Implementation of a leaky bucket module for simulations in NS-3

P. Baltzis[2], C. Bouras[1,2], K. Stamos[1,2,3], G. Zaoudis[1,2]
[1]Computer Technology Institute and Press "Diophantus"
Patra, Greece

[2]Computer Engineering and Informatics Dept. University of Patras, Greece

[3]Technological Educational Institute of Patras, Greece

E-mail: mpaltzis@ceid.upatars.gr, {bouras, stamos, zaoudis}@cti.gr

Abstract: **Simulation has always been a valuable tool for experimentation and validation of models, architectures and mechanisms in the field of networking. In the case of the DiffServ framework, simulation is even more valuable, due to the fact that an analytical approach of mechanisms and services is infeasible because of the aggregation and multiplexing of flows. In this paper we present the implementation of a module for the extensible network simulator ns-3 in order to conduct experiments simulating a leaky bucket shaping mechanism for incoming flows, which is missing in ns-3. We describe the way the module has been implemented within the framework of the simulator and we discuss the merits of using a leaky bucket mechanism for shaping traffic. We also use the newly developed module in order to conduct experiments, verify its proper behavior and demonstrate its usefulness in studying the leaky bucket through simulation.**

1. INTRODUCTION

The value of simulation for enabling faster and easier experimentation and validation of models, architectures and mechanisms in the field of networking has long been recognized. A long list of networking protocols have been studied using simulation environments and then subsequently tested in real circumstances and production environments. Using simulation, a researcher can validate and fine-tune a proposal to scales that are not easily and cheaply achieved otherwise, and can overcome practical difficulties that would otherwise prove insurmountable in an initial stage of an idea's development.

Evaluation and testing in real networks would require big investments in software development and very often hardware development. In addition, a lot of equipment would be needed to conduct experiments on a large scale having multiple sources and destinations of different types. Small-scale evaluation in a lab, wide-area scenarios, and custom simulators can all be valuable; however each has some significant shortcomings. It is difficult to reproduce the actual mix of traffic and topologies found in real networks, they may require substantial effort and expense, and repetition of experiments under controlled conditions may be difficult. Therefore, real network testing is too costly for most research institutes at least in the first phases of development. Testing in a simulation environment is a more appropriate and inexpensive solution for evaluating networking mechanisms and protocols.

Perhaps the most widely used simulation environment in research has been the ns-2 simulator [1], which provides a robust simulator core with a very large number of either built-in or additional extensions by researchers. The value of ns-2 as a simulation environment has long been recognized in the field of network research.

However, ns-2 is not without its shortcomings [2] due to its long history and initial architecture assumptions, which have led to the initiative for developing an eventual replacement architecture, called ns-3 [3]. Although ns-3 is an improvement in many aspects over ns-2, they are essentially different (and incompatible) environments, which means that most add-on extensions that are available for ns-2 are not for ns-3. Ns-3 is a new simulator written from scratch and is not an evolution of ns-2 [7]. It is a modern approach of a simulator which aims to achieve much better performance than ns-2.

One of the architectures that is supported in ns-2 is DiffServ. The Differentiated Services (DiffServ) framework is one of the basic architectures that have been proposed for QoS provisioning in the Internet and is widely supported by network equipment vendors. It introduces the definition of different service classes to which aggregates are appointed and the implementation of mechanisms for differential treatment by network elements of the packets belonging to each service class. In compliance with the aggregation model, packets entitled to a certain service or belonging to a specific aggregate are marked with a distinctive value of the so-called Differentiated Services CodePoint (DSCP), a single packet header field, on which routers are based for providing differentiated services. The DiffServ architecture consists of mechanisms for traffic classification, marking, policing and shaping.

