Trends in Applied Sciences Research

ISSN 1819-3579

Academic
Journals Inc.

www.academicjournals.com

Trends in Applied Sciences Research 5 (3): 177-187, 2010
ISSN 1819-3579

Setting up a Virtual Laboratory for Evaluation of Congestion Control Algorithms in TCP/IP Networks

[1]Sogol Babainejad, [2]Sasan Babainejad, [3]N. Bigdeli and [4]K. Afshar
[1]Department of Electrical Engineering, Semnan University,
Damghan Road, Semnan, Iran
[2]Department of Communication and Information Technology,
National Iranian Oil Company, Tehran, Iran
[3]Linear Systems Control Lab. Department of Electrical Engineering,
Imam Khomeini International University, Qazvin, Iran
[4]Laboratory of Linear Systems Control, Department of Electrical Engineering,
Imam Khomeini International University, Qazvin, Iran

Abstract: This study proposed a method for synthesizing a virtual laboratory for evaluation of congestion control algorithms in TCP/IP networks (CCVL). In CCVL (i.e., congestion control virtual laboratory), the required hardware and the risk of hardware malfunctioning is noticeably reduced. The result is a low-cost compact portable virtual laboratory which also has the capability of connecting to the real internet and interchanging and gathering test data with/from it, if applicable. This method can be greatly useful in congestion control laboratories at universities and research institutes.

Key words: Congestion control, virtual laboratory, network simulation, network emulation

INTRODUCTION

Congestion control has become a major problem in computer network management area in the recent years and a wide range of algorithms have been proposed for solving this problem in the literature (Bigdeli and Haeri, 2009a; Chrysostomou *et al*., 2009; Manfredi *et al*., 2009; Aweya *et al*., 2008; Wang *et al*., 2008). These algorithms are mainly categorized as heuristic, mathematical and control theory-based models (Kelly *et al*., 1998). The TCP congestion control methods being TCP Reno, Tahoe, New Reno, Sack, Vegas, Westwood and control-theory based AQM-controllers such as P, PI, PD, sliding mode and Coefficient Diagram Method (CDM) (Bigdeli and Haeri, 2007; Bigdeli and Haeri, 2009b; Sun *et al*., 2003; Hollot *et al*., 2002; Fengyuan *et al*., 2002; Ryu *et al*., 2003) as well as heuristic methods like Blue (Feng *et al*., 1999) and Purple (Pletka *et al*., 2003) are examples of developed congestion control schemes. In spite of such a wide range of developed congestion control methods, just a few of them has been employed/implemented practically. The reason is that there is no proper and reliable evaluation test bench for the proposed congestion control methods. The evaluations are mainly based on simple scenarios which are implemented via Network Simulator 2 (NS-2) (refer to The Network Simulator, ns-2) scripts on one computer. Therefore, the derived results are not sufficiently reliable (Floyd and Kohler, 2002).

Corresponding Author: Sogol Babainejad, Department of Electrical Engineering,
Semnan University, Damghan Road, Semnan, Iran

In synthesizing an evaluation test bench three different aspects should be considered. The structure of the computer network, the topology and the traffic passing through the network are the two main subjects of interest. In Bigdeli (2007), these two parts of an evaluation test bench has been considered closely and a simulation test bench based on characterization of the topology and traffic of the real networks has been synthesized. Implementation complexity and packet-level examination of a congestion control scheme is another matter of interest which should be considered. However, examination of these properties via a simple NS-2 script on one computer is not possible. Due to the complexity of real traffic synthesizing and implementation, setting up a real evaluation test bench is not a straight-forward task. On the other hand, implementation of such a network is not cost-effective. The main portion of the cost in a network is the cost of hardware components such as computer sets, hubs, switches, cables and etc. Besides, the implementation of user-defined congestion control algorithms is impossible in commonly available routers. In such cases, computers should be programmed as routers, which is a professional task and cannot be performed easily. Nonetheless, special software and protocols are required to connect the machines together.

