Bandwidth Control

Document revision 1.6 (Wed Dec 08 12:40:17 GMT 2004)
This document applies to MikroTik RouterOS V2.8

Table of Contents

General Information

Summary

Queuing is a mechanism that controls data rate allocation, delay variability, timely delivery, and delivery reliability. The MikroTik RouterOS supports the following queuing mechanisms:

- **PFIFO** - Packets First-In First-Out
- **BFIFO** - Bytes First-In First-Out

- **SFQ** - Stochastic Fair Queuing
- **RED** - Random Early Detection
- **HTB** - Hierarchical Token Bucket
- **PCQ** - Per Connection Queue

The queuing can be used for limiting the data rate for certain IP addresses, protocols or ports. The queuing is performed for packets leaving the router through a real interface. It means that the queues should always be configured on the outgoing interface regarding the traffic flow. There are two additional virtual interfaces in queue tree which are used to limit all the traffic coming to (**global-in**) or leaving (**global-out**) the router regardless of physical interface.

Specifications

Packages required: ***system***
License required: ***level1 (limited to 1 queue), level3***
Home menu level: ***/queue***
Standards and Technologies: ***None***
Hardware usage: *significant*

Related Documents

- ***Package Management***
- ***IP Addresses and ARP***
- ***Firewall Filters***

Description

Clasless Queues

There are four types of simple queues implemented in RouterOS: PFIFO, BFIFO, SFQ and RED. With Bytes First-In First-Out (BFIFO) and Packets First-In First-Out (PFIFO) packets are served in the same order as they are received. The only difference between BFIFO and PFIFO is that PFIFO has a length measured in packets, BFIFO in bytes. Generally, you do not want to use BFIFO or PFIFO as traffic shapers. It's better to use them just for statistics as they are pretty fast. The only exception is when you are running out of resources with RED and/or with complicated queue tree.

Stochastic Fair Queuing (SFQ) cannot limit traffic at all. Its main idea is to equalize sessions (not computer traffic, but session traffic, it is sometimes mentioned as SFQ drawback) when your link is completely full. It works in round-robin fashion, giving each session a chance to send **sfq-allot** bytes. Its algorithm can distinguish only 1024 sessions, and that is why several sessions can be treated as one. Each **sfq-perturb** seconds it drops internal table mixing all the connections and creates a new table. As it is very fast, you may want to use it as a child queue.

To address the imperfectness of SFQ, PCQ was created. PCQ is an advanced SFQ implementation without its stochastic nature - it is more precise, lets you choose classifiers and put a limit (it is called pcq-rate) on each subqueue it has (limits are applied on each subqueue simultaneously, you can not make different limits on particular subqueues). It degrades into a precise version of SFQ if

you put no limit and choose all classifiers at once. PCQ type is used for limiting data rate for each connection. These connections can be classified by the **pcq-classifier**:

- **src-address** - source address
- **dst-address** - destination address
- **src-port** - source port
- **dst-port** - destination port

You can use multiple parameters in the **pcq-classifier**. The **pcq-limit** is number of packets which can hold a single PCQ queue. Data rate for each connection is limited by the **pcq-rate** parameter(in bytes per second).

The combination of src-address, src-port, dst-address, dst-port (also known as tuple) uniquely identifies a connection (i.e., there can not be two different connections with the same tuple).

Note: for using PCQ you have to use queue tree.

Note: to equalize not each particular user (by IP address), but each particular connection, specify all pcq-classifiers at once.

The normal behavior of queues is called tail-drop. Tail-drop works by queuing up to a certain amount, then dropping all traffic that 'spills over'. Random Early Detection (RED is also known as Random Early Drop because it actually works that way) statistically drops packets from flows before it reaches its hard limit. This causes a congested backbone link to slow more gracefully. It starts dropping packets when threshold reaches **red-min-threshold** mark randomly with increasing probability as threshold rising. Maximum probability is used when traffic reaches **red-max-threshold** mark. Then packets are simply thrown away. burst parameter is the number of packets allowed to burst through the interface when the link is empty (generally value of (**red-min-threshold+red-min-threshold+red-max-threshold**)/3 works fine). The minimum value that can be used here is equal to the value of **red-min-threshold**.

Classful Queues

Classful queues are very useful if you have different kinds of traffic which should have different treatment. Generally, we can set only one queue on the interface, but in RouterOS even simple queues (known as classless queues) are attached to the main (attached to the root, which represent real interface) Hierarchical Token Bucket (HTB) and thus have some properties derived from that parent queue. With classful queues it is possible to deploy hierarchical queue trees. For example, we can set a maximum data rate for a workgroup and then distribute that amount of traffic between the members of that group as we can do with simple queues attached to the main HTB, but with upper limit.

Each queue represents a virtual interface with the allowed data rate. It can be borrowed from sibling queues (queues that are children of one queue) when **max-limit** is greater than **limit-at**. If so, the queue would use over the allocated data rate whenever possible. Only when other queues are getting too long and a connection is not to be satisfied, then the borrowing queues would be limited at their allocated data rate.

When a parent is allowed to send some amount of traffic, it asks its inner queues in order of **priority** (priorities are processed one after another, from 1 to 8, where **1** means the highest priority). When a queue reaches its **limit-at** value, its priority is not to be taken in account, such a queue will

be less-prioritative than the ones not reached this limit.

