Using CAIA's NS-3/FreeBSD Network Stack VM Appliance (Version 0.1)

Jonathan Kua Tze Hwei, Lawrence Stewart, Grenville Armitage
Centre for Advanced Internet Architectures, Technical Report 140702A
Swinburne University of Technology
Melbourne, Australia
jtkua@swin.edu.au, lastewart@swin.edu.au, garmitage@swin.edu.au

***Abstract*—This technical report discusses the architecture, design and use of CAIA's NS-3/FreeBSD VM Appliance (Version 0.1), developed as part of our "Exploring Possible Mitigation for Incast TCP Congestion in Data Centres" project. It describes the purpose and the operation of the simulator in the context of incast congestion control in data centres. This report also provides a detailed explanation of the operation of the switch model and how virtual output queueing mechanism is implemented. It also shows how to control simulation parameters, understand the debugging output and interpret the results obtained.**

***Index Terms*—Incast Congestion, Switch Model, Head of Line (HoL) Blocking, Virtual Output Queuing (VOQ), Network Simulator 3 (NS-3) and Network Simulation Cradle (NSC).**

I. Introduction

Network simulators are used for simulation, testing and results validation of large scale data networks in a laboratory environment. The Network Simulator 3 (NS-3) and Network Simulation Cradle (NSC) are developed for computer networks and data communications research purposes. NS-3 is a discrete time event simulator which models the behavior of real world networks closely [1]. NSC provides an environment for researchers to build and test networks by using actual network stacks instead of the idealised NS-3 network stack. This enables simulation results to be more accurate and resemble real world data.

As part of our project "Exploring Possible Mitigation for Incast TCP Congestion in Data Centres" we developed the "CAIA NS-3/NSC FreeBSD 9 Based Simulation Environment Virtual Machine Image v0.1" [2]. This image can be downloaded from http://caia.swin.edu.au/urp/incast/tools.html. It is a FreeBSD 9 based VirtualBox Appliance containing NS-3 and a version of NSC patched to work with the actual Transmission Control Protocol (TCP) stacks from FreeBSD 9-STABLE and FreeBSD-HEAD. This enable simulations to use the same TCP congestion control algorithms as implemented in real FreeBSD systems.

In the context of incast congestion in large data centres, the switch plays a major role in transferring the received packets from the receiving ports to the output port. The switch implements an algorithm that intelligently transfer the packet accross the switch fabric to the transmitting port. Virtual Output Queuing (VOQ) is implemented in the switch to overcome the Head of Line (HoL) blocking problem in older implementations of switches. However, congestion can still occur when multiple responders transmit packets simulataneously to the querier. This can cause packet loss when the queue is full.

The purpose of this technical report is to demonstrate how the simulator can be used in the context of TCP incast. The rest of report is organised as follows. Section II introduces the incast topology which is common in data centres and the architecture of the switch model. It then shows how VOQ is implemented in the simulator to handle incoming packets to overcome HoL blocking problem. Section III introduces specific details of the switch model as implemented in our NS-3/FreeBSD Virtual Machine (VM) environment. It shows how to run this simulation by controlling different parameters and discusses some modifications made to the NS-3 source codes. Section IV presents an incast simulation trial. It shows how to interpret the debugging output and the queue occupancy information from the simulation. Section V concludes the report.

II. Incast and Switches

A. The Incast Topology

TCP incast is a phenomenon in distributed storage clusters first described and identified by Nagle *et al* [3].

It is a network transport pathology that affects many-to-one communication patterns in large data centres which is caused by a complex interaction between data centres applications, underlying switches, network topology and the dynamic behaviour of TCP [4]. A typical incast topology is shown in Figure 1.

