Bandwidth Reservation in Deterministic Networks (DETNET)

Vamsi Addanki
dept. Infres
Telecom-Paristech
Paris, France
f2013790[at]goa[dot]bits-pilani[dot]ac[dot]in

Luigi Iannone
dept. Infres
Telecom-Paristech
Paris, France
luigi[dot]lannone[at]telecom-paristech[dot]fr

***Abstract*—Packet delivery in the Internet works as a best effort service. It has no guarantees. The end-to-end latency, jitter and packet-loss probability have no fixed bounds. In contrast, deterministic networks which is of interest in special cases, aims at guaranteed bounded latency and zero packet loss for priority set of flows. The techniques adopted in such kind of networks such as bandwidth-reservation and packet-replication are well described in the RFC. In this paper, we discuss the selection of algorithms for each technique and show the comparison between deterministic networks vs best effort service by simulations using ns-3. We mainly focus on bandwidth reservation part of deterministic networks. Although bandwidth is reserved for priority flows, there is still some unreserved bandwidth for non-priority flows. We aim at zero congestion loss for priority flows by bandwidth reservation, and fair usage of unreserved and/or total unused bandwidth for non-priority flows. In contrary to the idea of increasing number of buffers to reduce the probability of packet loss, we show that the algorithm we adopt for Deterministic networks can achieve zero packet loss with the same existing number of buffers (only one shared buffer for all flows in our experiments).**

***Index Terms*—Best effort service, Deterministic networks, NS-3, Bandwidth reservation, Packet-replication.**

I. Introduction

Deterministic networks is part of time sensitive networking. In this type of networks, a bounded latency and zero congestion loss is guaranteed to a set of priority flows. This can be achieved by mechanisms such as explicit bandwidth reservation, packet replication [1]. The transmitter on the other end is also limited to a specific bandwidth. Although the type of mechanisms to be used in such networks are well described in the RFC and white-papers, there is no study in the literature about the possible algorithms that can be used in Deterministic Networks. [1], [2] suggest that increase in number of allocated buffers and explicit bandwidth reservation reduce the probability of packet-loss close to zero.

Limiting the transmitter to a specific bandwidth and explicitly allocating resources in the network can be viewed as having virtually a separate link for each priority flow with a bandwidth equal to the reserved bandwidth. This leads to under-usage of bandwidth when the priority flows are inactive. In this paper, we propose an algorithm inspired from [**?**]. The concept of virtual queues allows us to explicitly allocate resources to priority flows without increasing the number of real packet buffers. While the bandwidth is shared by priority and non-priority flows, we show that our algorithm optimizes the usage of unreserved and unused-reserved bandwidth by fairly sharing it among the non-priority flows. Our algorithm acts a dropper before en-queuing a packet in the queue.

Bandwidth sharing has always been a classical problem in networking. Numerous algorithms have been proposed for fair-sharing of bandwidth like FRED, CHOKe. In our specific case, we aim at virtual bandwidth reservation (meaning, guaranteed maximum bandwidth) for priority flows in combination with fair sharing of the remaining bandwidth among non-priority flows. This makes it very complex and difficult to implement and make use of any of the algorithms proposed in the literature. In view of Deterministic Networking, our algorithm, inspired from X, eliminates the problem of increased number of packet buffers in the network and also eliminates the unused bandwidth problem by allowing the non-priority flows to use the bandwidth. The algorithm is also extremely simple to be implemented in software routers which are gaining popularity in the recent times.

In this paper, we use NS-3, an industry standard network simulator to implement our algorithm. We introduce a new queue-disc which implements our algorithm under the traffic-control module of ns-3. We show by simulations, per-flow bandwidth usage, latency profiles, jitter and queue length. We make comparison between a network with our proposed queue-disc, a network with simple fifo queues and a network with explicit bandwidth reservation. We show that our simple algorithm can be implemented easily on hardware or software to achieve the notion of explicit bandwidth-reservation for priority flows in deterministic networks.

In the following section we describe the essential components required for flow level bandwidth reservation. In sec 3 we present the algorithm and in sec 4 we describe in short, the implementation in ns-3 and present the results from simulations.

II. Bandwidth reservation

In the context of bandwidth sharing, numerous algorithms from literature like RED, CHOKE are stateless. This makes it difficult to achieve exact per-flow fair sharing of bandwidth.

