Object-Oriented Modeling and Simulation of Optical Burst Switching Networks

Joel J. P. C. Rodrigues, Nuno M. Garcia, Mário M. Freire
Department of Informatics
University of Beira Interior
Covilhã, Portugal
{joel, mario}@di.ubi.pt; ngarcia@ngarcia.net

Pascal Lorenz
IUT
University of Haute Alsace
Colmar, France
lorenz@ieee.org

Abstract— **Optical Burst Switching (OBS) is becoming an interesting technology for the optical Internet, since it does not need optical buffers like Optical Packet Switching (OPS), and is capable of a better performance than Optical Circuit Switching (OCS). Although OBS has been recently object of intense research, it still raises a number of important questions. Due to the high costs of an OBS network infrastructure, simulators are a good choice for predicting the behavior of this kind of networks. In this paper, we describe the proposal, implementation and validation of a simulator for OBS networks. The simulator, named OBSim, mimics the behavior of OBS networks in an Object-Oriented approach.**

Keywords-Modeling; Simulation tool; Optical Burst Switching

I. INTRODUCTION

Recent research activities in the area of Optical Burst Switching (OBS) [1-7] show that this technology may be probably an important part of the near future Internet. OBS is a compromise between Optical Circuit Switching (OCS) and Optical Packet Switching (OPS) [8], and may be implemented through various signaling protocols. OBS has some special characteristics [9], such as: i) Granularity: the size of a transmission unit in OBS is between OCS and OPS; ii) Data and control separation: control information is transmitted in a separate channel; iii) Unidirectional reservation: resources are reserved using a unidirectional messaging system (assuming one-way reservation scheme); iv) Burst with variable size: the size of each burst may not be fixed [10]; v) No buffering of data: once de data is sent, it must reach destination only with the delay inherent to the medium – the propagation delay of the signal in the optical fiber (assuming that no limited buffering (e.g. fiber delay lines) is used).

OBS node equipment includes, besides the Optical Cross Connect (OXC), a signaling engine, that processes the bursts control messages, and the switching matrix. Up to now, several signaling protocols have been proposed. Here, we consider the following five signaling protocols with one-way reservation schemes: JIT [3], JET [1], Horizon [2], JumpStart [4], and JIT$^+$ [6]. These protocols follow two types of reservation of the OXC resources: immediate or delayed. JIT and JIT$^+$ perform immediate reservation of the wavelength at the OXC while JET and Horizon delay the reservation of the wavelength at the OXC just before the data burst arrives. JumpStart has a set of messages that allow the protocol to reserve the resources in the OXC either delayed or immediately. Some protocols are more demanding on the signaling engine hardware and software than others, but not necessarily more efficient.

OBS technology raises a number of significant questions, as to analyze the performance of different wavelength routing schemes, different signaling protocols, and relate these with user profiles and network topologies, considering various shapes of traffic load on each node. These questions may be answered with a simulator to mimic the behavior of an OBS network, given the inexistence of such networks in the real world. However, there are some OBS networks demonstrators reported in [4, 11, 12]. Previous works in optical networks simulators are based on packet traffic (e.g. IP networks), which is significantly different from the aggregated packet traffic in an OBS network, since bursts are transmitted through the OBS network in a transparent way, in the sense that the network does not recognizes neither the end of burst nor its content. Therefore, a new tool is needed in order to include the specific features of OBS traffic at the network layer. This paper proposes an object-oriented approach for the development of an OBS simulator.

The remainder of this paper is organized as follows. In section II we present an overview of the modeling and simulation techniques. In section III, we describe the design of OBSim simulator, including a detailed overview of its main characteristics, design considerations. Validation of the simulator results is discussed in section IV. Main conclusions are presented in section V.

