Package QOS
Version 3.10.5

Frank Meyer
email: frank@fli4l.de

The fli4l-Team
email: team@fli4l.de

February 16, 2016

Contents

1. Documentation Of Package QOS

1.1. QoS - Quality of Service

By QoS the available bandwidth can be regulated and for example be distributed to several ports, IP addresses and more.

A modem manages a packet queue where packets are stored that exceed the available bandwidth. With DSL modems for example these queues are rather big. The advantage is a constant usage of maximum bandwidth. If the router sends lesser packets for a short period of time the modem has packets in the queue that it can send. Such a queue is a simple thing sending packets first in first out - rather fair, isn't it?

This is where QoS comes into play. QoS also manages a packet queue in the router itself which makes it possible to decide which packets are to be sent at first and which to hold back. If everything is configured the right way QoS sends packets at the speed the modem sends them out not filling the modem queue at any time. This is like moving the modem queue into the router.

Something general on speed units: QoS supports Mibit/s (mebibit/s) and Kibit/s (kibibit/s), where applies 1Mibit = 1024Kibit.

1.1.1. Configuration

OPT_QOS Set this to 'yes' to enable and 'no' to disable `OPT_QOS`.

QOS_INTERNET_DEV_N Number of devices routing data to the Internet.

QOS_INTERNET_DEV_x List of devices that route data to the Internet. Examples:

`QOS_INTERNET_DEV_N='3'`	device number
`QOS_INTERNET_DEV_1='ethX'`	for cable and other ethernet connections
`QOS_INTERNET_DEV_2='ppp0'`	for DSL over PPPoE
`QOS_INTERNET_DEV_3='ipppX'`	for ISDN

The ISDN device for the first circuit should be named ippp0 the second ippp1. If channel bundeling is activated for the first circuit the second channel of the first circuit is named ippp1 and the second circuit ippp2. QOS should only be used with ISDN if channel bundeling is deactivated for the circuit used.

QOS_INTERNET_BAND_DOWN Maximum downstream bandwidth of the Internet connection. See above: Something general on speed units (Page 3).

Hint: For time-critical jobs like preferring ACK packets it is necessary to limit the bandwidth to the actual value. Else packets will be sorted correctly in the packet queue but are withheld in the modem's packet queue. It may be possible that bandwidth declared by your provider is not in accordance with real conditions. It could be a little less or a little more. At the end only trying helps here.

QOS_INTERNET_BAND_UP Maximum upstream bandwidth of the Internet connection. See above: Something general on speed units (Page 3).

Also see hint at `QOS_INTERNET_BAND_DOWN`.

QOS_INTERNET_DEFAULT_DOWN Set the default class for packets coming from the Internet here. All packets which are not classified by a filter will end here.

If no class is specified for which variable

```
QOS_CLASS_x_DIRECTION='down'
```

is set specify:

```
QOS_INTERNET_DEFAULT_DOWN='0'
```

Example:

Two classes have been created and a filter puts all packets for a certain IP address into the first one. All other packets should go to the second one. This must be set like this:

```
QOS_INTERNET_DEFAULT_DOWN='2'
```

Pay attention to set a class for `QOS_INTERNET_DEFAULT_DOWN` where its `QOS_CLASS_x_DIRECTION` variable contains the argument 'down'.

QOS_INTERNET_DEFAULT_UP Set the default class for packets going out to the Internet here. All packets which are not classified by a filter will end here.

If no class is specified for which variable

```
QOS_CLASS_x_DIRECTION='up'
```

is set specify:

```
QOS_INTERNET_DEFAULT_UP='0'
```

This works in analog to `QOS_INTERNET_DEFAULT_DOWN`.

Pay attention to set a class for `QOS_INTERNET_DEFAULT_UP` where its `QOS_CLASS_x_DIRECTION` variable contains the argument 'up'.

QOS_CLASS_N Set the number of classes to be created.

QOS_CLASS_x_PARENT By this variable classes can be stacked. Set the number of the parent class here. Bandwidth allocated to the parent class can be spread between the subclasses. Maximum subclass layer depth is 8 whereas the interface itself is the first layer which leaves a maximum of 7 layers to be configured.

