Design and Implementation of the Traffic Control Module in ns-3

Pasquale Imputato and Stefano Avallone

Università degli Studi di Napoli Federico II
Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione
Via Claudio 21, 80125, Napoli

WNS3, June 15-16, 2016, Seattle, WA, USA

Table of Contents

Introduction: Linux Traffic Control 1/2

The Traffic Control (TC) is a component of the Linux network subsystem. TC supports operations needed to provide Quality of Service, including:

- shaping
- policing
- dropping

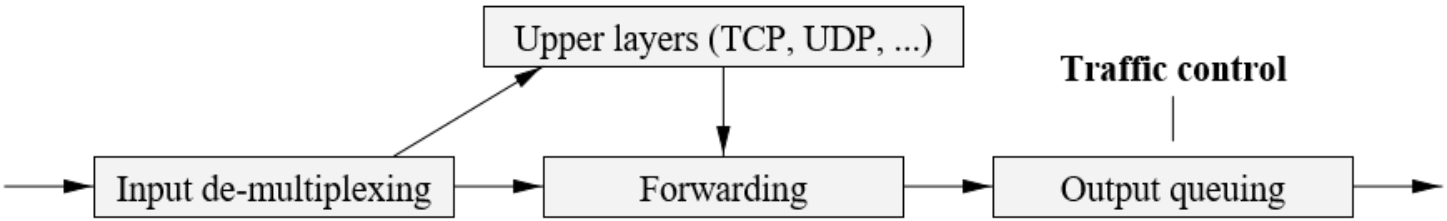

Figure: Processing of network packets in Linux.

Introduction: Linux Traffic Control 2/2

Three fundamental components in Linux TC:

- queue discs
- classes
- filters

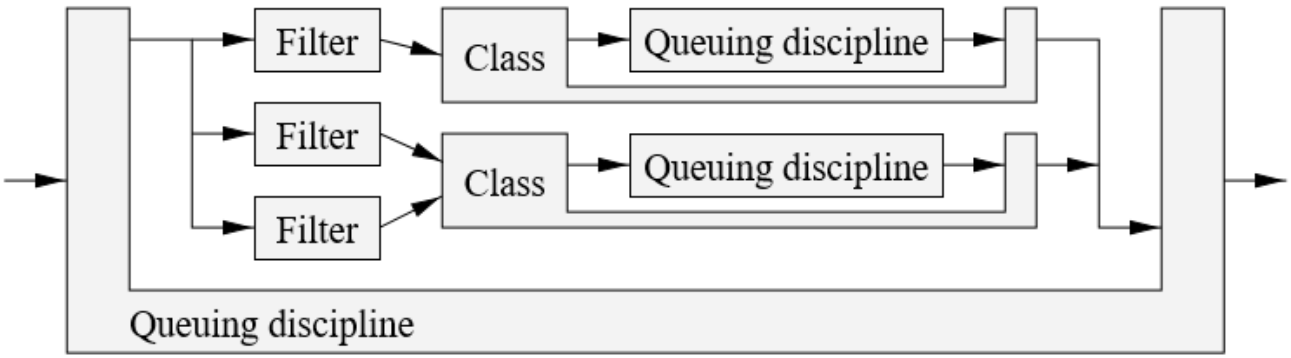

Figure: A queue disc with multiple classes in Linux.

Introduction: ns-3 has no Traffic Control

In ns-3, packets are not queued at the network layer, but only within netdevices:

- scheduling algorithms (e.g., RED and CoDel) implemented as subclasses of Queue ⇒ only available on those netdevices using a Queue subclass to store packets
- packets stored in netdevice queues already have the data link header added ⇒ operations like ECN are difficult

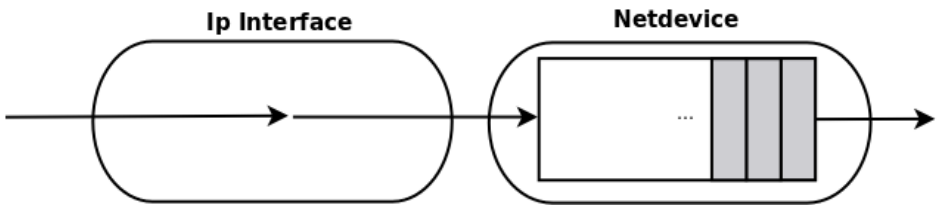

Figure: Queue in ns-3.

