Optical Wavelength Division Multiplexing (WDM) Network Simulator (*OWns*): Architecture and Performance Studies

Bo Wen, Nilesh M. Bhide, Ramakrishna K. Shenai, and Krishna M. Sivalingam *
School of Electrical Engineering & Computer Science
Washington State University, Pullman, WA 99164

Abstract

Optical wavelength division multiplexing (WDM) networking technology has been identified as a suitable candidate for future wide area network (WAN) environments, due to its potential ability to meet rising demands of high bandwidth and low latency communication. Networking protocols and algorithms are being developed to meet the changing operational requirements in future optical WANs (OWAN). Simulation is used in the study and evaluation of such new protocols, and is considered a critical component of protocol design. In this paper, we present an optical WDM network simulation tool called the optical WDM network simulator (*OWns*), that facilitates the study of switching and routing schemes in WDM networks. *OWns* is designed as an extension to the network simulator *ns2*, a multi-protocol network simulator that is widely-used for networking research and available in the public domain. Our goal is to incorporate the key characteristics of WDM networks in the simulator, such as optical switching nodes, multi-wavelength links, virtual topology constructions, related switching schemes and routing algorithms. This paper presents the architecture and design of the simulator, and a representative performance analysis to demonstrate how the simulator can be used. The software is available at [1].

1 Introduction

Advances in optical wavelength division multiplexing (WDM) technology [2, 3], which partitions the available bandwidth into a number of manageable smaller bandwidth channels, have made it a key technology for deployment in future network environments. There is considerable research and commercial activity to achieve optical WDM based networks.

Simulation plays an important role in network protocol design, providing researchers with a cost effective method to analyze and study the behavior of proposed protocol models. However, a lack of uniformity in the choice of simulation platforms for optical WDM networks makes it difficult for researchers to exchange and compare obtained results under a common simulation environment. To address this need, we developed a WDM network simulation tool, called optical WDM network simulator (*OWns*). An earlier version of the simulator was presented in [4]. Since then the simulator architecture has been redesigned as described later. In this paper, we present the details of this revised *OWns* architecture, design and implementation issues and describe the related validation techniques.

*Corresponding Author: Dr. Krishna M. Sivalingam. The work was supported in part by Cisco systems, San Jose, CA. An earlier version of this work was presented at the First Workshop on Optical Networks, Dallas, TX, Jan. 2000. The authors can be reached at {bwen,nbhide,rshenai,krishna}@eecs.wsu.edu.

OWns is designed as an extension of the network-simulator (*ns2*) (also referred to as *ns* in this paper), a network simulator built as a part of the Virtual Internet (VINT) project [5]. The *OWns* architecture encompasses the key characteristics of WDM networks including optical switching nodes, multi-wavelength links, routing and wavelength assignment (RWA) algorithms. *OWns* views the physical and logical topology of WDM networks being implemented as the *physical layer* and the *logical layer* respectively. *OWns* adopts a certain level of abstraction to build the specific switching schemes of WDM networks (e.g. circuit switching) based on the packet switching framework of *ns*. Additionally, a new class of traffic sources termed the session traffic is implemented to generate traffic sessions suitable for WDM circuit switching simulations. The simulator has been designed to allow addition of other RWA algorithms.

The simulator is validated by implementing existing algorithms in the *OWns* framework, followed by a comparison with simulation results produced by alternate implementations. The paper presents a set of representative results that demonstrate the utility of the tool. The latest implementation of *OWns* is publicly available at [1].

The rest of the paper is organized as follows. Section 2 discusses the major issues encountered in WDM network simulator design. Section 3 presents the *OWns* architecture, the design and implementation techniques. Section 4 describes the simulation of a circuit-switched optical network using *OWns*. This is followed by the validation results obtained by testing existing algorithm on *OWns*. Section 5 concludes the paper and indicates directions for future work.

2 Issues of optical WDM network simulator design

In this section, we discuss the major issues encountered in the WDM network simulator design, which include the requirement for a uniform WDM network simulation environment.

2.1 A uniform simulation environment

Simulation has become an indispensable tool in WDM network research (and in networking research, in general). It helps researchers to quickly and inexpensively evaluate the performance of new protocols. Simulation packages and tools that model the physical layer characteristics are available [6]. The focus of this work is on the network layer (i.e. routing, wavelength assignment and related problems). Almost all prior work on these aspects of WDM networks have been based on simulation models designed specifically to that problem. Since each group used its own simulation platform and assumptions, it is difficult to reuse existing protocol modules and compare simulation results under a common simulation environment.

A uniform WDM network simulation environment that can provide the foundation for incorporating the key characteristics of WDM networks becomes essential. This would provide WDM researchers with substantial benefits, which includes reusing existing protocol suites and simulation components, an open framework for easily accommodating new protocols and the characteristics of rapidly developing WDM technologies, and easier comparison of results across research efforts.

2.2 Design considerations

Existing network simulators provide a variety of functionality. Hence, instead of designing *OWns* from scratch, we tried to utilize capabilities provided by existing network simulators. We studied the designs and functionality of network simulators such as *ns* (available from UCB/LBN), *REAL* [7] (available from Cornell University), *NEST* [8] (available from Columbia University), *OPNET* [9] (commercially available from MIL 3), and *BONeS* [10] (commercially available from Cadence Design Systems). The increasing use of *ns* by network researchers, and its extensible design influenced the choice of using *ns* as the base platform for *OWns*.

Like any network simulator, *ns* has two key components: (i) *Building blocks* such as nodes, links, traffic models, and existing protocol suites, and (ii) *Glue* in the form of simulation description languages that configure simulation scenarios. A common problem encountered by network simulators is that the *building blocks* and the *glue* are often implemented with a uniform programming model. This uniform programming model generally cannot satisfy the diverse requirements of both components at the same time. Another common problem is the inflexibility of the description language when it comes to defining dynamic network simulation scenarios (e.g. *REAL's NETLanguage* which defines static configuration files and *OPNET's* schematic capture mechanism). *ns* tries to avoid these problems by using a split programming model for the implementation of these components. It uses C++ to implement efficient *building blocks* as well as transmission mechanisms, and a scripting language OTcl, an object-oriented extension of Tcl [11] (from MIT/LCS) as a *glue*.

Since *OWns* is based on the *ns* framework, we are be able to utilize most of the important features provided by *ns*, such as the capability to reuse existing components and the rich infrastructure for accommodating new protocols and characteristics.

3 Design of *OWns*

In this section, we present the design and implementation specifics of *OWns*. We focus on the details of the *OWns* architecture and components followed by an examination of the abstraction technique used to implement circuit-switched and large-scale scenario simulations. This section also introduces the session traffic source, suited for WDM circuit-switched simulations. Simulation scenario generation and visualization tools are discussed at the end of the section.

3.1 *OWns* architecture and components

This section discusses in detail, the *OWns* architecture in detail which includes the components that have been implemented in the current version of *OWns*.

3.1.1 Switching architecture of *OWns*

The *OWns* architecture is designed to accommodate specific characteristics of WDM network simulations. As shown in Figure 1 the *OWns* architecture views the physical and logical topology of WDM networks

being implemented as the *physical layer* and the *logical layer*, respectively. The *physical layer* consists of optical switching nodes and multi-wavelength links. Packet transmission mechanisms are implemented at the physical layer. The *logical layer* comprises the routing module and wavelength assignment (WA) module, which together create and maintain the virtual topology. These modules are described further in Section 3.1.2.