Based on the above arguments, in this study we propose a method for synthesizing a virtual laboratory for evaluation of congestion control algorithms in TCP/IP networks (CCVL). In CCVL, the traffic exchange is based on emulation capability of NS-2 (refer to the Network Simulator, ns-2) and the structure of a simplified real network is implemented in one machine via Xen (http://www.cl.cam .ac.uk/research/srg/netos/xen/readmes/user/user.html). Therefore, the required hardware is noticeably reduced. In this approach, the physical features of an actual network are transferred to an artificial network where NS-2 (refer to The Network Simulator, ns-2) is used as the network simulator and manager for testing. That is, NS-2 is connected to the artificial network where performance of the network is studied prior to complete fabrication. In this manner, the potential risk of hardware malfunctioning is reduced considerably. The result is a low-cost compact portable virtual laboratory which also has the capability of connecting to the real internet and interchanging and gathering test data to/from it, if applicable. This method can be greatly useful in congestion control laboratories at universities and research institutes.

MATERIALS AND METHODS

Congestion Control Virtual Laboratory (CCVL) Implementation

Hardware Implementation

Xen kernel facilities are used to create and run the real network and virtual machines. Virtual machines are employed to act as the real nodes in the virtual laboratory (Barham *et al*., 2003; Fraser *et al*., 2004). Figure 1 shows the schematic of the connected virtual Ethernet interfaces via Xen. Xen creates, by default, seven pairs of connected virtual Ethernet interfaces for use by the domain "dom 0", "veth 0" is connected to "vif 0.0", "veth 1" is connected to "vif 0.1", etc., up to "veth 7" which is connected to "vif 0.7". It is possible to use them by configuring IP and MAC addresses on the "veth #" end and then attaching the "vif 0 #" end to a bridge (Barham *et al*., 2003; Fraser *et al*., 2004; http://www.cl.cam.ac.uk/research/srg/netos/xen/readmes/user/user.html).

Every time a running "domU" instance is created, it is assigned a new domain id number. It is not possible to manipulate the number. The first "domU" will be id #1. The second one started will be #2, even if #1 is not running anymore.

For each new "dom U", Xen creates new connected virtual Ethernet interface, where one end of each pair is within the "dom U" and the other end exists within "dom 0". For Linux