Traffic shaping is the control of computer network traffic in order to optimize or guarantee performance, improve latency, and/or increase usable bandwidth for some kinds of packets by delaying other kinds of packets depending on certain criteria. If a link becomes saturated to the point where there is a significant level of contention (either upstream or

downstream), latency can rise substantially, thus a shaping algorithm is needed. An important algorithm for shaping, which is missing in ns-3, is the Leaky Bucket, which is the focus of our work in this paper. In this paper the Network Simulator 3 (ns-3) was used as a basic tool of our simulation environment to simulate a leaky bucket shaping mechanism for incoming flows over wired and wireless networks.

The rest of the paper is organized as follows. Section II gives an overview of the leaky bucket mechanism. Section III describes the way that the new leaky bucket module has been implemented and integrated within the ns-3 simulator framework. Section IV then presents the simulation testbed that was used for verifying the behavior of the implemented module and demonstrating its usefulness, while section V discusses the obtained results. Finally section VI concludes the paper with a summary and ideas for future work.

2. LEAKY BUCKET

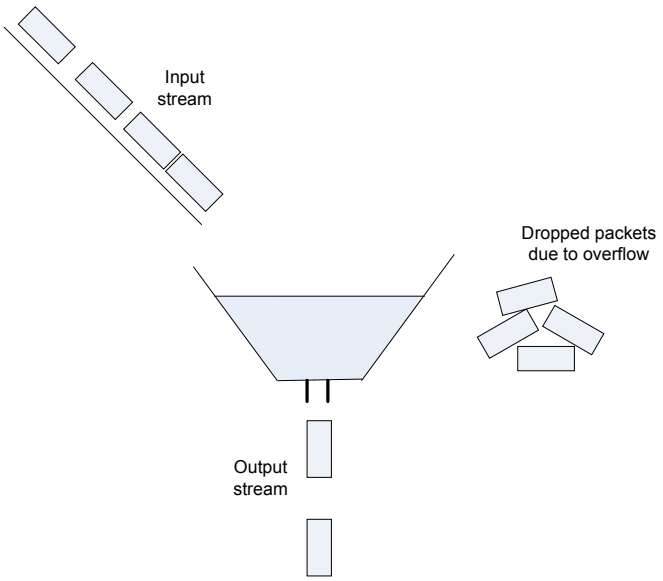

Figure 1 Leaky bucket operation

Leaky Bucket [5] [6] is a shaping algorithm, according to which packets arriving with random rate are shaped to a configurable constant rate. The Leaky Bucket anticipates for a buffering capacity with constant size. When packets arrive and the "bucket" is full, they are dropped.

Leaky bucket operates as a FIFO queue, as seen in *Figure 1*, where incoming packets leave with a constant bit rate. The queue size has a set value and when the queue is full, all subsequent packets are dropped until there is available space for the incoming packets to be stored. The leaky bucket algorithm is typically deployed at the entry points of the core network, thus protecting the network from packet bursts that may be produced from peripheral nodes. Therefore, the traffic levels at the network backbone are kept to acceptable levels, avoiding congestion and offering a type of Quality of Service.

```
while (incoming packet):
    Add tokens to bucket: rate * time_passed
    Check if we have any tokens
    if (no full token available):
        drop packet
    else:
        send packet
        available_tokens = available_tokens – 1
```

3. IMPLEMENTATION OF LEAKY BUCKET MODULE IN NS-3

In order to implement the leaky bucket in ns-3 [4], a series of modifications had to take place so that the simulator user can deploy the algorithm at the desired nodes. The leaky-bucket class that was implemented inherits the Queue class characteristics and implements all its virtual functions that are related to the storage and forwarding of packets from the queue. Also a leaky-bucket-helper class was used for defining the leaky bucket characteristics and deploying it at a specified node. All net-device subclasses also contain functions for handling leaky-buckets so that any device may be equipped with the introduced mechanism. Initially the mechanism implementation was tested on the point-to-point-net-device, which can easily be extended for any other net-device. Finally in order for the references to leaky buckets and the containing nodes to be stored the basic node class was also appropriately modified.