Identify applicable funding agency here. If none, delete this.

In a preliminary evaluation, we found that Fair-drop algorithm [**?**] is stateful and achieves exact fair sharing of available bandwidth among concurrent flows. The concept of virtual queues introduced in this algorithm makes it easier for us to adopt some of the concepts into our algorithm to achieve per-flow bandwidth reservation and per-flow bandwidth sharing at the same time. The essential components of our algorithm are one global flow-table and two active-lists, one for each priority/non-priority set of flows. Fig. [1] shows the structure of flow-table and active-list we use.

A. Flow Table

At the beginning, the flowtable is initialized with TABLESIZE rows and 1 column. Flows are classified based on rss hash value (HASH). RSS hash value is calculated by most of the hardware NICs in the recent times. So there is no overhead for the calculation of hash value.

Algorithm [1] explains the flow classification. On packet arrival, HASH%TABLESIZE is calculated, which gives the row index for the flow entry in the flow table. Then a linear search is applied in the row to find the flow entry for the current packet. If there is no entry for this flow yet, a new flow entry is created at the end of the row. When a new entry is created, the entry is initialized the following values: packet-size, type (priority/non-priority), bandwidth reservation, last-arrival, virtual queue and threshold (related to algorithm).

The scalability of the flow-table comes from the fact that deterministic network is a small-scale network typically a local-area-network. Given this scenario, a good Hash function typically yields a widely spaced range of Hash values for IP address (2 fields of the 5-tuple) in same or adjacent subnets. So the scale of flow-table collisions possible for this kind of network is expected to be low.

Algorithm 1 flow_classify(pkt)

```
1: Given: queue item of the packet = pkt; TABLESIZE; flow_table
2: hash = pkt.gethash;
3: mod = hash%TABLESIZE;
4: for i in 0 to n entries in flow_table[mod] do
5:    flow_entry=flow_table[mod][i];
6:    if hash == flow_entry.hash then
7:       return flow_entry;
8:    end if
9: end for
10: create a new flow table entry flow_table[mod][n+1];
11: flow_entry = flow_table[mod][n+1];
12: flow_entry.hash=hash
13: flow_entry.vqueue=pkt.getSize();
14: flow_entry.type=pkt.getType();
15: flow_entry.last_arrival=now();
16: flow_entry.bwreq=pkt.Getbwreq();
17: flow_entry.flow_threshold=THRESHOLD;
18: return flow_entry;
```

B. Active-list

Active-list is simply a list of all flows which have their virtual queue greater than zero. On packet-arrival and flow-classifications, it's virtual queue is checked. If it is zero, the flow is added to active-list corresponding to the type of the flow.

