Traffic Differentiation and Multiqueue Networking in ns-3

Pasquale Imputato
Università degli studi di Napoli "Federico II"
Via Claudio 21, 80125, Napoli, Italy
pasquale.imputato@unina.it

Stefano Avallone
Università degli studi di Napoli "Federico II"
Via Claudio 21, 80125, Napoli, Italy
stefano.avallone@unina.it

ABSTRACT

The Linux networking subsystem provides fundamental abstraction to send and receive packets or to perform other operations. At socket layer, the user can set the socket priority used by the networking stack to prioritise the packets. The kernel sends a high priority packet before a low priority packet but the exact behaviour depends on the traffic control layer. A priority based queueing discipline uses that value of priority to enqueue the packets while a multiqueue aware queueing discipline uses a priority mapping defined by the device to enqueue the packets in its queues. The enqueue event triggers a number of consecutive dequeues based on the implemented device flow control mechanism. In case of a WiFi device, an additional layer, named the Soft MAC layer, sits in between the networking API and the hard device MAC. This layer defines the priority mapping and the device driver uses the API provided by that layer to notify the kernel about the status of their queues. In this paper, we present the introduction of the socket priority and of the multiqueue networking infrastructure in ns-3 and the design of the new flow control infrastructure. Finally, we report a preliminary evaluation of our work, consisting of a number of tests that highlight the new behaviour introduced by our models.

CCS CONCEPTS

• **Networks → Network simulations**; *Network performance modeling*;

KEYWORDS

ns-3, Traffic Control, Queueing Discipline, Active Queue Management, Traffic Differentiation, WiFi

ACM Reference format:
Pasquale Imputato and Stefano Avallone. 2017. Traffic Differentiation and Multiqueue Networking in ns-3. In *Proceedings of the 2017 Workshop on ns-3, Porto, Portugal, June 2017 (WNS3 2017),* 8 pages.
DOI: http://dx.doi.org/10.1145/3067665.3067677

1 INTRODUCTION

The socket layer of the Linux kernel enables to send and receive packets or to perform other useful operations. The user sets the socket options available to determine how the Linux kernel network stack processes the packets along the incoming and outgoing paths. An important option is the socket priority value. Each packet sent over that socket takes the socket priority value. Then the packets are passed to the Linux traffic control module [1]. The traffic control infrastructure enqueues packets in the queue of the queueing discipline (queue disc) installed on the device. A number of queue discs have been introduced in Linux, e.g. the simple pfifo_fast queue disc, the flow queueing queue disc fq_codel [3], the multiqueue aware queue disc mq.

A queue disc can use the priority of the packet to enqueue the packet in its queues. The recently introduced multiqueue aware queue disc enqueues packets in their queues according to the device queue mapping. Traffic control redefines the packet priority according to the device rules. Indeed, a multiqueue device driver defines a driver hook which returns the index of the queue the packet is destined to. Hence, queue discs enqueue packets in their internal queues according either to the socket priority, e.g. pfifo_fast queue disc, or to the device index of queue, e.g. mq queue disc. The enqueue event triggers a number of dequeue events. Traffic control dequeues a number of consecutive packets to pass to the device from the queue disc where it enqueued the packet. The process can be stopped by the device driver, i.e., the implemented flow control mechanism, or by the networking stack, i.e., the Byte Queue Limits (BQL) algorithm (if supported by the device driver). For a WiFi device, Linux defines the mac80211 subsystem that performs some of the operations of the MAC layer, e.g. the rate control mechanism, as defined by the IEEE 802.11 standard. The layer sits in between the kernel multiqueue networking API and the hard MAC layer. It defines the Soft MAC layer. This layer defines the Access Category (AC) mapping for a quality of service WiFi device which uses the multiqueue networking select queue hook. For each packet, the implemented select queue determines the AC starting from the differentiated services (DS) field of the IP header. Also, a WiFi device driver implements its flow control by using the ieee80211_{start | stop | wake}_queue functions of this layer.

The goal of this work is to address a number of shortcomings in the ns-3 implementation:

- lack of the sockets priority option. IPv4 implements a protocol specific solution to define the Type of Service (ToS) byte of the IP packet header but its use is hidden to the user. Indeed, usually the user cannot have a socket pointer when the socket is created by means of the socket helper;
- lack of multiqueue aware queue discs. The multiqueue information, potentially defined by multiqueue devices, is unused;
- the User Priority (UP) in WiFi is kept by a tag that the user had to attach somehow to the generated packet;
- each device needs to define its flow control mechanism by calling the traffic control API.

WNS3 2017, June 2017, Porto, Portugal

DOI: http://dx.doi.org/10.1145/3067665.3067677

In brief, only protocol and device-specific solutions were provided to support traffic differentiation and multiqueue networking and no multiqueue aware queue disc was provided.