The user running the simulation can tune the algorithm by configuring it with a set of operation parameters, and install it on the desired device. The operation parameters contain the IP of the node which sends the traffic to be shaped, the outgoing data rate, the mode to count the queue size (packets or bytes) and the maximum packet or byte size of the queue. Finally a DSCP value may be used for introduction of the algorithm within an overall DiffServ architecture including time scheduling and packet removal from the queue. In the current implementation the leaky bucket algorithm is a drop tail queue which stores the incoming packets and sends them in a FIFO manner with a constant bit rate (configurable by the simulator user). There is also the possibility of installing more than one leaky buckets in a single or multiple nodes for more accurate simulation of actual networks. Our ns-3 implementation can be downloaded for experimentation at [4].

4. TESTBED

For our experimentation we used two different topologies.

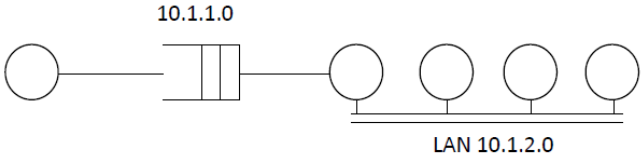

Figure 2 Simple topology

In the first topology presented in *Figure 2*, we simulate packet traffic originating from the left side node and destined for the right side node within the LAN. At the LAN gateway a leaky bucket implementation had been added. A simple way to install our leaky bucket module in a net device such as a LAN gateway is presented in the example code snippet below.

```
LeakyBucketHelper myBucket;
myBucket.SetAttribute ("DataRate", StringValue ("1Mbps"));
myBucket.SetAttribute ("Ipv4Shape", Ipv4AddressValue ("10.1.1.1"));
myBucket.SetAttribute ("MaxPackets", UintegerValue (5));
myBucket.Install(p2pDevices.Get(1));
```

Several experiments took place by modifying the leaky bucket size and keeping all other characteristics constant, or modifying the sending application's CBR (constant bit rate) and keeping all other characteristics constant. The constant characteristics are detailed below.

For the CBR application (implemented using ns-3 OnOffApplication):

- Constant bit rate at 10 Mbps
- Packets are generated with the above rate for 5sec and then for another 5sec the source remains idle
- The application is active from 0 to 15 second

For the leaky bucket:

- Outgoing packet rate at 1Mbps
- Leaky bucket size was different for each experiment

For the second set of experiments the constant characteristics are detailed below.

For the CBR application (implemented using ns-3 OnOffApplication):

- Constant bit rate was different for each experiment
- Packets are generated with the above rate for 1sec and then for another 1sec the source remains idle
- The application is active from 0 to 10 second

For the leaky bucket:

- Outgoing packet rate at 1Mbps
- Leaky bucket size at 1220 packets

The simulation duration in all cases was long enough for the leaky bucket to get empty and all outgoing packets to reach their destination.

The second and more complex topology models a more realistic network, as can be seen in *Figure 3* and a multitude of traffic streams were simulated:

1. From node i) to node v)
2. From node ii) to node vi)
3. From node iii) to node iv)
4. From node iv) to node i)

All traffic streams are shaped by different leaky buckets except for traffic stream 2 which is shaped by no leaky bucket. Using this topology, we again ran two sets of experiments. For the first set of experiments, their characteristics are following:

- Leaky bucket outgoing packet rate at 1Mbps
- Leaky bucket size at 6103 packets
- CBR application rate at 10 Mbps
- Packets are generated with the above rate for 5sec and then for another 5sec the source remains idle
- The application is active from 0 to 15 second

For the second set of experiments, leaky bucket parameters were differentiated as follows:

- Outgoing packet rate at 1Mbps for all leaky buckets
- Leaky bucket size at 800 packets for (a), 6103 packets for (b) and 3000 packets for (c)
- CBR application rate at 10 Mbps
- Packets are generated with the above rate for 5sec and then for another 5sec the source remains idle
- The application is active from 0 to 15 second