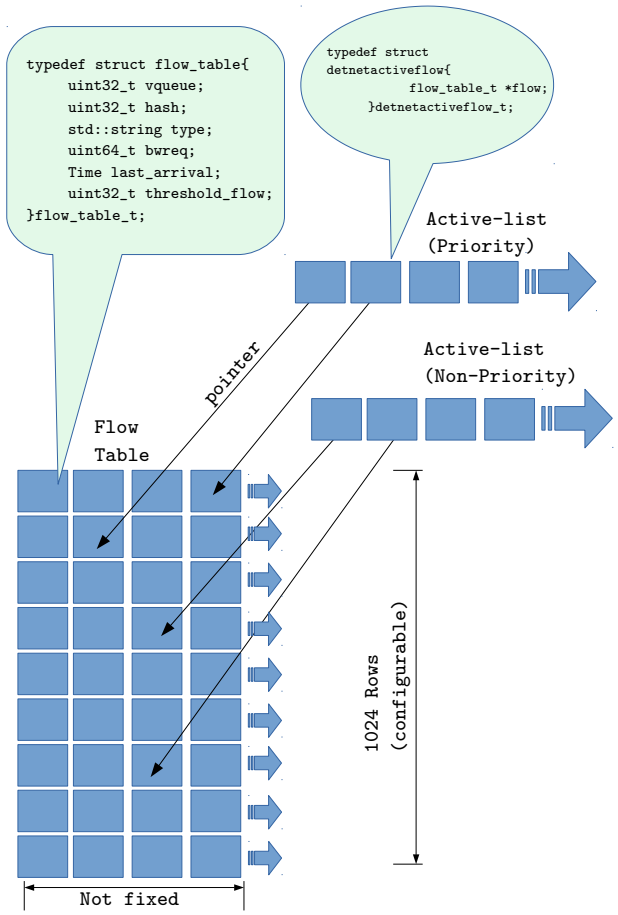

Fig. 1. Flow-table and active-list structures

III. Algorithm

Among the numerous concurrent flows, we have priority flows (PF) and non priority flows (N_PF). In the context of deterministic networks, we need to reserve bandwidth for priority flows or guarantee a maximum bandwidth utilization for the flow. We achieve this by defining a maximum bandwidth utilization variable (bwreq) for each priority flow as shown in Fig. [1]. As long as the incoming flow bandwidth is less than bwreq, we make sure that the flow experiences close to zero packet loss. The un-utilized bandwidth (independent of total reserved bandwidth) is fairly shared among all the concurrent non-priority flows. The algorithm consists of two stages.

A. stage-1

Stage-1 of the algorithm occurs at each packet arrival before en-queuing the packet in the output queue. The total available bandwidth (credit), is calculated by the difference between the arrival times of the current packet and the previous packet multiplied with the total output bandwidth. An additional credit type is also calculated for all the priority flows in the priority active-list. Priority-flow available bandwidth (priority-credit) calculated by the difference between the arrival times of the current packet and the last packet arrival of the corresponding priority flow multiplied with its reserved bandwidth.

$$credit = (t_n - t_{n-1}) * (BW_{output}) \quad (1)$$

$$priority\_credit_{pf} = (t_{npf} - t_{npf-1}) * (BW_{reserved}) \quad (2)$$

First, the virtual queue of all the entries in the priority activelist are decremented by the priority credit corresponding to the priority flow. If the virtual queue is decremented to zero, the flow is removed from the activelist. Otherwise the flow's activelist position is changed to the end of the list. During this step, while decrementing the virtual queues of all active priority flows, the priority credit which is unused is added to "remaining-credit" cumulatively.

Then, the remaining-credit is added to credit and this is fairly divided among the entries of the activelist of non-priority flows. Similar to the previous step, if the flow under consideration in the loop reaches to zero size virtual queue, it is removed from activelist. Otherwise the flow's activelist position is changed to the end of the list.

Algorithm 2 BW Reservation (Stage-1)

Given: priority and non-priority activelists, flowtable;
now=time_now();
$priority\_consumption$ = 0;
if size of priority activelist $>$ 0 **then**
  for i in 0 to size of activelist **do**
    $flow$ = get the first entry in activelist;
    Δt = now - $flow.last\_arrival$;
    $priority\_credit$ = Δt * $flow.bwreq$;
    if $flow.vqueue > priority\_credit$ **then**
      $flow.vqueue$ -= $priority\_credit$;
      $priority\_consumption$ += $priority\_credit$;
      move the entry to the end of the activelist;
    else
      $flow.vqueue$ = 0;
      $priority\_consumption$ += $flow.vqueue$;
      remove the entry from the activelist;
    end if
  end for
end if

B. stage-2

After stage-1, if the sum of current packet size and it's flow virtual queue is less than a THRESHOLD, the packet is enqueued in the output buffer and the packet size is added to it's virtual queue. Otherwise, the packet is dropped.

Algorithm 3 Remanining BW fair sharing (Stage-1)

Given: priority and non-priority activelists, flowtable, variables from Algorithm 2;
$credit$ = now - $LastPacketArrival$;
if $credit > priority\_consumption$ && size of non-priority activelist > 0 **then**
  $remaining\_credit$ = $credit$ - $priority\_consumption$
  while $SizeofNon - PriorityActivelist > 0$ and $remaining\_credit > 0$ **do**
    $fairbw$ = $remaining\_credit$ / $sizeofActivelist$
    $remaining\_credit$ = 0;
    for each $flow$ in Activelist **do**
      if $flow.vqueue > fairbw$ **then**
        $flow.vqueue$ -= $fairbw$
        move the entry to the end of the activelist;
      else
        $remaining\_credit$ += $fairbw$ - $flow.vqueue$;
        $flow.vqueue$ = 0;
        remove the entry from activelist;
      end if
    end for
  end while
end if

Algorithm 4 On packet arrival (Stage-2)

Given: packet's flow entry $flow$, $packet\_size$;
if $flow.vqueue > THRESHOLD$ **then**
  drop();
  exit();
else
  if $flow.vqueue$ == 0 **then**
    add the flow to the corresponding activelist by type.
  end if
  $flow.vqueue$ += $packet\_size$;
  $flow.last\_arrival$ = now();
end if

By this, we make sure that the priority flows are always served first based on their reserved bandwidth and only then the non-priority flows are served. As a matter of fact, if the transmitter obeys the limit of reserved bandwidth, we observe close to zero packet loss for all priority flows. Thus we achieve a combination of bandwidth-reservation and bandwidth-fairsharing in deterministic networks.

