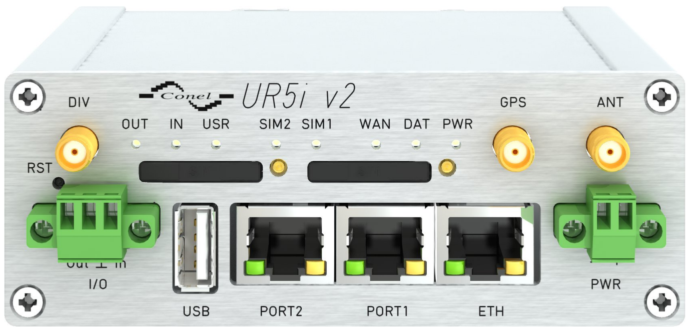
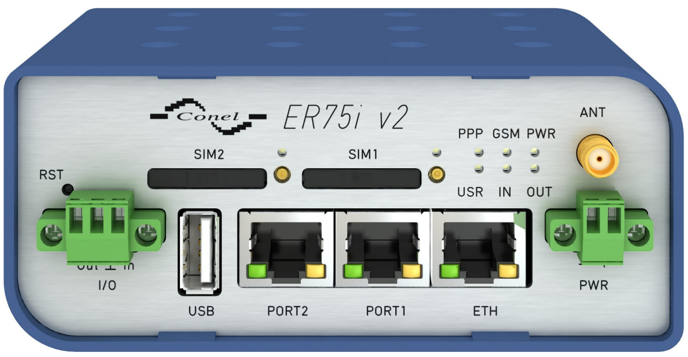
A B&B ELECTRONICS Company

Quality of Service

APPLICATION NOTE





Used symbols

⚠ *Danger* – important notice, which may have an influence on the user's safety or the function of the device.

❗ *Attention* – notice on possible problems, which can arise in specific cases.

ℹ *Information, notice* – information, which contains useful advice or special interest.

✎ *Example* – example of function, command or script.

GPL license

Source codes under GPL license are available free of charge by sending an email to:

info@conel.cz.

Conel s.r.o., Sokolska 71, 562 04 Usti nad Orlici, Czech Republic
Manual issued in CZ, May 15, 2014

Contents

1. Introduction to QoS

Quality of Service (QoS) is a method to devide an available bandwidth into classes so that the quality of network services doesn't decrease at the network congestion. Basically it is the ability to provide different priority to different applications, users or data flows and the packets are sent from each class in right order. This also means that due to QoS can be guaranteed a required bit rate, delay, jitter or packet dropping probability. Quality of Service guarantees are important if the network capacity is insufficient, especially for real-time streaming multimedia applications such as VOIP (Voice over IP) nad IP-TV, since these often require fixed bit rate and are delay sensitive.

1.1 Packet queues

Packet queue is the order in which packets are sent from network interface. Packets that are waiting in the queue can be affected by **qdisc** disciplines. It is an implementation of various QoS algorithms. Individual packets in the queue can be dropped, delayed or their order can be changed. There are two basic types of disciplines – *classless qdisc* and *classful qdisc*.

1.2 Classless qdisc

The disciplne of this type can't contain another qdiscs. Classless qdisc only determines whether the packet is classified, delayed or dropped.

- **FIFO** – Simplest usable qdisc, pure First In First Out (FIFO) behaviour. It is not too fair or adjustable qdisc.
- **TBF** – Token Bucket Filter is suited for slowing traffic down to a precisely configured rate. Packets are stored in the "buckets" and arrive at a steady rate, until the bucket is full. If the bucket overflows, packets are dropped. The resulting effect is the resistance to small dropouts.
- **SFQ** – The SFQ qdisc attempts to fairly distribute opportunity to transmit data to the network among an arbitrary number of flows. It accomplishes this by using a hash function to separate the traffic into separate (internally maintained) FIFOs which are dequeued in a round robin fashion. Because there is the possibility for unfairness to manifest in the choice of hash function, this function is altered periodically. *Perturb* parameter sets this periodicity.
- **RED** – Random Early Detection (RED) simulates physical congestion by randomly dropping packets when nearing configured bandwidth allocation. This type of qdisc is useful on a backbone network.
- **WRED** – RED with support of priorities.