This paper presents the work done to introduce an equivalent of the Linux socket priority and multiqueue networking and a new flow control infrastructure in ns-3. We introduced the socket priority option in ns-3 and the priority to ToS and DSCP (Differentiated Services Code Point) mapping. Then, we introduced a multiqueue queue disc, i.e. mq queue disc, in ns-3. Finally, we designed a new mechanism to add support for flow control to every device that uses `Queue` objects to implement their queues. We believe that our work will allow researchers to carry out more realistic simulations with traffic differentiation and multiqueue devices in ns-3.

The remainder of this paper is organised as follows. In Section 2 we provide an overview of traffic differentiation and multiqueue networking in Linux and of the status of ns-3. Section 3 describes the introduction of socket priority and multiqueue networking and the new flow control infrastructure in ns-3. In Section 4 we present some usage examples. In Section 5 we present the experiments we performed with the new models and the results we obtained. In Section 6 we conclude our work.

2 BACKGROUND

In this section, we provide an overview of traffic differentiation and multiqueue networking in Linux and of the status of ns-3.

2.1 Linux Traffic Differentiation and Multiqueue Networking

In Linux, the networking socket layer is a uniform interface between the user process and the network protocol stack in the kernel. A user process uses the socket functions to send or receive packets or to perform other operations. Users set socket options with the `setsockopt` or get socket options with the `getsockopt` functions. The SO_PRIORITY option sets the socket priority for all packets to be sent on that socket. Linux uses this value to select the networking queue. The kernel representation of the packet, i.e., the socket buffer (SKB), keeps the priority value for the packets in the priority field. Valid user priority is in the range 0 to 6. On a socket, users can specify protocol defined option. The IP Linux implementation supports the IP_TOS option. This option sets the ToS field that is sent with every IP packet originating from the socket. When the user sets the socket ToS, Linux determines the socket priority by means of a mapping from ToS to priority. Packets with higher priority may be processed first depending on the selected device queue disc.

Packets sent over a socket are enqueued in the traffic control queues. The traffic control layer is the networking subsystem which manages the enqueues and the dequeues from the queue disc installed on device. Each network device has a queue disc installed on it whence traffic control enqueues and dequeues packets. Before a packet is enqueued, the index of the queue used by device is determined. A multiqueue device can define its rules to provide the queue mapping for a packet. The kernel uses a device driver hook to define the queue mapping if defined, or a fallback mechanism by a hash calculation otherwise. That information is kept in the queue mapping field in the SKB. The packet priority is redefined according to device queue mapping. In case of a WiFi device, the mac80211 sublayer selects the correct AC based on the DS field of the IP packet. Then traffic control enqueues the SKB to the queue disc installed on the device. The queue disc configured on the device can prioritise the packets based on its rule. A classless queue disc such as pfifo_fast selects its internal band based on the priority value in the SKB. The recent introduction of the multiqueue aware classful queue disc, e.g. mq queue disc, enqueues the packet in the class grafted to the correspondent device queue based on the queue mapping defined in the SKB.

2.2 ns-3 Status

The Socket class in ns-3 has not a priority field. Users cannot set a priority value for the packet generated from that socket. ns-3 provides only protocol or device specific support to traffic differentiation. The IP protocol implementation in ns-3 provides support to set the ToS field of the IP header. The user can set the ToS value by `SetIpTos` of the socket only when it has a pointer to the socket, which generally is created by the application helpers. When the user sets the ToS, a packet tag is attached to the packet to keep the ToS value. The ns-3 WiFi device implements a support to traffic differentiation. WiFi defines a `QualityOfServiceTag` packet tag to provide support to traffic differentiation in its four AC. The user can set the user priority of a packet, kept in the quality of service tag and attach it to the packet generated destined to a WiFi device. Based on this tag, the device defines the mapping to the correspondent AC. In ns-3 the traffic control module is modelled after the Linux traffic control subsystem. It provides support to traffic differentiation to define the internal queue of the queue disc where packets are enqueued. The `QueueItem` enqueued within the queue disc keeps a queue mapping value defined according to the device. A multiqueue device provides a hook to define the index of the queue for the item. The default pfifo_fast queue disc uses the ToS field, potentially defined in IP, to enqueue the packets in their internal band. Pfifo_fast requires a packets filter to choose one of its internal band. Traffic control lack of multiqueue aware queue disc. The multiqueue information, potentially defined by multiqueue device, e.g., a QoS WiFi device, is unused. Finally, note that the pair of queues (the traffic control queues and device queues) must be a conservative system. Each device has to implement its flow control mechanism to make traffic control aware of device queue status. The current approach to adding the flow control support in each device leads to redundant code and is an error-prone activity.

3 MODELS

In this section, we present the new models introduced in ns-3.

3.1 Socket Priority

The basic idea is to introduce the equivalent of Linux sockets SO_PRIORITY option in ns-3. In order to provide sockets priority option, the ns-3 sockets API is extended with the addition of the methods `SetPriority` and `GetPriority`. Valid values of sockets priority are in the range from 0 to 6. For each packet, a `SocketPriorityTag` keeps the priority value. The socket implementation adds a `SocketPriorityTag` to each packet sent through

Table 1: Mapping of the four least significant bits of the priority to pfifo_fast band.