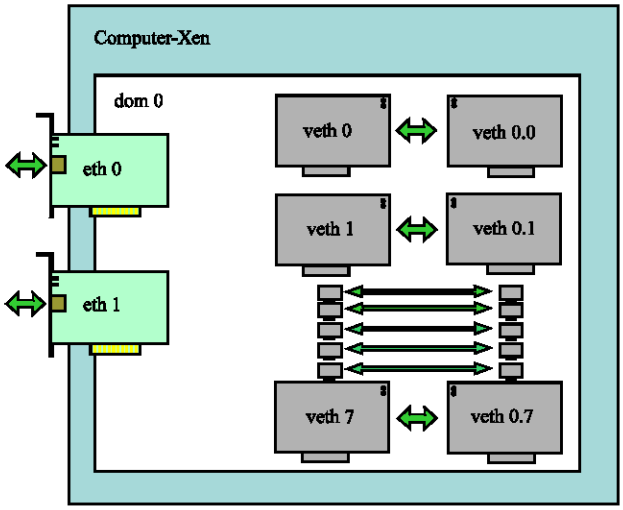

Fig. 1: Schematic of the connected virtual Ethernet interfaces via Xen

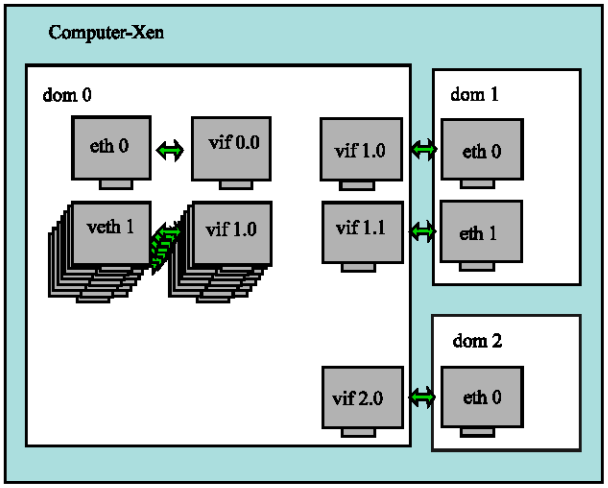

Fig. 2: Logical network card connected between dom 0 and dom 1

"dom U" 's the device name it sees is named "eth 0". The other end of the virtual Ethernet interface pair exists within "dom 0" as interface "vif<id#>.0". For example, the "eht 0" of "dom 5" is attached to "vif 5.0". If multiple network interface for a "dom U" is created, it's ends will be "eth 0", "eth1", etc, whereas the "dom 0" end will be "vif<id#>.0", "vif<id#>.1" , etc. Figure 2 shows such a logical network card connected between "dom 0" and "dom 1").

When "dom U" is shutdown, the virtual Ethernet interfaces for it are deleted. Virtualized network interfaces in domains are given Ethernet MAC addresses. By default Xen will select a random address. This will differ between instantiations of the domain. If it is required to have a fixed MAC address for a domain (e.g., for using DHCP) then this can be configured using the "mac=" option to the "vif" configuration directive (e.g., vif=['mac=aa:00:00:00:00:11'].

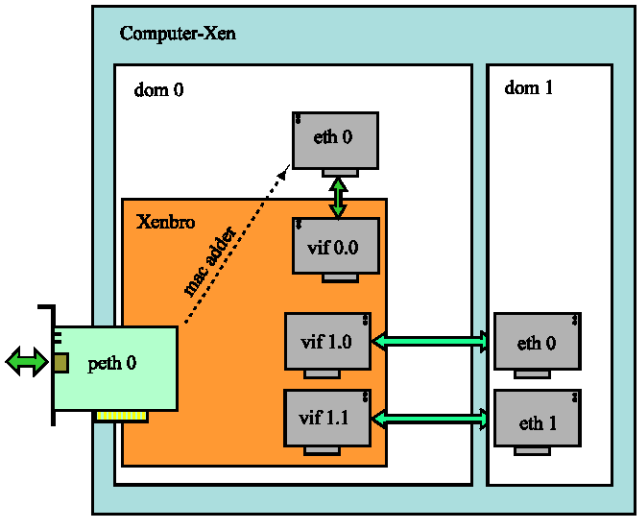

Fig. 3: Illustration of Network Bridge and Vif Bridge

When MAC addresses should be used, it is necessary to ensure that a unicast address is chosen. That is, one with the low bit of the first octet set to zero. It is the best to keep to the range of addresses declared to be locally assigned (rather than allocate globally to hardware vendors).

These have the second lowest bit set to one in the first octet. The correct form of the MAC address is "XY:XX:XX:XX:XX:XX" where, "X" is any hexadecimal digit and "Y" is one of the 2,6,A or E. It is recommended to use a MAC address inside the range "00:16:3e:xx:xx:xx". This address range is reserved for using by Xen.

The default Xen configuration uses bridging within domain 0 to allow all domains to appear on the network as individual hosts. If extensive use of IP tables ("iptable") is made in domain 0 (e.g., a firewall) then this can affect bridging because bridge packets pass through the PREROUTING, FORWARD and POSTROUTING IP table chains. This means that packets being bridge between guest domains and the external network will need to be permitted to pass those chains. The most likely problem is the FORWARD chain being configured to DROP or REJECT packets (this is different from IP forwarding in the kernel) (Barham *et al*., 2003).

