# A Simulation Tool for Traffic Engineering Methods and QoS Evaluation of MPLS Networks

Sylwester Kaczmarek
Gdansk University of Technology
ul. G. Narutowicza 11/12
80-952 Gdańsk, Poland
+48 58 3472767
sylwester.kaczmarek@eti.pg.gda.pl

Krzysztof Nowak
Nokia Siemens Networks
ul. Żupnicza 11
03-821 Warszawa, Poland
+48 604 490336
krzysztof.nowak@nsn.com

## ABSTRACT

The architecture of Multiprotocol Label Switching (MPLS) is successfully deployed in networks of many service providers, including telecommunication companies. The technology promises to provide Quality of Service (QoS) in IP networks and introduce a mechanism for effective traffic control. Because of the great complexity of analytical models, simulation is an effective technique for evaluating traffic engineering methods and new algorithms. In this paper, we present a model and the application *msim*, a C++-based simulator of MPLS networks. Its open structure allows for easy extension of new models and algorithms. Measurements results generated by the tool for networks of up to 100 nodes in size were presented at several international traffic engineering conferences. We present the logical structure of the program as well as example measurements scenarios and results, for packet and connection level simulations. We also describe shortly the *Vims*, a graphical extension to the *msim* simulator.

## Categories and Subject Descriptors

I.6.3 [**Simulation And Modeling**]: Applications.

## Keywords

Simulation tool, Modeling, QoS, MPLS.

## General Terms

Measurement, Performance, Experimentation.

## 1. INTRODUCTION

### 1.1 MPLS history

Multiprotocol Label Switching (MPLS) is a network architecture developed in the late 90's. MPLS was originally proposed as a remedy to insufficient speed of IP routers facing fast growing of IP networks. The idea came from different technologies, like Asynchronous Transfer Mode (ATM) or Cisco Tag Switching.

Several years later it became clear that the routing speed is no longer a problem. In the meantime fast hardware-based line modules for routers were developed, which greatly increased the routing speed. However, support for traffic engineering methods offered by MPLS met growing demand for quality of service and better control of network resources. This made MPLS a very attractive solution for many network providers.

### 1.2 MPLS architecture

The architecture of MPLS network is defined in details in RFC3031 [1]. Here we want to give very basic information about it, especially on the routing principle.

Data transport in MPLS requires having the complete path established in advance. If this is provided, IP packets entering an MPLS network are encapsulated with an additional header, which is used for forwarding instead of the IP header. The main field in the MPLS header is the 20-bit label, which is the identifier of the path which the packet belongs to. Routing is performed by changing the value of the label before sending the packet to the next hop. The actual label value for every hop is set locally between the neighboring routers and in general has no relation to the IP address.

The MPLS networks are designed for the core of the network, rather than from, and to the end user. Paths usually gather many IP streams so the traffic carried in a path is normally quite large. One of the practical examples of a path may be an IP connection of two local offices of a company. The path works in this case like a virtual link between the two locations.

Paths must have bandwidth reserved for them in the control plane to guarantee the QoS and enable traffic engineering methods. As a consequence, one must record the reserved bandwidth values on every port and check free bandwidth value when choosing a route for a path. That is possible thanks to extensions to existing link-state routing algorithms, like OSPF-TE and ISIS-TE, where TE denotes the traffic engineering extension.

The nodes in MPLS networks are IP routers extended by MPLS functionality and protocols. They use so called label distribution protocols to automatically create the necessary paths without bandwidth reservation. (It is difficult to automate bandwidth reservation because the traffic amount is usually unknown at the time of path creation.) The most common label distribution protocols are RSVP-TE (Reservation Protocol with Traffic Engineering) [2] and LDP (Label Distribution Protocol) [3]. The

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5689
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5689

LDP relies entirely on the information provided by the IP routing protocol when selecting routes, whereas the RSVP-TE is more flexible and gives an option for the explicit routing, where the route is either completely or partly pre-defined. This in turn gives possibilities to compute the route outside the protocol instance, eg. by a specialized offline tool.

## 1.3 Evaluation methods in MPLS

Research in MPLS can be based on an analytical or simulation approach. However, the application of analytical methods is very restricted. This is due to the great complexity of the nodes architecture, especially buffering systems employed in the routers. An additional problem is the complexity of IP traffic, which does not follow any classical models and requires novel and complicated tools. Indeed, the mathematical models are very complex and difficult to analyze. In contrast, the simulation approach is much more suitable for research of MPLS networks.

