Design and Implementation of Switch Module for NS-3

XIA Yu
School of Info. Science & Tech.
Southwest Jiaotong University
Chengdu 610031, China
rainsia@gmail.com

ZENG Huaxin
School of Info. Science & Tech.
Southwest Jiaotong University
Chengdu 610031, China
huaxinzeng1@126.com

SHEN Zhijun
School of Info. Science & Tech.
Southwest Jiaotong University
Chengdu 610031, China
shensljx@sina.com

ABSTRACT
In this paper, we first show the importance of switch module to NS-3. And then, we design a switch module for NS-3, to do so, some principles are first proposed. Based on these principles, an abstract switch module is designed and implemented for NS-3. By extending this abstract module, many specific switches using different switch architectures and arbitration or scheduling algorithms can be implemented. Two specific switches are implemented in this paper as examples. One is based on crossbar and uses iSLIP as its scheduling algorithm, the other is the CICB (Combined Input and Crosspoint Buffered) switch which uses the simplest round-robin as its input and output arbitration algorithm. Finally, several simulations are done, to verify the correctness of our switch module. By comparing the results to that within the original papers, we say that our designs and implementations are correct, and the switch module can be used for both the switch and network simulations. Then based on the switch module, some simulations are designed to show the network performance with different switches.

Keywords
switch module, NS-3, network simulation, traffic source

1. INTRODUCTION
Packet switches/routers have been the key components that significantly affect network performance more than any other components. The performance analysis of packet switches is important to network design. However, the performance analysis fully based on mathematics is complicated, and is only feasible when traffic sources are simple and can be represented as stochastic processes. When the traffic source itself becomes complex, for example the self-similar traffic [1], the analysis becomes impossibly hard. Thus simulation plays a dominant role in this area.

Some software kits are designed for packet switch simulations such as the famous SIM [2] by Stanford University. However, it is dedicated for crossbar switches with fixed-length packets, and can simulate only the switches in isolation. In most of the situations, we would like to put our switch into a real-world network topology and to evaluate its performance with the interacting between network components. Some researchers choose to write switch simulation modules with general purpose programming languages (such as C/C++ and Java) from scratch, it is free to choose switch architectures, but the above mentioned drawbacks still exist and are even worse. NS-2 [3] is a well-developed network simulator. It is widely used by the academia and industry in the past decade. It has been one of the most successful open-source projects in the past ten years. NS-3 [4] is designed to replace NS-2 with its elegantly designed architecture and many easy-to-use features. Both NS-2 and NS-3 can conveniently build network simulations which have complex network topologies and environments. Unfortunately, neither NS-2 nor NS-3 has a build-in switch module. The forwarding node of NS-2/3, by default, is equal to an Output-Queued (OQ) packet switch which is proven to be not implementable in modern high-speed network environment because of the low access rate of commercial static memories [5].

For the above reasons, we designed a packet switch module for NS-3. For switch designers, their switch structures can be involved into a more real network environment to test the performance with many complicated traffic models or even real traffic traces. For network designers, instead of using OQ switches, more concrete switch architectures can be used to estimate the performance of the networks, which makes the results more accurate and more reliable.

The rest of this paper is organized as follows: Section 2 analyses the pros and cons of the main stream packet switch simulation methods and why do we need a switch module in NS-3. In Section 3 we first design a generic switch model, and then implement, as examples, two specific switches based on this generic module. Section 4 describes how to integrate our switch module into the forwarding node of NS-3 so that it can be placed into a network topology. Several simulations are done in Section 5 to verify our designs and implementations and to demonstrate the use of switch module in network scenarios. Section 6 concludes this paper.

2. THE NEED OF SWITCH MODULE
There are mainly three ways to simulate packet switches: 1) To use a switch simulation software kit; 2) To build a simulation module with general purpose programming languages; 3) To use existing network simulators.

The advantages of the first method are obvious, both the underlying code of the simulation architecture, switch architectures and some simple traffic models are already implemented, the users only need to add their own algorithms and configurations to complete a simulation. But most of these

NSTOOLS 2009, October 19, Pisa, Italy

DOI 10.4108/ICST.VALUETOOLS2009.7658

software kits only provide the ability to simulate a single node switch. Furthermore, they are usually independent of protocol stacks and the states of the networks.

The second method allows users to freely choose switch architectures and traffic models as they want. However, there are a lot of things to care about: simulation architecture, statistics collecting framework, switch architectures, scheduling algorithms, traffic generating, and so on. It is time-consuming to implement all of these, and hard to make sure that they are correctly coded.

There are a lot of network simulators, such as NS-2/3 and OPNET [6], which provide advanced simulation architectures. With these simulators, users can build complicated network topologies conveniently as they want, use existing traffic generating modules or even import the real traffic traces. Future more, some of them also provide easy-to-use statistics collection and analysis frameworks. Some projects are done to add switch simulation functions to network simulators, such as in reference [7], but those works are not able to seamlessly integrate packet switches into network topologies and take full advantage of the existing protocol stacks, routing algorithms and other useful components of the simulators.

The forwarding node in NS-2/3, by default, is equal to an OQ switch, which is not commonly used in current network design. And as we can see later by simulation, the performance of OQ switch is quite different from that of other popular switches. So a generic packet switch framework with some popular packet switches is needed in NS-3 not only for the switch designers but also for the network designers. Switch designers can extend our framework to provide more switch architectures and scheduling algorithms, and then test the new architectures or algorithms in a real network environment including network traffic, protocol stacks, routing algorithms and the interactions between each node. Network designers can seamlessly integrate appropriate switches into their network topology instead of just using OQ switch, since OQ switch shows the best performance, which is very different from other switch architecture, especially when the AQM (Active Queue Management) and congestion control mechanisms are involved. So if more realistic switches are used to estimate the performance of the networks, the results should be more accurate and reliable.

As the very reasons, we design a switch module for NS-3, which will make NS-3 more suitable for either switch simulation or network simulation.

3. GENERIC SWITCH MODULE FOR NS-3

3.1 The Principles of Adding New Modules

Before we start designing our switch module, we should first describe our principles of adding new modules into NS-3.