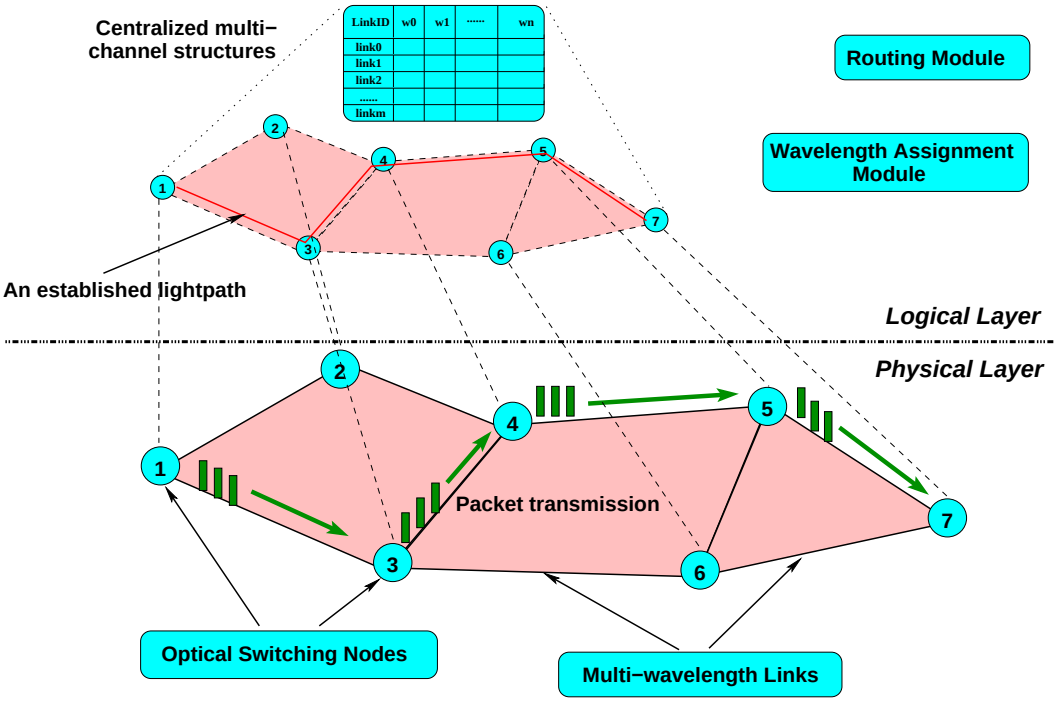

Figure 1: *OWns* architecture and layers

The *OWns* circuit-switched architecture is composed of the routing module, the WA module, optical switching nodes, and the multi-wavelength links. The multi-channel structures of multi-wavelength links are centrally maintained in the *logical layer*. The WA module works along with the routing module to compute wavelength assignment, set up lightpaths, and construct the virtual topology. Relying on the above results, optical nodes forward incoming traffic to the corresponding next hops through multi-wavelength links.

The current version of *OWns* supports circuit switching. The development of specific optical burst switching [12], photonic packet switching [13] and multi-protocol lambda switching [14] switching modules is planned for future work.

3.1.2 Components of *OWns*

Figure 2 illustrates component organization and interactions of *OWns*. The optical switching node, multi-wavelength link, routing module, and WA module are implemented as the `WDMNode`, `duplex-FiberLink`, `RouteLogic/Wavelength`, and `WAssignLogic` objects respectively. The session traffic source objects that are used to generate packets are composed of a transport-level agent, `Agent/WDM` and an application-

level traffic source, `Application/Session Traffic`. These are discussed in a later section.

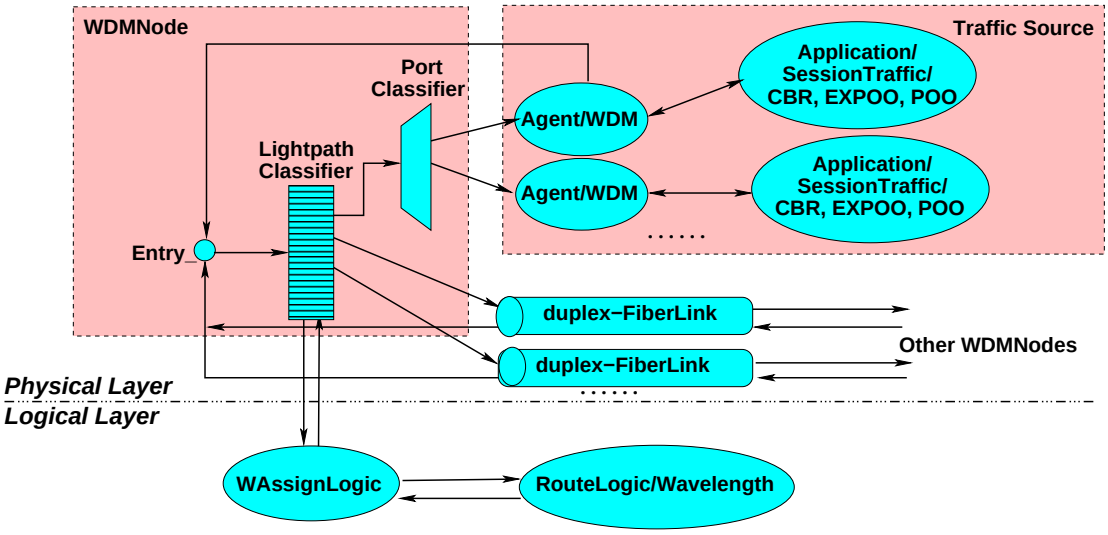

Figure 2: *OWns* Components Organization and Interactions

- **Optical switching node:** The `WDMNode` object is derived from the *ns* `Node` object, and it represents an instance of the optical switching node. It consists of a port classifier and a lightpath classifier. The port classifier [15] de-multiplexes and relays incoming packets to their respective sinks, which are the `Application/SessionTraffic` object. The lightpath classifier interacts with the `WAssignLogic` object to establish lightpaths for incoming traffic sessions and updates the current state of the virtual topology. The lightpath classifier at a source node always attempts to issue lightpath requests for its generated traffic to the `WAssignLogic`. On the other hand, a lightpath classifier at an intermediate node of the traffic path simply consults the `WAssignLogic` to determine forwarding information. The lightpath classifier also simulates the delay introduced by wavelength conversion. The lightpath classifier is implemented as the object `Classifier/Addr/Lightpath`.

- **Multi-wavelength link:** The duplex link of *ns* is extended to form the duplex multi-wavelength link, `duplex-FiberLink`. It has additional properties, such as the number of wavelengths and per-wavelength bandwidth, used to model the characteristics of optical links. Another major difference from traditional links is the absence of queuing components in multi-wavelength links due to the characteristics of lightpath communication. As described in section 3.1.3, an abstraction technique is adopted to reduce the complexity of implementing detailed multi-channels on multi-wavelength links. All multi-channel structures, wavelength usage, and virtual topology information are centrally maintained by the `WAssignLogic`.

  Error models for the links that allow transmission errors to be simulated are left for future work.

- **Wavelength assignment (WA) module:** The WA module is responsible for computing wavelength assignment, establishing lightpaths and constructing virtual topologies. The simulator represents the

WA module as an object `WAssignLogic`, which stores information necessary for wavelength assignment calculation. The *first-fit* [16] wavelength assignment algorithm is implemented in `WAssignLogic` as the default wavelength assignment mechanism in *OWns*. Implementation of a new wavelength assignment algorithm can be done by using classes inherited from `WAssignLogic`.

- **Routing module:** The routing module computes the routes needed to establish lightpaths, using certain specified routing algorithms. Since the routing algorithms of traditional networks and those of WDM networks are similar in functionality, the routing module is implemented as the object `RouteLogic/Wavelength`, which is inherited from the *ns* routing logic `RouteLogic` object. The simulator centrally maintains the routing information across networks in the `RouteLogic/Wavelength` object. *OWns* currently uses the *fixed-alternate shortest path* routing algorithm as the default routing. The `RouteLogic/Wavelength` supplies the `WAssignLogic` with route information to route incoming traffic. New wavelength routing algorithms can be easily implemented as derived classes based on the `RouteLogic/Wavelength` object.

3.1.3 Abstraction in *OWns*

Abstraction [17] is a useful technique that helps in implementing large-scale scenario simulation and complex protocols. By abstracting necessary details, memory and time consumption can be greatly reduced without sacrificing accuracy. In WDM circuit-switched scenarios, each lightpath needs to be mapped by a sequence of continuous (or converted) channels (wavelengths) on the physical multi-wavelength links across all intermediate nodes between a source and a destination node. Wavelength routing involves making decisions for every channel on every link. As a result, the simulator has to faithfully implement the detailed transmission of every packet on its corresponding channel, which inevitably causes a large amount of resource consumption. In *OWns*, we adopted the abstraction technique to achieve satisfactory performance results.