If the class is no subclass write the following:

```
QOS_CLASS_x_PARENT='0'
```

It will get the maximum bandwidth set in QOS_CLASS_x_PORT_TYPE depending on its direction (in- or outbound, see QOS_CLASS_x_PORT_TYPE)

Important: If this is not '0' pay attention to define the parent class before (in numbering)

QOS_CLASS_x_MINBANDWIDTH Bandwidth to allocate to the classes. See above: Something general on speed units (Page 3).

Example: A class with a bandwidth limited to 128Kibit/s:

```
QOS_CLASS_1_MINBANDWIDTH='128Kibit/s'
QOS_CLASS_1_MAXBANDWIDTH='128Kibit/s'
QOS_CLASS_1_PARENT='0'
```

Three subclasses of our parent class above where QOS_CLASS_x_MINBANDWIDTH- and QOS_-CLASS_x_MAXBANDWIDTH settings look like this:

```
QOS_CLASS_2_PARENT='1'
QOS_CLASS_2_MINBANDWIDTH='60Kibit/s'
QOS_CLASS_2_MAXBANDWIDTH='128Kibit/s'

QOS_CLASS_3_PARENT='1'
QOS_CLASS_3_MINBANDWIDTH='40Kibit/s'
QOS_CLASS_3_MAXBANDWIDTH='128Kibit/s'

QOS_CLASS_4_PARENT='1'
QOS_CLASS_4_MINBANDWIDTH='28Kibit/s'
QOS_CLASS_4_MAXBANDWIDTH='128Kibit/s'
```

All subclasses have the same (or no) priority (see QOS_CLASS_x_PRIO). If traffic on all subclasses exceeds their QOS_CLASS_x_MINBANDWIDTH they all get QOS_CLASS_x_MINBANDWIDTH allocated. If class 2 has only 20Kibit/s traffic there are 40Kibit/s "left". This overrun will be split in 40/28 ratio between class 3 and 4. Each class is limited to 128Kibit/s by QOS_CLASS_x_MAXBANDWIDTH and because all classes are subclasses of a parent class limited to 128Kibit/s the maximum bandwidth consumed is 128Kibit/s.

QOS_CLASS_x_MAXBANDWIDTH Maximum bandwidth to be allocated to the class. There is no sense in entering a lower value than in QOS_CLASS_x_MINBANDWIDTH. If nothing is set here this variable automatically will be set to the value of QOS_CLASS_x_MINBANDWIDTH. Such a class can't use any free bandwidth resources.

See above: Something general on speed units (Page 3).

QOS_CLASS_x_DIRECTION This variable sets the direction the class belongs to. If upstream regulation is intended set:

```
QOS_CLASS_x_DIRECTION='up'
```

for downstream in analog:

```
QOS_CLASS_x_DIRECTION='down'
```

QOS_CLASS_x_PRIO Set the priority of the class here. The lower the number the higher the priority. Values between 0 and 7 are allowed. Leaving this empty equals to setting '0'.

Priorities are used to determine which class can use free bandwidth resources. Lets change our example in `QOS_CLASS_x_MINIMUMBANDWIDTH` to reflect this: Nothing was changed for the first class. Classes 2 to 4 get a priority:

```
QOS_CLASS_2_PARENT='1'
QOS_CLASS_2_MINBANDWIDTH='60Kibit/s'
QOS_CLASS_2_MAXBANDWIDTH='128Kibit/s'
QOS_CLASS_2_PRIO='1'

QOS_CLASS_3_MINBANDWIDTH='40Kibit/s'
QOS_CLASS_3_PARENT='1'
QOS_CLASS_3_MAXBANDWIDTH='128Kibit/s'
QOS_CLASS_3_PRIO='1'

QOS_CLASS_4_PARENT='1'
QOS_CLASS_4_MINBANDWIDTH='28Kibit/s'
QOS_CLASS_4_MAXBANDWIDTH='128Kibit/s'
QOS_CLASS_4_PRIO='2'
```

Like in the original example class 2 consumes only 20Kibit/s and leaves free bandwidth of 40Kibit/s. Classes 3 and 4 both need more bandwidth than available. Class 3 now has a higher priority than class 4 and may use the free bandwidth of 40Kibit/s.

If class 3 needs only 20Kibit/s of the free bandwidth of 40Kibit/s class 4 will get the remaining 20Kibit/s.

Lets assume something different: Class 4 needs no bandwidth at all and class 2 and 3 both need more than exists. So they both get the bandwidth set in `QOS_CLASS_x_MINBANDWIDTH` and the remaining will be divided in 60/40 ratio between them because both classes have the same priority.