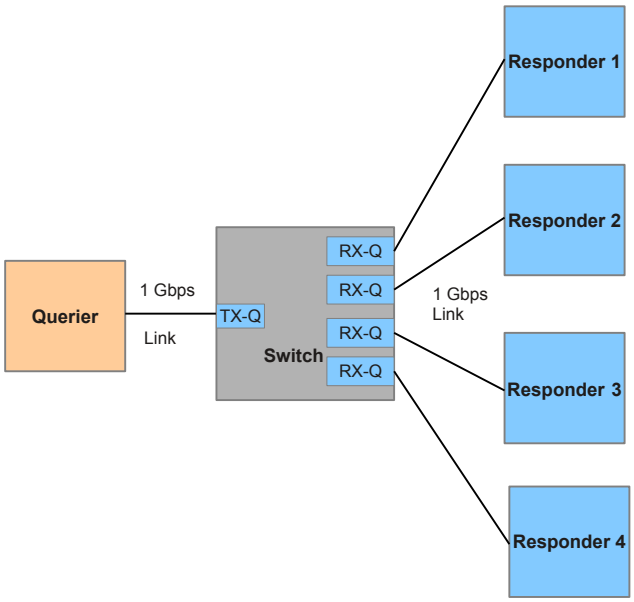

Figure 1. Typical Incast Topology in Data Centre Networks

In data centre networks, a querier is connected to multiple responders via a switch. In Figure 1, all physical network links are configured with 1Gbps. There are 4 responders and 1 querier. The architecture of the switch plays an important role in transferring incoming packets from the ports to the desired output port. In the context of incast network topology, after a querier sends a query, there is a possiblity that multiple responders will respond to the query. Since there are multiple high speed (1Gbps) links connected to the incoming ports and only one high speed link for the outgoing port, packets will arrive at the switch at a rate faster than the processing rate of the switch. Therefore, packets will be queued at the *receiving* or *ingress queue (RX-Q)* of incoming ports and waited to be processed accordingly. The *transmitting* or *egress queue (TX-Q)* is generally a small queue compared to the RX-Q.

Incast congestion is an issue because the responders are sending packets much faster than the processing speed of the switch and at some point it will overflow the ingress queue buffer. In this case, packets will be dropped, causing packet loss. Consequently there is an increase in the queuing delay of packet flows and a decrease in application level throughput to a level far below the link bandwdth.

B. The Switch Model

Our NS-3/FreeBSD Network Stack VM Appliance implements a switch architecture that resembles the switch architecture in reality. Figure 2 illustrates the architecture of the switch implemented in this simulator.

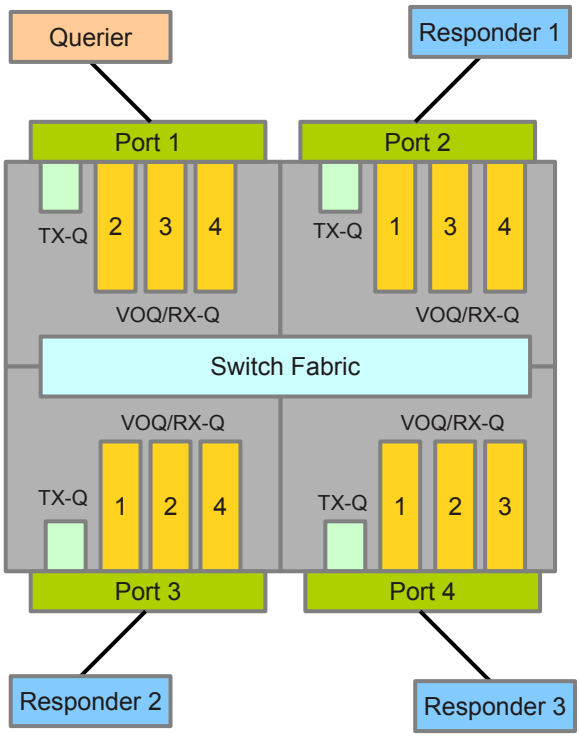

Figure 2. Architecture of the Switch Model

In Figure 2, the switch has 4 ports with 1 connected to the querier and the other 3 connected to the responders. Each port has 3 virtual output queues and a transmit queue buffer. It is noted that VOQs are not independent memory space (they are not physically separated), they share a common receive queue buffer determined by the number of bytes and the number of cells. VOQs are logically separated, where the number in the virtual output queues as shown in Figure 2 denotes the destination port of the incoming packets. Since the destination port of each packet is determined before the packet enters the queue, this information is tagged on the packet and placed in the queue buffer. The switch fabric in the middle of the switch is where all the packets traverse across from the receive queue to the transmit queue of the destination port. Subsection II-C will discuss why VOQs are implemented instead of the traditional queuing mechanism.

C. Oversubscription and Head of Line (HoL) Blocking Problem

In order to understand the importance of the switch architecture as described in II-B, we must first address the oversubscription and Head of Line (HoL) Blocking problem.