IV. NS-3 Simulations

A. Implementation

We introduced a new queue disc "BW_RESV" under the traffic-control layer of ns-3. The algorithm is implemented in the en-queuing operation of this queue disc. We use On-off application of ns-3 as packet-generator. To simplify the configurations done by network administrators in the real world, we send a payload in each packet. The payload is of

the format “XXXXKbps Y”. Here X are 4 numeric digits for the reserved bandwidth/bitrate, Y is the type of flow where D corresponds to priority flow and O corresponds to non-priority flow (for example “1000kbps D”). When a packet arrives at “BW_RESV” queue disc for the first time, the packet payload is read to identify the type of the flow and it’s bandwidth requirement.

B. Topology

We use a simple network with 4 hosts and 2 routers. Two hosts are connected to each router and the two routers are connected to each other, as shown in Fig. [**??**]. Host-1, Host-2 generate traffic and Host-3, Host-4 act as sink. All the links in the network are 30 Mbps except the link between the routers, which is 10 Mbps. The input link of Router-1 is configured high so that the output link becomes a bottleneck when there is a heavy traffic at the input. Router-1 is installed with either “FIFO” or “BW_RESV” queue disc in the simulations.

C. Simulations

We generate 12 flows out of which 10 flows are priority flows and 2 flows are non-priority flows. The priority flows are configured with bandwidth reservation of 100Kbps, 200Kbps and so on upto 1Mbps for priority flow 1-10. All the priority flows are generated at 1000Kbps bitrate and non-priority flows are generated at 8000Kbps. With this traffic, we have no congestion in any of the links, except the 10Mbps link, which is the output of Router-1. Non-priority flow-1 and flow-2 start at t=0 and t=2 respectively. Priority flows start at t=1, t=3, t=4, t=5 and so on, one flow at each second. We run simulations using this traffic configuration with our “BW_RESV” and “FIFO” queues to make a comparison and show that we can achieve bandwidth reservation and sharing at the same time using same resources.

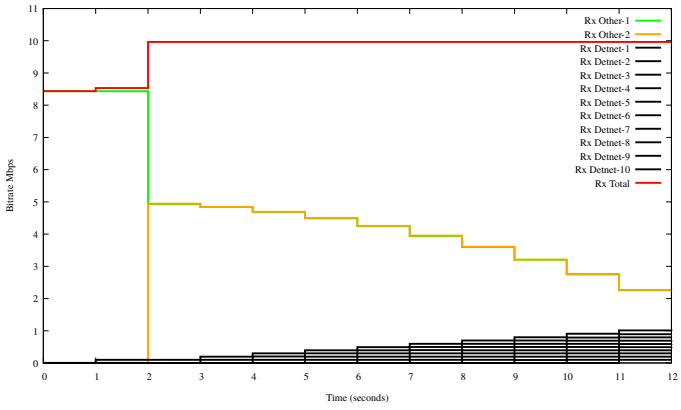

Fig. 2. Output rate per flow when bwresev queue is activated on the router with bottleneck link.

1) Throughput: Throughput of flows at the output of Router-1 (with bottleneck link) is measured. From Fig. [2], we can see that between t=0 and t=2, there is no congestion and no flow experiences drops. Starting from t=2, when the non-priority flow-2 starts, we see that priority flow-1 is given