Introduction: Traffic Control Module in ns-3

The proposed ns-3 Traffic Control module intercepts packets on the transmission and receive path to enable packet scheduling, filtering and policing. It is modelled after the Linux Traffic Control.

- AQM algorithms are now derived from a new QueueDisc class and can be tested on any netdevice.
- The impact of the additional queueing within netdevice queues on the effectiveness of the scheduling algorithms can now be evaluated.

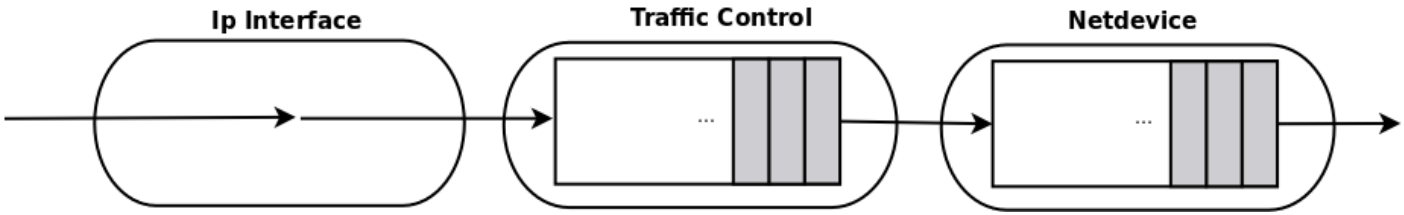

Figure: Queues in ns-3 after the introduction of Traffic Control.

Traffic Control Module: design 1/3

The traffic control layer keeps a reference to the root queue discs installed on the netdevices.
When it receives a packet from the IP interface, it requests the root queue disc:

- to enqueue the packet
- to dequeue a number of packets, until a predefined number of packets have been extracted or until the netdevice stops its transmission queue

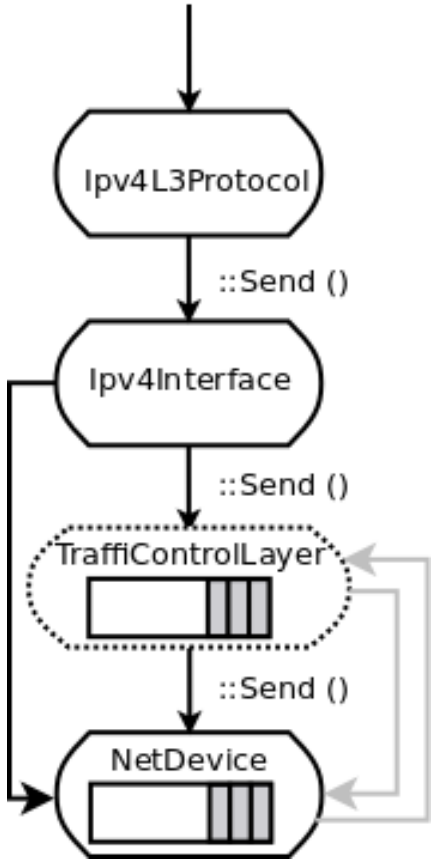

Figure: The transmission path on Internet enabled nodes with TC (IPv4 case).

Traffic Control Module: design 2/3

For each netdevice queue, it is necessary to keep a status bit which indicates if further packets can be passed for the transmission. The netdevice can:

- stop the passing of further packets when a resource becomes unavailable
- wake up the upper layer when the resource becomes available again

The netdevice wakes the root queue disc or a child queue disc depending on the operating mode.