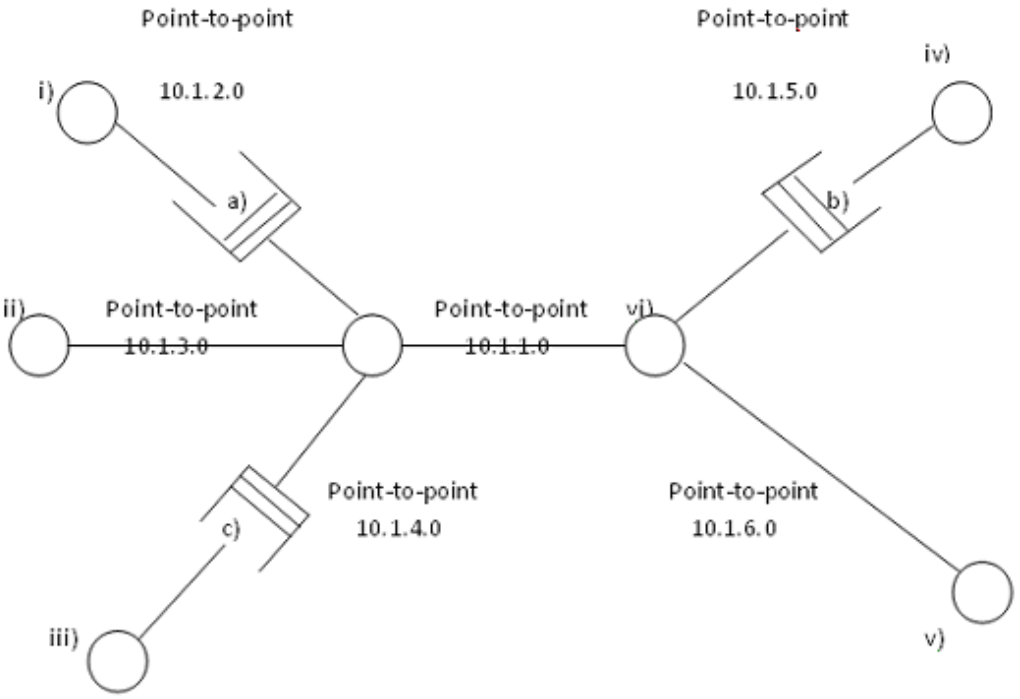

Figure 3 Complex topology

5. EXPERIMENTS AND RESULTS

In this section we summarize the results from the experiments described above that provide a verification of the proper operation of the leaky bucket algorithm in ns-3 and also enable us to study the algorithm's effect on packet flows. The metrics that will determine whether a leaky bucket algorithm is operating as expected can be summarized in two areas: First, a larger bucket should result in the ability to handle larger packet flows without overflowing and therefore without packet drops. Secondly, as a packet flow gets larger, it should meet a rate threshold at the point where the packet flow rate meets the bucket leaking rate.

Similar results were obtained in both types of topologies described above, and the ones from the simpler one are presented in *Figure 4*. It shows how the number of received packets is affected by the size of the leaky bucket buffer. As we can see, there is a threshold below which packets are lost, and this threshold corresponds to the packets gathered in the queue due to the smoothing of bursty traffic that is performed by the leaky bucket mechanism. If the size of the leaky bucket buffer is large enough to hold all pending packets, they are later transmitted and no packets are lost. However the traffic rate never surpasses the one set by the leaky bucket, and thus the "smoothness" of the potentially burst traffic is achieved.