II. OBS NETWORK MODELING AND SIMULATION

Regarding network simulation, research tools fall into three different categories: analytical tools, *in situ* measurements, and simulators [13]. OBSim is an event driven, stochastic, symbolic simulator. Event driven simulators are a class of models in which data flows to the pace of events of some type; other simulators may be activity driven or time driven, namely when the simulator responds to some kind of user interaction (internal or external, user initiated or not) and the last, when the

0-7803-8798-8/04/$20.00 ©2004 IEEE

software runs at the tick of a clock [14]. In OBSim, the events that run on the simulator are messages. These may be sent by the users, or generated at a node, as defined by the signaling protocols presented previously. Stochastic simulators, opposed to deterministic simulators, rely on random entities (usually random variables of numerical value) to simulate the randomness of real-life events. In OBSim, we used the Java class Random, who generates pseudo-random values (of several types), using a congruential algorithm. Pseudo random variables must pass two tests that certify, first, the homogeneity of its distribution, and second, the independence of the generated values [14]. Java class Random satisfies these conditions. Symbolic simulators use some type of symbols to copy the behavior of real elements. In OBSim, theses symbols are Java classes, which are instantiated as needed by the software, according to the input data provided initially by the user.

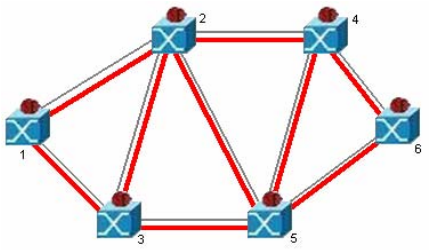

Figure 1. OBS network with 6 nodes and 9 links.

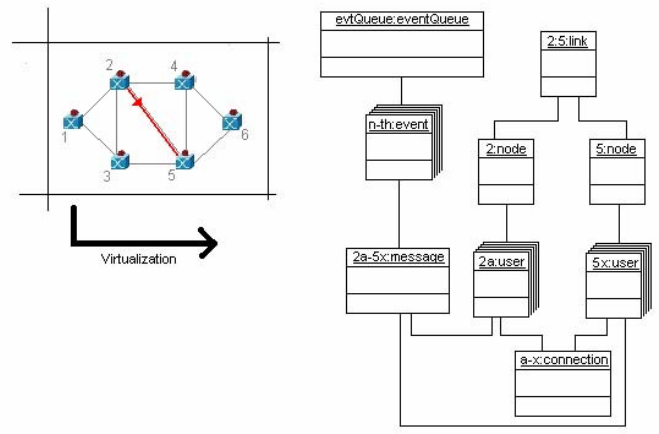

Figure 2. Classes instantiated when user 2a sends burst to user 5x.

Fig. 1 shows a simple example of an OBS network, used in this paper to support and present the discussed design and entities. We consider a network with 6 nodes (the OXC and its corresponding signaling engine, numbered from 1 to 6) and 9 links (data and signaling channels are shown separately). Fig. 2 presents a scheme and an Unified Modeling Language (UML) diagram that illustrates how a burst being sent from one user (user 'a' connected to node '2') to another user (user 'x' connected to node '5') deploys and uses a set of class instances in the simulator.

III. DESIGN OF THE OBSIM SIMULATOR

To study the problem and the characteristics of burst traffic in OBS networks, we need to evaluate the performance of different signaling protocols. This is achieved by studying its performance and behavior under different traffic conditions and network topologies. Our simulator mimics the behavior of a custom OBS network defined by the user. This simulator, called OBSim, allows us to assess and compare the performance of signaling protocols and load profiles to a given network topology. Previous works focus on simulating traffic for several types of networks and were primarily designed to simulate TCP/IP traffic. The main example is ns-2 (network simulator version 2) [15], developed on C++ and based on a project started in 1989, which has been widely used for network protocol performance studies [16]. While developing OBSim, we had access to the simulator developed by Teng and Rouskas from North Caroline State University [6, 7], and to the OBS-ns simulator released by DAWN Networking Research Lab from University of Maryland [17]. Both simulators were developed under C++ programming language. There have been also other developments in the area of simulation, such as OWns [16, 18], being this simulator an extension to the ns-2. In [19] we have found the IND Simulation Library, which is an object-oriented class library for event-driven simulation implemented in C++. These classes have been designed for support of performance evaluation of communication networks. Our OBSim is a tool that also gathers contributes from all these previous work in the area of network simulation.