In simulations we use a model of MPLS networks, which takes advantage of some simplifications regarding the node architecture and the traffic structure. We don't simulate the protocols, like RSVP or OSPF, as this would not bring enough added value to the quality of the results. Instead, our goal is to concentrate on different traffic engineering methods and new ideas and their influence on quality of service.

## 1.4 Contents

The paper is organized as follows. In the second chapter we present the MPLS network model used for implementation. The third chapter we devoted to the structure of the simulation program. We also highlight there some implementation aspects. In the fourth chapter we included examples of research areas where the program proved its usefulness. In the last chapter we introduce the *Vims*, a graphical extension to the simulator. Finally, we present conclusions and features we want to introduce to *msim* in the near future.

# 2. NETWORK MODEL

## 2.1 Simulation

We use a discrete event simulation with a common event queue. The simulation can be performed either on a packet level or on a connection level. The first one is used when we are interested in measurements of traffic characteristics. It provides detailed information about the quality of service, but the simulation time is often restricted to a fraction of a second. In contrast, the connection level simulation can be used to simulate long-term network behavior, especially to evaluate routing or preemption mechanisms.

## 2.2 Node architecture

Packet forwarding is based on the RFC3031 [1]. The node is a pure MPLS node, without IP routing functionality. The MPLS forwarding routine uses three arrays: NHLFE, ILM and FTN, as shown in fig. 1. The NHLFE (Next Hop Label Forwarding Entry) contains information about the next hop and a new label, which is assigned to the packet when it leaves the router. The ILM (Incoming Label Map) contains bindings between the incoming label value and the proper NHLFE entry. If the router is the access node and incoming packets have no MPLS header yet, then the FTN (FEC To NHLFE, FEC – Forwarding Equivalence Class) entries are checked, where the translations from the IP address prefix and the NHLFE entry are stored.

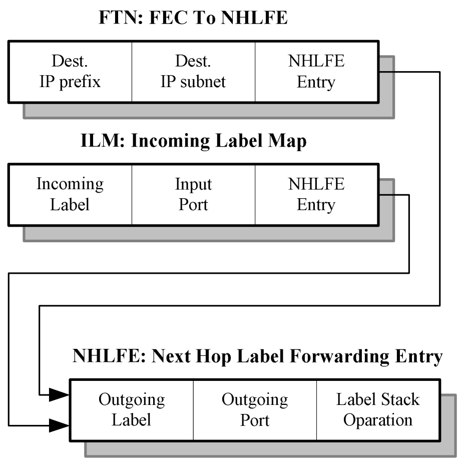


**Figure 1. Structure of forwarding arrays.**

After the forwarding decision is made and the outgoing label assigned, the packet is sent to a buffer in an output port. The buffering system is defined in the node description part of the configuration file. Our implementation offers flexibility in building different combinations of buffers by connecting smaller logical blocks. Currently there are limitations because of simplified syntax of this part of the configuration file and there are three possibilities: simple queue, PQ (Priority Queuing) and the combination of PQ and WFQ (Weighted Fair Queuing). The latter can be used with the DiffServ class model (fig. 2). The assignment of the outgoing packets to the specific queue is based on the class of service assigned to them. The queue length is defined in the configuration file.

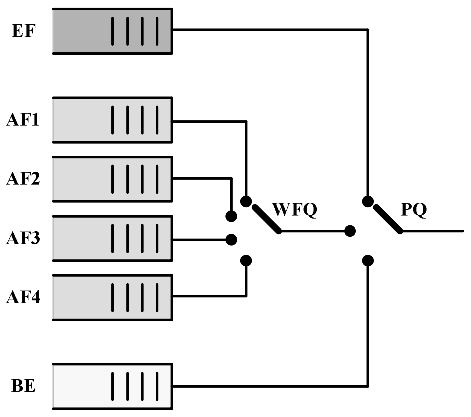


**Figure 2. Implemented buffering scheme. EF - Expedited Forwarding, AF - Assured Forwarding, BE - Best Effort.**

The packet delay in the model is the sum of three values: waiting time in queues, transmission times and propagation delays. The last value is calculated from the declared physical length of the output link, assuming that the link is a fiber, for which the speed of light is ca. 2e+8 m/s. That gives about 0.5ms propagation delay for every 100 km of fiber.

Packet losses in our model are caused by buffer overflow only. Currently we do not simulate losses in the transmission channel, e.g. caused by crosstalk or errors, as we focus on fiber links.

### 2.3 Sources

The sources in the model simulate aggregated data streams, which is the typical case in MPLS networks. We implemented a quite simple but open for extensions source definition. How it works, depends on the simulation level, which is indicated in the configuration file.