Information Rates and Contention Ratios

Quality of Service (QoS) means that router should prioritize and shape network traffic. QoS is not so much about limiting, it is more about providing quality. The main terms used to describe the level of QoS for network applications are:

- **CIR (Committed Information Rate)** - the guaranteed data rate. It means that traffic not exceeding this rate should always be delivered
- **MIR (Maximal Information Rate)** - the maximal data rate router will provide
- **Contention Ratio** - the ratio to which the defined data rate is shared between users (i.e., data rate is allocated to a number of subscribers). For example, the contention ratio of 1:4 means that the allocated data rate may be shared between no more than 4 users
- **Priority** - the order of importance in what traffic will be processed. You can give priority to some traffic in order it to be handeled before some other traffic.

MikroTik RouterOS may be used to provide CIR and MIR with some contention level and priority. Here we will talk in terms of queues (which represent either real or virtual interface) and classes (children of a queue; each class has an another queue attached to it):

- **limit-at** property is used to specify CIR. If the queue will be able to provide that data rate, it will (i.e, the parent queue (and the link the router is connected to) should be able to provide the total data rate equal or greater that the sum of all CIRs the queue should satisfy in order to quarantee these CIRs). CIRs will be satisfied in order of their **priority**.
- **max-limit** property is used to specify MIR. If the queue has satisfied all the CIRs and it is able to provide some additional data rate, it will try to distribute that additional data rate between all its classes regardless of their priorities and not exceeding their MIRs.
- Filters in RouterOS are very powerful and flexible. Providing Contention Ratio is only one application of what they can do. Using firewall mangle you can mark some a number of hosts with a flow-mark, so the data rate allocated for that mark will be shared between these hosts.

Virtual Interfaces

In addition to real interfaces, there are two vitrual interfaces you can attach tree queues to:

- **global-out** - represents all the output interfaces in general. Queues attached to it applies before the ones attached to a specific interface.
- **global-in** - represents all the input interfaces in general (INGRESS queue). Please **note** that queues attached to **global-in** applies to incomming traffic, not outgoing. **global-in** queueing is taking place just after mangle and **dst-nat**.

Queue burst

A queue burst is a way to 'overcome' the queue limit for a certain amount of time and packets. A queue with burst allows peaks of data rate up to **burst-limit** value, but if average data rate is higher than **burst-threshold** for **burst-time** (in seconds) time, the queue is collapsed to the **limit-at** value. The queue size is expanded back to **burst-limit** value when average data rate becomes lesser than

burst-threshold.

This type of behaviour can be extremely useful for prioritizing small rapid packet sequences like these coming from http www sessions.

For queues that limit traffic flow in both directions, **total-burst-time**, **total-burst-limit** and **total-burst-treshold** properties can be used to apply bidirectional bursts.

Additional Documents

- ***Home of Hierarchical Token Bucket (HTB)***
- ***Paper on Random Early Detection (RED)***
- ***More complete information on Traffic Cotrol***

Queue Types

Home menu level: ***/queue type***

Description

The queue types are used to specify some common argument values for queues. There are four default built-in queue types: **default**, **ethernet-default**, **wireless-default** and **synchronous-default**. The built-in queue types cannot be removed.

Property Description

bfifo-limit (*integer*; default: **15000**) - BFIFO queue limit. Maximum byte count that queue can hold

kind (*pfifo* | *bfifo* | *red* | *sfq* | *pcq*; default: **pfifo**) - kind of the queuing algorithm used:

- **pfifo** - Packets First-In First-Out
- **bfifo** - Bytes First-In First-Out
- **red** - Random Early Detection
- **sfq** - Stohastic Fair Queuing
- **pcq** - Per Connection Queuing

name (*name*) - name for the queue type

pcq-classifier (*multiple choice: dst-address, dst-port, src-address, src-port*; default: **""**) - the classifier of grouping traffic flow

pcq-limit (*integer*; default: **50**) - how many packets to hold in a PCQ

pcq-rate (*integer*; default: **0**) - maximal data rate (in bits per second) assigned to one group

- **0** - do not limit data rate

pfifo-limit (*integer*; default: **10**) - PFIFO queue limit. Maximum packet count that queue can hold

red-burst (*integer*; default: **20**) - RED burst

red-limit (*integer*; default: **60**) - RED queue limit

red-max-threshold (*integer*; default: **50**) - RED maximum threshold

red-min-threshold (*integer*; default: **10**) - RED minimum threshold

sfq-allot (*integer*; default: **1514**) - amount of data in bytes that can be sent in one round-robin round

sfq-perturb (*integer*; default: **5**) - how often to change hash function

Notes

For small limitations (64kbps, 128kbps) RED is more preferable. For larger speeds PFIFO will be as good as RED. RED consumes much more memory and CPU than PFIFO & BFIFO.