Priority & 0xf	Band
0	1
1	2
2	2
3	2
4	1
5	2
6	0
7	0
8	1
9	1
10	1
11	1
12	1
13	1
14	1
15	1

Table 2: Mapping of ToS bits 3-6 to Linux priority.

Bits 3-6	Means	Linux priority
0	Normal Service	Best Effort (0)
1	Minimize Monetary Cost	Best Effort (0)
2	Maximize Reliability	Best Effort (0)
3	mmc+mr	Best Effort (0)
4	Maximize Throughput	Bulk (2)
5	mmc+mt	Bulk (2)
6	mr+mt	Bulk (2)
7	mmc+mr+mt	Bulk (2)
8	Minimize Delay	Interactive (6)
9	mmc+md	Interactive (6)
10	mr+md	Interactive (6)
11	mmc+mr+md	Interactive (6)
12	mt+md	Int. Bulk (4)
13	mmc+mt+md	Int. Bulk (4)
14	mr+mt+md	Int. Bulk (4)
15	mmc+mr+mt+md	Int. Bulk (4)

that socket in the outgoing path. Conversely, it removes the `SocketPriorityTag` along the incoming path. The priority value for a socket is used to determine the priority of the packets sent through that socket. Currently, the socket types that support setting the packet priority are `UdpSocketImpl`, `TcpSocketBase` and `PacketSocket`. Every packet sent through these sockets is assigned a priority equal to the priority associated with the socket. In case of UDP sockets, if the packet is an IPv4 packet with non null ToS field, the packet is assigned a priority based on such ToS value according to the `IpTos2Priority` function. This function is implemented after the Linux rt_tos2priority function, which takes an 8-bit value as input and returns a value which is a function of bits 3-6 (where bit 0 is the most significant bit) of the input value according to Table 2. The rationale is that bits 3-6 of the ToS field were interpreted as the TOS subfield by RFC1349 [2]. The packet priority is used, e.g., by queue discs such as the default priority queue pfifo_fast to classify packets into distinct band. Currently, pfifo_fast queue disc does not require an external filter. Packets are enqueued in three priority bands (implemented as FIFO droptail queues) based on their priority. The four least significant bits of the priority are used to determine the selected band according to Table 1.

In order to simplify the setting of the ToS value for the IP protocol, the `InetSocketAddress` is extended with an additional field for the ToS desired for the flow destined to that address. It provides an equivalent of IP_TOS option of the Linux sockets. In this way, the user can set the ToS when it does not have a pointer to that socket. The applications eventually call the socket connect method to connect to the provided destination address. The connect method of the particular socket type sets the ToS associated with a socket by using the socket `SetIpTos` method. Currently, the socket types that support setting the ToS are `UdpSocketImpl` and `TcpSocketBase`. The ToS associated with a socket is then used to determine the value of the ToS field (renamed as Differentiated Services field by RFC2474 [5]) of the IPv4 header. For an IPv4 packet sent through a connected socket, the ToS field is set to the ToS value associated with the socket. In case of not connected UDP sockets, the ToS field is set to the value specified in the destination address passed to the `SendTo` method and the ToS associated with the socket is ignored. The DS field is used by multiqueue devices to define their queue mapping. Currently, WiFi models a DS to AC mapping. Finally, setting the ToS associated with a socket also sets the priority for the socket to the value that the `IpTos2Priority` function returns when it is passed the ToS value. The ToS to Linux priority mapping is reported in Table 2.

3.2 Multiqueue Networking

We introduce multiqueue networking support in ns-3 modelled after Linux networking subsystem. The traffic control module in ns-3 [4] introduced the `NetDeviceQueueInterface` class which keeps queue related information, such as the queue status, into a per queue class `NetDeviceQueue`, which is modelled after the Linux `netdev_queue` struct which represents the device queue. A device in ns-3 has as many `NetDeviceQueue` as the number of transmission queues.

We introduced in ns-3 the queue selection mechanism modelled after the Linux select queue. A multiqueue device in ns-3 provides a `SelectQueue` method which, when invoked with a packet, returns the index of the device queue the packet is destined to. This method is modelled after the Linux `select_queue` callback which a multiqueue device driver registers. A multiqueue driver in Linux can provide a method to determine which queue to use for a SKB. The device chooses the index based on its rules, e.g. based on the DS field of the IP header. Traffic control in ns-3 stores the `SelectQueue` method, if provided, for each device. When a packet is passed to the traffic control along the outgoing path, traffic control sets the index of the queue according to the device `SelectQueue` method. Then, it sets the value into the queue mapping field of the queue disc item. We prototype the select queue for a QoS WiFi device modelled after the mac80211 sublayer of Linux. The selection of the AC for an MAC service data unit (MSDU) is based on the value of the DS field

Table 3: Mapping user priorities values to access categories.

UP	Access Category
7	AC_VO
6	AC_VO
5	AC_VI
4	AC_VI
3	AC_BE
0	AC_BE
2	AC_BK
1	AC_BK

Table 4: Mapping DSCP and TOS values onto user priorities and access categories.