A. *Objectives*

OBSim is designed to implement a model of OBS networks based on objects, that allows to estimate the burst loss probability performance of different signaling protocols, to study the influence of different network profiles on the performance of OBS networks, to evaluate the performance of OBS networks for network topologies defined by the user, and compare this with the performance of other technologies, and to test new signaling protocols, easily programmed in an Object-Oriented Programming (OOP) built model.

B. *Design Considerations*

As we needed a simulator independent of existing network data encapsulation protocols, we built OBSim from scratch. Java was the programming language chosen to build OBSim for several reasons, namely: 1) the quality and ease of use Java available programming tools; 2) the robustness of Java in object and memory handling; and 3) the wide platform portability of the code.

We made several assumptions while building OBSim. These assumptions occur in respect of the definition of the signaling protocols. Concerning network modeling, these are the following: 1) All the nodes work in an independent and similar way. 2) All time scales are normalized in time-slots. 3) A path is used by a burst or by a control message, independently of the state of the network. 4) When a signal (burst or signaling message) arrives to a node, it follows a predefined path calculated previously by the Dijkstra algorithm [20]. 5) Between two consecutive nodes, the wavelength used is chosen by the algorithm defined for the source node initially by the user: random or first-free [21].

Concerning the modeling of network traffic, the basic assumptions are: 1) Messages arriving to a node follow an

exponential distribution (or a variant of the exponential distribution) [22-26]. 2) Users are responsible for the burst generation process, that is, neither the ingress node nor any other node processes the burst. 3) One node may generate, at the most, one message by time-slot or time period. 4) Bursts are sent uniformly to every node in the network, with the exception that a node cannot send messages to itself. 5) The size of a burst is limited [10].

OBSim maintains an event queue that accepts and removes events, and forwards each event to its corresponding object (that with other objects compose the virtual network) so it can be processed.

C. *Abstraction in OBSim*

Abstraction in OBSim is achieved by the behavior of the objects of the model. The OBS network we want to simulate, is a set of defined real-world objects (eventually, real-world objects yet to be real-objects), each having its function and behavior, each interacting with the remain objects of the network according to a defined set of rules (e.g. the algorithms of the signaling engines). As an example, the network topology defined initially by the user is processed according to the Dijkstra Algorithm, and for each pair of nodes, is defined a route. Fig. 3 shows some of the routes found by the algorithm.

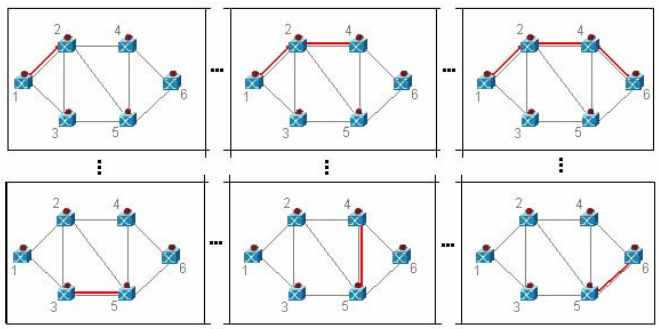

Figure 3. Six of the fifteen defined routes for topology of Fig. 1

D. *Components*

To simulate an OBS network, we defined several objects (Java classes), each having methods that may be activated by other objects. These mechanisms, common in Object Oriented Programming (OOP) languages such as Java and C++, namely, inheritance, polymorphism, encapsulation, and also, dynamic instantiation and dynamic memory management, allow us to create a working model that behaves like the real OBS network would. Time flow is simulated through a queue of events, ruled by a clock, and is introduced forward. Fig. 2 and Fig. 4 show how a potentially existing OBS network is virtualized in Java classes. As shown, each link is composed of two nodes, and a node may belong to more than one link (e.g. node '5'). Along with node and link, other classes were created to model the behavior of an OBS network. Fig. 5 shows the UML class diagram with the most important classes of the simulator.