Arrived packets, at hardware, are handled by "dom 0" Ethernet driver and appears on "peth 0", "peth 0" is bound to the bridge, so it is passed to the bridge from there. This step is run on Ethernet level, no IP addresses are set on "peth0"or bridge. Therefore, the bridge distributes the packets, just as the manner a switch would. Filtering at this stage would be possible with "ebtables". Now there is a number of "vifx.y" connected to the bridge, it decides where to put the packet based on the MAC address of the receiver. The "vif" interface puts the packet into Xen, which then puts the packet back to the domain that the "vif" leads to. The target device in the "dom 0/domU" finally has an IP address. It would be possible to apply IP tables filtering here. As shown in Fig. 3 when Xen starts up, it runs the network-bridge script as (Barham *et al*., 2003; Fraser *et al*., 2004):

- Creates a new bridge named "xenbr 0"
- Real Ethernet interface "eth 0" is brought down

- The IP and MAC addresses of “eth 0” are copied to virtual network interface “veth 0”
- Real interface “veth 0” is renamed “peth 0”
- Virtual interface “veth 0” is renamed “eth 0”
- “peth 0” and “vif 0.0” are attached to bridge “xenbr0”
- The bridge, “peth 0”, “eth 0” and “vif 0.0” are brought up

It would be better to have the physical interface and the “dom 0” interface separated. It would however, be possible for example by setting up a firewall on “dom 0” that does not affect the traffic to the “domU” 's (just for protecting “dom 0” alone). When a “domU” starts up, xend (running in “dom 0”) runs the vif-bridge script, which:

- Attaches “vif<id#>.0” to “xenbr 0”
- “vif<id#>.0” is brought up

Software Implementation (Emulation)

Once, the real and virtual Ethernet interfaces are set up, the traffic passing and manipulation among them should be created and/or managed via proper software. The employed software in this study is NS-2 (refer to The Network Simulator, ns-2), which is capable and free-access software. It has good emulation capabilities which are vital for constructing the virtual laboratory.

Emulation refers to the ability to introduce the simulator into a live network. There are two type of use for such a facility, depending on whether the simulator appears to the end station as a router or as another end station (Fall and Varadhan, 2007). In the first mode the simulator can grab the live traffic from the real network and pass them through a simulated network and then inject them back to the live network (the opaque mode). In the second mode the simulator include the traffic source or sinks that communicate with the real-world entities (the protocol mode) (Fig. 4). In the opaque mode, the live network packets are passed through the simulator without being interpreted. Network packets maybe dropped, delayed, re-ordered or duplicated by the simulator.

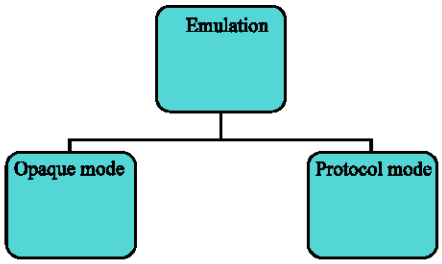

Fig. 4: Emulation diagrams

The opaque mode is useful in evaluating the behavior of the real-world implementations when subjected to adverse network conditions that are not protocol specific. Notice that in the opaque mode, the live traffic can come from a real network or a trace file. In both cases it is necessary to specify the source of the live traffic in the related module in the Emulation file (For example Pcap/File) (Alefiya, 2002). Figure 5 shows the interaction of simulator with the real network in the opaque mode.

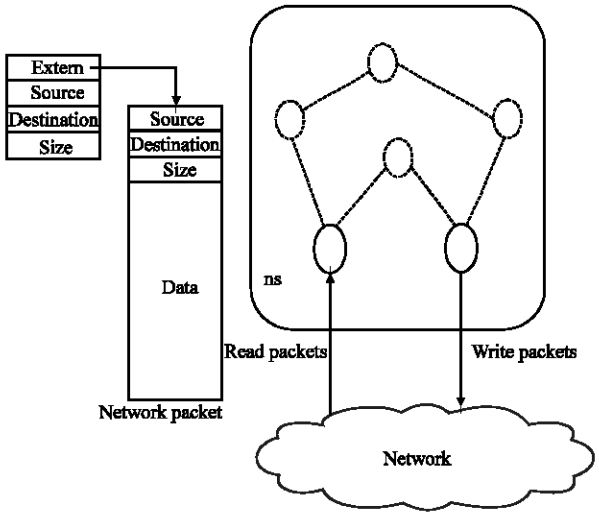

Fig. 5: The interaction of NS-2 simulator with the real network in the opaque mode