If we perform simulations on the connection level then the source is acting only as a trigger to create a path with declared bandwidth reservation. This mode is used by the topology manager, which automatically creates and shuts down the sources.

For the simulations on the packet level, the source generates packets at declared average speed and length. Both time and length can be set constant or selected randomly with uniform or exponential distribution. The implementation can be easily extended to other distributions.

The configuration file contains definitions of a given number of source types, including parameters like speed and packet length characteristics, class of service and activity duration (for connection level simulations).

## 3. SIMULATOR APPLICATION

### 3.1 Introduction

The *msim* simulator evolved from an ATM connection level simulator *casino*, written from scratch to evaluate connection admission control (CAC) methods. With growing popularity of MPLS, we decided to implement MPLS logic and introduce the packet-level simulations. Since then virtually any module of the *msim* has undergone major changes to make it more useful and universal.

There are many popular simulation tools available, including open community based *ns-2* [4] and *OMNeT++* [5] or commercially available *OPNET* [6]. Unlike these universal tools, the *msim* has been developed and optimized as an MPLS simulator only and currently it is not planned to extend it to other technologies or transport techniques. Thanks to this principle, configuration files of *msim* are usually much simpler, because the simulated network is preconfigured as an MPLS network. An important feature is the way the results are gathered by our application. It automatically calculates a set of network-wide parameters so as the user only needs to declare the list of requested parameters. After the simulation finishes, the average values are collected and written to the file, along with the confidence periods for every measured value. This also greatly fastens the process of interpreting the results. The fact that the tool is specific to MPLS, makes it less complex than the other tools and therefore easier to troubleshoot.

The primary goal of the implementation has been to create a tool which would serve the authors in research of traffic engineering methods at a network level, including new routing or preemption methods. The application should be easy to configure and capable to perform series of simulations and the results from different series should be presented in a common report file.

The *msim* application has been implemented as a C++ program consisting of 25 modules and ca. 11,000 lines of code. Currently the running platform is the Windows XP operating system. Though it should work properly in different Windows versions, these were not tested. The source code has been written using class hierarchy: most of the classes inherit from the class “basis” and every object capable of emitting and interpreting events inherits from the network object class “netob”. The program has a very basic user interface (fig. 3) and it records more detailed messages to the log file.

```
MPLS Network Simulator, K. Nowak, 2003-2008.

Creating configuration manager.
Creating connection manager.
Creating report manager.
Reading configuration and creating the network.
Checking network integrity.
Creating topology manager.
Begin of simulation.
 1i>....>....>
End of simulation.
Finished without problems!
```

**Figure 3. Typical console output of *msim* for single iteration.**

The simulation starts with a preparation phase, in which the control objects and the network are created. The second phase is the simulation of the network, and in the last phase the results are written to the files and the objects are destroyed.

The simulation phase begins with the initial period, which is used to reach the stable state of the network. After the initial period finishes a number of regular (simulation) periods of equal duration pass. At the end of each of them partial results are collected (fig. 4). Thanks to such division the final results consist not only of the average value, but also the confidence periods, calculated from the differences between the partial results using the T-Student distribution.

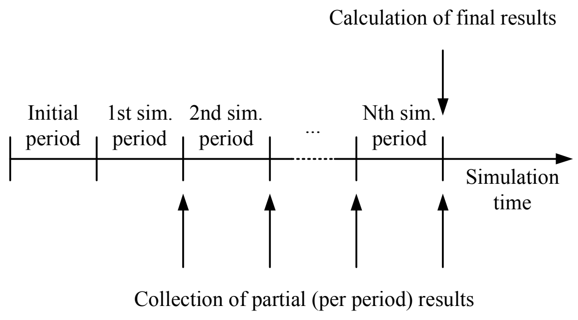


**Figure 4. Initial period and simulation periods.**

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5689
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5689