First, we should never modify existing code of NS-3. If we want to extend any functionality of NS-3, we always choose to add new code. There are two obvious benefits of this principle: the extension can be easily migrated to a new version of NS-3 by simply copying the new code to that new version; other existing modules or especially the third-party modules which depend on some of the modules we want to change will not fail.

Second, for any module, we first design a highly abstract super class, and then extend it by inhering to add new functions into the sub classes. Thus, other people can extend our module to implement their own functions in a much easier way.

Finally, for some common modules, such as traffic sources, we want them to be highly reusable. In ordinary, traffic modules are coupled with applications, which means we cannot reuse them between different applications. To avoid these modules becoming application dependent, we have tried to separate core functions from those application-related modules to make them reusable for any other applications.

3.2 Generic Packet Switch

An $N \times N$ packet switch, as shown in Figure 1, has N input units and N output units. The input ports and output ports have the same data rate, and are interconnected by the switch fabric. The switch fabric usually processes fixed-length packets (called cells). For packets with variable-length, an ISM (Input Segmentation Module) in Input Unit should segment them into fixed-length cells. In an output unit of the switch, cells belonging to the same packet are reassembled by an ORM (Output Reassembly Module) before they are sent out. The switch fabric may transfer cells faster than the data rate of input ports. In this case, we say that the switch fabric has a speedup. The time slot is defined as the time used to receive a cell from an input port. With a speedup ratio "1" (i.e. without speedup), only one packet may arrive at an output unit every time slot, so queuing is only needed at the input side, this kind of switches are called Input-Queued (IQ) switches. When the speedup ratio is greater than 1 and less than *N,* there might be more than one cell arriving at an output port every time slot, so packets may need to be queued at both

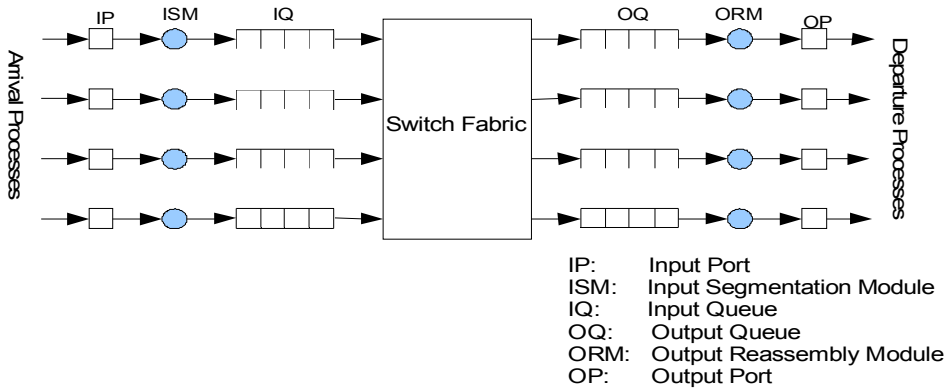

Figure 1. A generic concept model of the packet switch

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7658
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7658

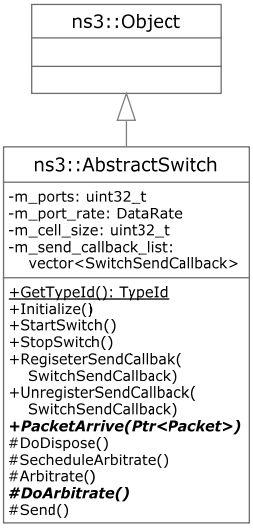

Figure 2. Class diagram of Abstract Switch

input and output sides. We call these switches Combined Input and Output Queued (CIOQ) switches. When the speedup ratio is equal to *N,* each packet can be forwarded to the corresponding output unit as soon as it arrives. Consequently, input queuing is unnecessary while output queuing is needed, and in this case, the switch is called Output Queued (OQ) switch.

According to our principle, we should first design an abstract class for the generic switch. The class diagram， which specifies the roles of the switch,in a network is shown in Figure 2. A brief description of the members in **`AbstractSwitch`** class is shown in Table 1.

Table 1. Description of members of the AbstractSwitch class

m_ports	The number of ports of a switch.
m_port_rate	The data rate of each input/output port.
m_cell_size	The cell size processed by the switch, Note that: timeslot=m_cell_size/m_port_rate
m_send_callback_list	A list of callback, the callbacks will be invoked when there is a packet being transmitted.
Initialize()	Initialize the switch parameters.
StartSwitch()	Start the switch.
StopSwitch()	Stop the switch.
RegisterSendCallback()	Add a send callback to the list.
UnregisterSendCallback( )	Remove a send callback from the list.
PacketArrive()	Pure abstract method. Is called when there is a packet arriving at any input port.
DoDispose()	Used to clear up when destroying.
SechduleArbitrate()	Schedule a call to Arbitrate() every time slot.
Arbitrate()	Call DoArbitrate() to run arbitration algorithm and ScheduleArbitrate() to schedule a call to itself next time slot.
DoArbitrate()	Pure abstract method. Will be implemented by its subclasses to provide various arbitration algorithms.
Send()	Send packet out from an output port.

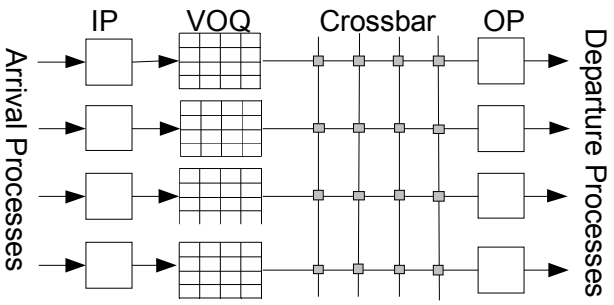

Figure 3. The concept model of the VOQ/Crossbar switch

3.3 Implementation of Two Specific Switches

Based on **`AbstractSwitch`** we can design and implement many different switch architectures and the corresponding scheduling algorithms. In this subsection, we are going to design and implement two different switch architectures: one is the most widely used crossbar switch, and the other is the most promising Combined Input and Crosspoint Buffered (CICB) switch.