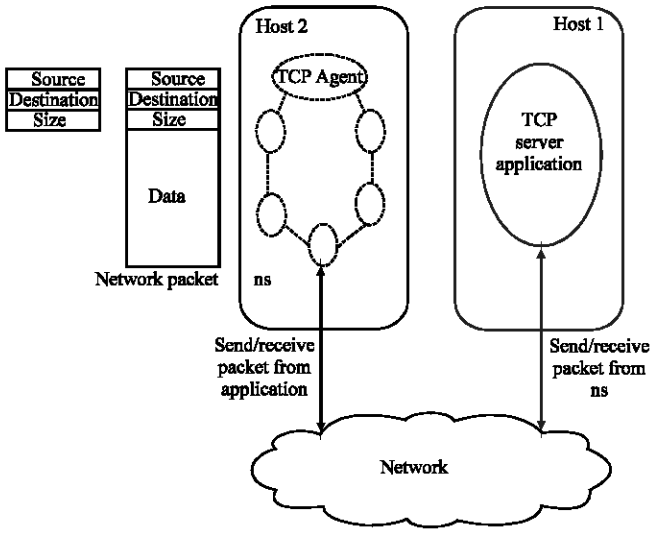

Fig. 6: The interaction of NS-2 simulator with the real network in the protocol mode

In the protocol mode, the simulator is used as an end-point to generate TCP traffic. A TCP agent within NS-2 interacts with a real-world TCP server and can receive data from the external applications. This mode can be used for end to end application testing, protocol and conformance testing (Alefiya, 2002). Figure 6 shows the interaction of simulator with the real network in the opaque mode.

Implementation Critiques

Connectivity via Xen

As stated earlier, Xen is used to create real network and virtual machine. Between different scenarios of making a network, the network-multinet is the most appropriate one for

this kind of studies. The virtual machine's Ethernet (eth 0) has to be configured to connect to the network bridge (xenbr 3) which is created in the network-multinet scenario. The real machine's Ethernet (veth 3) is also connected to xenbr 3. Network-multinet sets the IP address and subnet mask of veth 3 to 172.23.0.1 and 255.255.0.0, respectively. So eth 0 of the virtual machine has to be configured with the IP address of the same class (Fraser *et al.*, 2004).

Emulation Components

In order to implement emulation in NS-2, different components should cooperate.

The first component is Real-time scheduler that synchronizes the simulation virtual clock with the system time and ties event execution within the simulator to real time. In this way it ensures that the packets, passing the simulator network, are delayed a proper amount of time.

The second component is Network object. Network objects provide access to a live network or to a trace file of captured network packets. In addition to the facilities providing by the host operating system, there are several forms of network objects, depending on the protocol layer specified for access to the underlying network. Network objects provide an entry point into the live network at a particular protocol layer (like raw IP, UDP, Pcap/bpf …) and with particular access mode like read-only, write-only or read-write.

Tap agent is the next component which is used to covert live packets in to simulated packets and vice versa. The tap agent handles the setting of the common header packet size field and the type filed. It uses the packet type PT-LIVE for packets injected in to the simulator. Each tap agent can have at most one associated network abject, although more than one tap agent may be instantiated on a single simulator node like what we have in our Emulation script in this study.

And the last component is Pcap/bpf network object. These objects provide an extended interface into the LBNL (Lawrence Berkeley National Laboratory) packet capture library (Libpcap). This library provides the ability to capture link-layer frames in the promiscuous fashion form network interface drivers. It also provides the ability to read and write packet trace files in the tcpdump format. The extended interface provided by NS-2 also allows for writing frames out to the network interface driver, provided that the driver itself allows this action. Use of the library to capture or create live traffic may be protected; one generally requires at least read access to the system's packet filter facility which may need to be arranged through a system administrator.