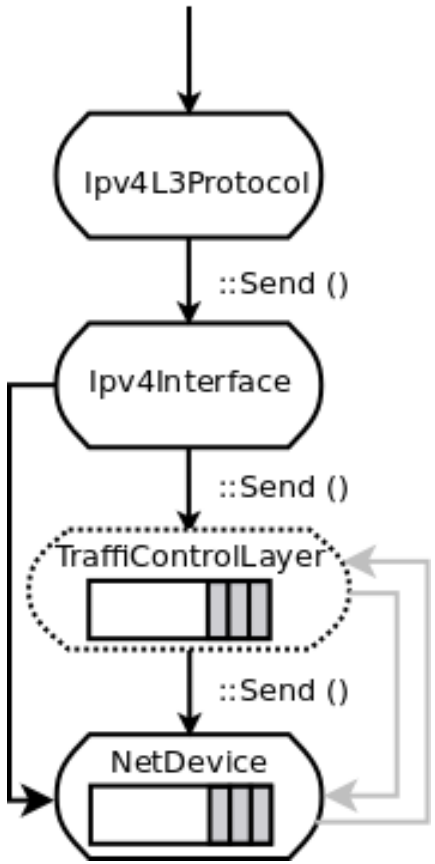

Figure: The transmission path on Internet enabled nodes with TC (IPv4 case).

Traffic Control Module: design 3/3

The Traffic Control module intercepts the packets incoming into the network node.
Currently, no additional operation is performed on incoming packets.

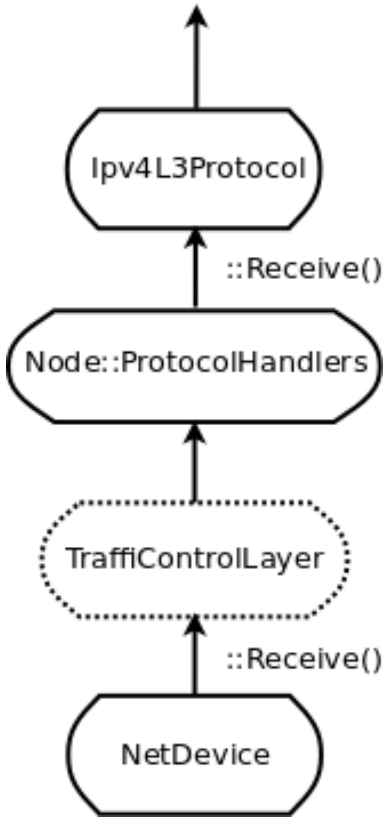

Figure: The receive path on Internet enabled nodes with TC (IPv4 case).

Traffic Control Module: implementation challenges

The introduction of the Traffic Control layer posed major implementation challenges:

- since packets arrive at the TC layer with the IP header already added, it is inefficient to manipulate (required by ECN) and access (required by Internet aware queue discs) L3 and L4 headers $\Rightarrow$ IP header and IP payload are now sent separately to TC
- the internet module depends on the traffic-control module $\Rightarrow$ to avoid a circular dependency, base classes (e.g., QueueDiscItem, PacketFilter) are defined in the traffic-control module and Internet specific classes are defined in the internet module

Traffic Control Module: helper

The Traffic Control Helper allows to build complex configurations and install them on netdevices.
A queue disc container allows to store the queue discs associated with the netdevices.

```
TrafficControlHelper tch;
uint16_t handle = tch.SetRootQueueDisc
    ("ns3::PfifoFastQueueDisc");
tch.AddInternalQueues (handle, 3,
    "ns3::DropTailQueue", "MaxPackets",
    UintegerValue (1000));
tch.AddPacketFilter (handle,
    "ns3::PfifoFastIpv4PacketFilter");
QueueDiscContainer qdiscs = tch.Install (devices);
```

Results: simulation settings

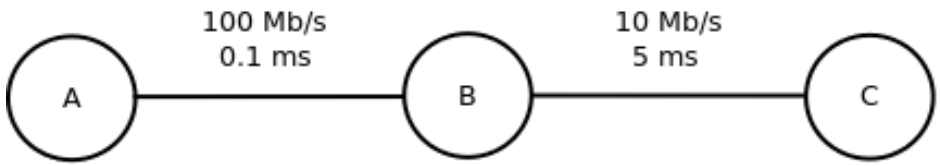

Figure: The network topology used for the validation tests.

Network settings for validation tests:

- OnOff generator of 100 Mbps TCP flow or 10 Mbps UDP flow on node A, sink on node C
- previous stack evaluated with DropTail, RED or CoDel as types of Queue
- new stack evaluated with PfifoFast, RED or CoDel as types of QueueDisc and DropTail as type of Queue

AQM strategies are employed on bottleneck netdevices.
Two scenarios have been considered.