3.3.1 Crossbar Switch

The crossbar switch architecture is the most widely used switch architecture. In such a switch, the switch fabric is crossbar, which connects input ports and output ports and is able to provide N parallel connections in the same time slot. By suffering from the HoL (Head-of-Line) blocking problem, the switch has very poor performance, until VOQ (Virtual Output Queue) is introduced [8]. With VOQ, each input port maintains a separate logical queue for each output port, the cells from the same input port and destined to different output ports are enqueued into different VOQs. A concept model of the VOQ/Crossbar switch is illustrated in Figure 3.

With VOQ, within a time slot, each input port has at most N cells

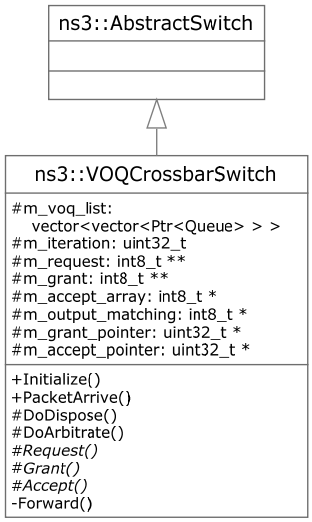

Figure 4. Class diagram of VOQCrossbarSwitch

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7658
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7658

waiting to be sent, and each output port also has at most N cells are destined for it. Thus additional scheduling algorithms are needed to solve the contention at the input ports and output ports. The design of high-performance scheduling algorithm for VOQ/Crossbar was once and continues to be a hot topic of switch design. Many famous algorithms have been proposed such as PIM (Parallel Iterative Matching) [9], RRM (Round-Robin Matching), SLIP [10], DRRM (Dual Round-Robin Matching) [11], etc. These algorithms are all consist of two or three steps, namely request, grant and accept (DRRM do not need an accept step). In order to find a maximal matching, most algorithms need to run for several iterations [12].

`VOQCrossbarSwitch`, whose class diagram is shown in Figure 4, is designed to represent a crossbar switch with VOQs as its input side queuing, and it is a subclass of **`AbstractSwitch`**. **`VOQCrossbarSwitch`** add some member variables and member functions to its base class, which are described in Table 2.

Table 2 Description of members of the VOQCrossbarSwitch class

m_voq_list	A list of virtual output queues.
m_iteration	How many iterations have been invoked.
m_request	The request matrix.
m_grant	The grant matrix
m_accept_array	The accept array.
m_output_matching	The input-output matching.
m_grant_pointer	A list of the grant pointer for each output port [10].
m_accept_pointer	A list of the accept pointer for each input port [10].
PacketArrive()	When a packet arrives from some input port, its destination port is retrieved, and then it is enqueued into the corresponding VOQ at that input port.
DoArbitrate()	Call the following four methods.
Request()	Pure abstract method. To send requests to each output port.
Grant()	Pure abstract method. To grant the requests of input ports.
Accept()	Pure abstract method. To accept the grants from output ports.
Forward()	Send the corresponding packets of matched input ports to related output ports.

Note that **`VOQCrossbarSwitch`** is still an abstract class since no scheduling algorithms are defined. Scheduling algorithms can be added by inheriting, for example **`SLIPSwitch`** and **`DRRMSwitch`** can be implemented by overriding the methods **`Request()`**, **`Grant()`** and **`Accept()`**.

3.3.2 Buffered Crossbar Switch

As the transmission rate becomes higher and higher and the cell length is unlikely to change, the time slot becomes shorter and shorter. The time needed for running complicated scheduling algorithms becomes very tight especially for iterative scheduling algorithms. The VOQ/Crossbar switch architecture is no longer a suitable choice for modern high-speed switches. Once, the buffered crossbar switch was considered to be the solution for high-speed switches since it can fully emulate an OQ switch. However, under current VLSI technology, only a small number of

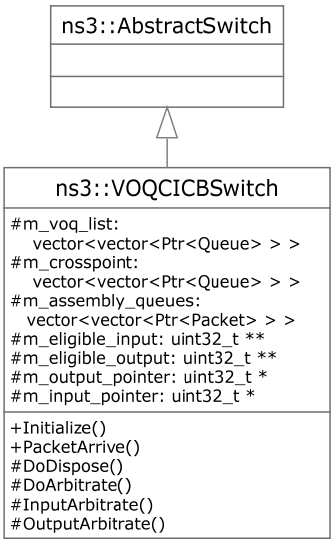

Figure 5. Class diagram of VOQ/CICB Switch

memories can be embedded into a crossbar, which are not enough to buffer all the incoming packets. Later, Combined Input and Crosspoint Buffered (CICB) switch architecture is proposed [13], and is proven to have very good performance, especially when the input side adopts VOQs. Only a very simple scheduling algorithm is need for a VOQ/CICB switch to achieve an average delay similar to that of an OQ switch [14].

An abstract class **`VOQCICBSwitch`**, which is a subclass of **`AbstractSwitch`**, is designed to represent VOQ/CICB switch. Its class diagram is shown in Figure 5, and the description of its members is in Table 3.

Table 3. Description of members of the VOQCICBSwitch class

m_voq_list	A list of VOQs.
m_crosspoint	A matrix of crosspoint buffers.
m_eligible_input	Indicates which crosspoint buffers are not full, and the corresponding input ports can send packets to them.
m_eligible_output	Indicates which crosspoint buffers are not empty, so that they can send packets to idle output ports.
m_output_pointer	Which crosspoint is chosen for each output port at the previous time slot.
m_input_pointer	Which crosspoint is chosen for each input port at the previous time slot.
InputArbitrate()	Pure abstract method. The scheduling algorithm for each input port.
OutputArbitrate()	Pure abstract method. The scheduling algorithm for each output port.