1) Oversubscription: High end network switches are designed in a way that have the potential of oversubscribing access to the switching fabric [5]. The oversubscription design allows the links connected to the switch ports to have higher link capacity than the switching fabric, assuming not all connections would be transmitting packets to switch at their maximum sending rate. The reason for doing this is to reduce the manufacturing cost of switches yet allow the links to send data at high speed. In a burst of traffic or during peak times in data centres, oversubscription and congestion occurs.

Some ways of dealing with oversubscription are port bandwidth reservation that allows dedication of specific amount of bandwidth to certain ports. Another option is to design a switch that operates as a full-rate, non-oversubscribed module that allows all links to transmit that their maximum rate. Oversubscription along with advanced traffic management capabilities enables cost optimization [5].

2) Head of Line (HoL) Blocking: In switches that handle high incoming traffic rates, a few congested output ports can impact the overall performance of the switch by the Head Of Line (HoL) blocking phenomenon. HoL Blocking occurs whenever traffic waiting to be transmitted prevents traffic destinated elsewhere to be transmitted.

Most switches have buffered input ports, a switch fabric and buffered output ports using the First-In-First-Out (FIFO) queuing mechanism at the input buffers. By using FIFO, the packet that arrives earliest at the receive queue buffer is processed [6]. If the packet cannot be processed due to a full transmit queue buffer at the destination port, all packets that arrives later cannot be processed, thus the earlier packet blocks the later packets.

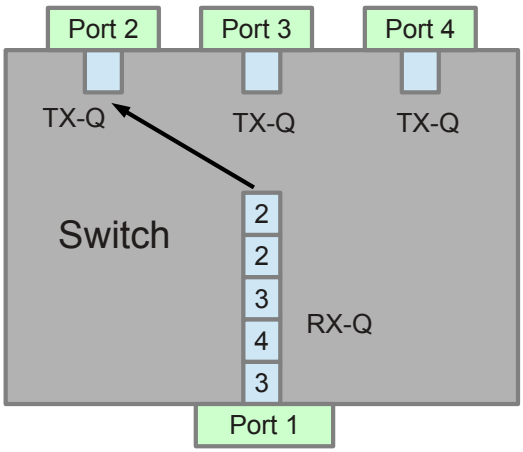

Figure 3. Head of Line (HoL) Blocking

As illustrated in Figure 3, packets arrived at Port 1 with the port destination sequence of 2, 2, 3, 4 and 3. Generally, the transmit queue buffer (TX-Q) at the destination port is much smaller than the receive queue buffer (RX-Q), usually just a packet or two long in data centre switches. In this example, assuming TX-Q is only 1 packet long which means that the transmitting port can only process 1 packet at a time, and no other packets are allowed to be at TX-Q when the queue buffer is full.

FIFO is implemented in this switch and the decision as to which packet is destinated to which port is only decided at the end of the receive queue buffer just before it is transmitted across the switch fabric. When the the earliest packet (packet destinated for Port 2) is processed, the TX-Q at Port 2 is full thus the next packet destinated for Port 2 cannot be processed. This can limit the performance and throughput of the switch because the switch cannot process subsequent packets destinated for Port 3 and 4, even though the queue buffer at Port 3 and Port 4 is empty. This phenomenon is known as HoL blocking, where a series of packets is blocked by the earlier packet. A HoL blocking architecture prevents the delivery of full amount of bandwdith even if individual switching modules are not oversubscribed or not all the ports are transmitting simultaneously. This limitation is overcomed by the implementation of Virtual Output Queuing (VOQ) in switches.

D. Virtual Output Queuing (VOQ)

A significant feature of VOQ is the assignment of logical queues for each output port to each input port. In other words, packets are tagged with information of destinated ports once they enter into the switch and then they are placed in logical RX-Qs at the input port. Each

input port maintains a separate logical queue for each output port. It is important to note that each RX-Q is still a shared input buffer with no defined queue capacity for each virtual output queue. Packets are queued into the RX-Q according to the destinated port number and processed with parallel processing mechanism.