Results: first scenario settings

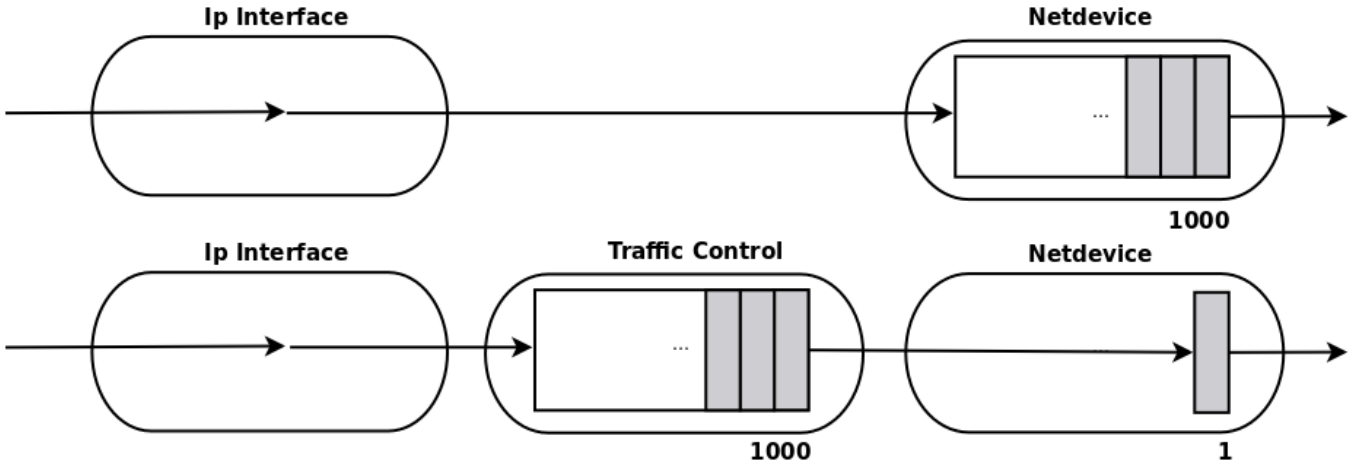

Figure: Old and new queue configuration in the first scenario.

The first scenario compares similar configuration of previous and new stack:

- the previous stack is evaluated with a netdevice queue having a size of 1000 packets
- the new stack is evaluated with a netdevice queue having a size of 1 packet and a queue disc having a capacity of 1000 packets

