The traffic-control module in ns-3

Stefano Avallone

WNS3 2017 Training
June, 12 – INESC Porto
Introduction

• The traffic-control module has been introduced in ns-3.25 (March 2016)
  - Advanced queue management and packet shaping
  - Modelled after the Linux Traffic Control (TC) infrastructure

• Some important pieces completed over the next releases
  - Packet priority handling, byte queue limit (ns-3.26)
  - Simplified flow control support (ns-3.27)

• traffic-control acts on the transmission of packets
  - Linux also allows to police incoming traffic
The Linux TC infrastructure

- 1 packet enqueued → multiple packets dequeued
- Device driver
  - Packet received: if there is no room for another packet, then *stop* the queue
  - Notification received from the device: if there is room for another packet, then *wake* the queue
- The size of the transmission ring can be dynamically adjusted by BQL
- Interrupt mitigation techniques or polling to reduce overhead
ns-3 network stack pre-3.25

- AQM algorithms (RED, CoDel) available as subclasses of Queue
- No flow control: packets discarded by the NetDevice if no room

Dependencies among modules:
- core
- network
- internet

Ipv{4,6}L3Protocol

Ipv{4,6}Interface

NetDevice

channel
ns-3 network stack pre-3.25: Limitations

- AQM algorithms only available for devices using Queue objects
  - No wifi, lte, wimax, ...
- ECN support difficult
  - IP header needs to be removed/modified/added
- Packet filtering based on L4 ports difficult
  - L4 header needs to be accessed
- Very difficult to reproduce the impact of BQL, interrupt mitigation, ...
Introducing the traffic-control module

Dependencies among modules:
Implementation details (1/5)

NetDeviceQueueInterface

m_txQueuesVector

NetDeviceQueue

m_wakeCallback

NetDeviceQueue

m_wakeCallback

NetDeviceQueue

m_wakeCallback

NetDevice 1

NetDeviceQueueInterface

m_txQueuesVector

NetDeviceQueue

m_wakeCallback

NetDeviceQueue

m_wakeCallback

QueueDisc

NetDeviceQueue

m_wakeCallback

QueueDisc

NetDeviceQueue

m_wakeCallback

NetDeviceQueue

m_wakeCallback

NetDevice N

NetDeviceQueueInterface

m_txQueuesVector

NetDeviceQueue

m_wakeCallback

NetDeviceQueue

m_wakeCallback

TrafficControlLayer

NetDeviceInfo

m_rootQueueDisc

m_queueDiscsToWake

1

N
void TrafficControlLayer::Send (Ptr<NetDevice> device, Ptr<QueueDiscItem> item) {
    ...
    uint8_t txq = 0;
    ... 
    Ptr<QueueDisc> qDisc = ndi->second.m_queueDiscsToWake[txq];
    qDisc->Enqueue (item);
    qDisc->Run ();
}

void QueueDisc::Run (void) {
    ...
    uint32_t quota = m_quota;
    while (Restart ()) {
        quota -= 1;
        if (quota <= 0) {
            break;
        }
    }
}
bool QueueDisc::Restart (void)
{
    Ptr<QueueDiscItem> item = DequeuePacket();
    if (item == 0)
    {
        NS_LOG_LOGIC ("No packet to send");
        return false;
    }
    return Transmit (item);
}

Ptr<QueueDiscItem> QueueDisc::DequeuePacket (void)
{
    Ptr<QueueDiscItem> item;
    if (m_devQueueIface->GetNTxQueues ()>1 || !m_devQueueIface->GetTxQueue(0)->IsStopped())
    {
        item = Dequeue ();
        if (item != 0)
        {
            item->AddHeader ();
        }
    }
    return item;
}
bool QueueDisc::Transmit (Ptr<QueueDiscItem> item) {
    m_device->Send (item->GetPacket (), item->GetAddress (), item->GetProtocol ());
    if (GetNPackets () == 0 ||
        m_devQueueIface->GetTxQueue (item->GetTxQueueIndex ())->IsStopped ())
    {
        return false;
    }
    return true;
}
After enqueuing a packet into one of its queues, the NetDevice has to **stop** the corresponding NetDeviceQueue if that queue cannot store another packet
- By calling `NetDeviceQueue::Stop()`

After dequeuing a packet from a queue, the NetDevice has to **wake** the corresponding NetDeviceQueue if there is room for another packet in that queue
- By calling `NetDeviceQueue::Wake()`
Flow control

• NetDevices should be modified to add support for flow control

• **Queue discs are useless if the NetDevice does not support flow control**
  - If NetDeviceQueues are never stopped, packets are enqueued and immediately dequeued from the queue disc

• As of ns-3.26, PointToPointNetDevice is the only NetDevice supporting flow control

• Adding flow control support to WifiNetDevice is tricky
  - Packets are dequeued at multiple points in the code
  - Packets are dequeued by subclasses that do not hold a pointer to WifiNetDevice
Simplified flow control support