As you see `QOS_CLASS_x_PRIO` only has influence on how an enventual overrun in bandwidth is spread.

QOS_CLASS_x_LABEL By using this optional variable a label for the class may be specified which will be used as the caption for the QOS-graphs created by an activated OPT_RRDTOOL.

QOS_FILTER_N Set the number of filters desired here.

A quick note on filters: Arguments of the different variables are AND-linked, arguments of the same variable are OR-linked. This means: If the same filter filters by an IP-address and a port only packets will get selected and put in the queue which match both conditions at a time.

Another example: Two ports (21 and 80) and an IP address are set in the same filter. Since a packet can't use two ports at a time the filter will match those packets that either use port 21 and the named IP address or port 80 and the named IP address.

Important: The ordering of filters is essential!

An example: All traffic on port 456 for **all** clients should be queued to class A. In addition all packets to a client with IP address 192.168.6.5 should be queued to class B, except those on port 456. If the filter on the IP is created first all packets (those on port 456 too) will be queued to class B and the filter on port 456 doesn't change this behavior. The filter on port 456 has to be created before the one on the IP address 192.168.6.5 to achieve our goals.

QOS_FILTER_x_CLASS This variable specifies the class a packet is queued in that matches the given filter. If for example packets should be put in the queue specified by `QOS_CLASS_25_MINBANDWIDTH` this should be done like this:

```
QOS_FILTER_x_CLASS='25'
```

By `QOS_CLASS_x_DIRECTION` it is set if a class belongs to up- or downstream. If a filter is set then queueing packets to an upstream class only upstream packets will be filtered and queued to the class mentioned. `QOS_CLASS_x_DIRECTION` defines the "direction " of filtering.

As of version 2.1 more than one target class can be set. If for example traffic on port 456 upstream and downstream is our target

```
QOS_FILTER_x_CLASS='4 25'
```

would be specified, if class 4 is the upstream class and 25 is the downstream one. It does not make sense to specifiy more than one up- or downstream class so there never will be more than two target classes.

QOS_FILTER_x_IP_INTERN Set IP addresses and IP ranges from internal networks here which should be filtered. They have to be separated by spaces and can be combined in any manner.

An example:

```
QOS_FILTER_x_IP_INTERN='192.168.6.0/24 192.168.5.7 192.168.5.12'
```

This filters all IP addresses that match 192.168.6.X and in addition IP 192.168.5.7 and 192.168.5.12.

This variable can be empty.

If this variable is used in conjunction with `QOS_FILTER_x_IP_EXTERN` only traffic between IPs or IP ranges defined by `QOS_FILTER_x_IP_INTERN` and `QOS_FILTER_x_IP_EXTERN` will be filtered.

Important: *If filtering by `QOS_FILTER_x_OPTION` for ACK, TOSMD, TOSMT, TOSMR or TOSMC takes place and variable `QOS_CLASS_x_DIRECTION` is of target class 'down' this variable will be ignored.*

QOS_FILTER_x_IP_EXTERN Specify IP addresses and IP ranges from external networks (being connected on QOS_INTERNET_DEV) here which should be filtered. They have to be separated by spaces and can be combined in any manner. This works the same way as QOS_FILTER_x_IP_INTERN.

This variable can be empty.

Important: *If filtering by QOS_FILTER_x_OPTION for ACK, TOSMD, TOSMT, TOSMR or TOSMC takes place and variable QOS_CLASS_x_DIRECTION is of target class 'down' this variable will be ignored.*

QOS_FILTER_x_PORT Ports and port ranges can be set here, separated by spaces and combined in any manner. If this variable is empty traffic on all ports will be limited.

Filtering for a port range from 5000 up to 5099 would look like this:

QOS_FILTER_x_PORT='5000-5099'

Another example: If traffic on ports 20 to 21, 137 to 139 and port 80 should be filtered to the same class this would look like this:

```
QOS_FILTER_x_PORT='20-21 137-139 80'
```

This variable can be empty.

Important:

- If filtering for ports QOS_FILTER_x_PORT_TYPE has to be set as well.
- Port ranges will be ignored if filtering for ACK, TOSMD, TOSMT, TOSMR or TOSMC takes place by the use of QOS_FILTER_x_OPTION.