A lot of algorithms are proposed for input and output side. The simplest one is Round-Robin, which is shown to have very good performance especially under uniform traffic [15]. For each algorithm, we only need to inherit the **`VOQCICBSwitch`** class and override **`InputArbitrate()`** and **`OutputArbitrate()`** methods. For example, we can implement a class called **`CICB_1_RR_RRSwitch`**, which means the switch can hold only

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7658
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7658

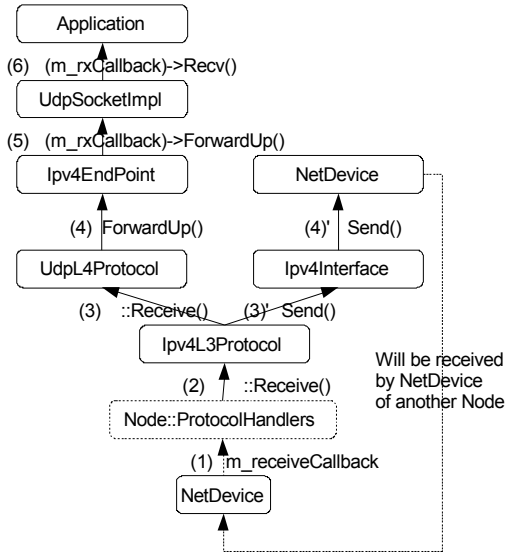

Figure 6. Process of receiving a packet in InternetStack

one cell at each crosspoint and use Round-Robin as its input and output arbitration algorithm.

4. INTEGRATING SWITCH MODULE INTO FORWARDING NODE

If we want to put our switch into a network topology and evaluate its performance using existing traffic sources, protocol stacks and routing algorithms. Or if the network designers need to use a switch other than the OQ switch. It is important to seamlessly integrate our switch module with current forwarding node of NS-3.

In this section, we introduce a way to integrate our switch module into a forwarding node and the corresponding protocol stack, so that any existing NS-3 code can still be used as before.

We take the InternetStack as an example to show how to integrate our switch module into a forwarding node.

When receiving a packet, the **`InternetStack`** will follow a process as shown in Figure 6. In Step (3) of Figure 6, the method **`Receive()`** of **`Ipv4L3Protocol`** is invoked, in this method, **`Ipv4L3Protocol`** removes the IP header, checks checksum (if implemented), if the node that the **`InternetStatck`** belongs to

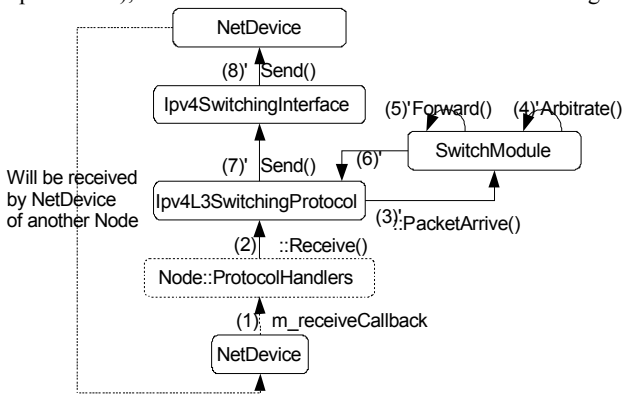

Figure 7. Process of forwarding a packet in InternetSwitchStack

is a switching node, the cell is forwarded by **`Forwarding()`** method, while if the node is an end system, the cell is forwarded to upper layer by calling **`ForwardUp()`** method.

To integrate our switch module into a forwarding node, we only need to have our switch module invoked before a packet is sent out from the forwarding node. To do so, we copy all the code of **`InternetStack`**, and create a new module called **`InternetSwitchingStack`**, the new process of packet receiving and forwarding is shown in Figure 7. By doing so, the delay and jitter caused by switching are added to the cell's transmission, which makes the performance more "real" than original process, since the OQ switch has the best performance but is impractical in real high-speed networks.

5. SIMULATION

5.1 Correctness of Switch Module

Before we can use our switch module to evaluate the performance of any network, we should first make sure that the designs and implementations of our switch module are correct.

To do so, we use several typical traffic models to test our implementations and compare the results to original papers to see whether our results are the same as the original ones. The test of switch module itself is independent of any protocol and network topology.

There are four famous traffic models, which are widely used to evaluate the performance of switches: uniform Bernoulli traffic, uniform bursty traffic, asymmetric traffic [16] and unbalanced traffic [15]. We use these models to check the correctness of our switch module.

5.1.1 Uniform Bernoulli Traffic

For the SLIP switch, we first simulate the single iteration version as McKeown has done in [10]. The simulation result of delay performance of the SLIP switch as a function of switch size is shown in Figure 8. By comparing our result to Fig. 11 in [10], we found that the curves are almost the same, and the minor differences may be caused by the different behaviors of the

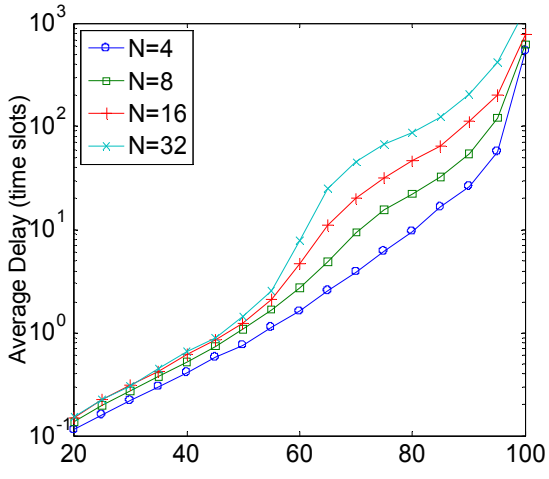

Figure 8. The delay performance of iSLIP as function of switch size under uniform i.i.d Bernoulli arrivals.

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7658
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7658

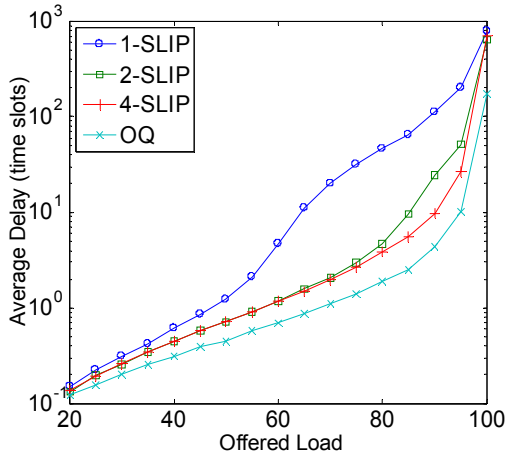

Figure 9. The delay performance of iSLIP for 1, 2 and 4 iterations under uniform i.i.d Bernoulli arrivals compared with OQ.