Our abstraction technique is the simplified implementation of circuit switching based on the packet switching framework. Instead of constructing detailed multi-channel structures on multi-wavelength links, centralized structures are implemented in the WA module that is maintained by the simulator. The multi-wavelength links simply carry out packet level transmission without knowledge of the multi-channel and virtual topology. As a result, high performance WDM networks with more than 64 wavelengths are efficiently simulated under restricted environment, and while a significant amount of memory and time is conserved by this abstraction.

3.2 Session traffic generation

Traffic generation plays an important role in simulation for proper understanding of system performance. In *ns*, existing traffic sources such as `CBR`, `Exponential`, and `Pareto` are designed to be suited for packet switching simulations. Thus, in order to facilitate circuit switching simulations, we need to modify the existing traffic sources to generate session traffic, which is most appropriate for WDM circuit switching simulations.

To describe a session traffic, we introduce two parameters: mean session arrival rate (`msar_`) and mean session holding time (`msht_`). Both are measured in terms of the traffic between a source-destination pair. Traffic session arrivals are modeled by a Poisson distribution with mean session arrival rate (`msar_`) as mean, and the holding time of each session has an Exponential distribution with mean session holding time as `msht_`. The product of mean session arrival rate and mean session holding time represents the traffic load measured in Erlangs. For each session, the packet inter-arrival time distribution is taken to be constant bit rate (CBR), Exponential, or Pareto. Therefore, in addition to specifying mean session arrival rate and mean session holding time for each session traffic, we also need to specify the parameters for each session depending on the packet arrival characteristics. For example, if a session traffic has exponential packet inter-arrival time, we need to specify mean packet size, mean packet rate, mean burst size and mean idle time as parameters.

The basic idea of generating session traffic is to control the start and stop time for each session by using the specified session arrival rate (`msar_`), and session holding time (`msht_`). When a session starts, its stop time is scheduled at `nextOffTime_` according to its session holding time. Similarly, when a session stops, its next start time is scheduled at `nextOnTime_` according to its session arrival distribution. We also noted that the packet rate of a session usually differs from others even in the same traffic session. We define the mean packet arrival rate of a session as (`rate_`). We chose the packet rate for each traffic session from a uniform distribution between (`rate_/2`) and (`rate_*2`). This way, using only one session traffic generator in a source and destination pair, we are able to simulate multiple session traffic that were needed for circuit switching based simulations.

In *OWns*, we developed a class of session traffic source objects namely `Application/SessionTraffic/CBR`, `Application/SessionTraffic/Exponential`, and `Application/SessionTraffic/Pareto`, which are designed to generate three types of session traffic. Traffic is identified and classified by its unique flow id, `fid_`. The `WAssignLogic` stores all active traffic session information, such as rate, flow id, traffic source and destination. Meanwhile, session traffic sources constantly send their session updates to the `WAssignLogic`. When a traffic session starts, it registers with the `WAssignLogic` object, and then unregisters when the session ends.

3.3 Scenario generation

We extend the scenario generation tool of *ns* to support the characteristics of WDM network simulations such as multiple wavelengths on each link, wavelength conversion parameters and traffic features. Scenario generation tools in *OWns* are mainly composed of two parts: (i) the *topology generator* and (ii) the *traffic generator*.

The *topology generator* provides the ability to create random topologies according to a set of specified parameters (e.g. degree of connectivity). By default, all generated multi-wavelength links have the same wavelength number and all wavelengths have the same bandwidth. To enhance flexibility, *OWns* also provides an alternate way to manually configure the network topology and connectivity. In this way, we can create the topology like NSFNET, and vary the wavelength number and bandwidth on the link. The following example creates a duplex multi-wavelength link between a source(`$src`) and a destination(`$dst`) with

the total bandwidth `$linkBW`, link delay `$linkDelay` and number of wavelengths(`$wvlenNum`).

```
$ns duplex-FiberLink $src $dst $linkBW $linkDelay Null $wvlenNum
```

While simulating WDM backbone networks, it is not sufficient just to generate sessions between fixed source-destination pairs. The *traffic generator* of *OWns* capable of generating random traffic source-destination pairs for a given WDM network topology according to the specified traffic models and related parameters. For a given WDM network topology, the *traffic generator* randomizes source and destination pairs according to their uniform distribution. As a result, traffic could flow between any source-destination pair.

3.3.1 Visualization tool of *OWns*

nam is an animation tool that reads the trace output generated by *ns* and produces a visualization. A specific extension to *nam*, *OWnam*, is being developed to address the needs of visualizing WDM network simulation scenarios based on OWns.

In *OWnam*, each traffic flow is still visualized by the conventional packet animation approach supported by *nam*. In order to support the characteristics of WDM networks, the *OWnam* extension introduces two components: the *events monitor* and *virtual topology statistics*.

The *events monitor* is used to capture and display dynamic events that occur in virtual topologies, such as lightpath requests for arrived traffic sessions, lightpath establishment and tear-down, etc. The *events monitor* can be activated as a separate window or as an annotation panel of main window. By clicking on a particular event, the playback can be immediately switched to the context where the event occurs. This allows us to detect and examine interesting occurrences.

The conventional animation component of *nam* only displays the details of data packet transmission over the simulated physical topology, and the dynamic information associated with the virtual topology cannot be visualized and presented to viewers. The *virtual topology statistics* component is designed to bridge this gap. The dynamic information of the virtual topology involves the state of lightpath establishment, wavelength usage on multi-wavelength links, etc. The *virtual topology statistics* component provides two ways to display this information. First, clicking on any of displayed links will pop out a one-shot panel showing the information for this link, including current wavelength usage and lightpaths established over this link. Second, a separate window is used to display the dynamic virtual topology.

4 Simulation Scenario and Results

In this section, we present an example based on circuit-switching to illustrate the usage of *OWns* as well as its validation by evaluating the performance of virtual topology construction algorithms for WDM networks. We describe a simulation example based on circuit switching to illustrate the usage of *OWns* and to validate the performance results obtained from testing existing algorithms.

4.1 Simulation scenario configuration

In this simulation example, we evaluate the RWA algorithm with *fixed-alternate shortest path* routing and *first-fit* wavelength assignment. We also enable the wavelength conversion capability for the nodes to study its effect on the network performance. *Sparse wavelength conversion* [18] scheme is used to model the wavelength conversion capability of a network. The wavelength conversion factor (`wvlen_conv_factor`) denotes the percentage of nodes in a network that have conversion capability. The wavelength conversion distance (`wvlen_conv_dist`) represents the limited-range wavelength conversion [19] capability of a node.

In the example, for simplicity, we utilize the scenario generation tool in *OWns* to generate random topology and traffic. Exponential session traffic is used in the simulation. We use the script presented in Figure 3 to invoke the topology generation tool and configure our simulation. The script mainly consists of two parts: the first part defines simulation variables and the second part conducts simulation configuration based on these simulation variables. A brief description of important variables and steps is given below[†].

In the first part of the script, as shown in the script file, *OWns* variables *wvln_routing* and *wvlen_assign* are assigned values `WDMStatic` and `FirstFit` respectively, to configure the routing and wavelength assignment algorithms. The parameters for topology generation are specified on lines 5 - 10 in the script file. The variables related to wavelength conversion start with a prefix *wvlen* and those related to traffic characteristics have *traf* as their prefix. The variables are self-explanatory and are explained by the comments following the variable definitions.

The second part of the simulation script invokes the topology and traffic generation tools and the commands to configure the simulation scenario. Lines 32-33 configure *OWns* to use the specified routing and assignment algorithms in the simulation. On lines 36-49, we utilize the topology and traffic generators to generate a WDM network topology and a random traffic. Lines 52-58 configure the start and stop time of the simulation and the traffic. The simulation is configured to stop either at the specified stop time, or when the number of received traffic requests reaches *traf_max_req* (in this case the stop time is set to zero). Before the simulation runs, `pre_run_wassignlogic`, on line 61, needs to be invoked to initialize the `WAssignLogic`.