QOS_FILTER_x_PORT_TYPE This variable is only valid and important in conjunction with QOS_FILTER_x_PORT (it is ignored in other cases).

Ports in client mode are different from those in server mode. Specify here if a port is of type server or client. Use PCs from your own net as a point of reference to decide what to use. Possible settings:

```
QOS_FILTER_x_PORT_TYPE='client'
QOS_FILTER_x_PORT_TYPE='server'
```

As of version 2.1 a combination of those two arguments is also valid to put traffic from the own net as well as traffic from the Internet into the same class on this port.

```
QOS_FILTER_x_PORT_TYPE='client server'
```

This equals to two similar filters once with QOS_FILTER_x_PORT_TYPE set to client and once set to server.

QOS_FILTER_x_OPTION This variable activates additional properties for the filter. Only one of the following arguments can be passed (a combination wouldn't make sense in the same filter). It is perfectly right and makes sense sometimes to set a filter for ACK packets and a second filter for TOSMD packets to move their packets to the same target class (see QOS_FILTER_x_CLASS).

ACK Acknowledgement packets.

A packet matching this option is sent as an acknowledgement to a data packet. If i.e. a huge download is running a lot of data packets will come in and for each of them an acknowledgement has to be sent to confirm it has reached you. If no acknowledgement packets reach the download source it will wait for them before sending the next chunk of data.

This is extremely important with asymetric connections (up- and downstream bandwidths differ) like used in most DSL lines. Those most likely have a small upstream that tends to reach its maximum rather fast. If ACK packets are normally queued this may end in the data server delaying its transaction to wait for ACK packets to come in. This results in download rates lower than they could be.

So ACK packets have to bypass the "normal" packets in order to get to the data server as fast as possible. How to combine this option in a meaningful way with a class is explained in the examples.

ICMP Ping packets (Protocol ICMP)

Ping packets are used to measure time a packet takes to get from A to B. To take influence on this value give ping packets a higher priority. This does not influence ping times for online games.

IGMP IGMP-Pakete (Protokoll IGMP)

If using IP-TV it makes sense to filter and priorize IGMP packets.

TCPSMALL Small TCP Packets

By using this filter outgoing HTTP(s)-Requests can be filtered and priorized. A combination with a destination port is possible and makes sense. Approx. size of this TCP packets is 800 Byte max.

TCP TCP packets (Protocol TCP)

Only packets using protocol TCP are filtered.

UDP UDP packets (Protocol UDP)

Only packets using protocol UDP are filtered.

TOS* Type of Service

An application can set one of four TOS bits for each packet transmitted. This specifies the intended handling for those packets. For example SSH can set TOS-Minimum-Delay for sending in- and output and TOS-Maximum-Troughput for sending files. Linux/Unix programs use these bits more often than Windows programs. A firewall can set TOS bits for certain packets as well. In the end it all depends on routers in the transport chain to honour TOS bits or not. Only TOS bits Minimum-Delay and Maximum-Throughput are really important for fli4l.

TOSMD - TOS Minimum-Delay Used for services that need packets to be transferred without time delays. It is recommended to use this TOS bit for FTP (control data), Telnet and SSH.

TOSMT - TOS Maximum-Troughput Used for services that need big amounts of data to be transferred with high speed. It is recommended to use this TOS bit for FTP data and WWW.

TOSMR - TOS Maximum-Reliability Used to ensure that data reaches its target without being resent. Recommended use for this TOS bit is SNMP and DNS.

TOSMC - TOS Minimum-Cost Used to minimize costs for transferring data. Recommended use for this TOS bit is NNTP and SMTP.

DSCP* Differentiated Services Code Point

DSCP is a marking according to RFC 2474. This process has replaced TOS marking mostly since 1998.

Filters on DSCP-classes can be configured as follows:

```
QOS_FILTER_x_OPTION='DSCPef'
QOS_FILTER_x_OPTION='DSCPcs3'
```

Please note that DSCP is in capital letters while the class is lower case.

The following classes can be used:

af11-af13, af21-af23, af31-af33, af41-af43, cs1-cs7, ef und be (Standard)

1.1.2. Examples

How do we configure `OPT_QoS` in detail? This will be shown below for some use cases:

- Example 1: targets at spreading bandwidth between 3 clients.
- Example 2: targets at spreading bandwidth between 2 clients and on the clients between one port and the rest of the traffic on this client.
- Example 3: targets at learning the general structures of working with QoS.
- Example 4: a configuration for preferring ACK packets in order to keep downstream high even if upstream is overloaded.