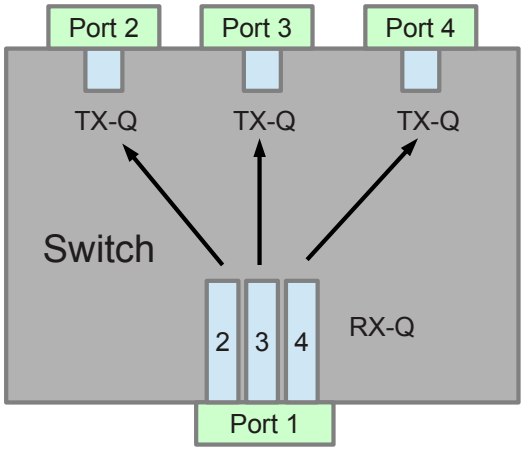

Figure 4. Virtual Output Queuing (VOQ)

As illustrated in Figure 4, dedicated VOQs for packets with different port destinations are assigned to Port 1. When a stream of packets enters Port 1, the placement of packets in RX-Qs is determined by the destinated port number of the incoming packets. HoL blocking problem is solved as multiple packets destinated for different ports can be processed simultaneously. The implementation of VOQ with an effective scheduling algorithm can achieve full throughput performance. VOQ queuing mechanism is implemented in our simulator. This will be discussed in detail in Section IV.

III. Simulating an Incast Scenario

A. Running an Incast Simulation

The incast simulation constructs a set of nodes connected via a virtual output queued Ethernet switch model, where one of the nodes is the querier and the other nodes are the responders. The traffic generated by all nodes (query traffic from the querier and response traffic from the responders) is designed to simulate TCP incast when the responses converge towards the querier's switch port. The commands to run the incast simulation by using the appliance are shown below:

```
# SSH from local machine
ssh -XY -p2222 ns3@localhost

# In VM environment
cd /usr/home/ns3/ns-3-allinone/nsc
./scons.py freebsd-9stable
cd /usr/home/ns3/ns-3-allinone/ns-3-dev
./waf --with-nsc=../nsc/freebsd-9stable
      configure
./waf build -j2
```

To see which command line parameters are available for run-time tweaking:

```
cd /usr/home/ns3/ns-3-allinone/ns-3-dev
./waf --run="incast --PrintHelp"
```

The user arguments available for run-time tweaking are:

```
-- queries: Number of query/response
-- responders: Number of responder nodes
-- query-size: Per-responder query size
   (bytes)
-- response-size: Response size (bytes)
-- rxqueue-cells: Ingress queue size in
   cells (number of cells)
-- txqueue-cells: Egress queue size in
   cells
-- bytespercell: Number of bytes per cell
   (Queue buffer size = number of cells x
   bytes per cell)
-- backplane-latency: Number of microseconds
-- data-store: Path to store simulation data
-- duration: How long to run simulation
-- nsc-stack: Name of the NSC stack to use
```

These parameters will be explained in detail in Section III-B.

A simulation trial with the following parameters is done.

- Number of responders = 2 responders
- Application level response size = 100000 bytes
- Ingress queue (RX-Q) size: 256 cells
- Egress queue (TX-Q) size: 16 cells
- Queue buffer cell size: 384 bytes
- Number of queries to a each responder: 1 query
- Query size: 500 bytes
- Backplane latency of switch: 6μs

To run this simulation, issue the command:

```
cd /usr/home/ns3/ns-3-allinone/ns-3-dev
./waf --run=incast --data-store=/home/ns3/
      simulation_data --responders=2
      --rxqueue-cells=256
      --txqueue-cells=16
      --response-size=100000"
```

The simulation data will be stored in `/home/ns3/simulation_data` as a directory with runtime date, time and other associated parameters. For example, the simulation data directory run on July 1, 2014, at 10:17am is:

```
simname=ns3-dev-incast-debug:
datetime=20140701-10h17m04s:
responders=2:qrylen=500:
rsplen=100000:rxcells=256:
txcells=16:cellbytes=384:
backplanelatency=6:queries=1
```

This directory contains packet trace files (*pcap)* captured at all nodes and net devices, queue trace files and a SIFTR [7] log file that logs the state of TCP for the connection.

B. Switch Model in the Simulator

The switch model described in Section II-B is defined as `BridgeNetDevice` in the `bridge-net-device.cc` [1] module. It is a major overhaul of the default NS-3 bridge module.