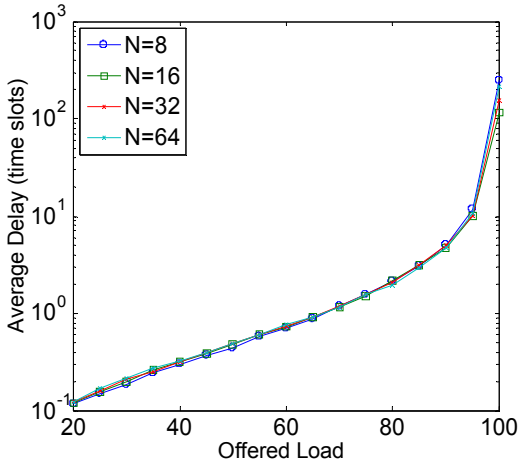

Figure 10. The delay performance of CICB switch with Round-Robin algorithm as function of switch size under uniform i.i.d. Bernoulli arrivals.

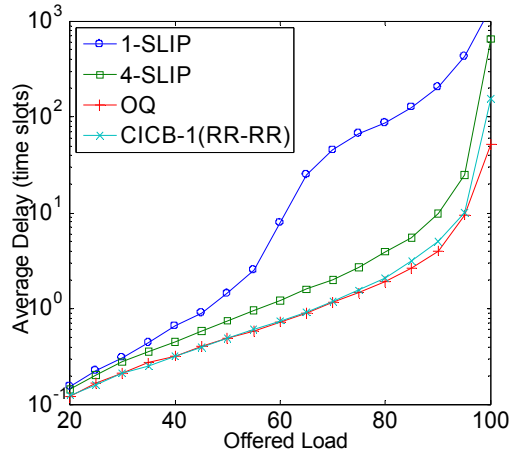

Figure 11. The delay performance comparison of 1-SLIP, 4-SLIP, CICB-1 and OQ under uniform i.i.d Bernoulli arrivals.

random number generators.

SLIP algorithm always be implemented with multiple iterations, and reference [10] suggest the number of iterations should be $log_2(N)$. For a 16x16 packet switch, the simulations of SLIP algorithms running multiple iterations (i.e. 4 iterations) are done, and the result is shown in Figure 9. We say that our implementation is the same as that in the original paper when comparing the result to Fig. 17 in reference [10].

For the CICB-1 switch with Round-Robin algorithm, we also simulate the performance as function of switch size, and the result is shown in Figure 10, and the curves are also comparable to Fig. 6 in reference [15]. This means our implementation has the same behavior as that of the original authors'.

The simulation result comparing the performance of the 32x32 packet switches of 1-SLIP, 4-SLIP, CICB-1 and OQ under uniform i.i.d. Bernoulli arrivals is shown in Figure 11, which also have been done by R. Rojas-Cessa et. al., and their result is shown by Fig. 3 in reference [13]. We found that our result is very similar to that of the original paper, which again proves the correctness of our design and implementation.

5.1.2 Uniform Bursty Traffic

For uniform bursty traffic, we use the ON/OFF model with the ON and OFF periods both obey the geometry distribution. The average value of the ON period is called the burst size, and the average value of the OFF period varies according to the offered load. Packets generated in the same ON period are destined to the same output port which is chosen uniformly.

For iSLIP with single iteration, the simulation result is shown in Figure 12. And for the iSLIP with $log_2 N$ iterations, the result is shown in Figure 13. With the comparison to Fig. 10 and Fig. 19 in reference [10] respectively, we know that our implementation of iSLIP is correct.

For the CICB-1 switch, simulation performance evaluation under bursty traffic is also done with the same configurations as in reference [15]: the switch size is 16, and the burst sizes are 1 (equivalent to Bernoulli arrivals), 10 and 100. The simulation result is shown in Figure 14, which is the same as Fig. 5 in reference [15]. So we can say that our implementation of CICB-1 is correct.

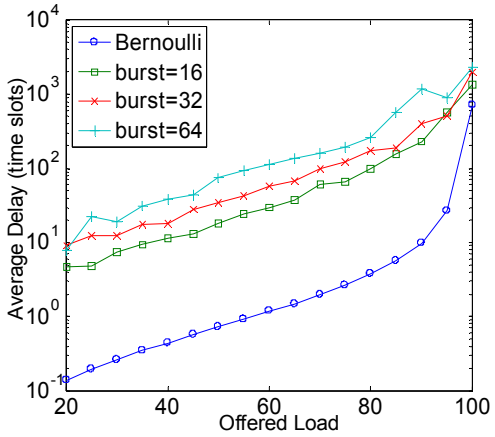

Figure 12. The delay performance of SLIP under uniform bursty arrivals, the switch size is 16.

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7658
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7658

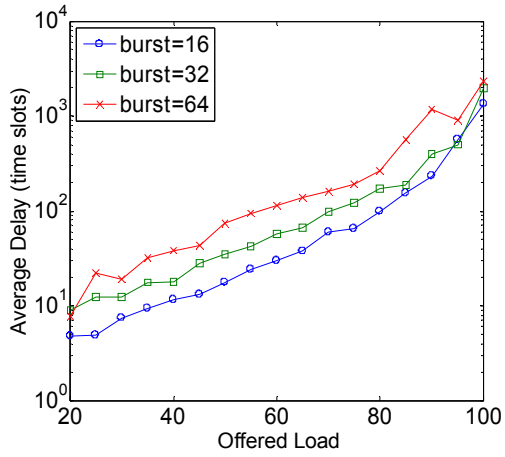

Figure 13. The delay performance of iSLIP with multiple iterations under uniform bursty arrivals. The switch size is 16, and the burst size varies from 16 to 64.