Example 1

Spreading bandwidth between 3 clients.

Four classes are created (see `QOS_CLASS_N`) with the following speeds (see `QOS_CLASS_x_MINBANDWIDTH` / `QOS_CLASS_x_MINBANDWIDTH`). They are subclasses of class 0 (see `QOS_CLASS_x_PARENT`) and therefore are bound directly to the interface for “up” res. “down” (see `QOS_CLASS_x_DIRECTION`).

The fourth class is only for visitors and gets lesser bandwidth. By `QOS_INTERNET_DEFAULT_DOWN`=’4’ all traffic not filtered is queued to the forth “guest” class. Because we seldom have visitors and the bandwidth is the same for the other 3 classes clients get 1/3 of the overall bandwidth (256Kibit/s each).

This configuration is only a beginning. It has to be specified how traffic is regulated by the classes.

We use two filters to assign traffic to the classes. We create 3 filters, one for each of the 3 clients (see `QOS_FILTER_N` a.s.o.) and attach a filter to each class (see `QOS_FILTER_x_CLASS`). By specifying `QOS_FILTER_x_IP_INTERN`, `QOS_FILTER_x_IP_INTERN`, `QOS_FILTER_x_PORT`, `QOS_FILTER_x_PORT` and `QOS_FILTER_x_OPTION` it can be defined what rules apply to each class.

Let’s call the interface 0, the 3 classes 1, 2 and 3 and the 3 filters F1, F2 and F3. The scenario looks like this image 1.1.

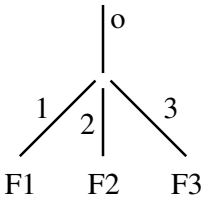

Figure 1.1.: Example 1

Configuration looks like this:
Three client PCs filtered by IP with 1/3 bandwidth each if visitors are absent:

```
OPT_QOS='yes'
QOS_INTERNET_DEV_N='1'
QOS_INTERNET_DEV_1='ppp0'
QOS_INTERNET_BAND_DOWN='768Kibit/s'
QOS_INTERNET_BAND_UP='128Kibit/s'
QOS_INTERNET_DEFAULT_DOWN='4'
QOS_INTERNET_DEFAULT_UP='0'

QOS_CLASS_N='4'

QOS_CLASS_1_PARENT='0'
QOS_CLASS_1_MINBANDWIDTH='232Kibit/s'
QOS_CLASS_1_MAXBANDWIDTH='768Kibit/s'
QOS_CLASS_1_DIRECTION='down'
QOS_CLASS_1_PRIO=''

QOS_CLASS_2_PARENT='0'
QOS_CLASS_2_MINBANDWIDTH='232Kibit/s'
QOS_CLASS_2_MAXBANDWIDTH='768Kibit/s'
QOS_CLASS_2_DIRECTION='down'
QOS_CLASS_2_PRIO=''

QOS_CLASS_3_PARENT='0'
QOS_CLASS_3_MINBANDWIDTH='232Kibit/s'
QOS_CLASS_3_MAXBANDWIDTH='768Kibit/s'
QOS_CLASS_3_DIRECTION='down'
QOS_CLASS_3_PRIO=''

QOS_CLASS_4_PARENT='0'
QOS_CLASS_4_MINBANDWIDTH='72Kibit/s'
QOS_CLASS_4_MAXBANDWIDTH='768Kibit/s'
QOS_CLASS_4_DIRECTION='down'
QOS_CLASS_4_PRIO=''


QOS_FILTER_N='3'

QOS_FILTER_1_CLASS='1'
QOS_FILTER_1_IP_INTERN='192.168.0.2'
QOS_FILTER_1_IP_EXTERN=''
```

```
QOS_FILTER_1_PORT=''
QOS_FILTER_1_PORT_TYPE=''
QOS_FILTER_1_OPTION=''

QOS_FILTER_2_CLASS='2'
QOS_FILTER_2_IP_INTERN='192.168.0.3'
QOS_FILTER_2_IP_EXTERN=''
QOS_FILTER_2_PORT=''
QOS_FILTER_2_PORT_TYPE=''
QOS_FILTER_2_OPTION=''

QOS_FILTER_3_CLASS='3'
QOS_FILTER_3_IP_INTERN='192.168.0.4'
QOS_FILTER_3_IP_EXTERN=''
QOS_FILTER_3_PORT=''
QOS_FILTER_3_PORT_TYPE=''
QOS_FILTER_3_OPTION=''
```

Option `QOS_INTERNET_DEFAULT_UP` is set to 0 because upstream should not be regulated.