Example

To add **red** queue type with minimum threshold of **0**, without any burst and named **CUSTOMER-def**:

```
[admin@MikroTik] queue type> add name=CUSTOMER-def kind=red \
\... red-min-threshold=0 red-burst=0
[admin@MikroTik] queue type> print
  0 name="default" kind=pfifo bfifo-limit=15000 pfifo-limit=50 red-limit=60
    red-min-threshold=10 red-max-threshold=50 red-burst=20 sfq-perturb=5
    sfq-allot=1514 pcq-rate=0 pcq-limit=50 pcq-classifier=""

  1 name="ethernet-default" kind=pfifo bfifo-limit=15000 pfifo-limit=50
    red-limit=60 red-min-threshold=10 red-max-threshold=50 red-burst=20
    sfq-perturb=5 sfq-allot=1514 pcq-rate=0 pcq-limit=50 pcq-classifier=""

  2 name="wireless-default" kind=sfq bfifo-limit=15000 pfifo-limit=50
    red-limit=60 red-min-threshold=10 red-max-threshold=50 red-burst=20
    sfq-perturb=5 sfq-allot=1514 pcq-rate=0 pcq-limit=50 pcq-classifier=""

  3 name="synchronous-default" kind=red bfifo-limit=15000 pfifo-limit=50
    red-limit=60 red-min-threshold=10 red-max-threshold=50 red-burst=20
    sfq-perturb=5 sfq-allot=1514 pcq-rate=0 pcq-limit=50 pcq-classifier=""

  4 name="CUSTOMER-def" kind=red bfifo-limit=15000 pfifo-limit=50
    red-limit=60 red-min-threshold=0 red-max-threshold=50 red-burst=0
    sfq-perturb=5 sfq-allot=1514 pcq-rate=0 pcq-limit=50 pcq-classifier=""

[admin@MikroTik] queue type>
```

Interface Default Queues

Home menu level: ***/queue interface***

Property Description

interface (*name*) - interface name

queue (*name*; default: **default**) - default queue for the interface

Example

To change the default queue type to **wireless-default** for the **wlan1** interface:

```
[admin@MikroTik] queue interface> print
  # INTERFACE                          QUEUE
  0 ether1                             default
  1 wlan1                             default
[admin@MikroTik] queue interface> set wlan1 queue=wireless-default
[admin@MikroTik] queue interface> print
  # INTERFACE                          QUEUE
  0 ether1                             default
  1 wlan1                             wireless-default
```

```
[admin@MikroTik] queue interface>
```

Configuring Simple Queues

Home menu level: ***/queue simple***

Description

Simple queues can be used to set up data rate management for the whole traffic leaving an interface or for certain target (source) and/or destination addresses. For more sophisticated queue setup use the queue trees described further on.

Property Description

burst-limit (*text*; default: **0/0**) - maximal allowed burst of data rate in form of in/out

burst-threshold (*text*; default: **0/0**) - average burst threshold in form of in/out

burst-time (*text*; default: **0/0**) - burst time in form of in/out

dst-address (*IP address/mask*) - destination IP address

interface (*name*) - outgoing interface of the traffic flow

limit-at (*text*; default: **0/0**) - allocated stream data rate (bits/s) in form of in/out, where in is the flow that matches the rule precisely, and out is the flow that matches the reverse rule (i.e. going from the specified interface with source and destination addresses swapped)

max-limit (*text*; default: **0/0**) - maximal stream data rate (bits/s) in form of in/out, where in is the flow that matches the rule precisely, and out is the flow that matches the reverse rule (i.e. going from the specified interface with source and destination addresses swapped)

name (*name*; default: **queue1**) - name of the queue

priority - flow priority, 1 is the highest priority

queue (*name*; default: **default**) - queue type. If you specify the queue type other than default, then it overrides the default queue type set for the interface under /queue interface

target-address (*IP address/mask*) - limitation target IP address (source address)

total-burst-limit (*text*; default: **0**) - maximal allowed total (bidirectional) burst of data rate (bits/s)

total-burst-threshold (*text*; default: **0**) - Total (bidirectional) average burst threshold (bits/s)

total-burst-time (*text*; default: **0**) - total (bidirectional) burst time

total-limit-at (*integer*; default: **0**) - allocated total (bidirectional) stream data rate (bits/s)

total-max-limit (*integer*; default: **0**) - maximal total (bidirectional) stream data rate (bits/s)

Notes

max-limit must be equal or greater than **limit-at**.

Queue rules are processed in the order they appear in the list. If some packet matches the queue rule, then the queuing mechanism specified in that rule is applied to it, and no more rules are processed for that packet.

The value **0** means that these settings will be ignored.

Example

To add a simple queue that will limit download traffic from **192.168.0.0/24** network to **128000** bits per second, and upload traffic to **192.168.0.0/24** network to **64000** bits per second on **ether1** interface:

```
[admin@MikroTik] queue simple> add dst-address=192.168.0.0/24 interface=ether1\
\... max-limit=64000/128000
[admin@MikroTik] queue simple> print
Flags: X - disabled, I - invalid, D - dynamic
 0    name="queue1" target-address=0.0.0.0/0 dst-address=192.168.0.0/24
      interface=ether1 queue=default priority=8 limit-at=0/0
      max-limit=64000/128000
[admin@MikroTik] queue simple>
```

Configuring Queue Trees

Home menu level: ***/queue tree***

Description

The queue trees should be used when you want to use sophisticated data rate allocation based on protocols, ports, groups of IP addresses, etc.

Property Description

burst-limit (*text*; default: **0**) - maximal allowed burst of data rate

burst-threshold (*text*; default: **0**) - average burst threshold

burst-time (*text*; default: **0**) - for how long the burst is allowed

flow (*name*; default: **""**) - flow mark of the packets to be queued. Flow marks can be assigned to the packets under '/ip firewall mangle' when the packets enter the router through the incoming interface

limit-at (*integer*; default: **0**) - maximum stream data rate (bits/s)

max-limit (*integer*; default: **0**) - maximum stream data rate (bits/s)

name (*name*; default: **queueN**) - descriptive name for the queue

parent (*name*) - name of the parent queue. The top-level parents are the available interfaces (actually, main HTB). Lower level parents can be other queues

- **global-in** - match all incomming traffic
- **global-out** - match all outgoing traffic

priority - flow priority, 1 is the highest

queue (*name*; default: **default**) - queue type

Notes

max-limit must be equal or greater than **limit-at**.