DiffServ PHB	TOS (binary)	UP	Access Category
EF	101110xx	5	AC_VI
AF11	001010xx	1	AC_BK
AF21	010010xx	2	AC_BK
AF31	011010xx	3	AC_BE
AF41	100010xx	4	AC_VI
AF12	001100xx	1	AC_BK
AF22	010100xx	2	AC_BK
AF32	011100xx	3	AC_BE
AF42	100100xx	4	AC_VI
AF13	001110xx	1	AC_BK
AF23	010110xx	2	AC_BK
AF33	011110xx	3	AC_BE
AF43	100110xx	4	AC_VI
CS0	000000xx	0	AC_BE
CS1	001000xx	1	AC_BK
CS2	010000xx	2	AC_BK
CS3	011000xx	3	AC_BE
CS4	100000xx	4	AC_VI
CS5	101000xx	5	AC_VI
CS6	110000xx	6	AC_VO
CS7	111000xx	7	AC_VO

in the IP header of the packet (ToS field in case of IPv4, Traffic Class field in case of IPv6). The select queue method sets the user priority of an MSDU to the three most significant bits of the DS field. The AC is then determined based on the UP according to Table 3. DSCP and TOS map onto user priorities and access categories according to Table 4. `SelectQueue` also sets the packet priority to the user priority, thus overwriting the value determined by the socket priority. Also, given that the traffic control calls `SelectQueue` before enqueueing the packet into a queue disc, it turns out that queue disc that classify packets based on their priority will use the user priority instead of the socket priority.

The traffic control interacts with the device queue disc. In case of a multiqueue device, traffic control can interact with each of the device queues by checking each queue status separately. Also, a multiqueue device can separately wake each of the queue discs grafted to its queues.

To best fit the multiqueue devices information in traffic control (i) the per queue status check, ii) wake queue disc grafted to device queue, iii) the index of the queue choosen by device for each packet), we introduced the multiqueue queue disc in ns-3. It is modelled after the mq Linux queue disc that is a classful multiqueue dummy scheduler. It presents device transmission queues as classes, allowing to attach different queue discs to them. Each class of mq is grafted to the corresponding device transmission queue. Traffic control enqueues a packet in the queue disc grafted to the device queue the packet is destined to. Then, traffic control makes a run on this queue disc to dequeue a number of consecutive packets to pass to the device. The process can be halted by the device, e.g. when the device queue occupancy is above a threshold, or by the networking stack, e.g. by BQL algorithm. When a queue occupancy is below a threeshold, the device can wake the queue disc installed on that queue. The wake mechanism of mq is a WAKE_CHILD mode and involves its classes. When a device has to wake the mq queue disc, it wakes the queue disc grafted to the corresponding queue it has to wake.

3.3 New Flow Control Mechanism

The basic idea is to provide a new flow control infrastructure to make it easier to add flow control support to devices. Indeed, adding support for flow control and BQL to devices requires to perform some actions in all the places where a packet is enqueued into or dequeued from the device queue. The current approach to adding such support leads to redundant code and is error-prone. The latter especially holds for devices such as WiFi where enqueue and dequeue operations are scattered throughout the whole module. With the new approach, adding support for flow control and BQL to a device is as simple as calling a (new) method of the `NetDeviceQueueInterface` class, provided that the device uses a subclass of the `Queue` class as device queue. The actions required when enqueuing or dequeuing packets to support flow control and BQL are performed by a few methods that we added to the `NetDeviceQueue` class.

The `Queue` class has been redesigned as a template class object to allow us to instantiate queues storing different types of items. The only requirement on the item type is that it must provide a `GetSize` method which returns the size of the packet included in the item. Currently, queue items can be objects of the following classes:

- `Packet`
- `QueueItem` and subclasses (e.g., `QueueDiscItem`)
- `WifiMacQueueItem`

The internal queues of the queue discs are of type `Queue <QueueDiscItem>`. A number of network devices (e.g., `SimpleNetDevice`, `PointToPointNetDevice`, `CsmaNetDevice`) use a `Queue<Packet>` to store packets to be transmitted. `WifiNetDevices` use instead queues of type `WifiMacQueue`, which is a subclass of `Queue` storing objects of type `WifiMacQueueItem`. Other devices, such as WiMax and LTE, use specialised queues and they still need to be converted to the new queue model.

The `Queue` class derives from the `QueueBase` class, which is a non-template class providing all the methods that are independent of the type of the items stored in the queue. The `Queue` class provides instead all the operations that depend on the item type, such as

enqueue, dequeue, peek and remove. The Queue class also provides the ability to trace certain queue operations such as enqueuing, dequeuing, and dropping. Queue is an abstract base class and is subclassed for specific scheduling and drop policies. Subclasses need to define the following public methods:

- `bool Enqueue (Ptr<Item> item)`: Enqueue a packet
- `Ptr<Item> Dequeue (void)`: Dequeue a packet
- `Ptr<Item> Remove (void)`: Remove a packet
- `Ptr<const Item> Peek (void)`: Peek a packet

The enqueue method does not allow to store a packet if the queue capacity is exceeded. Subclasses may also define specialised public methods. For instance, the WifiMacQueue class provides a method to dequeue a packet based on its TID (Traffic ID) and MAC address.