The packet capture library works on several UNIX-based platforms. It is optimized for use with the Berkeley Packet Filter (BPF) and provides a filter compiler for the BPF pseudo machine code. On most systems supporting it, a kernel-resident BPF implementation processes the filter code and applies the resulting pattern matching instructions to received frames. Those frames matching the patterns are received through the BPF machinery.

RESULTS AND DISCUSSION

Here, the creation and the connectivity of real and virtual machines for implementing congestion control algorithms via Xen and NS-2 are described via a simple example. The simulated scenario is a four-node network with the topology of Fig. 7, where the directions of arrows show the traffic passage direction. Two of the nodes are virtual nodes made inside NS-2 whereas the other two nodes are real ones. The real machine and the virtual machine

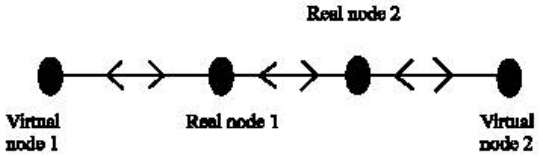

Fig. 7: Topology of the implemented network

```
set ns [new Simulator]
$ns use-scheduler RealTime
set myd [open out.tr w]
$ns trace-all $myd
set entry_node [$ns node]
set tcp_node [$ns node]
set $tcp_node "10.0.0.1"
$ns duplex-link $entry_node $tcp_node 10Mb 1ms DropTail
set tcp [new Agent/TCP/FullTcp]
$ns attach-agent $tcp_node $tcp
set bpf [new Network/Pcap/Live]
set dev [$bpf open readonly veth3]
set capture_tap [new Agent/TCPTap]
$capture_tap network $bpf
$ns attach-agent $entry_node $capture_tap
$ns simplex-connect $capture_tap $tcp
set rawsocket [new Network/IP]
$rawsocket open writeonly
set inject_tap [new Agent/TCPTap]
$inject_tap advertised-window 512
$inject_tap extipaddr "172.23.0.2"
$inject_tap extport 8000
$inject_tap network $rawsocket
$ns attach-agent $entry_node $inject_tap
$ns simplex-connect $tcp $inject_tap
$bpf filter "dst 172.23.0.1"
proc finish {} {
global ns myd nmyd
$ns flush-trace
close $myd
exit 0
}
$ns at 0.01 "$tcp advance 1"
$ns at 2.0 "exit 0"
$ns run
```

Fig. 8: The NS-2 script of the first real machine

act as the real nodes in this scenario. It should be noted that the second real machine has been actually implemented via Xen in the same computer as the first real one. The generated traffic is TCP and the congestion control method is the well-known Random Early Discard (RED) method (Christiansen *et al.*, 2000). The capacity and the delay of the links are considered as 10 Mb and 1ms, respectively.

The objective is to send a packet of data from the first virtual node via the real machines to the second virtual machine and then receive the acknowledgment by the first virtual node. The NS-2 scripts for implementing such connections in the different machines are shown in Fig. 8 and 9, respectively.

The script in Fig. 8 should be run on the first real machines and the script in Fig. 9, should be run on the virtual machine. The IP address of the first real node is 172.23.0.1 and the IP address of the second real node is 172.23.0.2. Based on these codes, a virtual node is created in the first real node when NS-2 runs within the first real node. This is the first virtual node shown in the topology. Then NS-2 begins to send a packet to the real network from this virtual node. The first real node receives the packet and then sends it to the second real node.

```
set ns [new Simulator]
$ns use-scheduler RealTime
set myd [open out.tr w]
$ns trace-all $myd
set entry_node [$ns node]
set tcp_node [$ns node]
$ns duplex-link $entry_node $tcp_node 10Mb 1ms DropTail
set tcp [new Agent/TCPSink/DelAck]
$ns attach-agent $tcp_node $tcp
set bpf [new Network/Pcap/Live]
set dev [$bpf open readonly eth1]
set capture_tap [new Agent/TCPTap]
$capture_tap network $bpf
$ns attach-agent $entry_node $capture_tap
$ns simplex-connect $capture_tap $tcp
set rawsocket [new Network/IP]
$rawsocket open writeonly
set inject_tap [new Agent/TCPTap]
$inject_tap advertised-window 512
$inject_tap extipaddr "172.23.0.1"
$inject_tap extport 8000
$inject_tap network $rawsocket
$ns attach-agent $entry_node $inject_tap
$ns simplex-connect $tcp $inject_tap
$bpf filter "dst 172.23.0.2"
proc finish {} {
global ns myd nmyd
$ns flush-trace
close $myd
exit 0
}
$ns at 2.0 "exit 0"
$ns run
```