To apply queues on flows, the mangle feature should be used first to mark incomming packets.

The router tries to apply queue trees before simple queues.

Example

To mark all the traffic going from web-servers (**TCP** port **80**) with **abc-http** mark:

```
[admin@MikroTik] ip firewall mangle> add action=passthrough mark-flow=abc-http \
\... protocol=tcp target-port=80
[admin@MikroTik] ip firewall mangle> print
Flags: X - disabled, I - invalid, D - dynamic
 0   target-address=:80 protocol=tcp action=passthrough mark-flow=abc-http
[admin@MikroTik] ip firewall mangle>
```

You can add queue using the **/queue tree add** command:

```
[admin@MikroTik] queue tree> add name=HTTP parent=ether1 flow=abc-http \
max-limit=128000
[admin@MikroTik] queue tree> print
Flags: X - disabled, I - invalid, D - dynamic
 0    name="HTTP" parent=ether1 flow=abc-http limit-at=0 queue=default
      priority=8 max-limit=128000 burst-limit=0 burst-threshold=0
      burst-time=0
[admin@MikroTik] queue tree>
```

Troubleshooting

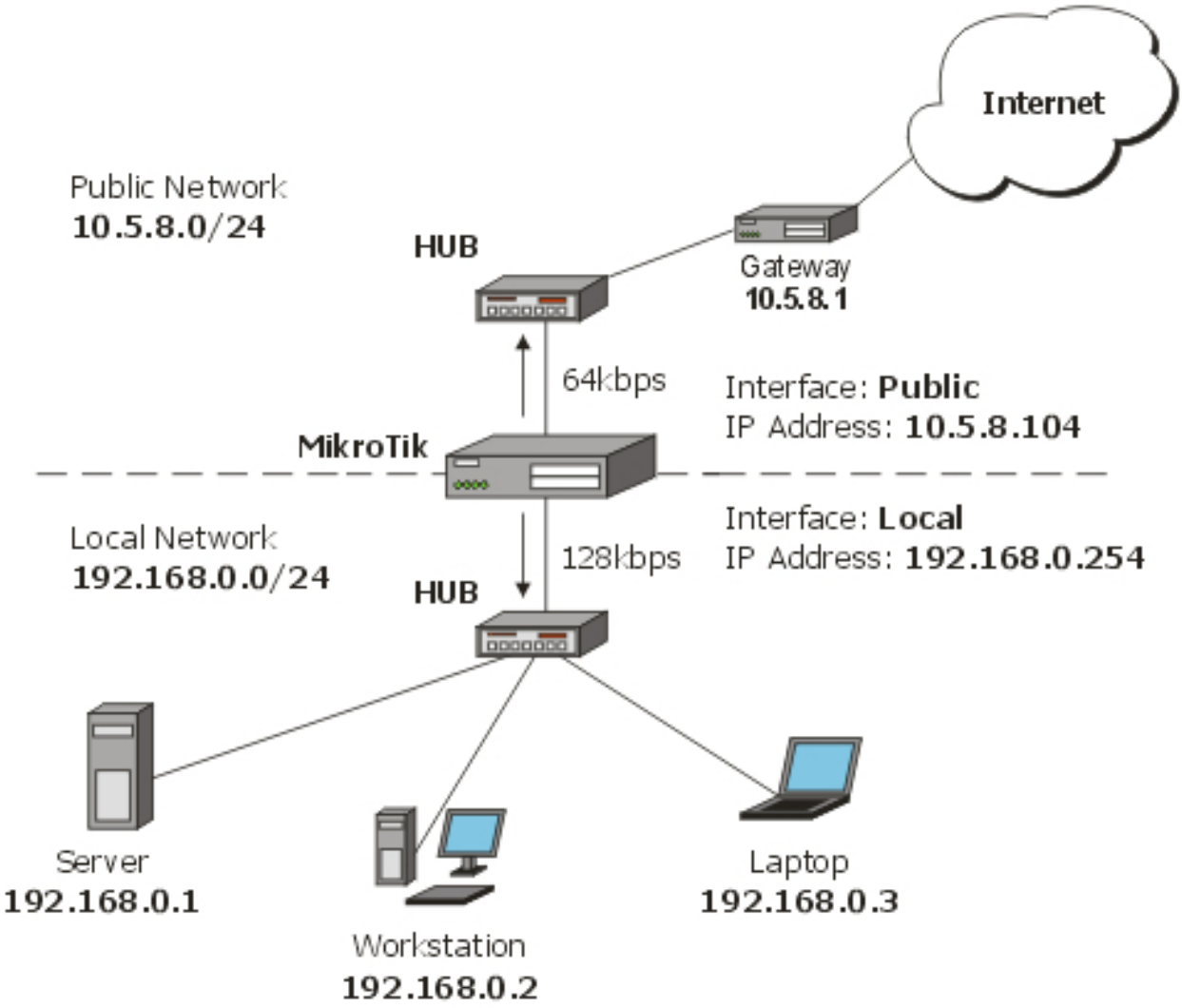
Description

- **The queue is not added for the correct interface**
  Add the queue to the interface through which the traffic is leaving the router. Queuing works only for packets leaving the router!
- **The source/destination addresses of the packets do not match the values specified in the queue setting**
  Make sure the source and destination addresses, as well as network masks are specified correctly! The most common mistake is wrong address/netmask, e.g., 10.0.0.217/24 (wrong), 10.0.0.217/32 (right), or 10.0.0.0/24 (right)
- **The priority setting does not work!**
  In order to take the priority setting in account, you have to specify **limit-at** parameter. Otherwise This setting will be ignored or will not work correctly