Two additional traced callbacks, DropBeforeEnqueue and DropAfterDequeue, are introduced in the Queue class. This is required by the new flow control approach because different actions must be performed when a packet is dropped before enqueue (the queue is full) or after dequeue.

To perform the actions required by flow control and BQL, the NetDeviceQueue class is extended with the addition of the followings methods:

- `template <typename Item> static void PacketEnqueued (Ptr<Queue<Item> > queue, Ptr<NetDeviceQueueInterface> ndqi, uint8_t txq, Ptr<const Item> item)`
- `template <typename Item> static void PacketDequeued (Ptr<Queue<Item> > queue, Ptr<NetDeviceQueueInterface> ndqi, uint8_t txq, Ptr<const Item> item)`
- `template <typename Item> static void PacketDiscarded (Ptr<Queue<Item> > queue, Ptr<NetDeviceQueueInterface> ndqi, uint8_t txq, Ptr<const Item> item)`

In order for a device to support flow control and BQL, the PacketEnqueued method must be connected to the Enqueue traced callback of a Queue object (through a bound callback), the PacketDequeued method must be connected to the Dequeue and DropAfterDequeue traced callbacks, and the PacketDiscarded must be connected to the DropBeforeEnqueue traced callback. The NetDeviceQueueInterface class provides the ConnectQueueTraces method which a device calls to properly connect the Queue traces to the new methods, thus gaining support for flow control and BQL.

4 USAGE

Setting the ToS applies to sockets using the IPv4 protocol. The InetSocketAddress class provides the methods to set and get the socket ToS value. Hence, the user can create an address of type InetSocketAddress with the desired ToS value and pass it to the application helpers. Then the networking stack sets the socket priority to the value that the IpTos2Priority function returns when it is passed the ToS value. A typical usage pattern is reported in the following example:

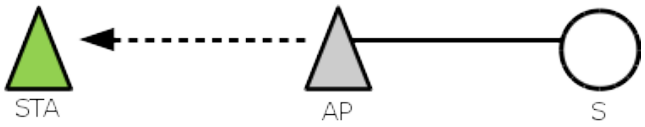


Figure 1: The network topology used for the validation tests.

```
InetSocketAddress dest (address, port);
dest.SetTos (tos);
OnOffHelper onoff ("ns3::TcpSocketFactory", dest);
```

The traffic control helper provides methods to add internal queues, classes and filters to a queue disc. To build a mq queue disc, the traffic control helper can be used to set the root queue disc of type MqQueueDisc and to add as many internal classes as the number of device queues, e.g. four classes for a QoS WiFi device. Additional components, as required by the specific queue disc, may need to be specified, e.g. fq_codel requires at least one packet filter. A usage pattern is reported in the following example, where an MqQueueDisc is built with four classes of type FqCoDelQueueDisc, each of which having an Ipv4PacketFilter:

```
TrafficControlHelper tch;
uint32_t handle = tch.SetRootQueueDisc ("ns3::MqQueue-
Disc");
TrafficControlHelper::ClassIdList cls = tch.AddQueue-
DiscClasses (handle, 4, "ns3::QueueDiscClass");
TrafficControlHelper::HandleList hdl = tch.AddChild-
QueueDiscs (handle, cls, "ns3::FqCoDelQueueDisc");
for (auto h : hdl)
{
tch.AddPacketFilter (h, "ns3::FqCoDelIpv4PacketFilter");
}
```