[†] Conventional *ns* commands are mostly not shown in the script example since we focus our attention on those commands, which are specific to *OWns*. For the further information regarding *ns* commands, the reader is referred to the *ns* manual [15].

```
# Define Simulation Variables
set val(result_file)              "demo.res"     ;# result file
set val(wvlen_routing)            WDMStatic      ;# wvlen routing protocol
set val(wvlen_assign)             FirstFit       ;# wvlen assignment protocol
set val(shortest_path_num)         1              ;# number of the shortest path
set val(node_num)                 25             ;# total nodes number in network
set val(conn_prob)                0.11           ;# nodes connection prob.
set val(topo_seed)                98765          ;# seed to generate random topology
set val(link_bw)                  16Mb           ;# bandwidth of link
set val(link_wvlen_num)           16            ;# wavelengths number on each link
set val(wvlen_conv_factor)        0.5           ;# wvlen conversion factor, between 0 and 1
set val(wvlen_conv_dist)          4             ;# wvlen conversion distance, <= wvlen num
set val(wvlen_conv_time)          0.024         ;# wvlen conversion time (relative time)
set val(traf_density)              0.6           ;# generated traffic density in networks
set val(traf_arrival_rate)        0.5           ;# mean of each session arrival rate
set val(traf_holding_time)        1.0           ;# mean of each session holdingtime
set val(traf_pkt_size)            1000          ;# session-traffic packet size
set val(traf_pkt_rate)            1Mb           ;# session-traffic packet arrival rate
set val(traf_type)                Exponential    ;# session-traffic type in network
set val(traf_exp_burst_time)       0.7          ;# expoo traffic average burst time
set val(traf_exp_idle_time)        0.1          ;# expoo traffic average idle time
set val(traf_max_req)             1000          ;# max traffic requests number
set val(traf_start_time)          0.0           ;# session-traffic starting time
set val(traf_stop_time)           0.0           ;# session-traffic stoping time
......


# Create a simulator object
set ns [new Simulator]

# Wvlen routing protocol and assigning mechanism
$ns wrouting-proto $val(wvlen_routing)
$ns wassign-proto $val(wvlen_assign)

# Generate the topology creation script
topology -outfile $val(topofile) -nodes $val(node_num)
        -connection_prob $val(conn_prob) -seed $val(topo_seed)
# Generate the traffic creation script
traffic $val(traf_type) $val(node_num) $val(traf_num) $val(traffile)
......

# Create random topology
create-topology ns WDMNode SessionTrafficRcvr $val(link_bw) $val(link_wvlen_num)
        $val(wvlen_conv_factor) $val(wvlen_conv_dist) $val(wvlen_conv_time)
        $val(wvlen_alloc_path2) $val(util_sample_interval) $val(traf_max_req)
# Create random traffics
create-traffic ns traffic WDMNode SessionTrafficRcvr $val(node_num) $val(traf_num)
        $val(traf_pkt_size) $val(traf_pkt_rate) $val(traf_arrival_rate) $val(traf_holding_time)
        $val(traf_exp_burst_time) $val(traf_exp_idle_time)

# Schedule session traffics
for { set i 0 } { $i < $val(traf_num) } { incr i } {
        $ns schedule-sessiontraffic $traffic($i) $val(traf_start_time) $val(traf_stop_time)
}
# Schedule ns stop
if { $val(traf_stop_time) > 0 } {
        $ns at [expr $val(traf_stop_time) + 1.0] "finish"
}

# before ns runs, prepare wassignlogic
$ns pre-run-wassignlogic
$ns run
```

Figure 3: Script file used in simulation.

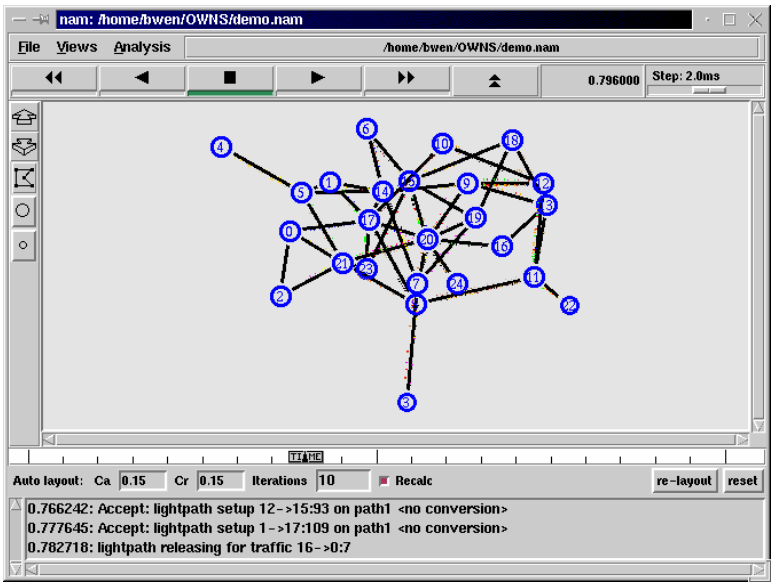

Figure 4: Topology of the WDM network used for simulation

Figure 4 shows the 25-node WDM network topology generated by the topology generator along with a snapshot of the simulation run. The bandwidth and the number of wavelengths on each multi-wavelength link are configured by the specified values. The propagation delay of each link is generated at random by the topology generator. The *Exponential* session traffic pairs are randomly distributed amongst all nodes, which are displayed by the packet flows in *OWnam*. In the snapshot, the update events of the virtual topology are captured and displayed in chronological order in the annotation panel of the main window by the *event monitor*. For instance, at 0.777645s, a lightpath is created for traffic session 109 from node 1 to node 17 and the lightpath is established on the shortest path (path1) between the source and the destination without wavelength conversion.

4.2 Simulation results

In order to validate *OWns*, we ran many simulations based on well known algorithms for various network topologies. We used randomly generated topologies and session traffic pairs to create a diverse set of scenarios. The topologies we tested had 25, 50, and 100 nodes, with each link having 16, 32, 48 and 64 wavelengths. Some typical simulation results obtained from simulations are presented.

Figure 5 and 6 show the results of performance measurement for two 100-node WDM networks, which have 48 and 64 wavelengths on each multi-wavelength link, respectively. The physical topology was generated by the *topology generator* with a connectivity probability of 0.03. The wavelength conversion distance was fixed at 4 in the simulations. Wavelength routing was performed on both the shortest path and the alternate path. The *Exponential* session traffic was used as the traffic source. We studied four major performance measures in the simulated WDM networks: blocking probabilities, average packet delays, average hop counts, and link utilization. The results in both figures are obtained from four cases that are described in the figures.

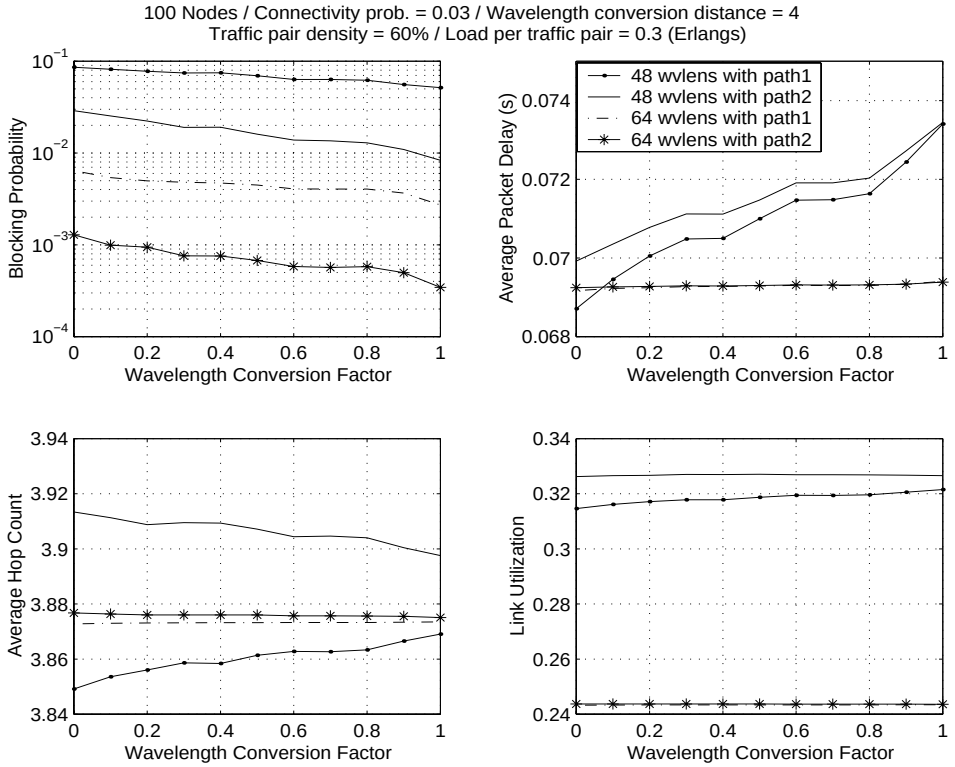

Figure 5: Effect of varying wavelength conversion factor on the network performance.