Results: first scenario TCP case

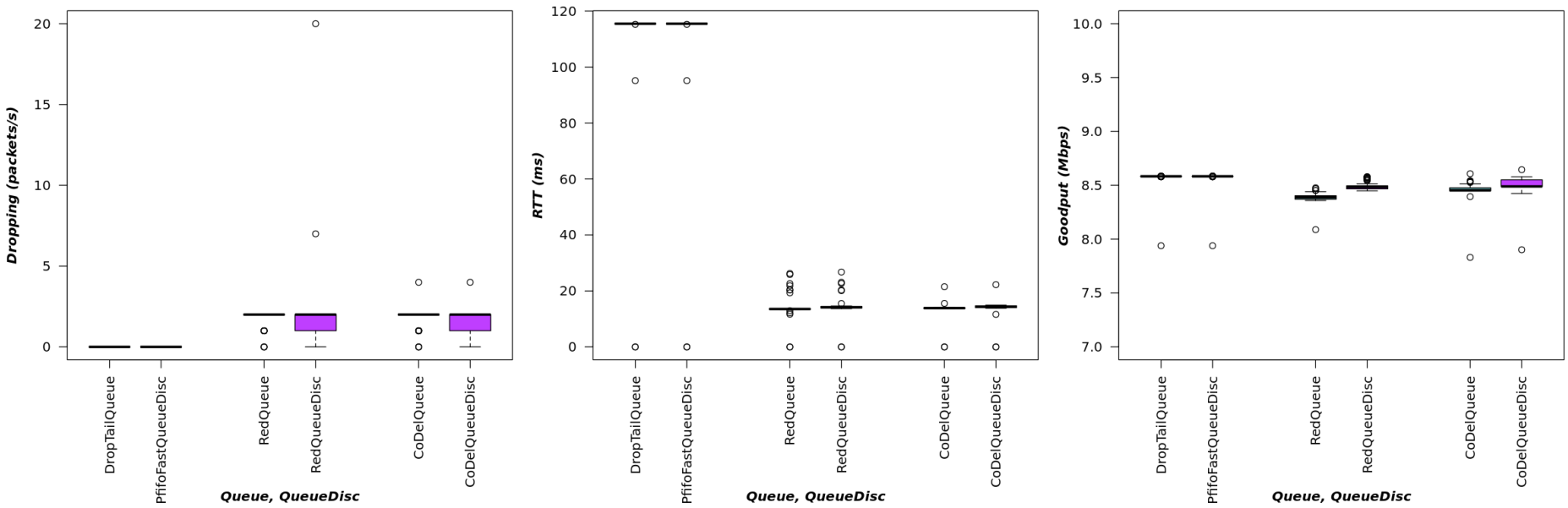

Figure: Dropping, RTT and Goodput for the first scenario in case TCP.

Minor and smoother dropping

Netdevice queue of one packet leads to a minor and smoother dropping activity in AQM.

Results: first scenario UDP case

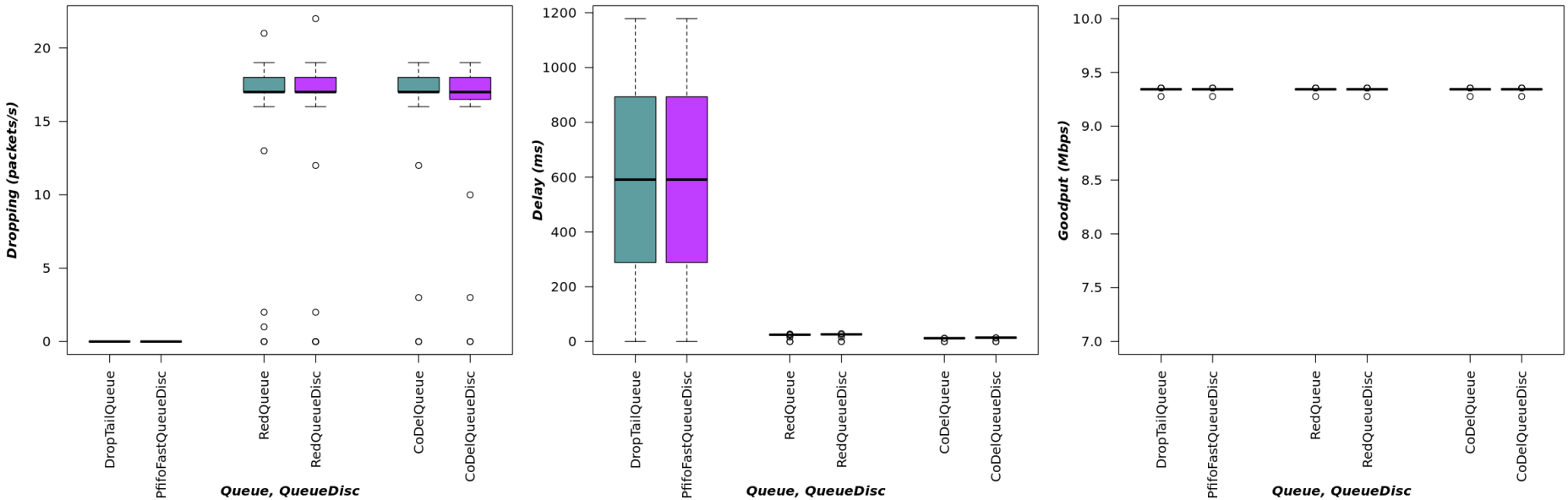

Figure: Dropping, Delay and Goodput for the first scenario in case UDP.

Similar behavior

The new stack in the first scenario behaves very similarly to the previous one.