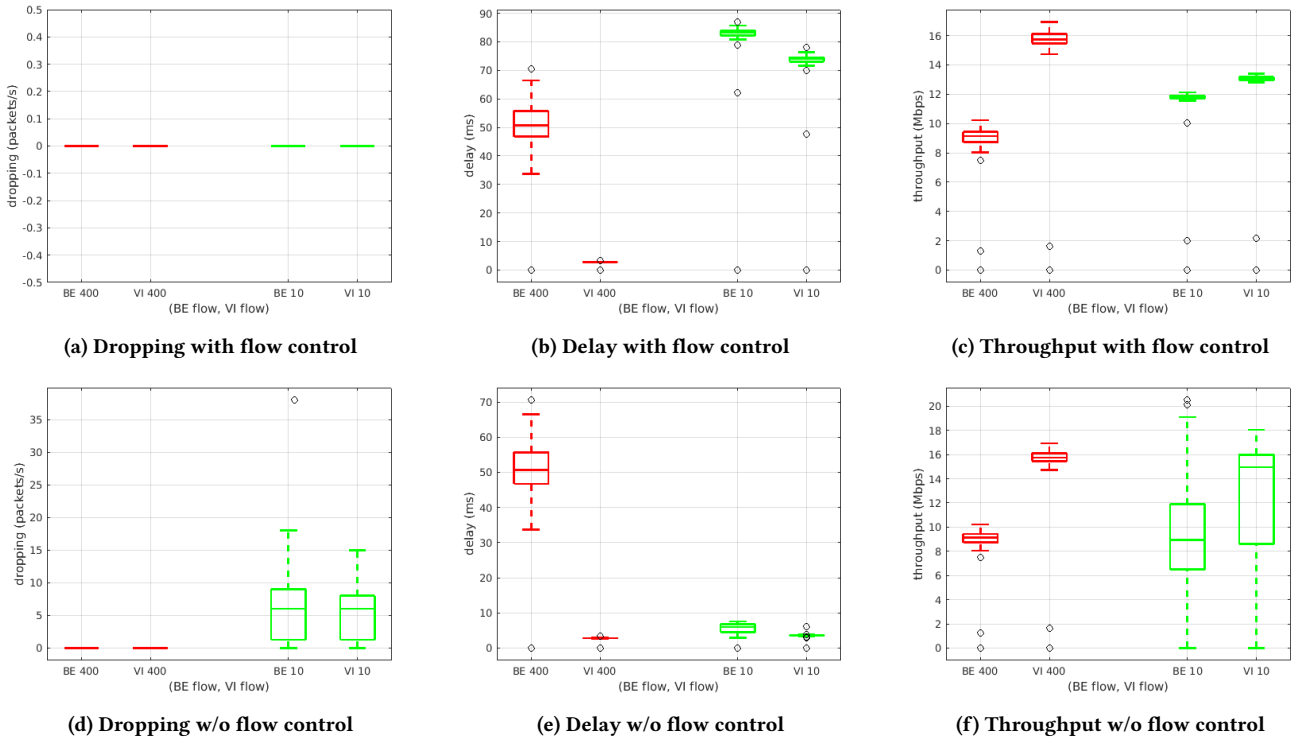
Note that the mq queue disc with classes of type fq_codel is the default traffic control configuration for multiqueue devices in some Linux distributions.

5 RESULTS

5.1 Simulations Setting

For all of the experiments reported hereinafter, the simple three node topology reported in Figure 1 was used. Source (S) and access point (AP) are connected by means of a point-to-point link having a data rate of 100 Mb/s and a delay of 2 ms. An 802.11g station (STA) is associated with the AP. The AP node uses a QoS MAC, while the STA node uses a non-QoS MAC. Both STA and AP use a MinstrelWifiManager and a ConstantPositionMobilityModel. The STA node is placed 5 m away from the AP. Two groups of experiments have been considered:

- the first group of experiments aims to validate the introduction of the new flow control model in WiFi. This group compares the new behaviour with flow control to the previous one without flow control. A pfifo_fast queue disc having a size of 1000 packets is installed on the AP;
- the second group of experiments aims to evaluate the new multiqueue networking models. They allow to highlight



(a) Dropping with flow control

(b) Delay with flow control

(c) Throughput with flow control

(d) Dropping w/o flow control

(e) Delay w/o flow control

(f) Throughput w/o flow control

Figure 2: Evaluation of the impact of flow control on WiFi performance.

behaviours encountered in real devices that could not be modelled previously. An mq queue disc, with two different types of queue discs attached to its classes, is installed on the AP. The first type is pfifo_fast queue disc having a size of 1000 packets and the second type is fq_codel having a size of 1000 packets and CoDel parameters Interval of 100 ms and Target of 5 ms.

Nodes S and STA use the default traffic control configuration of type pfifo_fast queue disc having a size of 1000 packets in all the experiments.

The experiments were conducted by using two TCP flows in downlink from S to STA. The first flow is generated by an OnOff traffic generator with data rate of 20 Mb/s and packet size of 1400 bytes marked with a ToS of 0x00 (DSCP code CS0) and therefore a priority of 0. The second flow is generated by an OnOff traffic generator with data rate of 20 Mb/s and packet size of 1400 bytes marked with a ToS of 0x98 (DSCP code AF43) and therefore a priority of 4. Note the flows are mapped to the same band of a pfifo_fast queue disc, band 1, while are mapped to different ACs of a QoS WiFi device, AC_BE and AC_VI respectively. The TCP version for all the experiments is New Reno. The simulation period is 60 seconds.

