year.

The simulation results for a configuration with 3 servers and a backup server with the initial data such chosen is well within the required parameters defined by ITU for telecommunication equipments.

By extrapolation it can be calculated using the same simulation data that it is still possible to meet this required unavailability even with a configuration of 32 servers with only one backup server.

IV. CONCLUSION

The data are relevant and repetitive simulation with different parameters for unavailability have been done to show that any topology with a backup server can be simulated in OMNet++ until compiler limit is reached. Also, by implementing the model into an additional function it is possible to calculate the liability of the simulated network.

The method proposed was to define the stated as stable or unstable and to base the model on transitions between these states so that we are able to calculate the total unavailability of a certain network. Along with the mathematical model it is possible to implement a network simulation using the powerful environment OMNet++ to describe the topology and run the simulation.

Calculations can be done easily for any topology of network and parameters can be changed from one simulation to another.

Simulations may use different entry populations, as described by the distribution functions presented in Chapter 1. The serving mechanism may be changed to add priority for certain queues and so to be able to simulate some heterogeneous networks.

This simulation environment is powerful and offers many development facilities, not only for defining network topologies but also for implementing new simple or compound modules that may be needed for simulation.

Liability calculation results with some simulation data show that it is fair enough to implement one backup server for 31 functional server for simulation purpose. Mathematical model can be applied for real networks to determine the liability. The three delays that are used in the formula of unavailability can be parameterized to obtain even more secure network or to extend the topology to a maximum number of servers that are backed up by one or more servers. For example if it is chosen a short delay for immediate intervention then we observe that unavailability is reduced, but in practice it is preferable to equilibrate intervention delays with waiting time and periodical or internal tests. Providing more often periodical tests may have the consequence to occupy the network by testing equipments instead of transporting traffic. But also rare periodical tests may come too late to detect a faulty equipment and then network may become unstable.

Simulation environment OMNet++ allows many options for implementing the parameters needed for network simulation, such as delays and probabilities, so we conclude that it is an appropriate tool for the study of network congestions. It is possible to simulate all the possible aspects of system instability, overload or unavailability.

REFERENCES

[1] http://www.omnetpp.org OMNet++ community web site

[2] A. Varga, "OMNet++ User Manual", Department of Telecommunications, Technical University of Budapest.

[3] A. Varga, "OMNet++ Discrete Event Simulation System", proceeding of the European Simulation Multiconference ESM2001, June 6-9 2001 Prague

[4] A. Varga, "Software Tool for Networking" IEEE Network Interactuve, July 2002 vol.16 no.4