Important incast simulation parameters to note as shown in Section III-A are:

- Number of responses per query (`queries`): Default value is 1.
- Number of responders (`responders`) : The number of responders will affect the amount of incast traffic. Default value is 2.
- Query size per responder in bytes (`query-size`): Default value is 500.
- Response size in bytes (`response-size`): The application level response size of the responder. The default value is 5000.
- Ingress queue size (RX-Q) in number of cells (`rxqueue-cells`): The size virtual output queues in number of cells for the incoming packets to be buffered. The default value is 256 cells.
- Egress queue size (TX-Q) in number of cells (`txqueue-cells`): The outgoing queue buffer in cells of the transmitting port. The egress queue size is much smaller than the ingress queue size. The default value is 16 cells.
- Number of bytes per cell (`bytespercell`): Default value is 384 bytes.
- Backplane latency (`backplane-latency`): The amount of time taken for a packet to traverse the switch fabric from the virtual output queue to the egress queue. The default value is 6 microseconds.
- Simulation duration (`duration`): The default value is 10 seconds.
- Name of NSC stack to be used in the simulation (`nsc-stack`): The default NSC stack is FreeBSD 9-STABLE.

The queue buffer size of the RX-Q or the VOQs for any port is defined as the number of cells multiplied by the number of bytes per cell. Therefore the default queue buffer size for the RX-Q is *256 cells * 384 bytes = 98304 bytes*.

1) Simulating Backplane Latency: In real world switches, the backplane latency is defined as the amount of time taken for a packet to traverse the switch fabric from the virtual output queue to the egress queue. There is a finite amount of time for a packet to traverse the switch fabric and this time plays a role in determining the processing speed and throughput of the switch. However, in this simulator, there is no simulated switch fabric so a backplane latency is simulated using a different mechanism. In the simulator, the backplane latency is simulated by holding the packet that is ready to be transmitted at the virtual output queue or the receive queue buffer for a finite amount of time defined by the backplane latency, then it instantaneously arrives at the transmit queue of the destinated port.

2) Simulating Back Pressure: Back pressure occurs when there is a buildup of packets behind the transmitting port. In other words, the TX-Q buffer is full and is incapable of receiving any more data. In this situation, there exist a back pressure from the RX-Q to put packets in the destinated transmit queue buffer. It halts its sending until the transmit queue is emptied and is capable of receiving packets.

In real switches, a switching algorithm is implemented so the scheduler knows that the transmit queue is full and is incapable of handling anymore packets, so it will stop the sending of other packets from the virtual output queues. This process is implemented in a slightly different way in this simulator. During the simulation, the scheduler will 'wakeup' and do a round robin, a process of checking all the RX-Qs to see if there any packets that are ready to be transmitted. If packets in the RX-Q are ready to be transmitted, the scheduler will take the packets and place it in the TX-Q of the destinated port. In the case where the TX-Q is empty, then the scheduler will successfully place the packets in the TX-Q and continue to do round robin scheduling. However, if the TX-Q is full, the scheduler will still attempt to

[1]The module is located at `/usr/home/ns3/ns-3-allinone/ns-3-dev/src/bridge/model/bridge-net-device.cc`

push the packet in the full TX-Q but without success. This unsuccessful attempt causes the scheduler to put the packet back to the RX-Q. This process of pulling out packets from the RX-Q and pushing it into a full TX-Q queue is not implemented in the real world switches. In order to resemble real world data as closely as possible, the simulation time is stopped when this situation occurs. Although there is a finite amount of real time elapsed when this process occurs, there is no time elapsed in the simulation. Therefore, from the perspective of observing the simulated time logged in the queue trace log files, it seems that the scheduler somehow knows that the TX-Q buffer is full and is incapable of handling anymore packets, so it does not attempt to push packets into a full queue, which is a real world behaviour of a switch.

C. Modifications Made to the Simulator

The major modifications [8] made are:

- Added a new `QueryResponseApplication` model to simulate one-to-many query-response network traffic patterns commonly found in data centres.
- Added a new `MultiQueue` model to simulate queues which have a fixed resource limit that can be shared between logically partitioned sub-queues. This model forms a building block for the `BridgeNetDevice` overhaul.
- Overhauled the `BridgeNetDevice` model to turn it into a round-robin, input queued device utilising virtual output queuing (VOQ) at each ingress port.
- Extended the interface between NS-3 and NSC to support socket upcalls, generic route manipulation and an enhanced mechanism for NSC to obtain the virtual time from the simulator.
- Added support for the RFC1323 TCP window scale option to NS-3's TCP model.
- Modified Queue tracing to make it possible to use a single callback function for selective per-queue and per-action tracing.
- Added an incast simulation which utilises the VOQ of `BridgeNetDevice` and `QueryResponseApplication` to simulate simple TCP incast scenarios.
- Added a "dumbbell" simulation which utilises the `PointToPointDumbbellHelper` and `BulkSendApplication` to simulate simple TCP dumbbell networks.