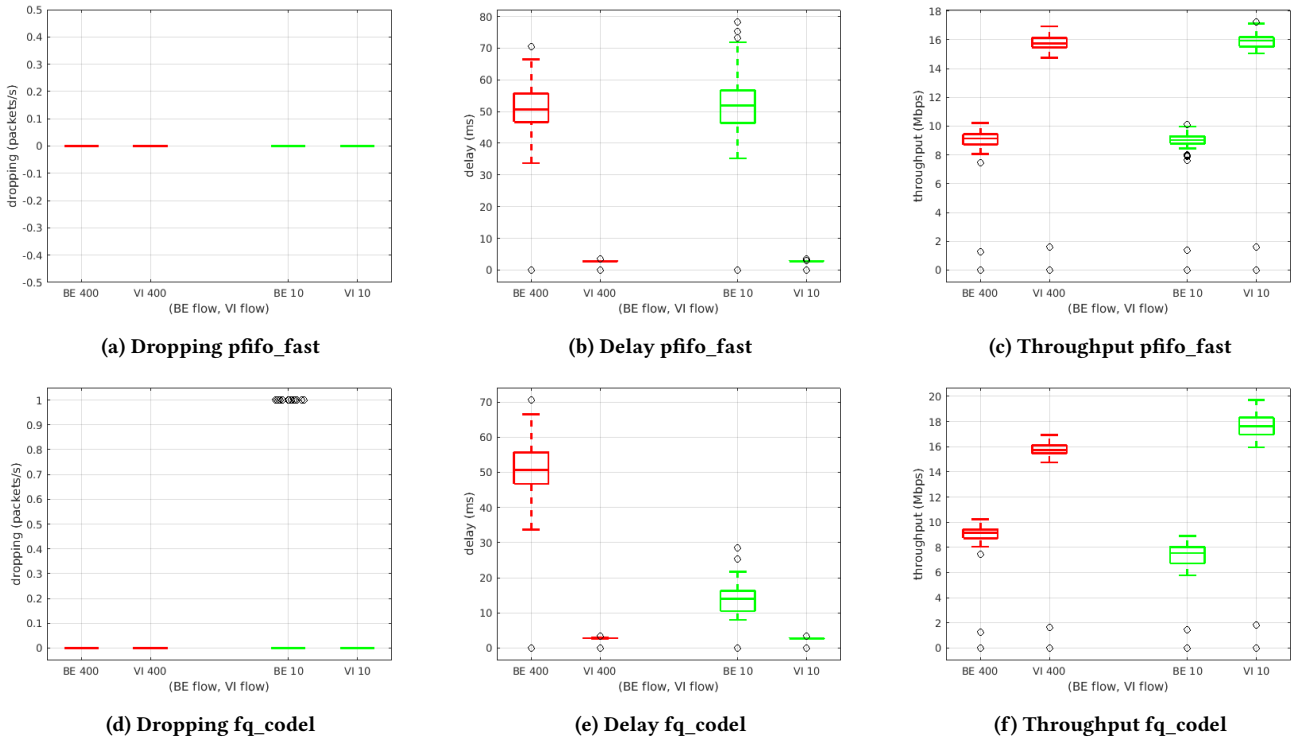
Each experiment evaluates the parameters of the downlink path for the two flows, i.e., the flow BE mapped to the AC_BE and the flow VI mapped to the AC_VI, for two AP MAC queue size. In all the experiments, we evaluate the default MAC queue size of 400 packets and a reduced MAC queue size of 10 packets.

5.2 Flow Control

To evaluate the introduction of the new flow control in WiFi, the current WiFi stack with no flow control is compared to the new one with flow control in the same AP MAC queue size configurations. In particular, the experiments consider two AP MAC queue sizes of 400 and 10 packets. The value of 400 is the default WiFi MAC queue size in ns-3.

Results are reported in Figure 2. With flow control, the pair of queues (the traffic control queues and the device queues) forms a conservative system and there are no losses. Indeed, packets are not passed to the device when it is unable to enqueue them in its queues. This is what occurs in real systems and could not be modelled previously in ns-3 for a WiFi device. Without flow control, packets passed to the device when it is unable to enqueue them are dropped by the device. This behaviour is not encountered in real systems.

In presence of flow control, there is no dropping for the two flows in both cases (Figure 2a). Larger device queues absorb all the packet bursts. With smaller device queues, traffic control stops the sending process to device when it is unable to store other packets. The device queue disc keeps the packets waiting to be sent to the device. The delay is reported in Figure 2b. With larger device queues, the delay of the VI flow (about 3 ms) is smaller, since this flow is prioritized by the AP MAC layer over the BE flow (experiencing a delay of about 50 ms). Although the flows are undifferentiated in the pfifo_fast queue disc, a MAC queue capable enough makes the MAC QoS prioritization effective. The undifferentiated effect



(a) Dropping pfifo_fast **(b) Delay pfifo_fast** **(c) Throughput pfifo_fast**

(d) Dropping fq_codel **(e) Delay fq_codel** **(f) Throughput fq_codel**

Figure 3: Plots of mq queue disc.

in the queue disc has a negligible impact. The case of small queue size presents a reduced advantage of the VI flow with respect to the BE one (with a VI delay of about 73 ms and BE delay of about 82 ms). Indeed, a reduced MAC queue size leads to a prevalence of the undifferentiated effect in the pfifo_fast queue disc. The throughput of the VI flow gain advantages in both cases, compared to the BE one (Figure 2c).

In absence of flow control, the dropping activity depends heavily on the MAC queue size (Figure 2d). In case of larger queues, the device queue is enough capable and absorbs totally the packet bursts and no dropping occurs. In case of smaller queues, the reduced MAC queue size leads to an outstanding dropping activity. This is due to the reduced queue size and therefore they are unable to absorb all the packets. Indeed, we consider a pfifo_fast queue disc and no dropping occurs in traffic control due to other reasons. The delay is reported in Figure 2e. The delay for the case of larger queues presents advantages for VI flow due to the better AC_VI parameters. The case of smaller queues presents a better delay of the BE flow (with a delay of about 7 ms) and a comparable one for the VI flow (with a delay of about 4 ms). These values are achieved by means of the dropping activity that notifies TCP to reduce its sending rate. This is a side effect of no flow control that does not match with behaviour encountered in real systems. The throughput is reported in Figure 2f. The throughput in case of larger queues of the VI flow gain advantages compared to the BE one. In case of smaller queues the throughput is highly variable for both flows with a reduced advantages for the VI flows compared to the BE one.