• Starting from the upcoming ns-3.27, a NetDevice using a Queue subclass to store its packets can gain support for flow control (and BQL) by calling:

\[
\text{NetDeviceQueueInterface::ConnectQueueTraces(\text{Ptr<Queue<Item>> queue, uint8_t txq})}
\]

• which connects:
  - NetDeviceQueue::PacketEnqueued to the “Enqueue” traced callback of the queue
  - NetDeviceQueue::PacketDequeued to the “Dequeue” and “DropAfterDequeue” traced callbacks of the queue
  - NetDeviceQueue::PacketDiscarded to the “DropBeforeEnqueue” traced callback of the queue
Flow control support status

- PointToPointNetDevice, CsmaNetDevice, SimpleNetDevice easily gained flow control support
  - These NetDevices were already using Queue
- The Queue class was reworked to define WifiMacQueue as a subclass of Queue and have WifiNetDevice gain flow control support
  - Queue is now a template class
    - The type parameter is the type of the elements stored in the queue (Queue<Packet>, Queue<QueueDiscItem>, …)
    - Allowed us to get rid of static casts in queue discs
- To add flow control support for other NetDevices
  - Let them use Queue<Item>
  - Add the required operations when packets are enqueued or dequeued from their queues
Multi-queue networking

- Many network device drivers have multiple queues
  - To exploit the availability of multiple cores (Ethernet drivers)
  - For QoS purposes (Wi-Fi drivers)
- Multi-queue aware queue discs have been introduced
  - mq, multiq, mq-prio
  - Create as many child queue discs as the number of queues used by the network device driver
  - Each child queue disc corresponds to a queue
  - Packets are enqueued in the queue disc corresponding to the queue in which the network device driver will enqueue the packet
  - Network device drivers have to provide a select callback
void TrafficControlLayer::Send (Ptr<NetDevice> device, 
Ptr<QueueDiscItem> item)
{
  ...
  uint8_t txq = 0;
  if (devQueueIface->GetNTxQueues() > 1)
  {
    if (!ndi->
      second.m_selectQueueCallback.IsNull ())
    {
      txq = ndi->
        second.m_selectQueueCallback (item);
    }
    Ptr<QueueDisc> qDisc = ndi->second.m_queueDiscsToWake[txq];
    qDisc->Enqueue (item);
    qDisc->Run ();
  }
}
uint8_t
WifiNetDevice::SelectQueue
(Ptr<QueueItem> item) const
{
    uint8_t dscp, priority = 0;
    if (item->GetUint8Value
(QueueItem::IP_DSFIELD, dscp))
    {
        priority = dscp >> 5;
    }
    // replace the priority tag
    ...
    return QosUtilsMapTidToAc (priority);
}

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

- The socket priority is set based on the socket ToS
  - Socket::SetIpTos (tos) calls Socket::IpTos2Priority (tos)
- The packet priority is set equal to the socket priority
  - Ipv4L3Protocol::IpForward

<table>
<thead>
<tr>
<th>PHB</th>
<th>TOS (binary)</th>
<th>bits 3-6</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>EF</td>
<td>101110xx</td>
<td>12-13</td>
<td>4</td>
</tr>
<tr>
<td>AF11</td>
<td>001010xx</td>
<td>4-5</td>
<td>2</td>
</tr>
<tr>
<td>AF21</td>
<td>010010xx</td>
<td>4-5</td>
<td>2</td>
</tr>
<tr>
<td>AF31</td>
<td>011010xx</td>
<td>4-5</td>
<td>2</td>
</tr>
<tr>
<td>AF41</td>
<td>100010xx</td>
<td>4-5</td>
<td>2</td>
</tr>
<tr>
<td>AF12</td>
<td>001100xx</td>
<td>8-9</td>
<td>6</td>
</tr>
<tr>
<td>AF22</td>
<td>010100xx</td>
<td>8-9</td>
<td>6</td>
</tr>
<tr>
<td>AF32</td>
<td>011100xx</td>
<td>8-9</td>
<td>6</td>
</tr>
<tr>
<td>AF42</td>
<td>100100xx</td>
<td>8-9</td>
<td>6</td>
</tr>
<tr>
<td>AF13</td>
<td>001110xx</td>
<td>12-13</td>
<td>4</td>
</tr>
<tr>
<td>AF23</td>
<td>010110xx</td>
<td>12-13</td>
<td>4</td>
</tr>
<tr>
<td>AF33</td>
<td>011110xx</td>
<td>12-13</td>
<td>4</td>
</tr>
<tr>
<td>AF43</td>
<td>100110xx</td>
<td>12-13</td>
<td>4</td>
</tr>
<tr>
<td>CS0</td>
<td>000000xx</td>
<td>0-1</td>
<td>0</td>
</tr>
<tr>
<td>CS1</td>
<td>001000xx</td>
<td>0-1</td>
<td>0</td>
</tr>
<tr>
<td>CS2</td>
<td>010000xx</td>
<td>0-1</td>
<td>0</td>
</tr>
<tr>
<td>CS3</td>
<td>011000xx</td>
<td>0-1</td>
<td>0</td>
</tr>
<tr>
<td>CS4</td>
<td>100000xx</td>
<td>0-1</td>
<td>0</td>
</tr>
<tr>
<td>CS5</td>
<td>101000xx</td>
<td>0-1</td>
<td>0</td>
</tr>
<tr>
<td>CS6</td>
<td>110000xx</td>
<td>0-1</td>
<td>0</td>
</tr>
<tr>
<td>CS7</td>
<td>111000xx</td>
<td>0-1</td>
<td>0</td>
</tr>
</tbody>
</table>