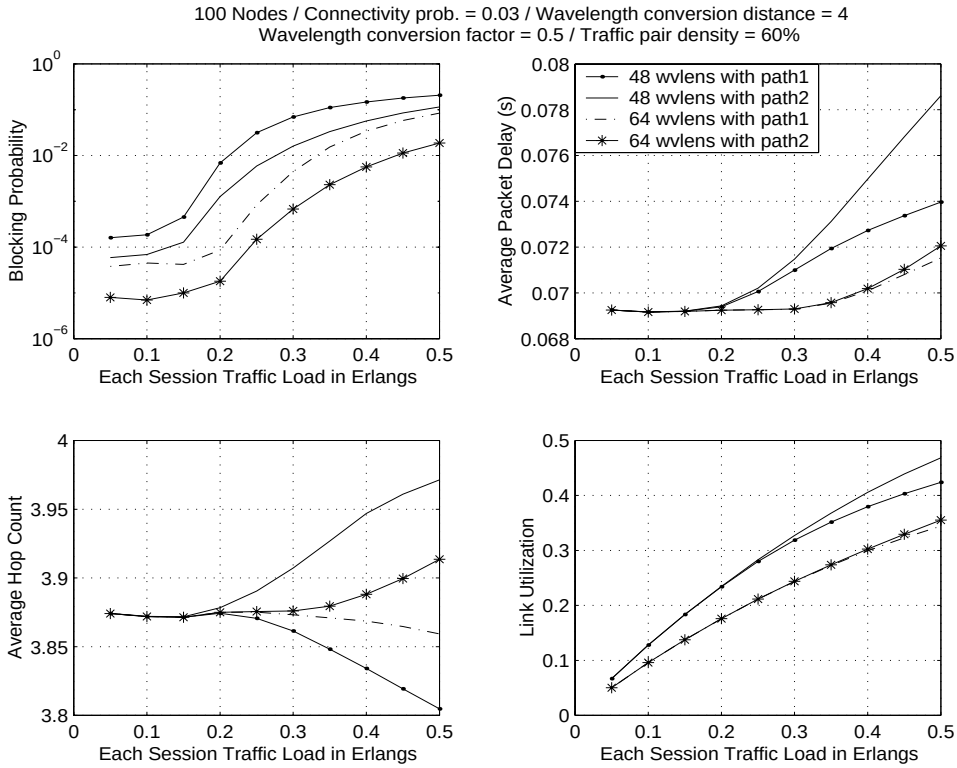

Figure 6: Effect of varying traffic load on the network performance.

Figure 5 presents the effects of varying the wavelength conversion factor on network performance. We generated random session traffic pairs with 60% traffic density. Each session traffic load was kept at 0.3 Erlangs. From the figure, we can easily see that by varying wavelength conversion factor from 0 to 1, all performance measures vary corresponding to their expected trends. Note that each of the four cases has different performance measures because of the different scenario variable specification.

Figure 6 shows the results of the effects of varying traffic load on the network performance. We kept the wavelength conversion factor at 0.5, and varied traffic load to see how they would affect performance measures. The traffic density of the random session traffic was fixed at 60%, and each traffic load was varied from 0.05 to 0.5 Erlangs. Similarly, we can conclude from the figures that all performance measures demonstrate expected trends with varying traffic load.

5 Conclusions

In this paper we presented the details of the design of *OWns*, a WDM network simulator based on *ns*. *OWns* provides the foundation for evaluating the routing and wavelength assignment algorithms in the OWAN environment. We also validated the simulator by running many simulations based on various scenarios. The results obtained from these simulations were found to be in accordance with the expected results. The usage of the simulator was demonstrated with the help of a sample simulation scenario based on a randomly generated WDM network topology.

We realize that *OWns* represents the initial steps in the implementation of a common simulation environment for WDM networks. Further development and improvement in *OWns* framework is warranted. Novel switching schemes such as photonic packet switching [13] and multi-protocol lambda switching [14] need to be implemented in the *OWns* framework. Also, support for multicasting in optical WDM environment needs to be provided. Furthermore, the basic *OWns* framework needs to be extended to provide support for QoS schemes such as Integrated services and Differentiated services. Finally, more realistic traffic sources for the wide area networks, e.g. self-similar traffic sources [20], should be integrated in *OWns* to achieve more accurate and valuable simulation results.

References

[1] K. Sivalingam, "DAWN – WSU Networking Research Laboratory: Optical WDM Network Simulator Software." http://www.eecs.wsu.edu/~dawn/software/owns.html, 2001. Email: krishna@eecs.wsu.edu.

[2] P. Green, "Progress in optical networking," *IEEE Communications Magazine*, vol. 39, pp. 54–61, Jan. 2001.

[3] K. Sivalingam and S. Subramaniam, eds., *Optical WDM Networks: Principles and Practice*. Boston, MA: Kluwer Academic Publishers, 2000.

[4] N. Bhide and K. M. Sivalingam, "Design of a WDM Network Simulator for Routing Algorithm Analysis," in *Proc. of First Optical Networking Workshop*, (Dallas, TX), Jan. 2000.

[5] "The Network Simulator - ns-2." http://www.isi.edu/nsnam/ns/index.html, 2000.

[6] B. Ramamurthy, D. Datta, H. Feng, J. P. Heritage, and B. Mukherjee, "Simon: A simulator for optical networks," in *Proc. of SPIE All Optical Networking: Architecture, Control and Management Issues*, vol. 3843, pp. 130–135, 1999.

[7] "REAL network simulator (version 5.0)." http://www.cs.cornell.edu/skeshav/real, 1998.

[8] "NEST network simulator (version 2.6)." ftp://ftp.cs.columbia.edu/nest, 1998.

[9] "OPNET network simulator." http://www.mil3.com, 1998.

[10] "BONeS network simulator (version 4.0)." http://www.altagroup.com, 1998.

[11] D. WetherHall and C.J.Lindblad, "Extending Tcl for Dynamic Objected-Oriented Programming," in *Proceedings of the Tcl/Tk Workshop*, (Ontario, Canada), July 1995.

[12] C. Qiao and M. Yoo, "Optical Burst Switching (OBS) - A New Paradigm," *Journal of Highspeed Networks, Special issue on WDM networks*, 1998.

[13] S. Yao, B. Mukherjee, and S. Dixit, "Advances in photonic packet switching: An overview," *IEEE Communications Magazine*, pp. 84–94, Feb. 2000.

[14] D. Basak, D. O. Awduche, J. Drake, and Y. Rekhter, "Multi-protocol Lambda Switching: Issues in Combining MPLS Traffic Engineering Control With Optical Cross-connects." Internet Draft, Feb. 2000. Expires August 2000.

[15] K. Fall and K. Varadhan, *The ns Manual*. The VINT Project, 2001.

[16] I. Chlamtac, A. Ganz, and G. Karmi, "Lightpath communications: An approach to high bandwidth optical WANs," *IEEE Transactions on Communications*, vol. 40, pp. 1171–1182, July 1992.

[17] P. Huang, D. Estrin, and J. Heidemann, "Enabling Large-scale Simulations: Selective Abstraction Approach to The Study of Multicast Protocols," in *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, (Montreal, Canada), IEEE, July 1998.