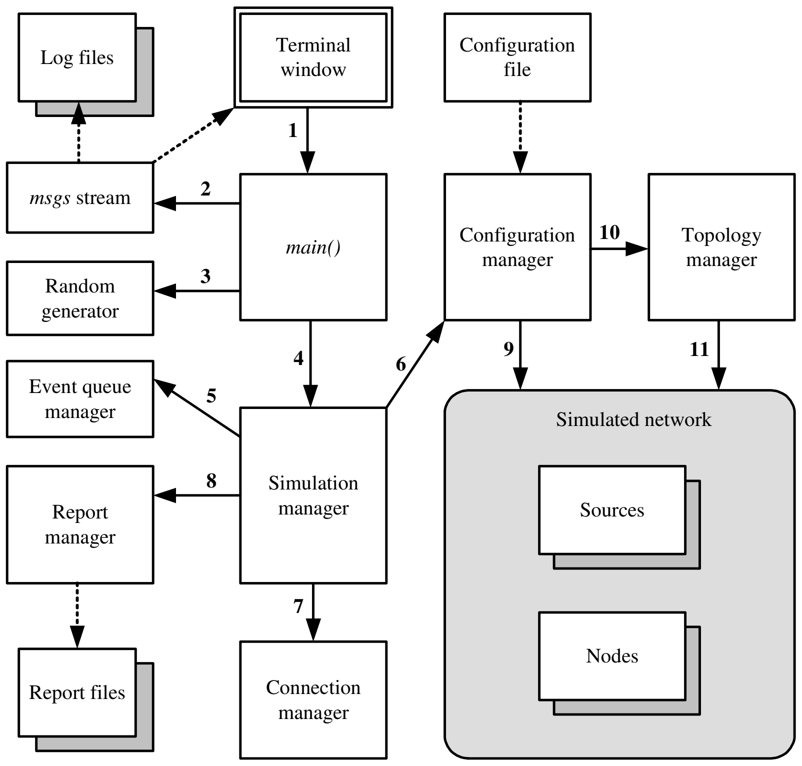


**Figure 5. Structure of the program and objects creation order.**

The paths can be either static ones, declared in the configuration file, or created automatically during the simulation. The latter case is necessary to perform nontrivial connection level simulations. The topology manager object is responsible for the changes.

## 3.2 Program structure

In fig. 5 we show the structure of the program that contains the major objects. The main object is the simulation manager, which is controlling the simulation flow, including creation of other major objects and dividing the simulation time between initial and regular periods.

The other important objects are: configuration manager, which reads the configuration and creates the network, connection manager, which stores information about the paths, and report manager, which collects and saves the measurement results. Here we describe how the simulation objects are constructed.

Just after its start (1) the program creates the message stream *msgs* (2) for debug and output messages, and the random generator object (3).

The first network object created is the simulation manager (4), which then takes over control of creating other objects and performing the simulation.

The first task of the simulation manager is to interpret the command line parameters, which always include the configuration file name and optionally other information, e.g. the debugging level. In the next step the fixed network objects are created. These are: event queue manager (5), configuration manager (6), connection manager (7) and report manager (8).

When the fixed objects are created, the network structure can be built. The configuration manager reads the configuration file and creates network objects (9) including nodes, sources and the topology manager. At that point the network structure is ready and the simulation can start, so control is passed to the event queue manager. Its main task is to distribute events in a loop, until any event remains in the queue. To stop the simulation at a certain time point, the simulation manager simply requests deleting all remaining events from the queue.

After the event loop ends, the next iteration can start, if requested. In this case the simulation manager deletes all the objects of the simulated network and resets the state of the fixed objects, what for most of the objects has a similar effect like deleting and creating them again (This is not the case for the report manager, which has to keep the results from previous iterations.) At that point the configuration manager has no information about the network and it reads the configuration file again, now using the values which correspond to the new iteration. Then the event queue starts again, like during previous iterations.

The ending of the last iteration is signaled with broadcast message CM_END_SIM, which triggers creating combined reports by the report manager. After the reports are created, the network is deleted and the simulation quits.

## 3.3 Main features

### *3.3.1 Discrete event simulation*

The simulation principle is based on events, which are distributed between the simulated objects. An event corresponds to an action in the network, like generation of a packet, creation of a new source, ending the simulation, etc. The event is put into the queue, which is sorted based on delivery time. A simplified form of the event loop looks like following:

```
1  while (queue->first_item != NULL) {
2    event = queue->fetch_first_item();
3    sim_time = event->time;
4    event->receiver->handle_event(event);
5    delete_event(event);
6  };
```

The loop is executed while the queue is not empty. The first event (which has the smallest delivery time) is fetched from the queue and its delivery time becomes the current simulation time (line 3). The event is sent to the receiver object for processing and than deleted. Such procedure works well, because of the principle, that only the events trigger changes in the network. It is also used in many other simulators, including OMNeT++ [7]. In our implementation the events can be addressed to a specific object or broadcasted to every network object. An example of the broadcast message is the event CM_END_SIM, which is generated by the simulation manager and delivered to every network object when the simulation must finish.