Bits 3-6 | Priority
---------|----------
0 to 3   | 0 (Best Effort)  
4 to 7   | 2 (Bulk)        
8 to 11  | 6 (Interactive) 
12 to 15 | 4 (Inter. Bulk)
PfifoFastQueueDisc

- PfifoFastQueueDisc behaves like the pfifo_fast qdisc
  - default qdisc in Linux
- Packets are enqueued into three priority bands (queues) based on (the four least significant bits of) the packet priority
  - Carried by the SocketPriorityTag
  - As modified by the select queue callback (e.g., wifi)
- Band 1 is the default band

<table>
<thead>
<tr>
<th>Priority &amp; 0xf</th>
<th>Band</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
</tr>
</tbody>
</table>
Default configuration

- A TrafficControlLayer object is aggregated to a node by InternetStackHelper::Install (node)

- **By default**, when an IPv{4,6} address is assigned to a NetDevice, a PfifoFast queue disc is installed on the NetDevice
  - Ipv{4,6}AddressHelper::Assign (const NetDeviceContainer &c)

- In order to install a queue disc other than PfifoFast
  - Install the queue disc before assigning IP addresses
    - Ipv{4,6}AddressHelper::Assign does not install PfifoFast on a NetDevice if a queue disc is already installed
  - Remove PfifoFast and install a different queue disc
Various queue disc types

- Classless, single queue
- Classless, multiple queues
- Classful
TrafficControlHelper API

uint16_t SetRootQueueDisc (string type, string n01, const AttributeValue &v01,...);

void AddInternalQueues (uint16_t handle, uint16_t count, string type,
   string n01, const AttributeValue &v01,...);

void AddPacketFilter (uint16_t handle, string type,
   string n01, const AttributeValue &v01,...);

ClassIdList AddQueueDiscClasses (uint16_t handle, uint16_t count, string type,
   string n01, const AttributeValue &v01,...);

uint16_t AddChildQueueDisc (uint16_t handle, uint16_t classId, string type,
   string n01, const AttributeValue &v01,...);

HandleList AddChildQueueDiscs (uint16_t handle, const ClassIdList &classes,
   string type, string n01, const AttributeValue &v01,...);

QueueDiscContainer Install (Ptr<NetDevice> d);

void Uninstall (Ptr<NetDevice> d);
Installing classless queue discs

```
TrafficControlHelper tch;

// PfifoFast
uint16_t h = tch.SetRootQueueDisc("ns3::PfifoFastQueueDisc");
tch.AddInternalQueues(h, 3, "ns3::DropTailQueue");  // optional

// RED
tch.SetRootQueueDisc("ns3::RedQueueDisc");

// CoDel
tch.SetRootQueueDisc("ns3::CoDelQueueDisc");

// PIE
tch.SetRootQueueDisc("ns3::PieQueueDisc");
```
Installing classful queue discs

TrafficControlHelper tch;

// FqCodel
// Classes and child queue discs are dynamically created by the Enqueue method

uint32_t h = tch.SetRootQueueDisc("ns3::FqCoDelQueueDisc",
                                     "PacketLimit", UintegerValue (1000));

tch.AddPacketFilter (handle, "ns3::FqCoDelIpv4PacketFilter");
tch.AddPacketFilter (handle, "ns3::FqCoDelIpv6PacketFilter");
Installing classful queue discs

```
TrafficControlHelper tch;

// Prio – ns.3-28
// An internal packet filter may be used

uint16_t h = tch.SetRootQueueDisc ("ns3::PrioQueueDisc");
TrafficControlHelper::ClassIdList cid = tch.AddQueueDiscClasses (h, 2, 
    "ns3::QueueDiscClass");
tch.AddChildQueueDisc (h, cid[0], "ns3::FifoQueueDisc");
tch.AddChildQueueDisc (h, cid[1], "ns3::RedQueueDisc");
```
Installing multi-queue aware queue discs

TrafficControlHelper tch;

// Mq installed on a WifiNetDevice
// Classes are created just for convenience

uint16_t h = tch.SetRootQueueDisc ("ns3::MqQueueDisc");
TrafficControlHelper::ClassIdList cls = tch.AddQueueDiscClasses (h, 4,
"ns3::QueueDiscClass");
TrafficControlHelper::HandleList hdl = tch.AddChildQueueDiscs (h, cls,
"ns3::FqCoDelQueueDisc");

for (auto hh : hdl)
{
    tch.AddPacketFilter (hh, "ns3::FqCoDelIpv4PacketFilter");
}
Thank you!