Application Examples

Example of Emulating a 128kbps/64kbps Line

Assume we want to emulate a 128k download and 64k upload line connecting IP network 192.168.0.0/24. The network is served through the Local interface of customer's router. The basic network setup is in the following diagram:



IP addresses on MikroTik:

```
[admin@MikroTik] ip address> print
Flags: X - disabled, I - invalid, D - dynamic
 #   ADDRESS            NETWORK         BROADCAST       INTERFACE
 0   192.168.0.254/24   192.168.0.0     192.168.0.255   Local
 1   10.5.8.104/24      10.5.8.0        10.5.8.255      Public
[admin@MikroTik] ip address>
```

And routes:

```
[admin@MikroTik] ip route> print
Flags: X - disabled, I - invalid, D - dynamic, J - rejected,
C - connect, S - static, r - rip, o - ospf, b - bgp
 #    DST-ADDRESS        G GATEWAY         DISTANCE INTERFACE
 0  S 0.0.0.0/0          r 10.5.8.1        1        Public
 1 DC 192.168.0.0/24     r 0.0.0.0         0        Local
 2 DC 10.5.8.0/24        r 0.0.0.0         0        Public
[admin@MikroTik] ip route>
```

Add a simple queue rule which will limit download traffic to 128kbps and upload traffic to 64kbps for clients on local network (**192.168.0.0/24**):

```
/queue simple add name=Limit-Local target-address=192.168.0.0/24 \
interface=Local max-limit=65536/131072
```

```
[admin@MikroTik] queue simple> print
Flags: X - disabled, I - invalid, D - dynamic
 0    name="Limit-Local" target-address=192.168.0.0/24 dst-address=0.0.0.0/0
      interface=Local queue=default priority=8 limit-at=0/0
      max-limit=65536/131072
```

The **max-limit** parameter defines maximum allowed bandwidth in form of upload/download (for clients, connected to interface **Local**). **target-address** is an additional matcher that specifies our local network. If you will not specify **target-address** and will attach a new network to interface **Local** it will also be limited.

You can also monitor the traffic flow through an interface while doing file transfer, using the **/interface monitor-traffic** command:

```
[admin@MikroTik] interface> monitor-traffic Local
    received-packets-per-second: 7
       received-bits-per-second: 68kbps
        sent-packets-per-second: 13
           sent-bits-per-second: 135kbps
```

If you want to exclude the server from being limited, add a queue for it without limitation (**max-limit=0/0** which means no limitation) and move it to the top:

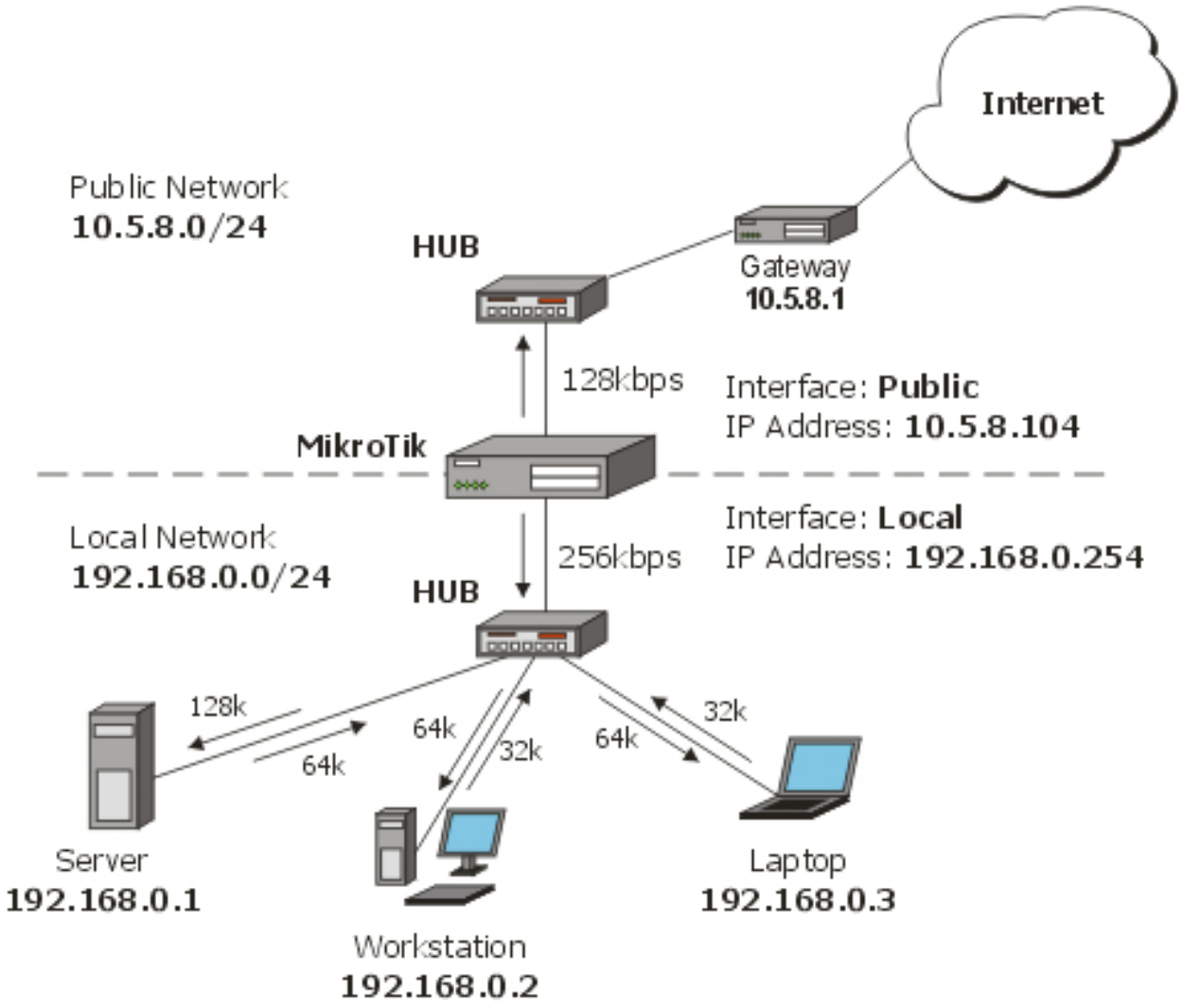
```
/queue simple add name=Exclude-Server interface=Local \
   target-address=192.168.0.1/32
/queue simple move 1 0
[admin@MikroTik] queue simple> print
Flags: X - disabled, I - invalid, D - dynamic
 0    name="Exclude-Server" target-address=192.168.0.1/32 dst-address=0.0.0.0/0
      interface=Local queue=default priority=8 limit-at=0/0 max-limit=0/0

 1    name="Limit-Local" target-address=192.168.0.0/24 dst-address=0.0.0.0/0
      interface=Local queue=default priority=8 limit-at=0/0
      max-limit=65536/131072
[admin@MikroTik] queue simple>
```