### *3.3.2 Network definition in a text file*

The simulation parameters and the topology are defined in a text file which is easy to edit and interpret. The file consists of simulation parameters, node and sources definition, static paths configuration, classes of service description and report files structure. Every line starts with the command name, followed by colon and the comma-separated parameters in the form of *name*=*value*, and finishes with semicolon, for example:

```
link: id=44, bw=1e+09, pt=0.001, [...] ;
```

### *3.3.3 Direct report generation*

The measurement results are written to a set of files. The included measurements and order of values are defined in the configuration file. The values are organized in tables with user defined separators. This allows for creating tab-delimited or comma-separated values (CSV) file. The number representation can be set to any locale to ease the post-processing of the results. Properly defined report files are ready to create graphs of them.

### *3.3.4 Iterations*

The simulation can be automatically repeated several times, using the same configuration file. Thanks to a special syntax virtually any configuration parameter can be defined as a list of values, so as the *n*-th element of the list is used as the current value for the *n*-th simulation repetition. At the end of the last simulation, a single report file is generated, which combines values of every iteration in a manner which makes it easy for graph generation or just for comparison. This greatly simplifies preparing and processing the results of a series of simulations, where only one parameter changes, which is a common scenario.

## 3.4 Main objects

### *3.4.1 Simulation manager*

The simulation manager is the engine of every simulation. It is created first and deleted as the last object. It carries the simulation flow and keeps record of every created network object. Its main tasks are the following.

1. Read and interpret the command line parameters.

2. Store pointers to every network object. This allows for mass operations performed for all objects, like deleting the simulated network or sending broadcast messages.

3. Store keywords, which correspond to network objects. Every object has associated with them one or more text strings called keywords. Any network object which needs to know the pointer to another object can get this information from the simulation manager by providing the keyword.

4. Control the simulation flow. The simulation manager executes the iteration loop, creates and deletes the network objects, and starts the event queue manager. It also generates most of the control events.

### *3.4.2 Event queue manager*

The event queue manager is a passive object that stores the event queue, which is sorted by the delivery time. When requested by the simulation manager, the object executes the event loop which in fact keeps the simulation running. It simply fetches the event from the queue head and runs the handler procedure in the receiver. That in turn generates an action in the receiver object and possibly generates another event.

### *3.4.3 Configuration manager*

The main task of the configuration manager is to read the configuration file, perform necessary syntax and consistency checks, store the read information and properly create the simulated network.

The process of reading the configuration file and creating the network is done every time the iteration starts.

### *3.4.4 Connection manager*

The connection manager keeps track of current connections (paths) created in the network. It performs creation of every connection, using implemented traffic engineering methods, including routing and admission control. If defined so in the configuration file, it also performs preemption of existing paths when there is no free bandwidth.

### *3.4.5 Report manager*

The report manager stores centrally the measurement results collected by every object and performs necessary calculations, e.g. average values and the confidence interval.

At the end of the simulation, after the last iteration, the report manager combines all the requested statistics and creates the result files. The contents and format of the files is defined in the configuration file.

*Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5689*
*http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5689*

### 3.4.6 *Topology manager*

The topology manager is an optional object, which is created only if there is a need for changing network topology during the simulation. Currently it supports mass creating and deleting sources to trigger creation of paths. One of the applications of this feature is to evaluate methods like routing or admission control, which need to be evaluated by creating new connections.

## 3.5 Generating reports

The configuration file contains definitions of the result files called reports. Such definitions include file name, list of measurements, separator character, locale used to format numbers and corresponding header labels, for example:

```
report: id=1, file="europe.pree.csv", obj=preem,
ires=line, sep=";", loc="Pl_PL",
params = "*|src+|fail+|pre+",
labels = " CoS|Src|+/-|Fail|+/-|Preem|+/-";
```

This definition creates the file "europe.pree.csv" to include values associated with preemption methods. Iteration results will be ordered line-by-line and the numbers converted to the locale "Pl_PL" with semicolons as value separators. Three measurements will be added (src, fail, pre) with corresponding period of confidence (+). The first item (*) is the default parameter used as the value of the first column, which in this case is the number of the class of service (CoS). (Another example of a default parameter can be the link number for link statistics.) The specified labels will be added as table headers. Assuming that four iterations have been performed, such definition would generate results similar to the following.

```
[...]
CoS;Src;+/-;Fail;+/-;Preem;+/-
0 [1];14 995,9;69,40;2 884,2;31,94;9 500,1;106,91
0 [2];14 990,5;64,30;3 000,2;48,15;9 787,2;113,90
0 [3];15 049;80,84;2 793,2;51,85;9 482,8;154,22
0 [4];14 943,9;70,49;2 980,3;33,10;17 496,9;337,54
[...]
```

Such format of report files can be directly used as input for further processing, e.g. for creating tables or graphs.