Fig. 9: The NS-2 script of the second real machine

Table 1: Output trace file of the first machine

Event	Time	Input node	Output node	Pkt type	Pkt size	Flag	Src addr	Dst addr	Seq num	Pkt id
+	0.010001	1	0	tcp	40	0	1.0	0.1	0	0
-	0.010001	1	0	tcp	40	0	1.0	0.1	0	0
r	0.011033	1	0	tcp	40	0	1.0	0.1	0	0
+	0.122543	0	1	tcp	80	0	0.0	1.0	-1	1
-	0.122543	0	1	tcp	80	0	0.0	1.0	-1	1
r	0.123609	0	1	tcp	80	0	0.0	1.0	-1	1

Table 2: Output trace file of the second machine

Event	Time	Input node	Output node	Pkt type	Pkt size	Flag	Src addr	Dst addr	Seq num	Pkt id
+	0.44285	0	1	live	40	0	0.0	1.0	-1	0
-	0.44285	0	1	live	40	0	0.0	1.0	-1	0
r	0.443884	0	1	live	40	0	0.0	1.0	-1	0
+	0.549212	1	0	ack	40	0	1.0	0.1	0	1
-	0.549212	1	0	ack	40	0	1.0	0.1	0	1
r	0.550248	1	0	ack	40	0	1.0	0.1	0	1

The second real node sends the packet to the second virtual node which is made inside NS-2 running on the virtual machine with the same scenario as the first virtual node. The acknowledgment packet is received by the first virtual node after passing the reverse path.

Tables 1 and 2 contain the output trace files of running the programs which have been saved in tarce files 'out.tr' files. The first column shows the event type. It is given by one of four possible symbols r, +, - and d which correspond, respectively to received (at the output of the link), enqueued, dequeued and dropped. The second column gives the time at which the event occurs. The third and the fourth columns give the input and output nodes of the link at which the event occurs, respectively. The fifth column shows the packet type

(Pkt type) and the sixth one gives the size of the packet (Pkt size). Seventh column is a kind of flag. The eighth and ninth columns are the source and destination addresses (Src addr and Dst addr). The tenth one is the sequence number (Seq num). The last column shows the unique id (Pkt id) of the packet. From these trace files it is observed that the packet exchange between desired machines has been done, satisfactorily.

CONCLUSIONS

In this study, a method for synthesizing a virtual laboratory for evaluation of congestion control algorithms in TCP/IP networks (CCVL) has been proposed. The main significant advantage of this method is the ability of virtualization of a small real network even with one machine. Also, by this method it is possible to construct big real networks within only a few numbers of machines at universities or research institutes where fabrication of such networks can be expensive in terms of both cost and volume. In CCVL, the required hardware and the risk of hardware malfunctioning is noticeably reduced. The result is a low-cost compact portable virtual laboratory which also has the capability of connecting to the real internet and interchanging and gathering test data with/from it, if applicable. This method can be greatly useful in congestion control algorithms evaluation at personal laboratories, universities and research institutes.

It should however, noted that the number of virtual machines or equivalently, the size of the implemented network are confined by the available memory and processing limitations. However, with current advanced technologies these limitations are of minor concern.

ACKNOWLEDGMENT

The authors would like to express their appreciation to Linear Systems Control Laboratory of Imam Khomeini International University for providing facilities for this study and also the reviewers for their helpful comments.

REFERENCES

Alefiya, H., 2002. Emulation in ns. http://www.isi.edu/nsnam/ns/ns-tutorial/tutorial-02/slides/emulation.pdf.