Queue tree example with masquerading

In previous example we dedicated 128kbps download and 64kbps upload to local network. In this example we will show you how to guarantee 256kbps download (128kbps for server, 64kbps for Workstation and Laptop) and 128kbps for upload (64kbps for server, 32kbps for workstation and laptop) for local network devices. Additionally, if there is bandwidth that is currently free, share it among users. For example, if we turn off the laptop (or it does not use nework resources), share its 64k download and 32k upload to Server and workstation.

When using masquerading, you have to mark the outgoing connection with **mark-connection** parameter and then mark all packets belonging to this connection with the **mark-flow** parameter.



1. Mark server's download and upload traffic. At first we will mark the outgoing connection and then all packets which belong to this connection.

```
/ip firewall mangle
    add src-address=192.168.0.1/32 action=passthrough mark-connection=server-con
    add connection=server-con action=accept mark-flow=server
```

2. The same for laptop and workstation:

```
/ip firewall mangle
    add src-address=192.168.0.2/32 action=passthrough \
        mark-connection=lap_work-con
    add src-address=192.168.0.3/32 action=passthrouth \
        mark-connection=lap_work-con
    add connection=lap_work-con action=accept mark-flow=lap_work
```

   As you can see, we marked connections that belong for laptop and workstation with the same flow.

3. Now add rules in **/queue tree**. The first rule will limit server's download and the second - upload traffic:

```
/queue tree
   add name=Server-Down parent=Local flow=server limit-at=131072 \
       max-limit=262144
   add name=Server-Up parent=Public flow=server limit-at=65536 \
```

```
        max-limit=131072
```