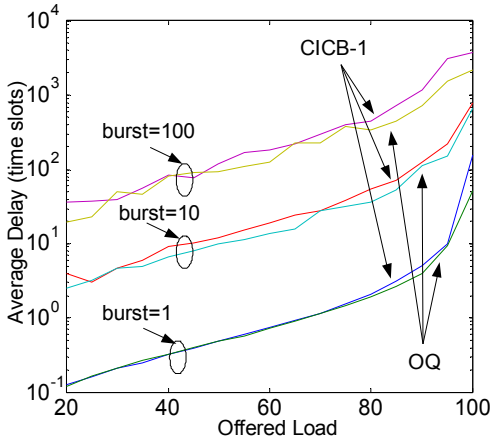

Figure 14. The delay performance with Bernoulli and burst sizes of 10 and 100 for 32x32 CICB-1 and OQ switches.

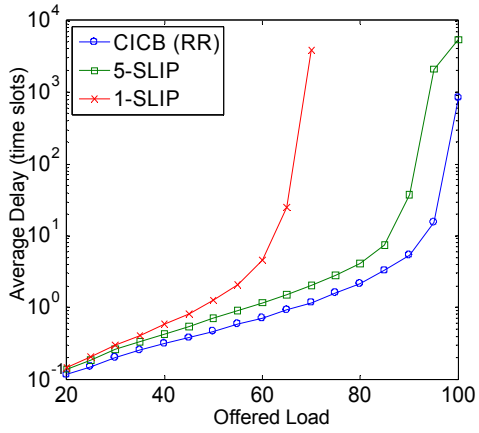

Figure 15. The delay performance of iSLIP and CICB under asymmetric arrivals.

5.1.3 Asymmetric Traffic

To further test the correctness of our implementation, we use other two nonuniform traffic models. The asymmetric traffic model is a widely used nonuniform traffic for evaluating performance of switches. The asymmetric traffic model can be defined as having different load for each input-output pair, such as

$$\rho_{i,(i+j)\%N} = a_j \rho$$

where

$$\begin{cases} a_0 = 0 \\ a_1 = (r-1)/(r^N - 1) \\ a_j = a_1 r^{j-1} & \forall j \neq 0 \end{cases}$$

and $r = (100:1)^{-1/(N-2)}$.

The simulation comparing the performance of SLIP and CICB switches is done, and the result is showing in Figure 15. The comparison between Figure 15 and Fig. 5 in reference [17] further proves the correctness of our implementations.

5.1.4 Unbalanced Traffic

Another well-known nonuniform traffic model is unbalanced traffic. The unbalanced traffic model uses a probability w as the fraction of input load directed to a single predetermined output, while the remaining load is directed to all outputs with a uniform distribution. Let's consider input port s, output port d, and the offered input load ρ for each input port. The traffic load from input port s to output port d, $\rho_{s,d}$ is given by:

$$\begin{cases} \rho_{s,d} = \rho\{w + [(1-w)/N]\} & s = d \\ \rho_{s,d} = \rho[(1-w)/N] & otherwise \end{cases}.$$

The throughput of different switch architectures with unbalanced traffic is shown in Figure 16, As we can see, the OQ switch can achieve 100% throughput, but the throughput of the iSLIP switch and the CICB-1 switch is heavily degraded with unbalanced traffic, which is the same as Fig. 4 in reference [17] shows.

To this end, we have verified our designs and implementations by comparing the various simulation results with those in published literatures, and found they are almost the same (the minor differences may be caused by the different random number generators, simulation times and different statistics collection

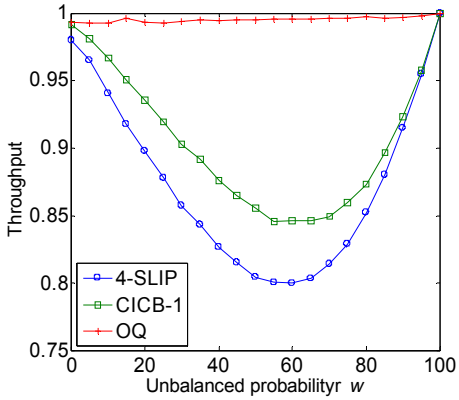

Figure 16. Throughput of iSLIP CICB-1 and OQ under unbalanced arrivals.

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7658
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7658

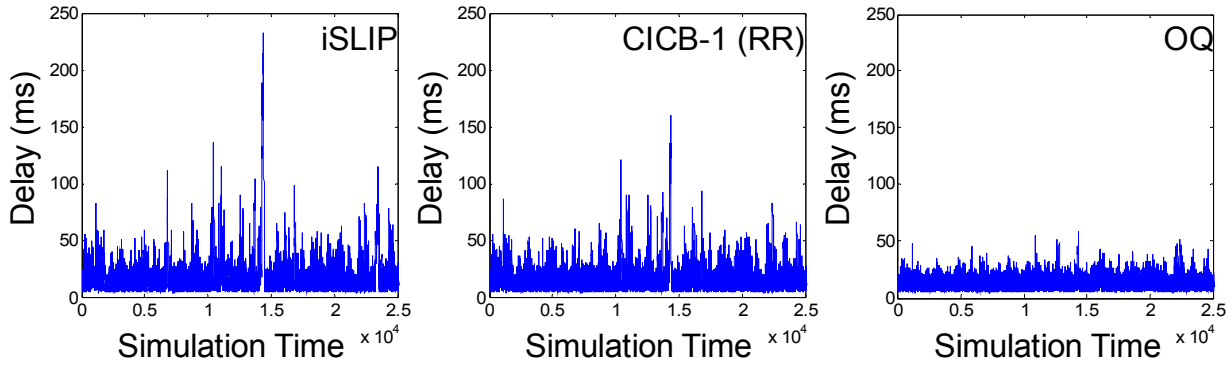

Figure 17. Delay traces with variable-length packets for different switch architectures.