IV. Understanding the Output

A. Debugging Output

Section III-A showed the commands for running an incast simulation. In order to run the simulation with the same parameters in debugging mode, the environment variable `env` is set to the name of the class associated with the model. In this case, we are interested in the debugging output of the switch model, `bridge-net-device.cc`.

```
env NS_LOG="BridgeNetDevice":INFO
./waf --run="incast
--data-store=/usr/home/ns3/simulation_data
--responders=2 --rxqueue-cells=256
--txqueue-cells=16 --response-size=100000"
```

Part of the debugging information is shown below:

```
BridgeNetDevice:DoDispose()
BridgeNetDevice:DoDispose(): [INFO ]
Port 0 txQueueVoqBytes=0
txQueueVoqBytesHighwater=101864
rxQueueBytesHighwater=1140
rxQueueCellsHighwater=10
rxQueuePacketsDropped=0

BridgeNetDevice:DoDispose(): [INFO ]
Port 1 txQueueVoqBytes=0
txQueueVoqBytesHighwater=640
rxQueueBytesHighwater=50252
rxQueueCellsHighwater=133
rxQueuePacketsDropped=0

BridgeNetDevice:DoDispose(): [INFO ]
Port 2 txQueueVoqBytes=0
txQueueVoqBytesHighwater=570
rxQueueBytesHighwater=51612
rxQueueCellsHighwater=136
rxQueuePacketsDropped=0

Port 0
txQueueVoqBytesHighwater Value =101864
totalPacketLoss =0
BridgeNetDevice:~BridgeNetDevice()
```

In order to understand the debugging information, consider Figure 5.

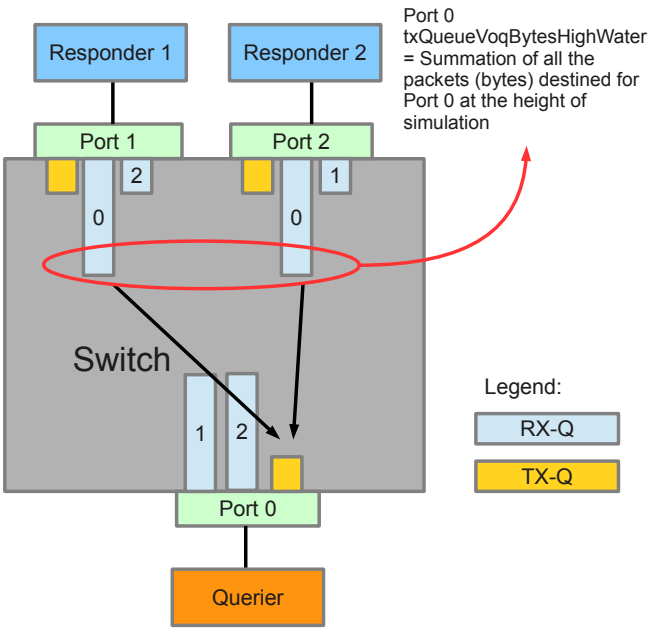

Figure 5. Debugging Information Illustration

Figure 5 shows the illustration of the switch architecture with 3 ports. Port 0 is connected to the querier whereas Port 1 and 2 are connected to the responders. Since all the packets arrived at the VOQs are destinated for the querier connected to Port 0, only the RX-Q '0' is filled with data packets, whereas other RX-Qs are empty (this is illustrated in 5 with long logical RX-Q '0' and short logical RX-Qs for the others).

From the debugging information, it is known that Port 0 at its worst case had 101864 bytes (`txQueueVoqBytesHighwater`) waiting to be transmitted out of it. In other words, all the RX-Qs at Port 1 and 2 had a sum total of 101864 bytes in them at the height of the simulation (worst case scenario). The total peak queue occupancy can be lower than `txQueueVoqBytesHighwater` since the peaks of RXQs connected to different responders can occur at different times (as illustrated in Section IV-B). The debugging information does not provide the information of the exact simulation time of this event, where the RX-Qs of the responders experience its height. It only provides the information about the maximum amount of packets in bytes that are waiting to be transmitted out of Port 0. The `rxQueueBytesHighwater` gives the information of the maximum amount of packets in bytes that are received by Port 0 (queries), it is the sum of all packets in the RX-Qs of Port 0. Whenever there is a queue buffer overflow at the RX-Q, packets are dropped and the variable `rxQueuePacketDropped` will give the information on the number of packets dropped.