## 4. SIMULATIONS

The program is developed as a universal tool and there are many different measurements scenarios. In this section we present two examples. The first one uses the packet level simulation to evaluate the performance of some static and dynamic routing mechanisms. In the second example we compare preemption algorithms by simulating long-term networks behavior on the call level only. The results presented in this section illustrate different applications of the program only so we do not provide analysis of the results here.

## 4.1 Dynamic routing

We implemented a dynamic routing algorithm which reroutes some of the paths which face excessive packet loss ratio. The idea is that after the initial period several next periods are used to evaluate packet loss ratio and delay. Several paths experiencing the worst QoS are rerouted at the end of the period (fig. 6).

In fig. 7 we show the network topology we use. We compared the resulting packet loss ratio value with corresponding results of the static routing method, i.e. without any rerouting. In both cases we performed a series of simulations with different priority of the routing algorithm.

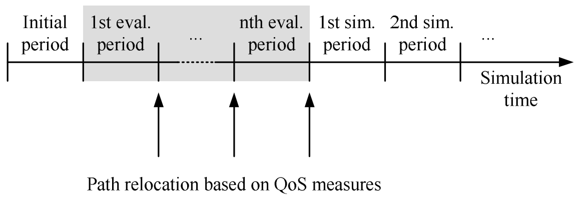


**Figure 6. Additional simulation periods are used for path relocation based on QoS measurements results.**

The hop count priority is the typical case when always the shortest path is selected, provided that there is enough free bandwidth. When bandwidth priority is used, then the path with the highest available bandwidth will be chosen even if it is not the shortest one. Additional constraint is defined using the parameter *d*, which is equal to the biggest possible difference in hop counts between the selected path and the shortest path. If *d*=0 then only the shortest path can be chosen, but with availability of more then one equal routes, the one which is less occupied will be chosen. If d=∞, then the free bandwidth is the only criterion and length of the path does not count.

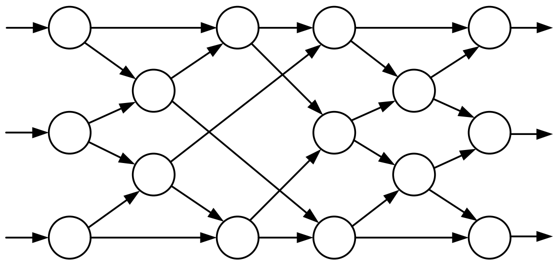


**Figure 7. Topology used for routing evaluation.**

Table 1 contains results obtained from such scenarios. We can see that using the static routing with the shortest path priority generates the worst results. The loss ratio can be greatly improved if dynamic routing is used or if longer paths are allowed to be chosen. What may be surprising is the fact, that the best results can be achieved for the bandwidth priority with d=1, but without the dynamic routing mechanism. In fact, the results scored by implementing the dynamic routing don't depend much on the routing priority.

**Table 1. Packet loss ratio PLR for different routing priorities**

|  | Static routing |  | Dynamic routing |  |
|---|---|---|---|---|
| **Priority** | **PLR** | **Conf. period** | **PLR** | **Conf. period** |
| Hop cnt. | 0,095610 | 0,004718 | 0,061996 | 0,006701 |
| Bw, *d*=0 | 0,088942 | 0,004672 | 0,077342 | 0,011955 |
| Bw, *d*=1 | 0,038858 | 0,008060 | 0,063055 | 0,012427 |

## 4.2 Preemption algorithms

Preemption is one of the traffic engineering methods and can be used in cases when there is not enough free bandwidth on selected route available. The remedy called preemption is to collect the missing bandwidth by removing some of the existing paths of lower priority. The procedure can be used if the new path has not the lowest priority and there are paths of lower priority with enough bandwidth available on the selected route.

It is possible, that there are many different sets of paths which satisfy the request. In fact it was proved, that the problem of selecting the best set of paths to be removed is NP-complete. For that reason, different heuristic algorithms have been developed. Currently we implemented two methods, one decentralized RFC [8] and one centralized KN [9]. As the two methods can be adjusted at different priorities, we performed two series of simulations for both methods. For the relocation count priority (REL) the aim is to minimize the number of preempted paths, whereas for the bandwidth priority (BW) the sum of preempted bandwidth must be minimized instead.