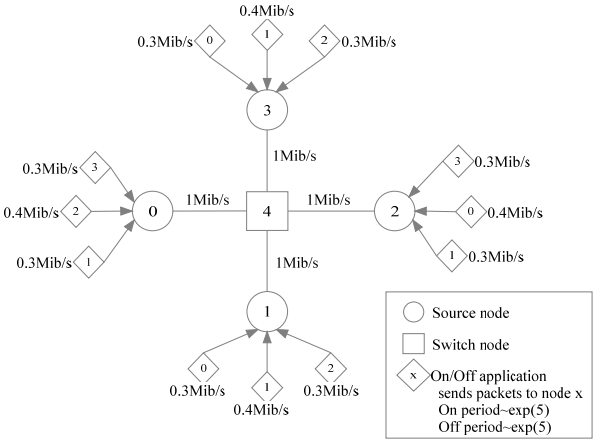

Figure 18. Network topology of scenario 1

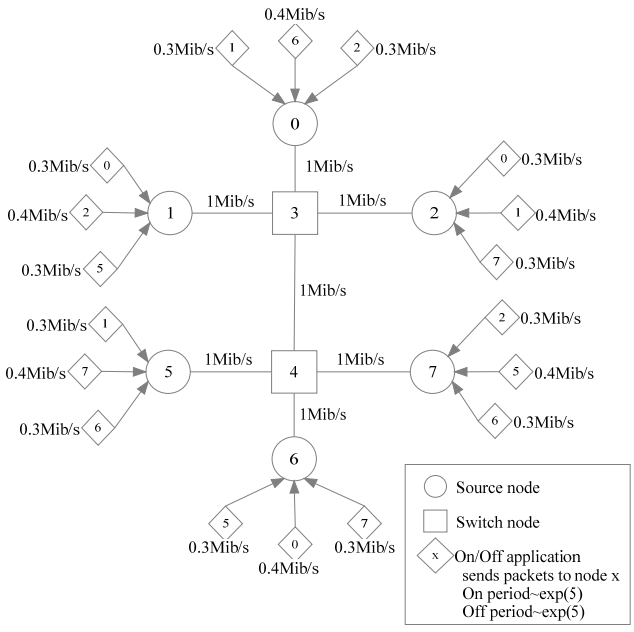

Figure 19. Network topology of scenario 2

mechanisms).

5.2 Variable-length Packet

Although switches use fixed-length cells inside the switch fabric, the IP packets are variable-length. So the ISM and ORM are implemented in our switch module and add reassembly buffer at the output side. In the next simulation, we let the packet lengths obey the exponential distribution and vary from 64 bytes to 1024 bytes, and the switches use 64 bytes as their internal cell length. Figure 17 illustrates the delay traces of different switches with variable-length packets.

5.3 Network Scenarios

Now let's build two network scenarios using existing NS-3 component, then integrate our switch modules into the forwarding node and see how the different switches affect the network performance. The first scenario is illustrated in Figure 18, where four end systems are interconnected by a switch. The end systems communicate with each other with ON/OFF bursty traffic. As shown in Figure 18, the ON and OFF periods obey exponential distribution and the average value is 5. The built-in global routing is used as the routing algorithm.

In the second scenario we consider a multiple-node case, the configurations are similar the previous scenario, except the number of nodes. The topology of scenario 2 is shown in Figure 19.

For scenario one, we are focusing on the delay of packets from node 0 to node 2. The packet generating rate of node 2 varies from 0.1Mib/s to 0.4Mib/s. The results of average packet delay of three different switches are shown in Figure 20. As we can see, the difference of the average delays is not so obvious. So in Figure 21 we show the delay traces for detail, as we can see that the traces are quite different from each other, which means the delay jitters are quite different. As expected, the OQ switch has the best performance, and the SLIP switch have the worst performance.

For the second scenario, we capture the packet delay from node 0 to node 6, and the delay traces are shown in Figure 22. As we can see that the OQ switch still have the best performance, and the difference of delay jitters between these switches are even larger.

6. CONCLUSION

In this paper, we showed that the switch module is important but is not included in NS-3 by default. Then we design a new switch module for NS-3. For our switch module, we design a highly

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7658
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7658

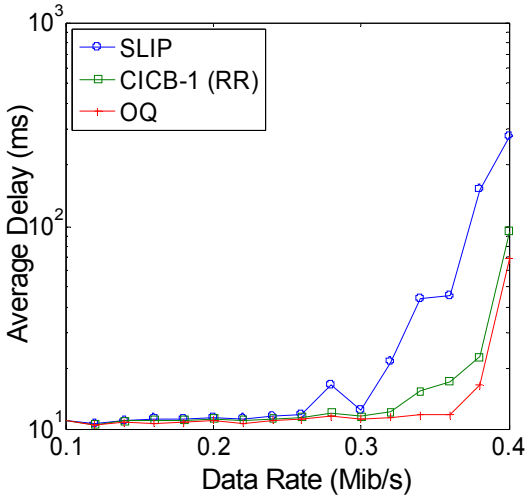

Figure 20. Average Delay in scenario 1.

abstract class **`AbstractSwitch`** which can be inherited and extended to implement different switch architectures and scheduling algorithms. We also implemented two different switches as examples.

We showed that our implementations are correct by comparing the simulation results to that of the original papers. This makes sure that the designs and implements of our module are correct and can be directly used by both switch designers and network designers. We also introduce a way to integrate our switch module into the forwarding node of NS-3 to replace the default OQ module, so that network designers can use different packet switches to build their networks.

Then we showed that the same network with different switch architectures will have quite different performance in both single node and multiple node cases. These scenarios also demonstrate how our switch module can be seamlessly integrated with exiting NS-3 protocol stacks, routing algorithms and other components.

7. ACKNOWLEDGMENTS

The work presented in this paper is supported in part by Chinese Natural Science Foundation (Project No. 60773102) and in part by Sichuan University Foundation (Project Name: Next Generation Internet). The authors would like to acknowledge the financial support from CNSF and SCU. The authors also would like to thank the anonymous reviewers, whose suggestions are very helpful for us to improve this paper.

8. REFERENCES

[1] V. Paxson and S. Floyd, "Wide area traffic: the failure of Poisson modeling," *Networking, IEEE/ACM Transactions on*, vol. 3, 1995, pp. 226–244.

[2] "SIM - a fixed-length packet switch simulator," *http://klamath.stanford.edu/tools/SIM/*.