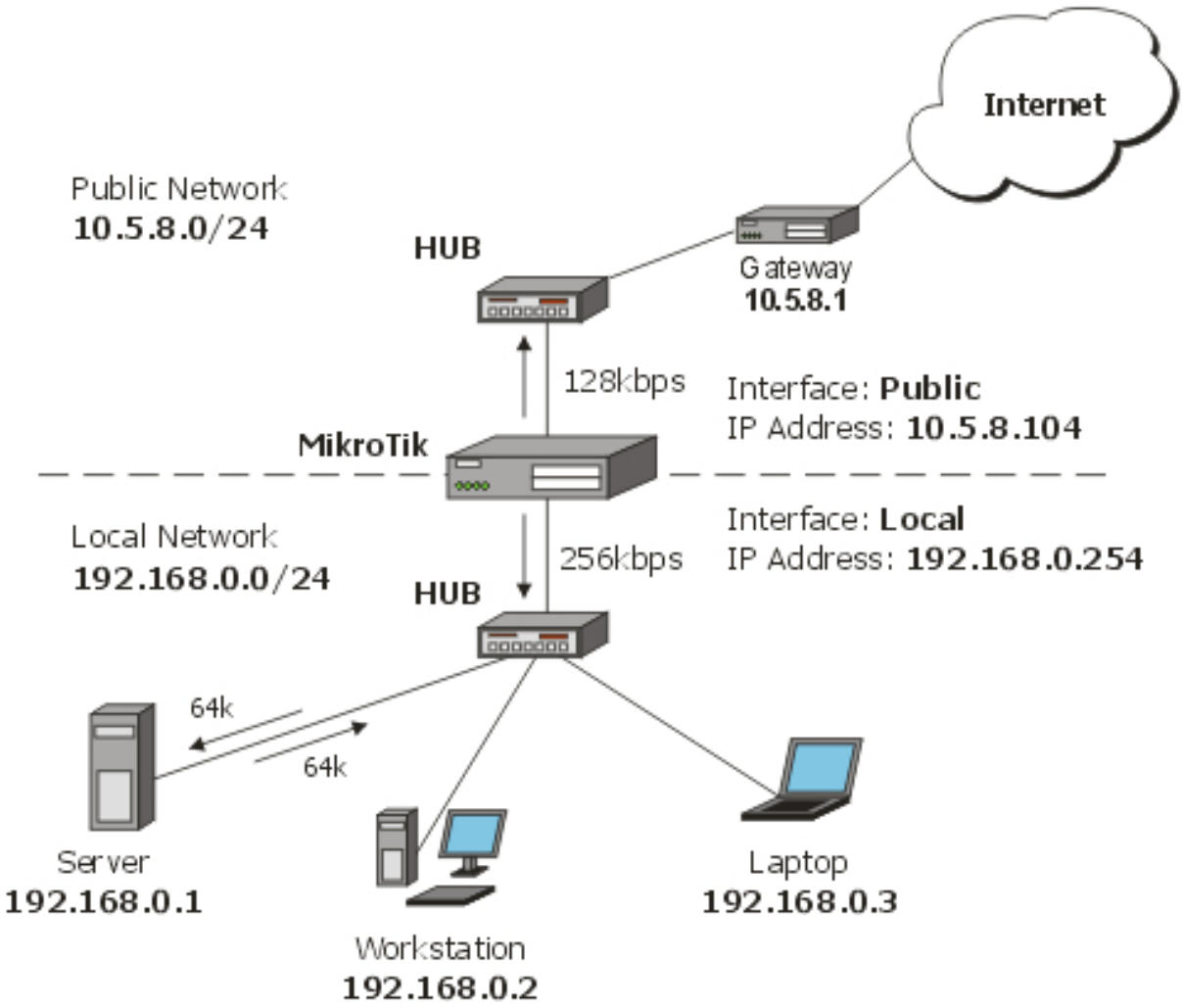
And the same for Laptop and Workstation:

```
/queue tree
    add name=Laptop-WorkStation-Down parent=Local flow=lap_work \
        limit-at=65536 max-limit=262144
    add name=Laptop-WorkStation-Up parent=Public flow=lap_work \
        limit-at=32768 max-limit=131072
```

Example of Guaranteed Quality of Service

This example shows how to limit data rate on a channel and guarantee minimum speed to the FTP server allowing other clients to use the rest of the traffic.

Assume we want to emulate a 256kbps download and 128kbps upload line connecting IP network **192.168.0.0/24** as in the previous examples. But if these speeds are the best that you can get from your Internet connection, you may want to guarantee certain speeds to the **FTP** server (192.168.0.1) so that your customers could download from and upload to this server with the speeds not dependent on the other traffic using the same channel (for example, we will guarantee this server the minimum data rate of 64kbps for each flow direction).



1. Limit the overall download (256k) and upload (128k) traffic:

```
/queue tree
    add parent=Local max-limit=262144 name=Download
    add parent=Public max-limit=131072 name=Upload
```

2. Mark FTP connection, initiated by FTP server (will not work for FTP passive mode!):

```
/ip firewall mangle add src-address=192.168.0.1/32 src-port=20-21 \
    mark-connection=ftp-con protocol=tcp action=passthrough
```

Mark all packets belonging to this connection with a mark **ftp**:

```
/ip firewall mangle add connection=ftp-con mark-flow=FTP_Server action=accept
```

Mark other traffic:

```
/ip firewall mangle add action=accept mark-flow=other
```

3. Add queues for FTP Server download and upload:

```
/queue tree add name=Server_Upload parent=Upload limit-at=65536 \
   flow=FTP_Server max-limit=131072 priority=7
/queue tree add name=Server_Download parent=Download limit-at=32768 \
   flow=FTP_Server max-limit=262144 priority=7
```

Add queues for other's download and upload:

```
/queue tree add name=Other_Upload parent=Upload flow=other
/queue tree add name=Other_Download parent=Download flow=other
```

Now, the FTP traffic destined to and coming from FTP Server will have a guaranteed bandwidth of 64kbps, and a higher priority than other traffic (**priority=7**).

Example of using global-in and global-out queues

Let us consider a situation when you are using a Web-Proxy on your MikroTik router and you want to use bandwidth limitation to/from Internet and allow the maximum speed available if the clients use proxy-data (or do uploads to the router). In this situation you can use **global-in** and **global-out** virtual interfaces. Remember that data from Web-Proxy is sent to clients from Local Process. See ***this*** diagram for a better understanding of packet flow through the router.