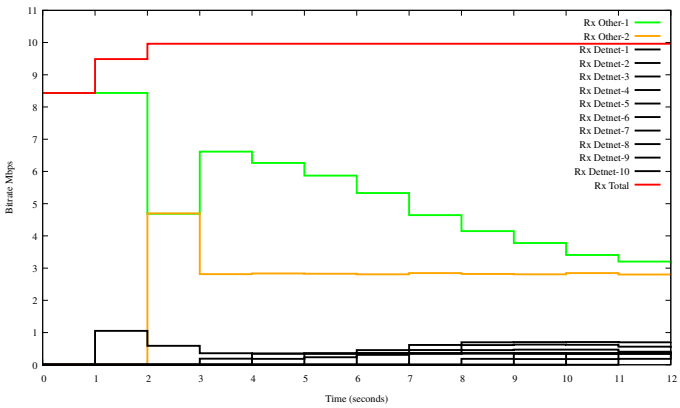

Fig. 3. Output rate per flow with fifo queues at the output of the router with bottleneck link.

exactly the configured bandwidth reservation i.e 100Kbps for flow-1 and the remaining bandwidth is shared by the two non-priority flows. The same pattern continues in the graph. When an additional priority flow is introduced, the remaining bandwidth reduces and it is fairly shared by the two non-priority flows. In this case, the difference between flow’s input and output throughput is entirely due to dropping of packets by the algorithm based on the reservation and sharing configurations.

Fig. [3] shows a simulation with Router-1 configured with “FIFO” queue at the output. This scenario can be compared to best-effort service. We see that there is specific pattern in how the output bandwidth is shared. Most importantly, the priority flows don’t always get the reserved bandwidth at the output due to the excessive usage of bandwidth by non-priority and other priority flows.

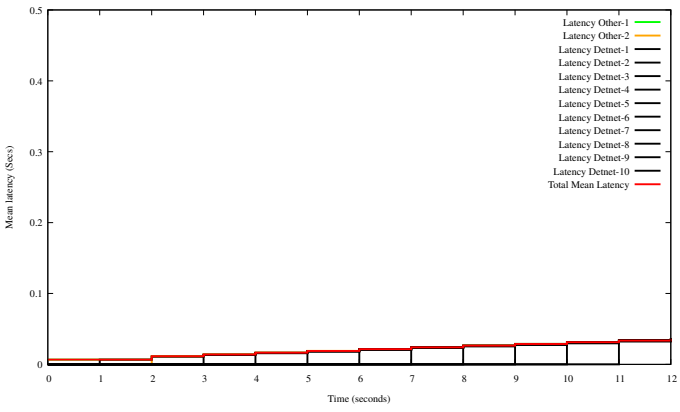

Fig. 4. Mean end-to-end latency when bw_resv queue is installed

2) Mean end-to-end Delay: The mean end-to-end Delay is measured by simulations using flow-monitor module of ns-3. Fig. [5] show a simulation with “FIFO” queues at Router’s output. We see that the mean-delay stays very low until the queue is saturated and reaches a maximum when the queue is

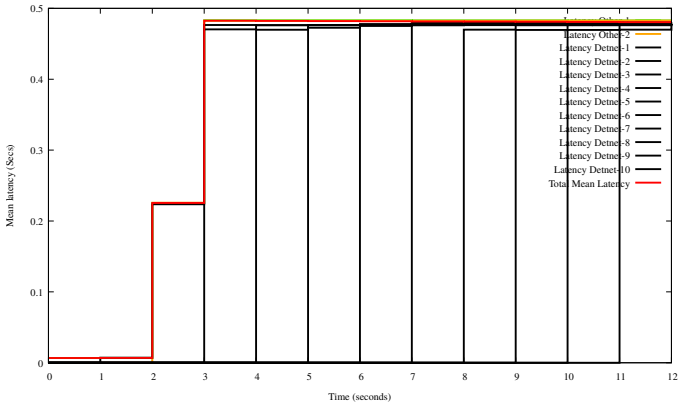

Fig. 5. Mean end-to-end latency with fifo queues

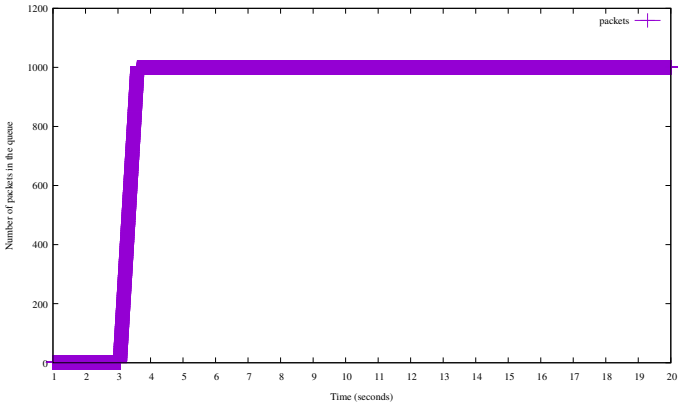

Fig. 7. Mean Queue length of fifo queue installed at the output of router-1

saturated.