The obtained results show that the flow control preserves the previous behaviour in case of device queue size enough capable while introduced a new behaviour encountered in real systems and not previously modelled in the simulator.

5.3 Multiqueue Networking

The second group of experiments aims to hightlight the introduction of multiqueue networking models on top of the new flow control model. The mq queue disc is a classful queue disc that enables to use a different queue disc for each device queue. This group of experiments evaluates two types of classes with mq. The first type of class is a pfifo_fast queue disc having a size of 1000 packets for each device queue. The second type is a fq_codel queue disc, an example of Active Queue Management (AQM) algorithm, for each device queue. An AQM is unaware of the queueing time spent in the device queue, which can limit the effectiveness of the AQM.

Results are reported in Figure 3. The mq queue disc enables to make an independent check of each device queue and an independent wake of each queue disc grafted to device queue.

In case of mq queue disc with classes of type pfifo_fast, the conservative system made of queue disc and device queue leads to no dropping activity in both cases (Figure 3a). The delay is reported in Figure 3b. In case of larger queues, the VI flow gets advantages compared to the BE flow and no difference occurs compared to the pfifo_fast without mq. In case of a reduced queue size, mq leads to flow separation at traffic control level and, therefore, the VI flow keeps the advantages compared to the BE flow (with delay values

comparable to the case of larger queues). Also, the throughput keeps the advantages of the VI flow compared to the BE flow in both cases with comparable values (Figure 3c).

In case of mq queue disc with classes of type fq_codel, the dropping activity highlights the ability of the AQM to contrast the increase of the queueing time (Figure 3d). When the device queue is enough capable, it absorbs all the packets and the AQM has no packets to drop. In case of reduced queue size the AQM drops packets to notify TCP to reduce its sending rate and therefore to reduce the queueing time. The dropping regards the BE flow that has a delay over the AQM queueing time target and the dropping is extended with a drop of one packet during the whole simulation period (as reported along the x-axis direction). The delay reflects the dropping activity of the AQM (Figure 3e). The delay in case of larger queues is the same of the pfifo_fast case, due to the inability of the AQM to contrast the queueing time. The delay in case of smaller queues is reduced for the BE flow (to about 15 ms) and preserves the advantages for the VI flow (with a delay of about 3 ms). The throughput in case of larger queues is the same of the pfifo_fast without mq, while reflects the dropping activity in case of smaller queues. In particular, the dropping on BE flow enables to better advantages of the VI flow compared to the BE one.

The obtained results show that the multiqueue networking introduced new behaviours in the simulator. The flow separation set up from device leads to a better flows prioritisation. The AQM algorithms can be applied to separate flows and improve the delay of each flow. Their effectiveness is strongly related to device queue size.

6 CONCLUSIONS AND FUTURE WORK

In this work, we presented the models of socket priority, multiqueue networking and a new flow control infrastructure and their preliminary validation. The socket priority and the flow control models have been integrated in ns-3.26 while the multiqueue networking model is queued for the inclusion in 3.27. The introduced models enable to carry out more realistic simulations on multiqueue devices and traffic differentiation. In particular, we can accurately evaluate the effectiveness of the AQM algorithm on multiqueue devices. The new flow control strategy introduced in ns-3 makes traffic control aware of the device queue status for all the devices that use the `Queue` class to implement their queues. Currently the `SimpleNetDevice`, `PointToPointNetDevice`, `CsmaNetDevice` and `WiFiNetDevice` are the traffic control aware devices.

REFERENCES

[1] W. Almesberger, J. Salim, and A. Kuznetsov. Differentiated Services on Linux. In *Proceedings of the Global Telecommunications Conference (Globecom)*, volume 1B, pages 831–836. IEEE, 1999.

[2] P. Almquist. Type of Service in the Internet Protocol Suite. RFC 1349, RFC Editor, July 1992. http://www.rfc-editor.org/rfc/rfc1349.txt.

[3] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet. The Flowqueue-codel Packet Scheduler and Active Queue Management Algorithm. Internet-Draft draft-ietf-aqm-fq-codel-06, IETF Secretariat, March 2016. http://www.ietf.org/internet-drafts/draft-ietf-aqm-fq-codel-06.txt.

[4] P. Imputato and S. Avallone. Design and Implementation of the Traffic Control Module in ns-3. In *Proceedings of the Workshop on ns-3*, WNS3 '16, pages 1–8, New York, NY, USA, 2016. ACM.

[5] K. Nichols, S. Blake, F. Baker, and D. L. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474, RFC Editor, December 1998. http://www.rfc-editor.org/rfc/rfc2474.txt.