B. *Queue Occupancy*

NS-3 is a discrete time event driven simulator, therefore the incast simulation script `incast.cc`[2] is written to log packet enqueue (packets entering the RX-Q), dequeue (packets leaving the RX-Q) and drop (packets dropped due to full queue buffer) events with a callback function. The queue occupancy of the RX-Q will be logged in a queue trace file when one of these events occur. These files stored in the simulation directory. Figure 6 and 7 are plotted from the simulation trial described in Section III-A.

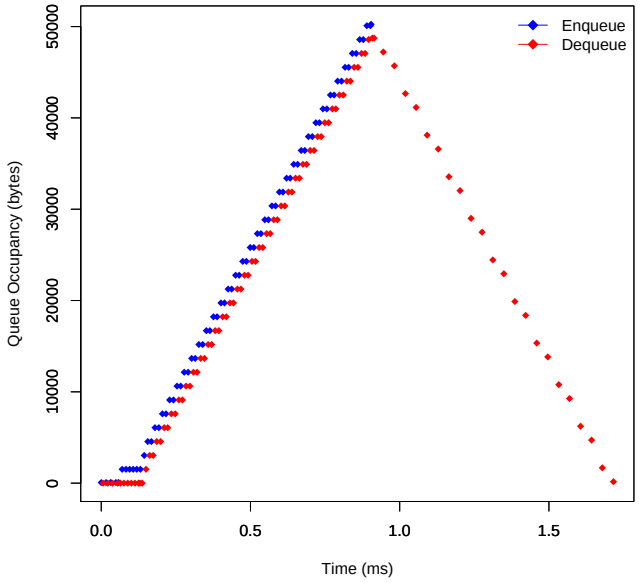

Figure 6. Queue Occupancy of Responder 1 *vs* Time

[2]The script is located at `/usr/home/ns3/ns-3-allinone/ns-3-dev/examples/tcp/incast.cc`

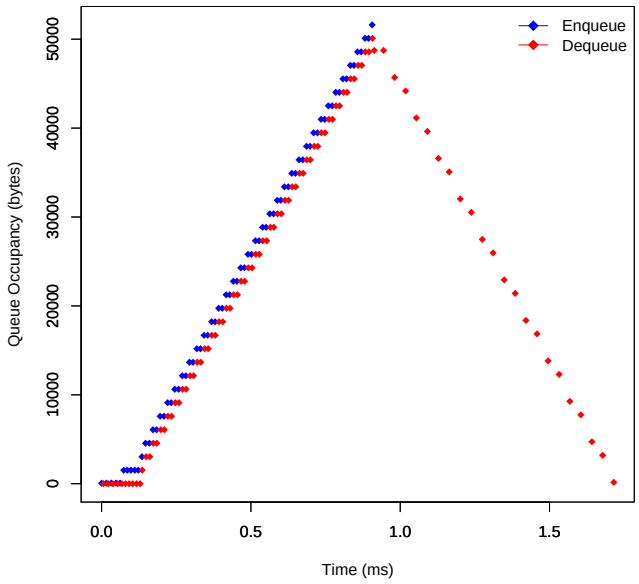

Figure 7. Queue Occupancy of Responder 2 *vs* Time

Figure 6 shows the queue occupancy of the RX-Q connected to Responder 1. The incast scenario is a leaky bucket scenario where the total input rate is greater than the output rate. Figure 6 shows packets enqueuing and dequeuing simultaneously before attaining its peak queue occupancy, which in this case is 50252 bytes.

The RX-Q of Responder 2 exhibits a very similar behaviour with a peak queue occupancy of 51612 bytes as shown in Figure 7. This yields to a total peak queue occupancy of 101864 bytes, which means that there are 101864 bytes waiting to be transmitted out the TX-Q connected to Port 0 at its worst case (`txQueueVoqBytesHighwater`).

Notice that both RX-Qs are configured with *256 cells * 384 bytes = 98304 bytes*, which is less than the application level response size of 100000 bytes. However, Figure 6 does not indicate a full queue, therefore no packet losses have occurred. This is due to the fact that there are only 2 responders in this scenario and the scheduler can process and transmit packets fast enough to prevent overflowing the queue buffer.