Fig. [4] shows as simulation with "BW_RESV" queue at Router's output. We see that the mean-delay is relatively very low and almost constant. Although, there is a small constant incremental value when the number of flows increase. This constant incremental value can be the overhead of measurement or the per-flow overhead of algorithm or a combination of both. It becomes extremely complex to isolate the cause since measuring time also consumes time.

The most-important aspect of deterministic networks is bounded latency. We can clearly see from simulations that our algorithm can be configured properly to achieve a specific bounded latency.

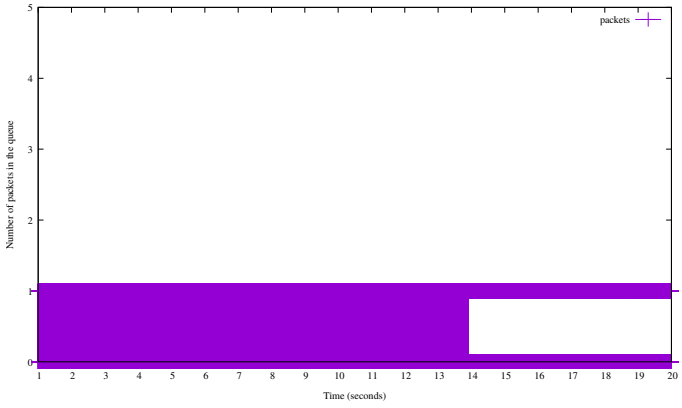

Fig. 6. Mean Queue length of bw_resv queue installed at the output of router-1

3) Queue length: We also measured the number of packets in the output queue of Router-1 at each packet arrival. We see that, in the case of "FIFO" queue from Fig. [7], the number of packets in the queue is close to zero until saturation, and starting from t=2, the queue length slowly reaches to full capacity and starts to drop packets. The increase in the queue length is one reason why the mean-delay of packets reaches a maximum due to the mean waiting-time in the queue.

We avoid this problem by making drop decisions at each packet arrival to avoid congestion in the output queue. This is done while ensuring the bandwidth reservation and sharing configurations. As a result, we see that the queue length stays close to zero.

4) Packet Loss: In this part, we run experiments with 12 flows consisting of 10 priority flows and 2 non-priority flows. All the priority flows transmit at 1Mbps with 1.1Mbps reservation and the non-priority flows transmit at 8Mbps. Each flow starts to transmit at times exactly same as in the previous experiments. (Non-priority-1 at t=0; Priority-1 at t=1; Non-priority-2 at t=2 and so on). When all the flows are in progress, they intersect at Router-1 with one output link of 12Mbps. As we can notice immediately, the output link is in an overload scenario. In this scenrio, we run experiments to check the overall packet loss experienced by the flows during the whole experiment. We can see from Fig. [8] that the priority flows have an exact zero packet loss as expected since they all have a reservation of 1.1Mbps and transmit at lower rate. We can also see that there is a heavy packet loss with "FIFO" queues.

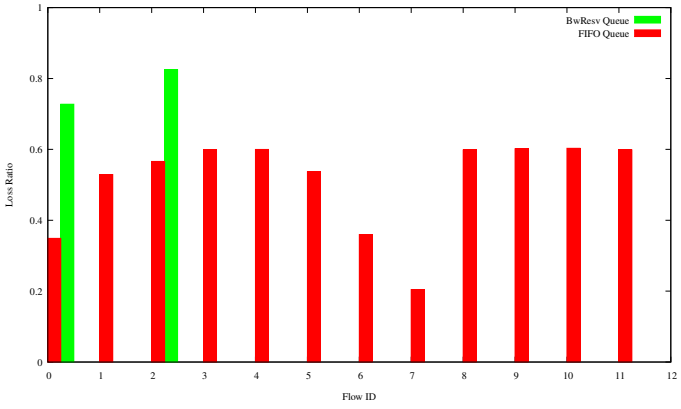

Fig. 8. Overall packet loss

V. Conclusions

Acknowledgment

References

[1] a
[2] b