The main method is defined in *Obs* class and this calls several other objects, in particular, the following: i) *NetworkFactory*, that builds the network from the topology file; this process is the virtualization that builds the class Network; ii) *Network*, composed by Links and Nodes; iii) *RouteBuilder*, that builds a route for any two nodes, and stores these routes in the *PathTable*; this class implements the Dijkstra algorithm [20, 27]; iv) *PathTable* manages the paths defined by *RouteBuilder*; v) *Global* stores and manages the global constants and variables of the simulator; v) *Simulator*, which starts the burst request actions from each user at every node; vi) *EventQueue* manages all the *Events* according to the *Clock* class; vi) And the classes *ErrorMsg* and *DebugMsg*, which manage the output of the debugging and error messages.

It may also be seen that the Node class is a generalization for classes *NodeJIT*, *NodeJET*, *NodeHorizon*, *NodeJumpStart*, and *NodeJITP*, which in turn, model the nodes of these signaling protocols. With this approach, the addition of a new protocol can be made by defining a new class and implementing it with its own set of specific algorithms.

NetworkObject is the generalization class for *Node*, *User* and *Event*. These are the main actors of an OBS network, and their behavior and interaction, as seen before, creates instances of classes like *Link*, *Connection*, *Event* and *Message*. In a non-parallel approach, classes *NetworkBuilder*, *Network*, *Simulator*, *EventQueue* (and *Clock*), *RouteBuilder* and *PathTable*, along with classes *ErrorMsg* and *DebugMsg* are instantiated only once. Classes *Node*, *Link* and *User* are instantiated as many times as defined in the network topology file. Other classes are instantiated as many times as needed, either by the stochastic workflow of the simulator, either by running algorithms like Dijkstra and WRAs.

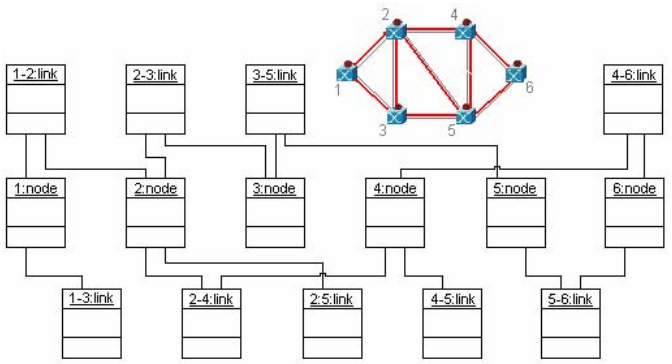

Figure 4. UML object diagram modeling an OBS network

E. *Session traffic generation*

Traffic generation is an important issue in the model. As we have an event driven simulator, we initially need to simulate the need to transmit bursts between nodes. As seen before, in a simulation, we assume that the bursts are sent evenly to every node in the network. Since every burst must be preceded by a signaling message, and since users connected to nodes send bursts at a random time, we considered that time between these messages follow an exponential distribution, simple or with an offset [24, 25]. The traffic is then simulated when the OBSim starts to process the event queue, which was, at the start of the program, loaded with requests (messages) from the users. These requests, when processed, normally generate more messages that are added to the queue. Each time a message is added to the queue, the simulator timer generates a time

interval according with the distribution defined by the user, and this time is added to the simulator clock, defining the time the event will be scheduled to happen.

F. Scenario generation

The network scenario is read from a text file that defines the number of nodes, the number of connections, the allowed attributes for each node, and the definition of the existing links in the network. The creation of the network abstraction – the network model that supports the simulation – is accomplished by the classes defined in the program. Fig. 4 is an UML diagram that partially illustrates the abstraction programmed in OBSim, for the example network showed in figure 1. In Fig. 5 we show the most important classes in the simulator. These classes are responsible for the virtualization of the model.