Results: second scenario settings

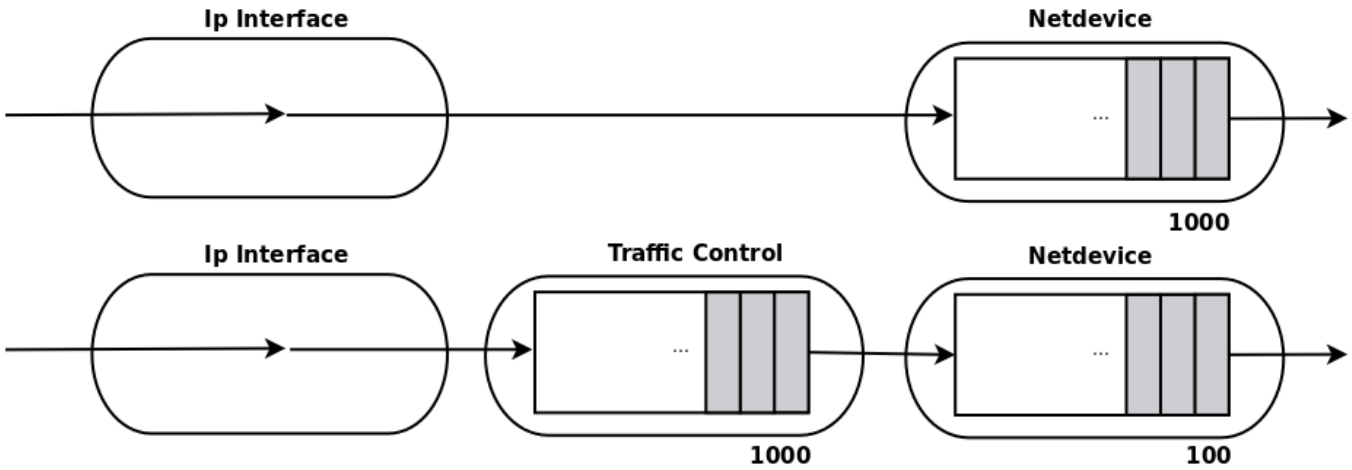

Figure: Old and new queue configuration in the first scenario.

The second scenario highlights the impact of netdevice queueing delay:

- the previous stack is evaluated with a netdevice queue having a size of 1000 packets
- the new stack is evaluated with a netdevice queue having a size of 100 packets and a queue disc having a capacity of 1000 packets