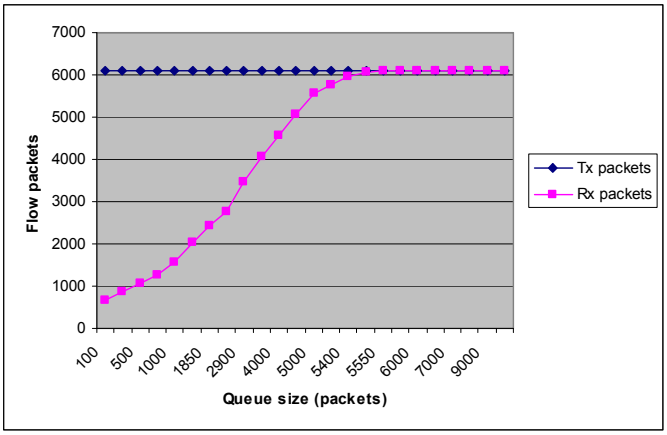

Figure 4 Flow behavior according to the size of the buffer

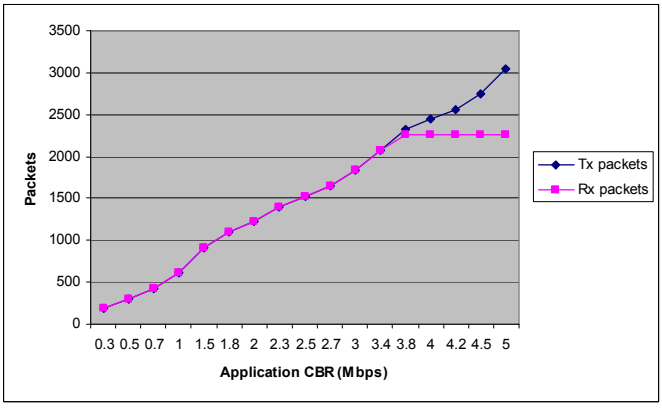

Figure 5 Results on the simple topology

Figure 5 above displays the results for the second set of experiments also on the simple topology. We can see that while the average traffic rate remains relatively low, traffic is smoothed but no packets are lost because the leaky bucket queue size is large enough to hold all pending packets. However, when the transmission rate exceeds a specific threshold, the leaky bucket buffer is no longer adequate, and some packets are lost. The number of lost packets gradually increases, as the average transmission rate also increases. In this case, traffic is "smoothed" but the overall number of transmitted packets is too large for all packets to be transmitted with the rate allowed by the leaky bucket. Therefore in this case, traffic is also policed to some degree.

6. CONCLUSION

In this paper we have presented the implementation of a leaky bucket module for the popular simulation environment ns-3, and we have shown how this algorithm may be used to shape traffic in the context of an overall DiffServ architecture. We have also provided an example of use for this extension and we have compared it with transmissions that are not shaped. The importance of our work lies on the fact that the scale of operation in DiffServ environments does not allow for analytical evaluation of models and mechanisms. Therefore, the use of simulation is necessary and particularly useful as a first step before implementation and testing in a real network topology.

Our future work in this area includes the enhancement of the simulation environment with mechanisms for scheduling such as the Modified Deficit Round Robin (MDRR) and other alternatives, the development of tools for metering and classification of packets, and the implementation of entities that handle DiffServ service management such as bandwidth brokers. We also intend to compare simulation accuracy between ns-2 and ns-3 environments and evaluate the possible improvements of the newer simulation architecture.

REFERENCES

[1] http://www.isi.edu/nsnam/ns/ (Accessed August 2011)

[2] Nino Kubinidze, Ivan Ganchev and Máirtin O'Droma: "Network Simulator NS2: Shortcomings, Potential Development and Enhancement Strategies", Springer US, 2006, p.p. 263-277.

[3] http://www.nsnam.org/ (Accessed August 2011)

[4] http://ru6.cti.gr/ru6/ns3.tar (Accessed August 2011)

[5] RFC 3290 - An Informal Management Model for Diffserv Routers

[6] Jiang Zhigang and Li Lemin: "Analysis of the leaky bucket algorithm for priority services", Springer, 1996, p.p. 333-338.

[7] Jianli Pan, Raj Jain: "A Survey of Network Simulation Tools: Current Status and Future Developments", 2008, http://www1.cse.wustl.edu/~jain/cse567-08/ftp/simtools.pdf (Accessed August 2011)