G. Input Interface

The input interface of the simulator allows the definition of several simulation attributes. The network is fully defined with these attributes and the attributes described for each node and each link in the text file mentioned earlier. The parameters used to configure the model of the OBS network are the following: Signaling protocol, Generation distribution function, Burst generation ratio, Available data channels per link, Switch signal process time, Switch setup time, User to node delay, User timeout, and Network topology file.

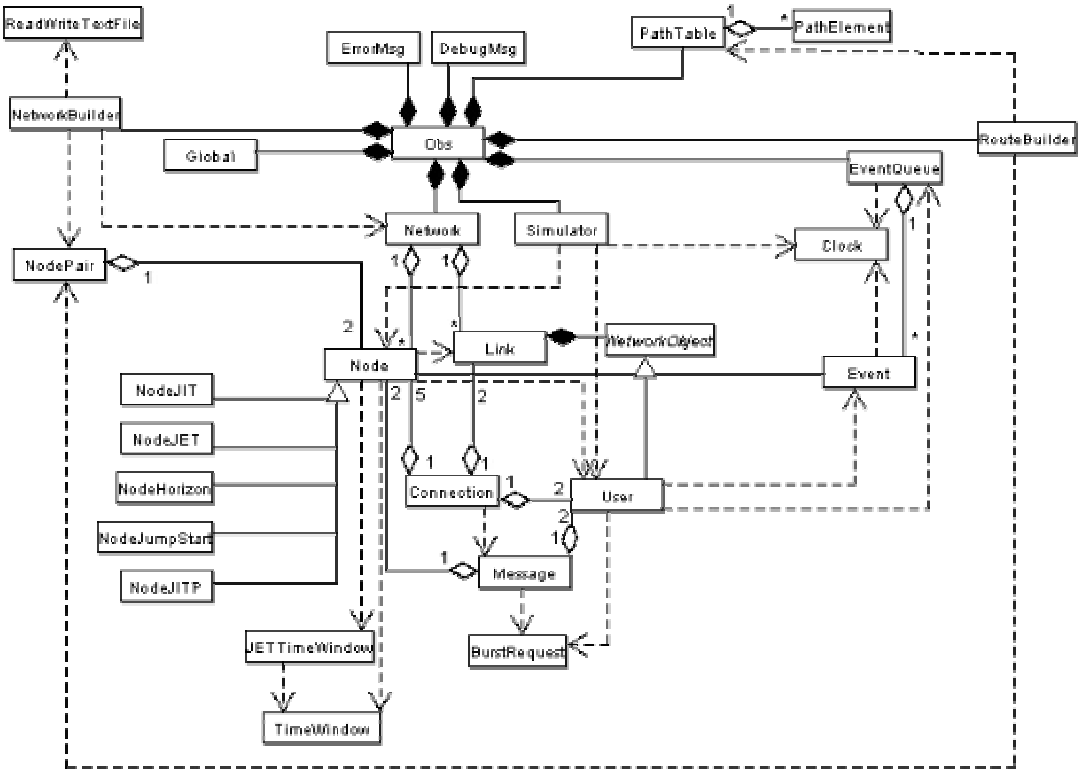

Figure 5. UML class diagram for OBSim

IV. SIMULATOR VALIDATION

Validation is a key issue to entrust the use of the results given by any simulator. Perros [14] defines the validation of the model as the verification of five steps: 1) Check the pseudo-random numbers generator. 2) Check the stochastic variable generator. 3) Check the logic of the simulation program. 4) Relationship validity. 5) Output validity.

In OBSim, the accuracy of the pseudo-random number generator is guaranteed by the Java language definition standards, and confirmed through the Qui-Squared test, and the Independence test performed on the Java class Random [14, 28]. The stochastic variable generators have been separately validated by [22, 25, 26]. The logic of the simulator and the validity of the relationships are inherent to the design of the signaling protocols, and to the Java programming environment, referred before. The output validity has been achieved through comparison with the results of [6]. For this purpose, we have run a sample simulation considering a single OBS node, in isolation, for JIT, JET, and Horizon signaling protocols. It is assumed that [6]: T_{OXC} = 10ms, T_{Setup}(JIT) = 12.5μs, T_{Setup}(JET) = 50μs, T_{Setup}(Horizon) = 25μs, the mean burst size 1/μ was set to 50ms, and burst arrival rate λ, is such that λ/μ = 32, assuming 64 users per node.