[18] S. Subramaniam, M. Azizoglu, and A. Somani, "All-Optical Networks with Sparse Wavelength Conversion," *IEEE/ACM Transactions on Networking*, vol. 4, no. 4, pp. 544–557, 1996.

[19] J. Yates, J. Lacey, D. Everitt, and M. Summerfield, "Limited-Range Wavelength Translation in All-Optical Networks," in *Proc. IEEE INFOCOM*, pp. 954–961, 1996.

[20] M. Yuksel, B. Sikdar, K. S. Vastola, and B. Szymanski, "Workload Generation for *ns* Simulations of Wide Area Networks and the Internet," in *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, (San Diego, CA), pp. 93–98, 2000.

Bo Wen is a M.S. candidate in Computer Science at Washington State University, Pullman. He received the B.E. degree in Electronic Engineering from Beijing Institute of Technology, Beijing, China in 1994 and the M.S. degree in Computer Science from the Sixth Research Institute of Ministry of Electronic Industry, China in 1997. His current research interests include Optical WDM networks and bandwidth management.

Nilesh Bhide is a Ph.D. candidate in Computer Science at Washington State University, Pullman. He received his B.Sc. degree in Physics and M.S. degree in Computer Applications from University of Bombay, India in 1993 and 1996 respectively. His research interests include architectures and protocols for optical WDM networks, and performance evaluation. He is a student member of IEEE and ACM.

Ramakrishna Shenai is a M.S. candidate in Computer Science at Washington State University, Pullman. He received his B.E. degree in Computer Science and Engineering from University of Madras, India in 1999. His research interests include Quality of Service (QoS) architectures for optical WDM networks and performance evaluation. He is a student member of the IEEE and ACM.

Krishna M. Sivalingam is Boeing Associate Professor of Computer Science in the School of Electrical Engineering and Computer Science, at Washington State University, Pullman, where he was an Assistant Professor from 1997 to 2000. Earlier, he was an Assistant Professor at University of North Carolina Greensboro from 1994 until 1997. He has conducted research at Lucent Technologies' Bell Labs in Murray Hill, NJ, and at AT&T Labs in Whippany, NJ. He received his Ph.D. and M.S. degrees in Computer Science from State University of New York at Buffalo in 1994 and 1990 respectively. While at SUNY Buffalo, he was a Presidential Fellow from 1988 to 1991. Prior to that, he received the B.E. degree in Computer Science and Engineering in 1988 from Anna University, Madras, India.

His research interests include wireless networks, optical wavelength division multiplexed networks, and performance evaluation. He has served as a Guest Co-Editor for a special issue of the IEEE Journal on Selected Areas in Communications on optical WDM networks. He is co-recipient of the Best Paper Award at the IEEE International Conference on Networks 2000 held in Singapore. He has published an edited book on optical WDM networks in 2000. His work is supported by AFOSR, Laboratory for Telecommunication Sciences, NSF, Cisco, Bellcore, Alcatel, Intel, and Washington Technology Center. He holds three patents in wireless networks and has published several papers including 18 journal publications. He has served on several conference committees including ACM Mobicom 2001, Opticom 2001, Opticom 2000, ACM Mobicom 1999, MASCOTS 1999, and IEEE INFOCOM 1997. He is a Senior Member of IEEE and a member of ACM. Email: krishna@eecs.wsu.edu

The figures are reproduced here, one per page, and expanded to full textwidth.