Example 2

This example targets at spreading bandwidth between 2 client PCs and then those client bandwidths between one port and the remaining traffic on the client.

We create 2 classes at first with the complete speed for each client and attach them directly to the interface for "up" res. "down" (see example 1). No we create two additional classes for the first client attached to the first class. These classes are created in the same way like the first ones directly attached to the interface except for one difference: `QOS_CLASS_x_PARENT` is not 0 but the number of the parent class it is attached to. If this for example is `QOS_CLASS_1` the classes' `QOS_CLASS_1` has to be set to 1. The same is applied for the second client PC. Attach two subclasses to the class for the second PC. This could be done for an infinite number of PCs if needed. Subclasses of a class can also be created in the amount needed.

This is our skeleton. Now filters have to be defined to assign traffic to the classes (see example 1).

2 filters have to be created for each client. One filter on a port and one for the remaining traffic of the client. Filter sequence is of essential importance here. First filter the port and then the rest. The other way round the filter for the remaining traffic would assign all traffic to its class (including the port traffic).

Let's call the interface 0, the 6 classes 1, 2, 3, 4, 5, and 6 and the 4 filters F1, F2, F3 and F4. The scenario looks like this image 1.1.

Configuration looks like this:

2 classes for 2 PCs getting 1/2 interface bandwidth each with 2 classes for a port getting 2/3 and the rest getting 1/3 of its parent class:

```
OPT_QOS='yes'
QOS_INTERNET_DEV_N='1'
QOS_INTERNET_DEV_1='ppp0'
QOS_INTERNET_BAND_DOWN='768Kibit/s'
QOS_INTERNET_BAND_UP='128Kibit/s'
QOS_INTERNET_DEFAULT_DOWN='7'
```

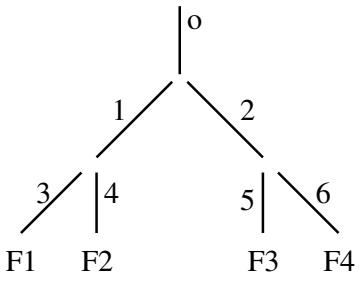

Figure 1.2.: Example 2

```
QOS_INTERNET_DEFAULT_UP='0'

QOS_CLASS_N='6'

QOS_CLASS_1_PARENT='0'
QOS_CLASS_1_MINBANDWIDTH='384Kibit/s'
QOS_CLASS_1_MAXBANDWIDTH='768Kibit/s'
QOS_CLASS_1_DIRECTION='down'
QOS_CLASS_1_PRIO=''

QOS_CLASS_2_PARENT='0'
QOS_CLASS_2_MINBANDWIDTH='384Kibit/s'
QOS_CLASS_2_MAXBANDWIDTH='768Kibit/s'
QOS_CLASS_2_DIRECTION='down'
QOS_CLASS_2_PRIO=''

QOS_CLASS_3_PARENT='1'
QOS_CLASS_3_MINBANDWIDTH='256Kibit/s'
QOS_CLASS_3_MAXBANDWIDTH='768Kibit/s'
QOS_CLASS_3_DIRECTION='down'
QOS_CLASS_3_PRIO=''

QOS_CLASS_4_PARENT='1'
QOS_CLASS_4_MINBANDWIDTH='128Kibit/s'
QOS_CLASS_4_MAXBANDWIDTH='768Kibit/s'
QOS_CLASS_4_DIRECTION='down'
QOS_CLASS_4_PRIO=''

QOS_CLASS_5_PARENT='2'
QOS_CLASS_5_MINBANDWIDTH='256Kibit/s'
QOS_CLASS_5_MAXBANDWIDTH='768Kibit/s'
QOS_CLASS_5_DIRECTION='down'
QOS_CLASS_5_PRIO=''

QOS_CLASS_6_PARENT='2'
QOS_CLASS_6_MINBANDWIDTH='128Kibit/s'
QOS_CLASS_6_MAXBANDWIDTH='768Kibit/s'
QOS_CLASS_6_DIRECTION='down'
QOS_CLASS_6_PRIO=''
```

```
QOS_FILTER_N='4'

QOS_FILTER_1_CLASS='3'
QOS_FILTER_1_IP_INTERN='192.168.0.2'
QOS_FILTER_1_IP_EXTERN=''
QOS_FILTER_1_PORT='80'
QOS_FILTER_1_PORT_TYPE='client'
QOS_FILTER_1_OPTION=''

QOS_FILTER_2_CLASS='4'
QOS_FILTER_2_IP_INTERN='192.168.0.2'
QOS_FILTER_2_IP_EXTERN=''
QOS_FILTER_2_PORT=''
QOS_FILTER_2_PORT_TYPE=''
QOS_FILTER_2_OPTION=''

QOS_FILTER_3_CLASS='5'
QOS_FILTER_3_IP_INTERN='192.168.0.3'
QOS_FILTER_3_IP_EXTERN=''
QOS_FILTER_3_PORT='80'
QOS_FILTER_3_PORT_TYPE='client'
QOS_FILTER_3_OPTION=''

QOS_FILTER_4_CLASS='6'
QOS_FILTER_4_IP_INTERN='192.168.0.3'
QOS_FILTER_4_IP_EXTERN=''
QOS_FILTER_4_PORT=''
QOS_FILTER_4_PORT_TYPE=''
QOS_FILTER_4_OPTION=''
```