[3] "The Network Simulator - ns-2,"

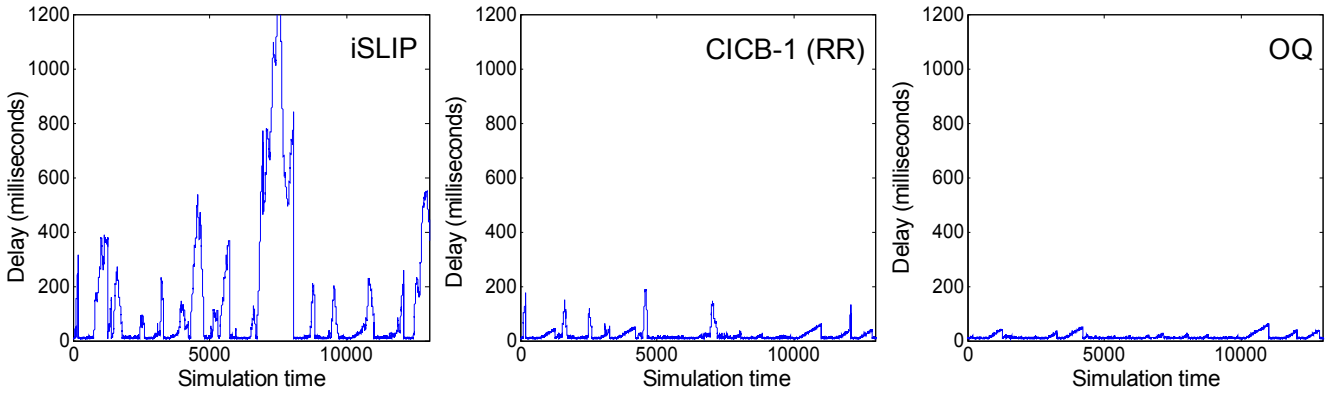

Figure 21. Delay traces in scenario 1

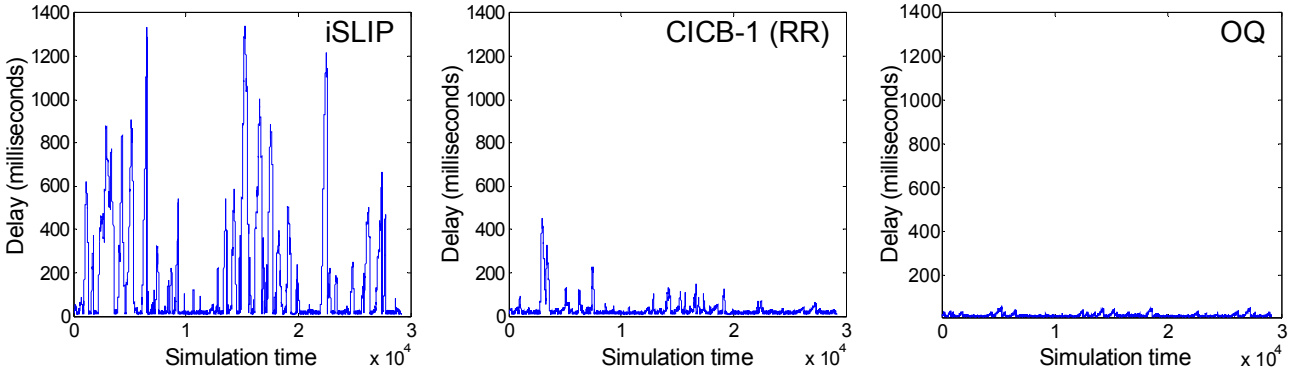

Figure 22. Delay traces in scenario 2

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7658
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7658

http://www.isi.edu/nsnam/ns/.

[4] "The ns-3 network simulator," *http://www.nsnam.org/*.

[5] S. Iyer, "Load Balancing and Parallelism for the Internet," Ph.D. dissertation, Stanford University, 2008.

[6] "OPNET Technologies - Making Networks and Applications Perform," *http://www.opnet.com/*.

[7] H. Zheng, Y. Zhao, and C. Chen, "Design and Implementation of Switches in Network Simulator (ns2)," *Innovative Computing, Information and Control, 2006. ICICIC '06. First International Conference on*, 2006, pp. 721–724.

[8] Y. Tamir and G. Frazier, "High-performance multiqueue buffers for VLSI communication switches," *Computer Architecture, 1988. Conference Proceedings. 15th Annual International Symposium on*, 1988, pp. 343–354.

[9] T.E. Anderson, S.S. Owicki, J.B. Saxe, and C.P. Thacker, "High-speed switch scheduling for local-area networks," *ACM Trans. Comput. Syst.*, vol. 11, 1993, pp. 319–352.

[10] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *Networking, IEEE/ACM Transactions on*, vol. 7, 1999, pp. 188–201.

[11] Y. Li, S. Panwar, and H. Chao, "On the performance of a dual round-robin switch," *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2001, pp. 1688–1697 vol.3.

[12] M. Bayati, B. Prabhakar, D. Shah, and M. Sharma, "Iterative Scheduling Algorithms," *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, 2007, pp. 445–453.

[13] R. Rojas-Cessa, E. Oki, and H. Chao, "CIXOB-k: combined input-crosspoint-output buffered packet switch," *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE*, 2001, pp. 2654–2660 vol.4.

[14] S. Chuang, S. Iyer, and N. McKeown, "Practical algorithms for performance guarantees in buffered crossbars," *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, 2005, pp. 981–991 vol. 2.

[15] R. Rojas-Cessa, E. Oki, Z. Jing, and H. Chao, "CIXB-1: combined input-one-cell-crosspoint buffered switch," *High Performance Switching and Routing, 2001 IEEE Workshop on*, 2001, pp. 324–329.

[16] R. Schoenen, G. Post, G. Sander, and G. S, "Weighted Arbitration Algorithms with Priorities for Input-Queued Switches with 100% Throughput," *IEEE INTERNATIONAL WORKSHOP ON BROADBAND SWITCHING SYSTEMS (BSS'99*, 1999).

[17] R. Rojas-Cessa, E. Oki, and H. Chao, "On the combined input-crosspoint buffered switch with round-robin arbitration," *Communications, IEEE Transactions on*, vol. 53, 2005, pp. 1945–1951.

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7658
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7658