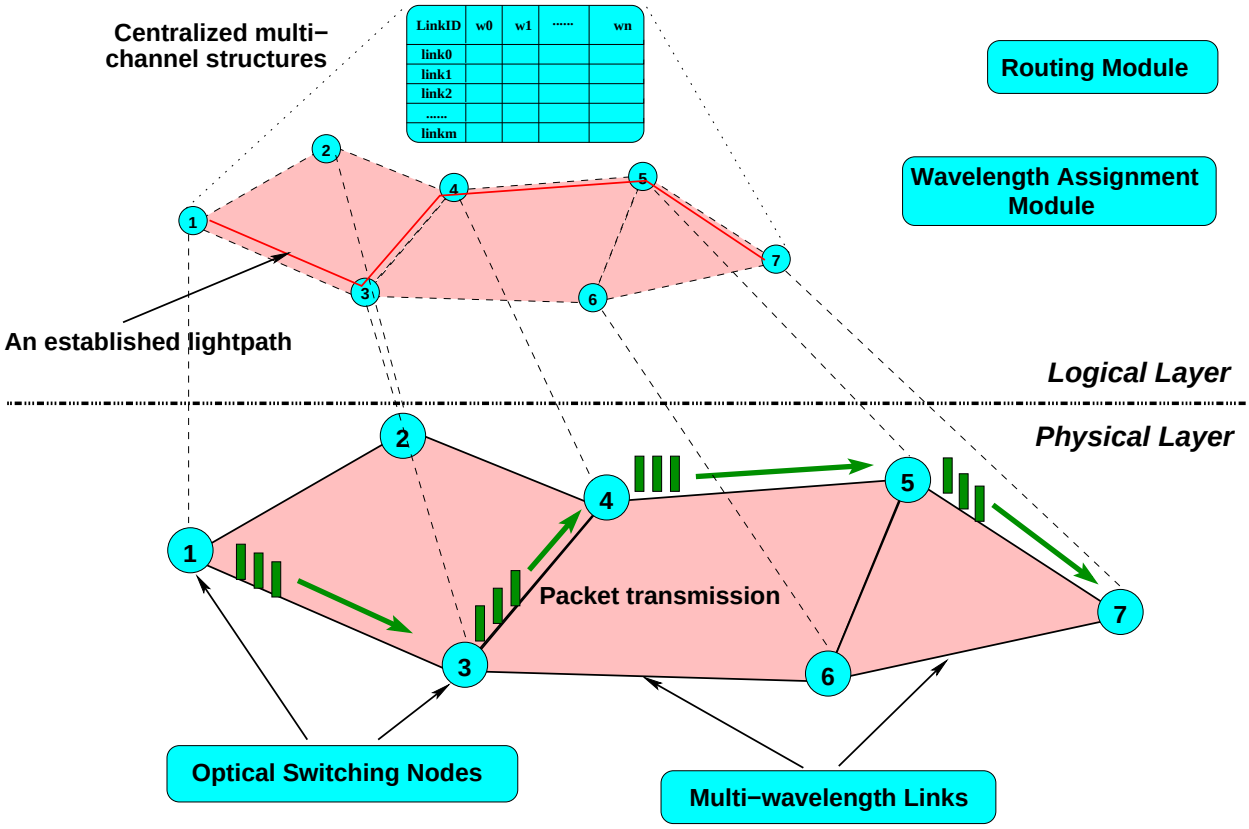

Figure 1: The *OWns* architecture and layers

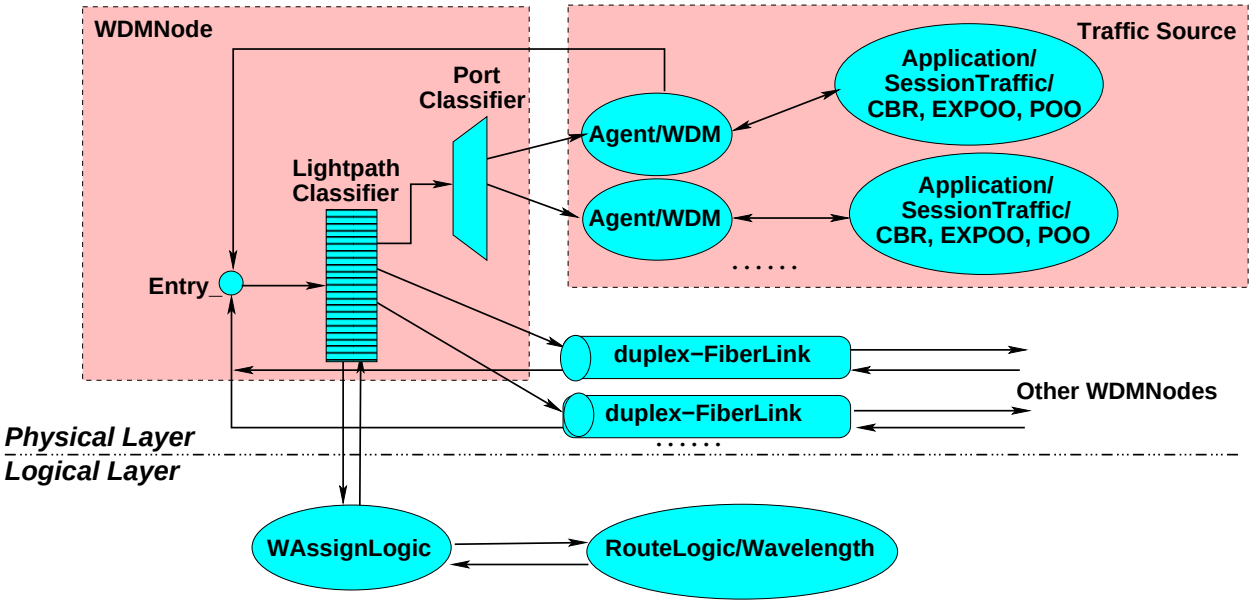

Figure 2: *OWns* Components Organization and Interactions

```
# Define Simulation Variables
set val(result_file)              "demo.res"     ;# result file
set val(wvlen_routing)            WDMStatic      ;# wvlen routing protocol
set val(wvlen_assign)             FirstFit       ;# wvlen assignment protocol
set val(shortest_path_num)         1              ;# number of the shortest path
set val(node_num)                 25             ;# total nodes number in network
set val(conn_prob)                0.11           ;# nodes connection prob.
set val(topo_seed)                98765          ;# seed to generate random topology
set val(link_bw)                  16Mb           ;# bandwidth of link
set val(link_wvlen_num)           16            ;# wavelengths number on each link
set val(wvlen_conv_factor)        0.5           ;# wvlen conversion factor, between 0 and 1
set val(wvlen_conv_dist)          4             ;# wvlen conversion distance, <= wvlen num
set val(wvlen_conv_time)          0.024         ;# wvlen conversion time (relative time)
set val(traf_density)              0.6           ;# generated traffic density in networks
set val(traf_arrival_rate)        0.5           ;# mean of each session arrival rate
set val(traf_holding_time)        1.0           ;# mean of each session holdingtime
set val(traf_pkt_size)            1000          ;# session-traffic packet size
set val(traf_pkt_rate)            1Mb           ;# session-traffic packet arrival rate
set val(traf_type)                Exponential    ;# session-traffic type in network
set val(traf_exp_burst_time)       0.7          ;# expoo traffic average burst time
set val(traf_exp_idle_time)        0.1          ;# expoo traffic average idle time
set val(traf_max_req)             1000          ;# max traffic requests number
set val(traf_start_time)          0.0           ;# session-traffic starting time
set val(traf_stop_time)           0.0           ;# session-traffic stoping time
......


# Create a simulator object
set ns [new Simulator]

# Wvlen routing protocol and assigning mechanism
$ns wrouting-proto $val(wvlen_routing)
$ns wassign-proto $val(wvlen_assign)

# Generate the topology creation script
topology -outfile $val(topofile) -nodes $val(node_num)
        -connection_prob $val(conn_prob) -seed $val(topo_seed)
# Generate the traffic creation script
traffic $val(traf_type) $val(node_num) $val(traf_num) $val(traffile)
......

# Create random topology
create-topology ns WDMNode SessionTrafficRcvr $val(link_bw) $val(link_wvlen_num)
        $val(wvlen_conv_factor) $val(wvlen_conv_dist) $val(wvlen_conv_time)
        $val(wvlen_alloc_path2) $val(util_sample_interval) $val(traf_max_req)
# Create random traffics
create-traffic ns traffic WDMNode SessionTrafficRcvr $val(node_num) $val(traf_num)
        $val(traf_pkt_size) $val(traf_pkt_rate) $val(traf_arrival_rate) $val(traf_holding_time)
        $val(traf_exp_burst_time) $val(traf_exp_idle_time)

# Schedule session traffics
for { set i 0 } { $i < $val(traf_num) } { incr i } {
        $ns schedule-sessiontraffic $traffic($i) $val(traf_start_time) $val(traf_stop_time)
}
# Schedule ns stop
if { $val(traf_stop_time) > 0 } {
        $ns at [expr $val(traf_stop_time) + 1.0] "finish"
}

# before ns runs, prepare wassignlogic
$ns pre-run-wassignlogic
$ns run
```

Figure 3: Script file used in simulation.