Option `QOS_INTERNET_DEFAULT_DOWN` was set in a way that traffic not being assigned to a class by a filter is put in a non-existent class. This is to simplify the example and because it is assumed that there is no unassigned traffic left. Traffic being sent to a non-existent class is forwarded very slow. If a rest of traffic exists always ensure that it is assigned to its own (existing) class.

Option `QOS_INTERNET_DEFAULT_UP` was set to 0 because upstream should not be regulated.

Example 3

An example targeting at learning the general structures of working with QoS.

Picture 1.3 once again shows the layout of example two but this time with an extension. Both subclasses of the second class have two more subclasses attached. You see that it is possible to have nested subclasses. The maximum for nested subclasses is a depth of 8 where 0 is the interface itself leaving 7 more possible subclass levels. “Width” is unlimited though. A subclass can have an unlimited number of classes attached.

The picture shows as well that it is possible to have more than one filter attached to a class as it is done in class 10. Pay attention that at the moment it is not possible to attach a filter in the middle of the “tree” as F8 intended to.

Lets have a closer look at the sense of classes and subclasses. Classes set and control speed of a connection. Spreading of speed is handled described as in `QOS_CLASS_x_MINBANDWIDTH`. This

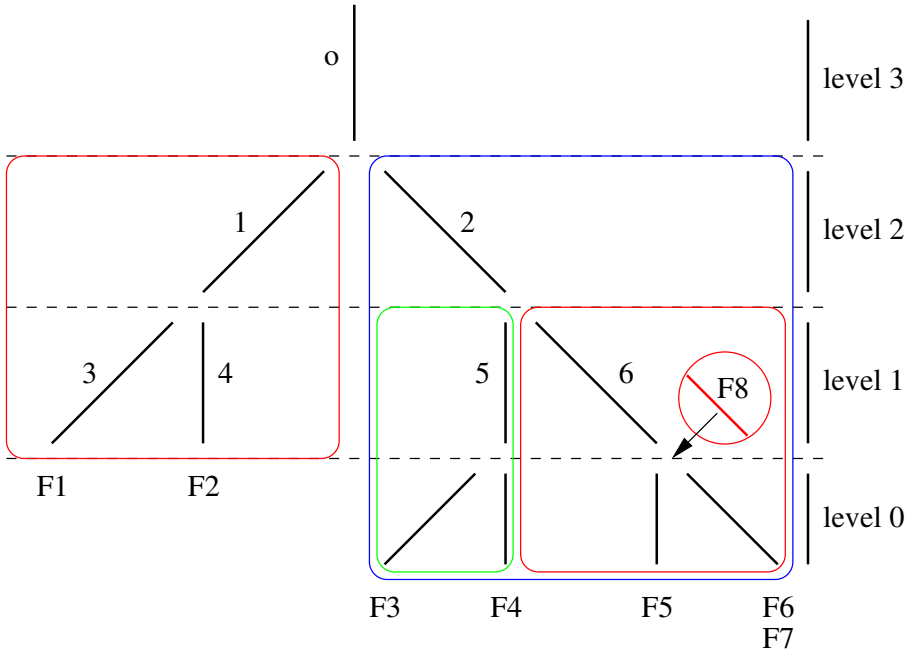

Figure 1.3.: Example 3

can have disadvantages if i.e. you attach all classes to one parent class. If it is for example intended that one client PC should have half of the available bandwidth and the other half is for a second client PC divided in 2/3 http and 1/3 for the rest (2/6 and 1/6 of the whole bandwidth) this would happen: If both clients run at full load both get their half of the bandwidth. If the second one is not transferring http 2/6 of unused bandwidth are distributed not only to the second but to both PCs as decribed above. To avoid this subclasses are created. Traffic of a class is at first distributed to its subclasses. Only if they don't use the complete traffic the rest is spread to the other classes. In the picture the areas that belong together are encircled (red = 1, blue = 2, green = 5 and orange = 6.

Example 4

Configuration for ACK packet priorization in order to keep downstream high if upstream has heavy load:

```
OPT_QOS='yes'
QOS_INTERNET_DEV_N='1'
QOS_INTERNET_DEV_1='ppp0'
QOS_INTERNET_BAND_DOWN='768Kibit/s'
QOS_INTERNET_BAND_UP='128Kibit/s'