1. Assume that you already have configured your web-proxy:

```
[admin@MikroTik] ip web-proxy> print
                 enabled: yes
             src-address: 10.5.8.104
                    port: 8080
                hostname: proxy
       transparent-proxy: no
            parent-proxy: 0.0.0.0:0
    cache-administrator: webmaster
        max-object-size: 4096 kB
            cache-drive: system
          max-cache-size: none
                  status: running
     reserved-for-cache: 100 MB
```

2. Add a mangle rule for marking all packets coming from interface **Public**:

```
/ip firewall mangle add in-interface=Public mark-flow=all-down action=accept
```

Add a mangle rule for marking all packets coming from interface **Local**:

```
/ip firewall mangle add in-interface=Local mark-flow=all-up action=accept
```

3. Add a queue tree rule that will limit all traffic coming from interface **Public** (flow=all-down) to 512kbps:

```
/queue tree add parent=global-in max-limit=524288 flow=all-down
```

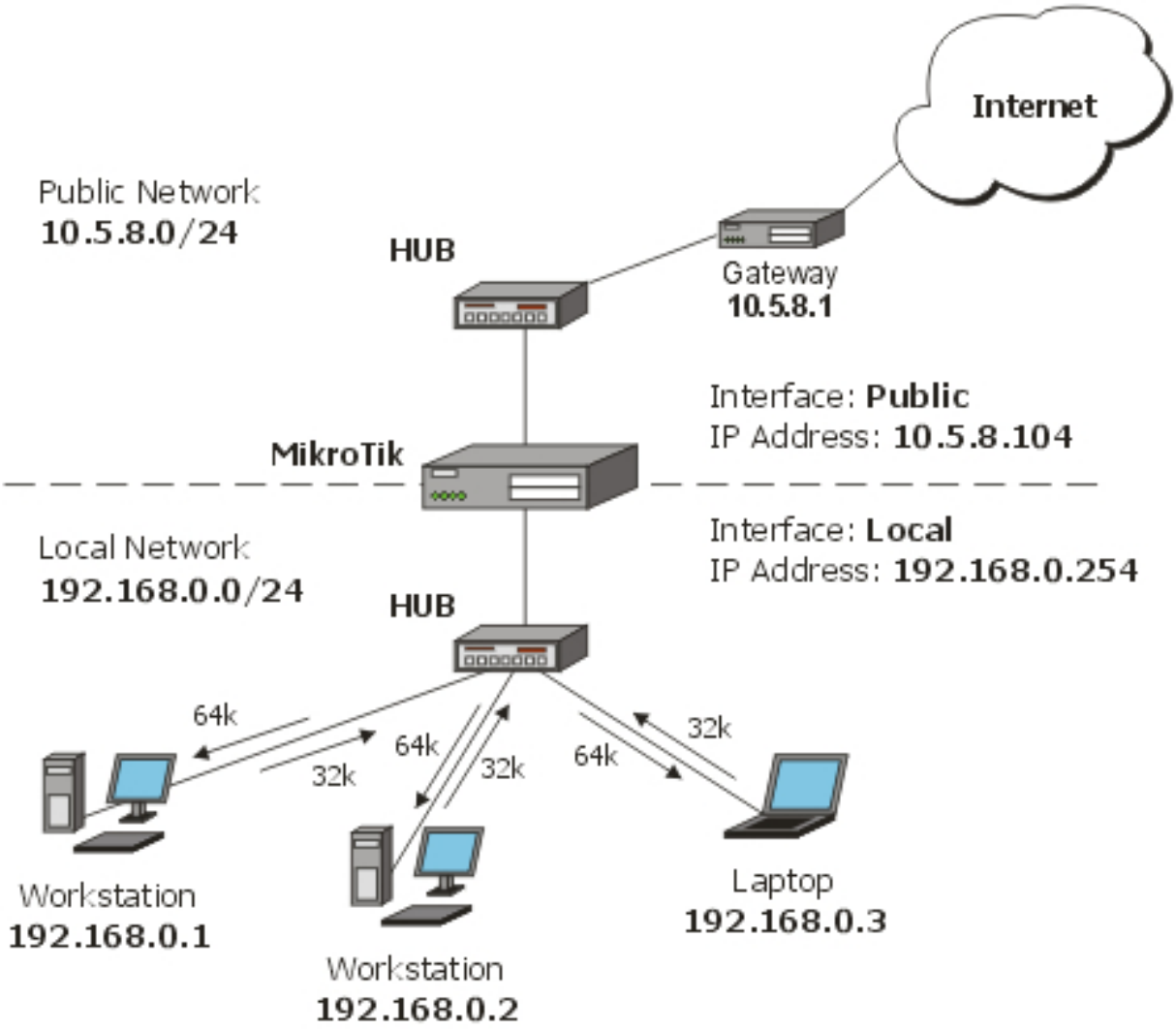
Add a queue tree rule that will limit all traffic coming from interface **Local** (flow=all-up) to 256kbps:

```
/queue tree add parent=global-out max-limit=262144 flow=all-up
```

Now the client downloads from the router (proxy) will be unlimited, but downloads from the Internet will be limited to 512K! The same goes for uploads - no limitation if you are uploading to router, but limit all uploads to Internet to 256K.

PCQ Example

In situations when you want to limit users in your network to a specific bandwidth, you can use PCQ. In this example we will show you how to configure the router so that all users have 64kbps download and 32kbps upload:



1. Mark all packets with flow **all**:

```
/ip firewall mangle add action=accept mark-flow=all
```

2. Create two PCQ queue types - one for download and one for upload. For download traffic queues will be classified by **dst-address** and for upload - by **src-address**:

```
/queue type add name=PCQ-Download kind=pcq pcq-rate=65536 \
   pcq-classifier=dst-address
/queue type add name=PCQ-Upload kind=pcq pcq-rate=32768 \
   pcq-classifier=src-address
```

3. Add two queue rules - one for download and one for upload:

```
/queue tree add parent=Local queue=PCQ-Download flow=all
/queue tree add parent=Public queue=PCQ-Upload flow=all
```