If there is defined no type of qdisc, the FIFO queue is the automatic default. Another queue can selected using the following command:

tc qdisc add dev <device> **root** <qdisc> <qdisc parameters>

To remove a queue use:

tc qdisc del dev <device> **root**

1.3 Classful qdisc

The disciplne of this type can contain another qdisc (classless or classful). This creates a tree of classes. Internal nodes (and root node) of this tree contain another classful qdiscs and leaves contain classless qdisc. Each classful qdisc discipline may have **filters** that determine qdisc, where packets will be transmitted. Therefore packets travel through a tree structure to the leaves.

- **PRIO** – The PRIO qdisc creates three classes with priorities 0, 1, 2. Classes are processed sequentially from the lowest priority to highest.
- **CBQ** – Class Based Queueing is a frequently used type of qdisc that implements a rich linksharing hierarchy of classes. CBQ supports priorities.
- **HTB** – This qdisc is meant as a more understandable and intuitive replacement for the CBQ qdisc. HTB facilitates guaranteeing bandwidth to classes, while also allowing specification of upper limits to inter-class sharing.

2. Configuration

Configuration is performed using **tc** (traffic control) utility from the iproute package. Disciplines are written in the *number:number* form. The first number (in front of the colon) is the qdisc number and the second number (after the colon) is the class number. These numbers must be unique, because it's used for identification.

In pursuance of creating a tree structure it is necessary to observe certain rules. Bandwidth of the parent node must be greater than the sum of data flows in leaves. It is not appropriate that the tree consists of a large number of leaves with much lower guaranteed throughput than the total throughput.

2.1 Shaping

Shaping is the mechanism by which packets are delayed before transmission in an output queue to meet a desired output rate. This is one of the most common desires of users seeking bandwidth control solutions.

We have a line with 1 Mbps download and we want to share it to three users so that the first user has a guaranteed minumum 512 kbps and the others 256 kbps. If any user does not utilize allocated capacity, free part is distributed among other users. The IP addresses of users are 10.0.0.10 – 10.0.0.12.

At first create a structure of HTB classes. The root class contains the entire capacity of the line and three subclasses correspond to three users.

```
tc qdisc del dev eth0 root
tc qdisc add dev eth0 root handle 1: htb default 1

tc class add dev eth0 parent 1: classid 1:1 htb rate 1024kbit

tc class add dev eth0 parent 1:1 classid 1:11 htb rate 512Kbit ceil 1024kbit
tc class add dev eth0 parent 1:1 classid 1:12 htb rate 256Kbit ceil 1024kbit
tc class add dev eth0 parent 1:1 classid 1:13 htb rate 256Kbit ceil 1024kbit
```

Classes with higher priority are usually stated before. *Rate* parameter specifies a guaranteed throughput (sum of children rates should not exceed the value of parent rate). *Ceil* specifies the maximum throughput. Children ceil should not be larger than parent ceil.

The default gueue for qdisc is the FIFO. However this type of queue is not too fair so it is better to use for instance SFQ. The *perturb* parameter allows you to specify how often SFQ changes its hashing algorithm.

```
tc qdisc add dev eth0 parent 1:11 handle 11: sfq perturb 10
tc qdisc add dev eth0 parent 1:12 handle 12: sfq perturb 10
tc qdisc add dev eth0 parent 1:13 handle 13: sfq perturb 10
```

Now it is necessary to determine in which class a packet will be enqueued. Whenever traffic arrives at a class with subclasses, it needs to be classified. Various methods may be employed to do so, one of these are the filters.

Classification based on a destination IP address:

```
tc filter add dev eth0 parent 1:1 protocol ip u32 match ip dst 10.0.0.10 flowid 1:11
tc filter add dev eth0 parent 1:1 protocol ip u32 match ip dst 10.0.0.11 flowid 1:12
tc filter add dev eth0 parent 1:1 protocol ip u32 match ip dst 10.0.0.12 flowid 1:13
```

Classification based on iptables marking:

```
iptables -t mangle -A FORWARD -d 10.0.0.10 -j MARK –set-mark 10
iptables -t mangle -A FORWARD -d 10.0.0.11 -j MARK –set-mark 11
iptables -t mangle -A FORWARD -d 10.0.0.12 -j MARK –set-mark 12
```

```
tc filter add dev eth0 parent 1: protocol ip handle 10 fw flowid 1:11
tc filter add dev eth0 parent 1: protocol ip handle 11 fw flowid 1:12
tc filter add dev eth0 parent 1: protocol ip handle 12 fw flowid 1:13
```

The first method of classification is primarily used in situations in which the complex rules are not needed. The second method can be used (for example) for classification based on MAC addresses. This method is slightly slower.