Fig. 7 shows the burst blocking probability as a function of the number of data channels per link for the OBS network presented above, given by OBSim and compared to the results presented in [6]. As may be seen in this figure, the results

obtained by OBSim are in a close range of those published by [6]. The small variation perceived is expectable because of the stochastic nature of the events that are modeled.

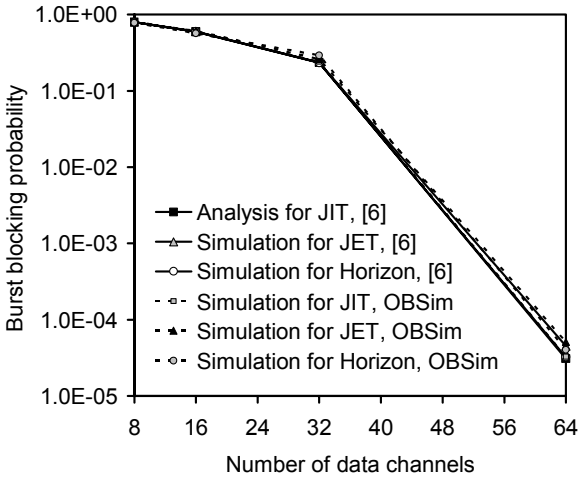

Figure 6. Burst blocking probability, as a function of number of data channels per link (*F*), in a single OBS node for JIT, JET and Horizon signaling protocols given by OBSim compared to results in [6]

V. CONCLUSIONS

In this paper we presented the objectives, design, implementation and validation of a simulator for OBS networks, we named OBSim. This simulator implements a model of OBS networks based on objects, which allows measuring the performance of a custom designed OBS network. The results of the simulator have been validated, and thus the simulator may be used as a tool to predict the behavior of the OBS networks.

ACKNOWLEDGMENT

Part of this work has been supported by the Group of Networks and Multimedia of the Institute of Telecommunications – Covilhã Lab, Portugal, in the framework of the Project PIONEER.

REFERENCES

[1] C. Qiao and M. Yoo, "Optical burst switching (OBS) - A new paradigm for an optical Internet," *Journal of High Speed Networks*, vol. 8, pp. 69-84, 1999.

[2] J. S. Turner, "Terabit burst switvhing," *Journal of High Speed Networks*, vol. 8, pp. 3-16, 1999.

[3] J. Y. Wei and R. I. McFarland, "Just-in-Time signaling for WDM optical burst switching networks," *Journal of Lightwave Technology*, vol. 18, pp. 2019-2037, 2000.

[4] I. Baldine, G. Rouskas, H. Perros, and D. Stevenson, "JumpStart - A Just-In-Time Signaling Architecture for WDM Burst-Switched Networks," *IEEE Communications Magazine*, vol. 40, pp. 82-89, 2002.

[5] C. Qiao and M. Yoo, "Choices, Features and Issues in Optical Burst Switching," 1999.

[6] J. Teng and G. N. Rouskas, "A Detailed Analisys and Performance Comparison of Wavelength Reservation Schemes for Optical Burst Switched Networks," *Submited for publication*, 2003.

[7] J. Teng and G. N. Rouskas, "A Comparison of the JIT, JET, and Horizon Wavelength Reservation Schemes on A Single OBS Node," presented at WOBS 2003, Dallas, Texas, 2003.

[8] C. S. R. Murthy and M. Gurusamy, *WDM Optical Networks, Concepts, Design and Algorithms*. New Jersey: Prentice Hall PTR, 2002.