We performed simulations of several published real topologies [10] as well as a 100 nodes random network (fig. 8). In fig. 9 we present results of the combined metric, which is defined as a normalized value, which calculates both the number of preempted paths and the amount of preempted bandwidth. The value is greater for preemption methods that perform closer to an ideal, hypothetical method. The most important conclusion is the following: with growing network size the effectiveness of preemption methods declines. More results and analysis can be found in [11].

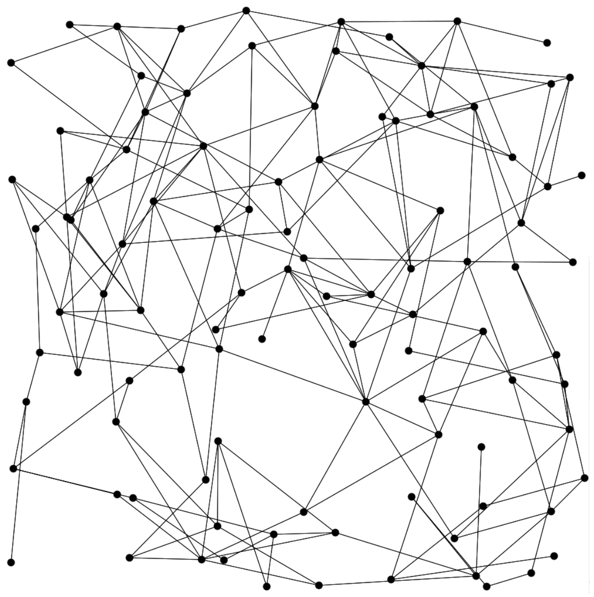


Figure 8. Random network of 100 nodes used in connection level simulations.

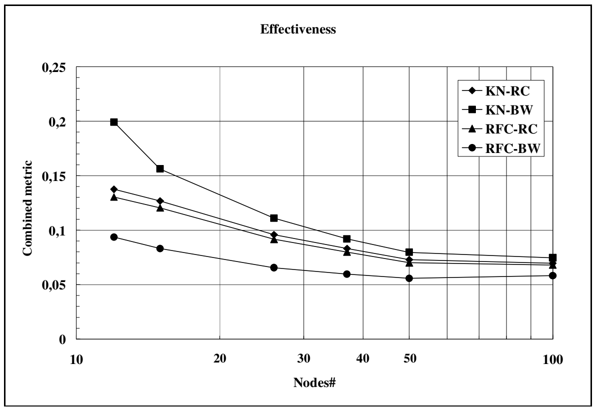
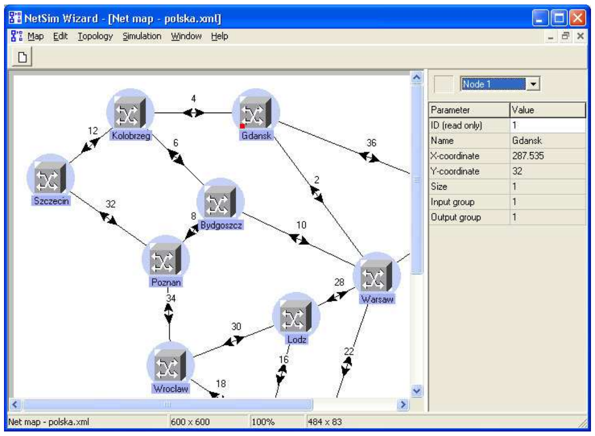


Figure 9. Performance of preemption methods for networks of different size.

## 5. VIMS: GUI FOR MSIM

When the simulated networks are bigger than several nodes, the process of creating the configuration file can become challenging. One problem is the large number of definitions and the second one is the growing chance for making mistakes. To overcome these problems, we developed the *Vims* – a visual extension to the *msim* application, which allows for creating the visual representation of a network and automate generation of proper configuration file. The main window of the application is shown in fig. 10.

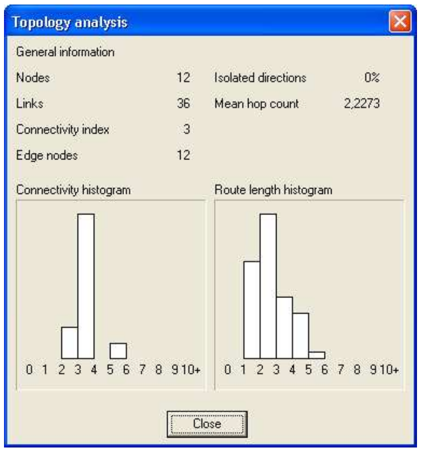


| Parameter | Value |
|---|---|
| ID (read only) | 1 |
| Name | Gdansk |
| X-coordinate | 287.535 |
| Y-coordinate | 32 |
| Size | 1 |
| Input group | 1 |
| Output group | 1 |

Figure 10. *Vims*: main window.