Aweya, J., M. Ouellette, D.Y. Montuno and K. Felske, 2008. Design of rate-based controllers for active queue management in TCP/IP networks. Comput. Commun., 31: 3344-3359.

Barham, P., B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho and R. Neugebauer *et al.*, 2003. Xen and the art of virtualization. Proceedings of the 19th ACM Symposium on Operating Systems Principle, Oct. 19-22, Bolton Landing, USA., pp: 164-177.

Bigdeli, N., 2007. Synthesis and modeling of a test bed for evaluation and improvement of AQM methods in internet congestion control. Ph.D. Thesis, Sharif University of Technology, Tehran, Iran.

Bigdeli, N. and M. Haeri, 2007. ARM-PFC, An optimized AQM congestion controller in TCP/IP networks. Iran. J. Sci. Technol., 31: 663-678.

Bigdeli, N. and M. Haeri, 2009a. CDM-based design and performance evaluation of a robust AQM method for dynamic TCP/AQM networks. Comput. Commun., 32: 213-229.

Bigdeli, N. and M. Haeri, 2009b. Predictive functional control for active queue management in congested TCP/IP networks. ISA Trans., 48: 107-121.

Christiansen, M., K. Jeffy, D. Ott and F.D. Smith, 2000. Tuning RED for web traffic. Proceedings of the ACM/SIGCOMM, Sept., 2000, Stockholm, Sweden, pp: 139-159.

Chrysostomou, C., A. Pitsillides and Y.A. Sekercioglu, 2009. Fuzzy explicit marking: A unified congestion controller for best-effort and diff-serv networks. Computer Networks: The Int. J. Comput. Telecommun. Network., 53: 650-667.

Fall, K. and K. Varadhan, 2007. The ns manual. http://www.isi.edu/nsnam/ns/ns-documentation.html.

Feng, W., D. Kandlur, D. Saha and K. Shin, 1999. Blue: A new class of active queue management algorithms. Technical Report CSE-TR-387-99, University of Michigan.

Fengyuan, R., L. Chuang, Y. Xunhe, S. Xiuming and W. Fubao, 2002. A robust active queue management based on sliding mode variable structure control. Infocom, 1: 13-20.

Floyd S. and E. Kohler, 2002. Internet research needs better models. Hotnets-I. http://www.icir.org/models/bettermodels.html.

Fraser, K., S. Hand, R. Neugebauer, I. Pratt, A. Warfield and M. Williamson, 2004. Safe hardware access with the Xen virtual machine monitor. Proceedings of the 1st OASIS ASPLOS 2004 Workshop.

Hollot, C.V., V. Misra, D. Towsley and W.B. Gong, 2002. Analysis and design of controllers for AQM routers supporting TCP flows. IEEE Trans. Automat. Control, 47: 945-959.

Kelly, F.P., A. Maulloo and D. Tan, 1998. Rate control in communication networks: Shadow prices, proportional fairness and stability. J. Operat. Res. Soc., 49: 237-252.

Manfredi, S., M. Bernardo and F. Garofalo, 2009. Design, validation and experimental testing of a robust AQM control. Control Eng. Prac., 17: 394-407.

Pletka, R., M. Waldvogel and S. Mannal, 2003. PURPLE: Predictive active queue management utilizing congestion information. Proceedings of the 28th Annual IEEE International Conference on Local Computer Networks, Oct. 20-24, IEEE Computer Society, Washington DC., pp: 21-30.

Ryu, S., C. Rump and C. Qiao, 2003. A predictive and robust active queue management for Internet congestion control. Proceedings of the 8th IEEE International Symposium on Computers and Communication, June 30-July 3, IEEE Computer Society, Washington DC., pp: 991-998.

Sun, J., K.T. Ko, G. Chen, S. Chan and M. Zukerman, 2003. PD-RED: To improve the performance of red. IEEE Commun. Lett., 7: 406-408.

Wang, J., L. Rong and Y. Liu, 2008. Design of a stabilizing AQM controller for large-delay networks based on internal model control. Comput. Commun., 31: 1911-1918.