[9] L. Xu, "Performance Analysis of Optical Burst Switched Newtorks," in *Department of Computer Science*. Raleigh: North Carolina State University, 2002.

[10] C. Qiao and M. Yoo, "A Taxonomy of Switching Techniques," in *Optical WDM Networks - Principles and Practice*, K. M. Sivalingam and S. Subramaniam, Eds.: Kluwer Academic Publishers, 2000.

[11] I. Baldine, M. Cassada, A. Bragg, G. Karmous-Edwards, and D. Stevenson, "Just-in-Time Optical Burst Switching Implementation in the ATDnet All-Optical networking Testbed," presented at Globecom 2003, San Francisco, Ca, 2003.

[12] L. McAdams, I. Richer, and S. Zabele, "TBONE: TestBed for all-Optical NEtworking," presented at IEEE/LEOS Summer Topical Meetings, 1994.

[13] P. Bartford and L. Landweber, "Bench-style Network Research in an Internet Instance Laboratory," *ACM SIGCOMM Computer Communications Review*, vol. 33, pp. 21, 2003.

[14] H. Perros, "Computer Simulation Techniques: The definitive introduction!," (at December 18th, 2003): North Carolina State University, http://www.csc.ncsu.edu/faculty/perros/, 2003.

[15] Dawn Networking Research Labs, "The network simulator ns-2," (at January 10th, 2004), http://www.isi.edu/nsnam/ns/, 2002.

[16] N. M. Bhide and K. M. Sivalingam, "Design of OWns: Optical Wavelength Division Multiplexing (WDM) Network Simulator," presented at First SPIE Optical Networking Workshop, Dallas, TX, 2000.

[17] DAWN Networking Research Lab, "DAWN Research Lab," (at January 15th, 2004): DAWN Research Lab, http://dawn.cs.umbc.edu/, 2004.

[18] B. Wen, N. M. Bhide, R. K. Shenai, and K. M. Sivalingam, "Optical Wavelength Division Multiplexing (WDM) Network Simulator (OWns): Architecture and Performance Studies," in *SPIE Optical Networks Magazine*, vol. Special Issue on "Simulation, CAD, and Measurement of Optical Networks, 2001, pp. 16-26.

[19] Institute of Communication Networks and Computer_Engineering, "IND Simulation Library," (at February 17th, 2004): University of Stuttgart, http://www.ikr.uni-stuttgart.de/INDSimLib/, 2004.

[20] A. V. Goldberg and R. E. Tarjan, "Expected Performance of Dijkstra's Shortest Path Algorithm," 1996.

[21] L. Li and A. K. Somani, "Dynamic wavelength routing techniques and their performance analyses," in *Optical WDM Networks - Principles and Practice*, K. M. Sivalingam and S. Subramaniam, Eds.: Kluver Academic Publishers, 2000.

[22] S. Ma and C. Ji, "Modeling heterogeneous network traffic in wavelet domain," *IEEE / ACM Transactions on Networking*, vol. 9, pp. 634-649, 2001.

[23] A. W. Moore, "Measurement based management of network resources," in *Computer Laboratory*. Cambridge: University of Cambridge, 2002.

[24] V. Paxson and S. Floyd, "Wide Area Traffic: The Failure of Poisson Modeling," *IEEE Transactions on Networking*, vol. 3, pp. 226-244, 1995.

[25] A. Schäefer, "Self-Similar Network Traffic," (at February 3rd, 2004), http://goethe.ira.uka.de/~andreas/Research/Fractal_Traffic/Fractal_Traffic.html, 2003.

[26] W. T. Willinger and R. M. S. Sherman, "Self-similarity through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the source level," *IEEE / ACM Transactions on Networking*, pp. 71-86, 1997.

[27] Y. Alavi, G. Chartrand, L. Lesniak, D. R. Lick, and C. E. Wall, *Graph Theory with Applications to Algorithms and Computer Science*. New York: John Wiley & Sons, 1985.

[28] Sun Microsystems Inc., "Class Math," (at December 12th, 2003), http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Math.html/, 2003.