The main purpose of the application is to create network maps, configure parameters of the objects and create the configuration file based on a predefined template. When the configuration file is created, the simulation can be started directly from the main menu of *Vims*.

The *Vims* program can be used not only to generate fully defined networks. It also includes procedures to generate random networks using different modifications of the Waxman algorithm [12]. Additional useful feature of the application is topology analysis function that builds histograms of nodes connectivity and route length, like shown in fig. 11.



Topology analysis

General information

| | | | |
|---|---|---|---|
| Nodes | 12 | Isolated directions | 0% |
| Links | 36 | Mean hop count | 2,2273 |
| Connectivity index | 3 | | |
| Edge nodes | 12 | | |

**Figure 11. *Vims*: topology analysis window.**

The application has typical editing and file manipulation features. We used the XML file as a standard format for topology files. Its main advantages are invulnerability to slight format changes, i.e. some parameters can be added to the file by future version of the application and still the old files will be read without problems. Another useful feature of the XML format is the possibility to easily view or edit the contents with any text file editor.

To increase importing and exporting functionality, we introduced two additional features. A customized XML filter is used to read the network topologies gathered in the library of Zusse Instatut Berlin (ZIB) [10]. We also implemented network topology exporting as the SVG graphic format for further processing. The topology shown in fig. 8 was created in this way.

## 6. CONCLUSIONS

The simulation application presented in this paper is an effective tool used in research of traffic engineering methods and QoS analysis of MPLS networks. Its main advantages are universality, support for quick result presentation and iteration mechanism for performing a series of simulations. The graphical extension program *Vims* greatly simplifies creation of configuration files, especially for bigger networks, and offers a topology analysis feature. The simulator program is serving the authors as the main research tool for analysis of MPLS networks and the results have been published at several scientific conferences.

The implementation based on the object model makes the program open for new extensions and improvements. In fact, we constantly improve it and do not consider it as a finished work. Some of the current implementation plans include additional preemption algorithms, histograms generation and changes in configuration file syntax to allow building any buffering systems.

The applications described in the paper are available for download [13]. There are also example configuration files and a description provided.

## 7. REFERENCES

[1] Rosen, E., Viswanathan, A. and Callon, R. 2001. Multiprotocol Label Switching Architecture. RFC 3031, January 2001.

[2] Awduche, D., Berger, L., Gan, D., Li, T., Srinivasan, V. and Swallow, G. 2001. RSVP-TE: Extensions to RSVP for LSP Tunnels. RFC 3209. December 2001.

[3] Andersson, L., Minei, I. and Thomas, B. 2007. LDP Specification. RFC 5036, October 2007.

[4] The NS-2 main page. http://nsnam.isi.edu/nsnam/index.php/Main_Page.

[5] The OMNeT++ Discrete Event Simulation System. http://www.omnetpp.org/.

[6] OPNET Technologies, Inc. http://www.opnet.com/.

[7] The OMNeT++ User Manual. http://www.omnetpp.org/doc/manual/usman.html.

[8] Oliveira, J. de, Ed. 2007. Label Switched Path (LSP) Preemption Policies for MPLS Traffic Engineering. RFC 4829, April 2007.

[9] Kaczmarek, S. and Nowak, K. 2006. A New Heuristic Algorithm for Effective Preemption in MPLS Networks. In proceedings of the Workshop on High Performance Switching and Routing, Poznań 2006, 337-342.

[10] SND network library. Zusse Institut Berlin. http://www.sndlib.zib.de/.

[11] Kaczmarek, S. and Nowak, K. 2008. Performance of LSP Preemption Methods in Different MPLS Networks. In proceedings of the 5th Polish-German Teletraffic Symposium (Berlin, Germany, October 6-7, 2008). Logos Verlag Berlin GmbH 2008, 195-204.

[12] Zegura, E. W., Calvert, K. L. and Donahoo, M. J. 1997. A Quantitative Comparison of Graph-based Models for Internet Topology. IEEE/ACM Transactions on Networking. Vol. 3, Issue 6, Dec. 1997, 770-783.

[13] The msim application page, http://www.eti.pg.gda.pl/katedry/kst/pracownicy/Sylwester.Kaczmarek/badania/msim/index.html.

*This work was supported in part by the Polish National Central for Research and Development under project PBZ MNiSW – 02/II/2007.*