For Responder 1, direct inspection of the queue trace file shows that the RX-Q achieved its peak queue occupancy at 0.9041ms and packets are completely drained out of the queue at 1.7161ms. For Responder 2, the RX-Q achieved its peak queue occupancy at 0.9060ms and the packets are completely drained out of the queue at 1.7152ms. This shows a high correlation between the burst of packets from 2 responders. Since the peak queue occupancy of both RX-Qs occurred at a slightly different time, the total number of bytes waiting to be transmitted out of TX-Q connected to Port 0 at the height of the simulation is lower than `txQueueVoqBytesHighwater`.

The dequeue events after the peak queue occupancy for both Responder 1 and 2 reflect each queue being drained at the share of the line rate of Port 0, which is roughly 0.5Gbps. The downward gradients of Figure 6 and 7 is consistent with each queue draining at the rate of roughly 0.5Gbps[3].

In an incast scenario, the number of responders increases to a point where the scheduler could not schedule and process the correlated responses from the responder fast enough, therefore the RX-Qs connected to the responders will be full and subsequent packets will be dropped, causing TCP timeouts and adding delays to the query-response process. As a result, the application level throughput decreases.

V. Conclusions

This technical report presents an introduction to the "Network Simulator 3 (NS-3)/FreeBSD Network Stack VM Appliance" and shows how this simulator can be used to simulate large scale incast topology networks commonly used in data centres. A significant advancement made in this simulator is the availablity of the option of selecting an actual FreeBSD 9-STABLE network stack to simulate data centre networks and the implementation of virtual output queues in the switch model. As a result, simulation data obtained are more likely to resemble the behaviour of real networks with end hosts operating FreeBSD 9-STABLE. Switching concepts such as HoL Blocking and VOQs are presented to show the importance of VOQs in data centre networks. It also shows how backplane latency and back pressure is simulated in the switch model, which varies from switch implementations in the real world. Illustrations and explanations of the debugging output are presented so that users can extract relevant information and interpret the simulation results. An incast scenario is presented to illustrate the fluctuations of queue occupancy in RX-Qs connected to the responders. Future work involves investigating the TCP incast problem in greater detail and testing modified TCP algorithms to mitigate incast congestion by using this simulator.

[3]The time required to drain roughly 50000 bytes at the rate of 0.5Gbps is approximately 0.8ms, which is close to the 0.81ms it took for both responder traces to drain their queues.

ACKNOWLEDGEMENTS

This project has been made possible in part by a gift from The Cisco University Research Program Fund, a corporate advised fund of Silicon Valley Community Foundation.

REFERENCES

[1] NS-3 Consortium, "Network Simulator 3 (NS-3) and Network Simulation Cradle (NSC)." [Online]. Available: http://nsnam.org

[2] G. Armitage and L. Stewart, "Incast Congestion Control," August 2013. [Online]. Available: http://caia.swin.edu.au/urp/incast/

[3] D. Nagle, D. Serenyi, and A. Matthews, "The Panasas ActiveScale Storage Cluster: Delivering Scalable High Bandwidth Storage," in *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, ser. SC '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 53–. [Online]. Available: http://dx.doi.org/10.1109/SC.2004.57

[4] Y. Chen, R. Griffit, D. Zats, and R. H. Katz, "Understanding TCP Incast and Its Implications for Big Data Workloads," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2012-40, Apr 2012. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-40.pdf

[5] P. Perazzo, "Switch Architectures and Highways," May 2009. [Online]. Available: http://blogs.cisco.com/datacenter/switch_architectures_and_highways/

[6] D. Barnes and B. Sakandar, *Cisco LAN Switching Fundamentals*. Indianapolis: Cisco Press, 2005.

[7] L. Stewart and J. Healy, "Statistical Information for TCP Research (SIFTR)," 2010. [Online]. Available: http://www.freebsd.org/cgi/man.cgi?query=siftr&sektion=4&manpath=FreeBSD+7.4-RELEASE

[8] G. Armitage and L. Stewart, "CAIA NS-3/NSC FreeBSD 9 Based Simulation Environment Virtual Machine Image v0.1 Readme," September 2013. [Online]. Available: http://caia.swin.edu.au/urp/incast/tools/README.caia-freebsd9-amd64-caia-ns3-release-0.1.txt