Results: second scenario TCP case

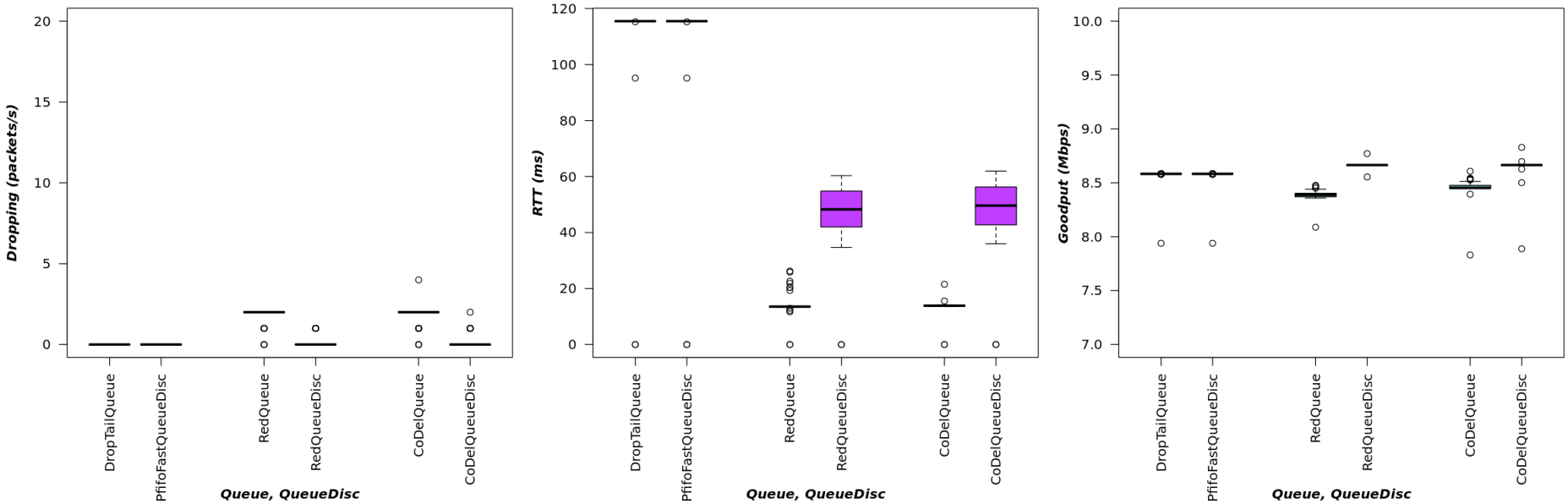

Figure: Dropping, RTT and Goodput for the second scenario in case TCP.

Reduction of AQM dropping

Netdevice queue of one hundred packets reduces AQM dropping activity.

Results: second scenario UDP case

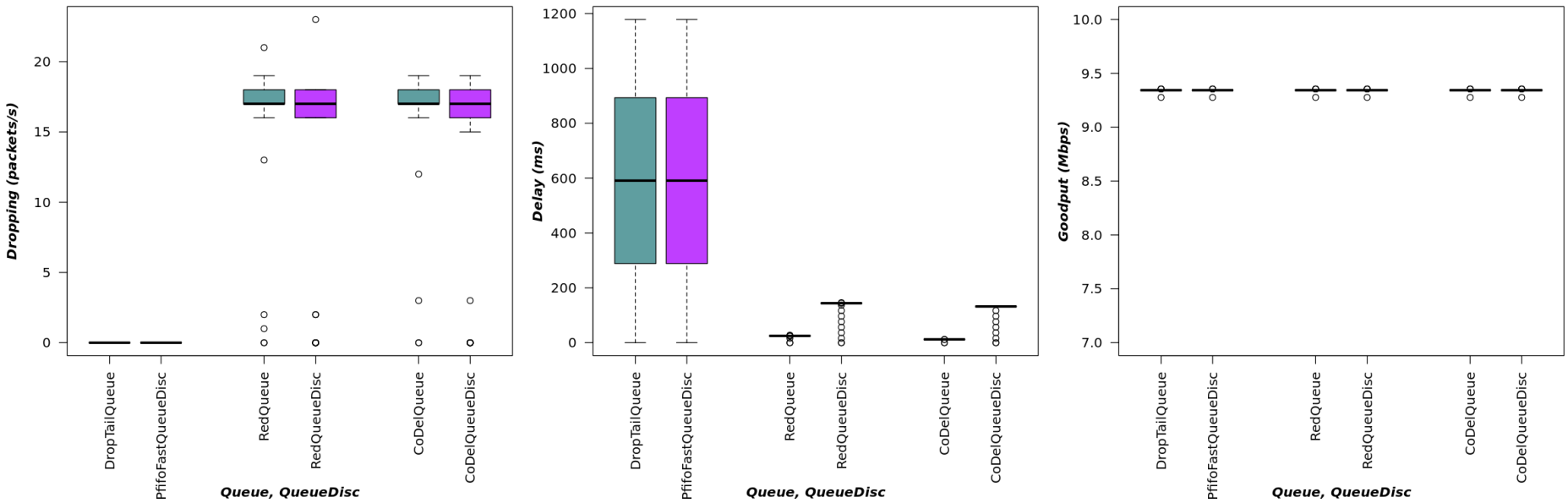

Figure: Dropping, Delay and Goodput for the second scenario in case UDP.

Disclosure of other behaviors

The behaviors highlighted by the second scenario cannot be observed with the previous stack. Such behaviors are encountered in real systems and require limiting netdevice queueing delay.

Conclusions and future work

Our module has been integrated into ns-3 starting from ns-3.25 release:

- more realistic simulations to evaluate AQM schemes more accurately
- netdevice flow control makes Traffic Control aware of the status and capabilities of the netdevice

Future work includes:

- the addition of FQ-CoDel queue disc, a new Internet aware queue disc
- the addition of Byte Queue Limits, to dynamically size netdevice queue
- validation through real systems