QOS_INTERNET_DEFAULT_DOWN='0'
QOS_INTERNET_DEFAULT_UP='2'
```

Configure ppp0 as the Internet device (DSL) and give it the usual up/downstream bandwidth for TDSL (and some other providers). It may be necessary to lower upstream bandwidth for some Kibibit for the trying.

No classes for downstream should be defined:

```
QOS_INTERNET_DEFAULT_DOWN='0'
```

For upstream class number two should be the default class. The network device eth0 is set to 10Mibit/s.

```
QOS_CLASS_N='2'

QOS_CLASS_1_PARENT='0'
QOS_CLASS_1_MINBANDWIDTH='127Kibit/s'
QOS_CLASS_1_MAXBANDWIDTH='128Kibit/s'
QOS_CLASS_1_DIRECTION='up'
QOS_CLASS_1_PRIO=''
```

This is the class for ACK (acknowledgement) packets. ACK packets are rather small and thus need only a minimum bandwidth. Because they should not be affected in any way they get 127Kibit/s. 1Kibit/s is left for the rest.

```
QOS_CLASS_2_PARENT='0'
QOS_CLASS_2_MINBANDWIDTH='1Kibit/s'
QOS_CLASS_2_MAXBANDWIDTH='128Kibit/s'
QOS_CLASS_2_DIRECTION='up'
QOS_CLASS_2_PRIO=''
```

This class is for everything else (except ACK packets). The bandwidth that is left is 1Kibit/s (128-127=1). We don't limit it to 1Kibit/s though, the class is limited by the entry

```
QOS_CLASS_2_MAXBANDWIDTH='128Kibit/s'
```

Because our first class never can use all of its bandwidth there will be something left over which then gets allocated to the second class. If upstream should be divided some more (prominent use case) all other classes have to be subclasses "under" this class. Of course `QOS_INTERNET_DEFAULT_UP` has to be adapted then.

```
QOS_FILTER_N='1'

QOS_FILTER_1_CLASS='1'
QOS_FILTER_1_IP_INTERN=''
QOS_FILTER_1_IP_EXTERN=''
QOS_FILTER_1_PORT=''
QOS_FILTER_1_PORT_TYPE=''
QOS_FILTER_1_OPTION='ACK'
```

This filter filters all packets matching option ACK (i.e. ACK packets). By specifying `QOS_FILTER_1_CLASS='1'` we achieve that all these packets filtered are sent to class 1.

For testing purposes look for one or more good up- and download sources that can produce full load for up- as well as for downstream. Have a look at the traffic display in ImonC. Try this once with and once without QoS.

Downstream should not or not as much decline as without this configuration. It could get even better by declining upstream bandwidth in steps of Kibibits and analyze the effect. I reached my optimum at 121Kibit/s (no declining downstreams anymore). Of course MAXBANDWIDTH- and MINBANDWIDTH- values of all classes have to be adapted accordingly.

A. Appendix For Package QOS

List of Figures

List of Tables

Index