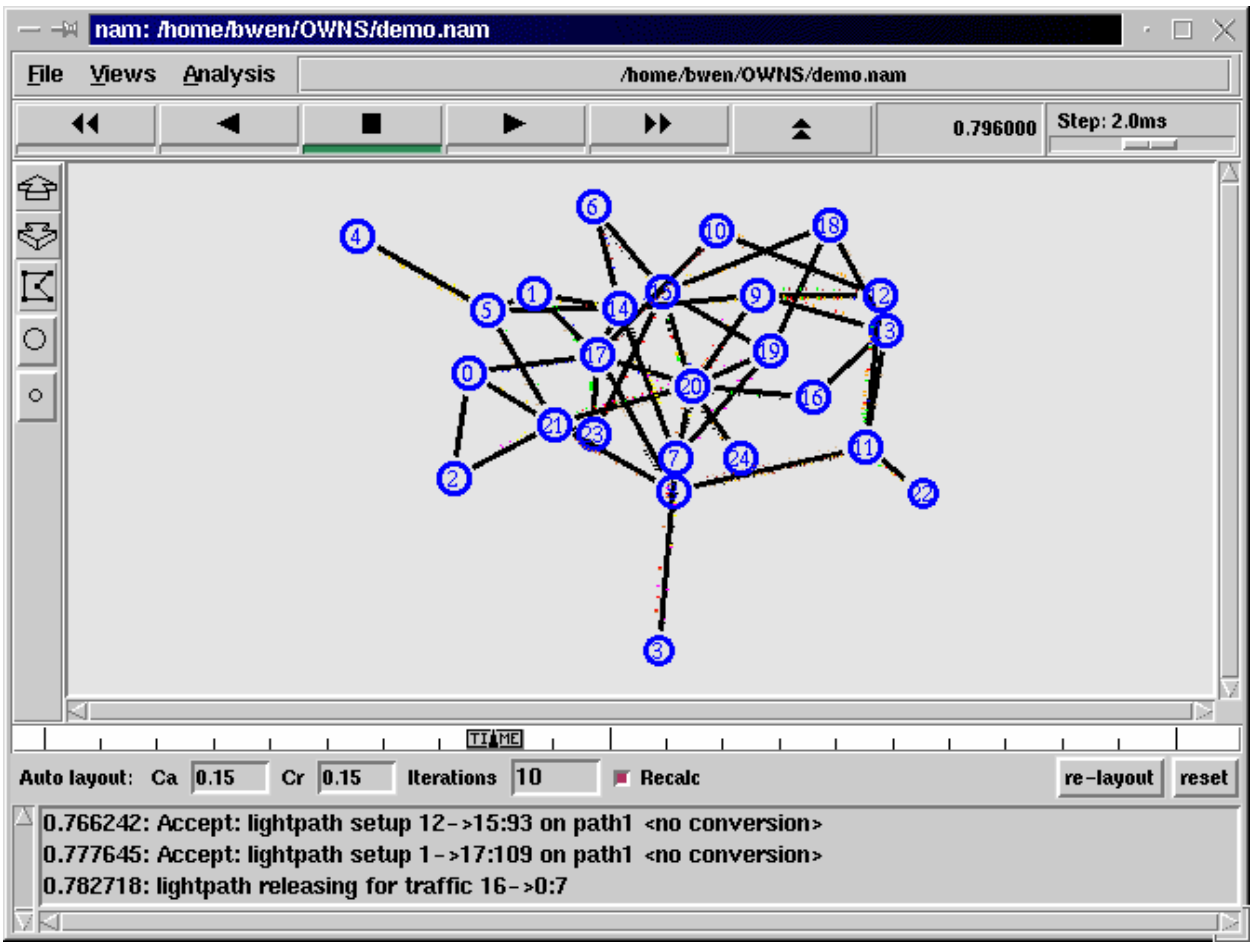

Figure 4: Topology of the WDM network used for simulation

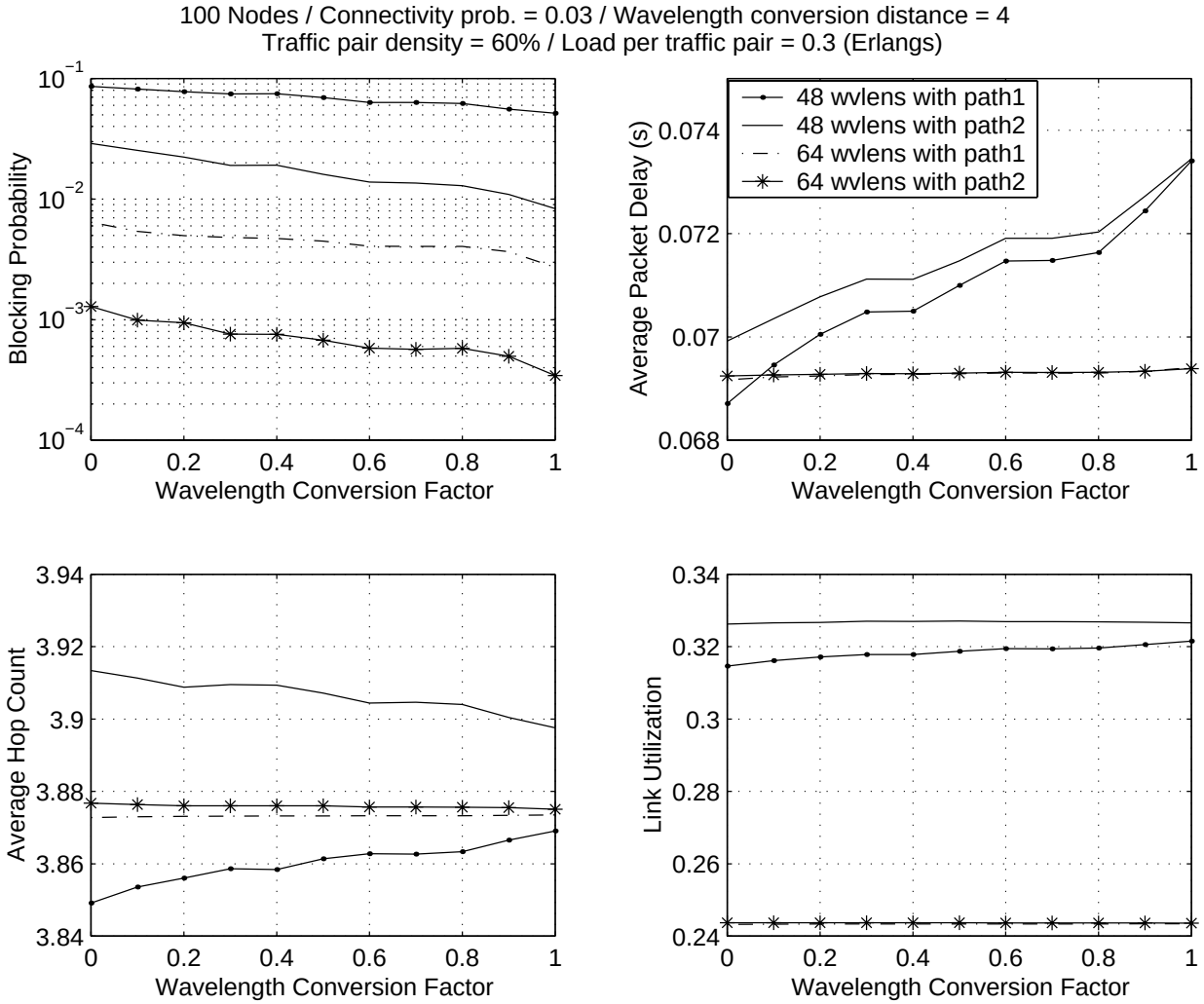

Figure 5: Effect of varying wavelength conversion factor on the network performance

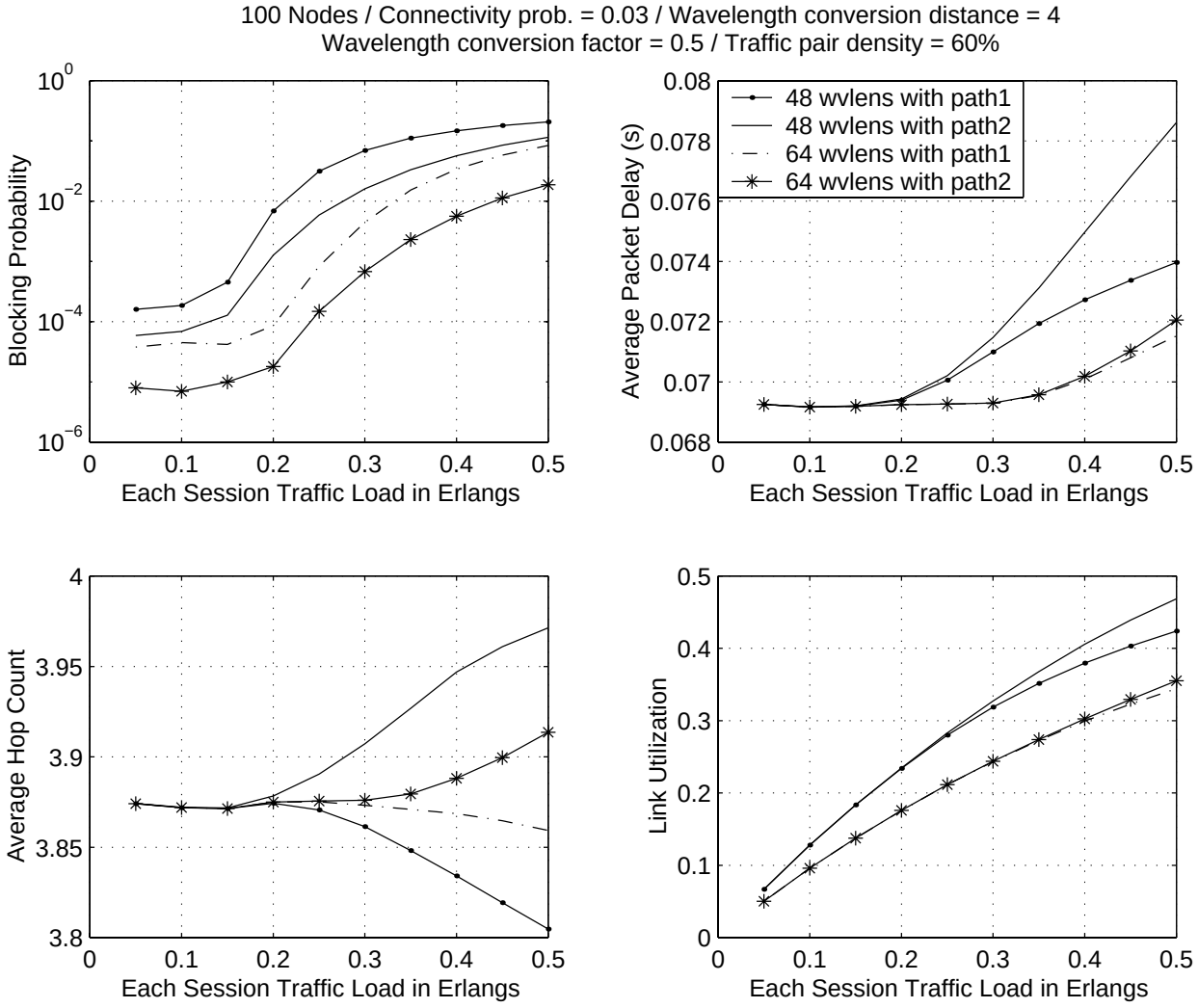

Figure 6: Effect of varying traffic load on the network performance

List of Figure Captions