

Supporting Excess Real-time Traffic with Active Drop Queue Software Manual

Yaqing Huang
yaqing@seas.upenn.edu
Dept. of ESE, University of Pennsylvania

Pranav Gupta
guptap@seas.upenn.edu
Dept. of CIS, University of Pennsylvania

I. INTRODUCTION

The requirements of real-time applications mean that they often stand to benefit from network service guarantees, and in particular delay guarantees. However, most of the mechanisms that provide delay guarantees do so by hard-limiting the amount of traffic the application can generate. This can be a significant constraint that conflicts with the operation of many real-time applications. ADQ overcomes such limit. It is designed for the following goals: (1) guarantee a delay bound to a contracted amount of real-time traffic, named “conformed traffic”; (2) transmit within the same delay bound as many excess real-time packets as possible; (3) enforce a given link sharing ratio between excess real-time traffic and other service classes, e. g. , best-effort. ADQ can also preserve the original ordering of the real-time traffic if such is desired, i. e. , all real-time packets will be transmitted, if not dropped, in the same order as they arrived.

ADQ is based on a combination of buffer management and scheduling mechanisms that are capable of guaranteeing delay bounds to a pre-specified amount of traffic, while allowing applications to transmit excess traffic. Two instances of ADQ were implemented in the form of Linux kernel queuing disciplines. One of the queuing discipline included in the package, named *jqueue*, uses SCED+ and SCFQ as scheduling algorithms with buffer management. The other queuing discipline, named *scfque*, differs only in the scheduling algorithm it uses. SCFQ alone is used as scheduling algorithm in *scfque*. For details on the buffer management and scheduling schemes used, see [1]. The queuing disciplines are built as kernel modules for the ease of modifications and future developments.

The software package we provide contains the source code of the two Linux kernel modules, which need to be used with *iproute2* [2] package. The software is developed for Intel-based Linux platform with kernel version 2.4.2. The machine we used is a PIII 1GHz machine with 256M RAM and two Intel 10/100 express NICs. RedHat7.2 operating system is used on the machine. We recommend not using 3com905x series NICs because they have caused problems under heavy load. To use our package requires *iproute2* package stalled on the machine.

This manual documents the installation instructions, parameter configurations and usage of the modules. For details on implementation overview, see [3].

Most of the files referred to in the rest of this document, if not specified, are in */installation/jqueue* if the files are used by module *jqueue*, or in */installation/scfque* if the files are used by module *scfque*, assuming */installation* is your installation directory.

II. INSTALLATION

Our package is available at <http://einstein.seas.upenn.edu/mnlab/software/ADQ.html>. To install our package, you need to do the following three things:

- 1) Rebuild your kernel.
- 2) Install the “iproute2” package.
- 3) Make and Install our package.

Instructions for the first two steps are available at [4]. The detailed instructions are as follows:

- 1) Rebuild your kernel.
 - *For 2.4 kernel*
Support for Diffserv is already integrated into 2.4 kernels. In order to enable it, you may have to reconfigure and rebuild your kernel, or at least some modules. The general procedure for this is described in the Linux Kernel HOWTO [5] <http://www.tldp.org/HOWTO/Kernel-HOWTO.html>.

The following kernel configuration options have to be enabled in the section Networking options:

Kernel/User netlink socket (CONFIG_NETLINK)

Network packet filtering (CONFIG_NETFILTER)

QoS and/or fair queuing (CONFIG_NET_SCHED)

The following kernel configuration options should be enabled in the section Networking options, QoS and/or fair queuing:

CBQ packet scheduler (CONFIG_NET_SCH_CBQ)

The simplest PRIO pseudoscheduler (CONFIG_NET_SCH_PRIO)

RED queue (CONFIG_NET_SCH_RED)

GRED queue (CONFIG_NET_SCH_GRED)

Diffserv field marker (CONFIG_NET_SCH_DSMARK)

Ingress Qdisc (CONFIG_NET_SCH_INGRESS)

QoS support (CONFIG_NET_QOS)

Packet classifier (CONFIG_NET_CLS)

TC index classifier (CONFIG_NET_CLS_TCINDEX)

Firewall based classifier (CONFIG_NET_CLS_FW)

U32 classifier (CONFIG_NET_CLS_U32)

Traffic policing (CONFIG_NET_CLS_POLICE)

In some cases, additional elements may be needed. It is therefore recommended to enable all options in QoS and/or fair queuing.

When the kernel is successfully built and booted, proceed with the instructions in the “iproute2” section below.

- *For 2.2 Kernel*

The Diffserv code for 2.2 kernels is no longer maintained. Several significant bugs have been fixed in the 2.4 kernel. It is therefore strongly recommended to use 2.4 instead of 2.2.

If you insist on using your 2.2 kernel, here is how Diffserv support is added to a 2.2 kernel:

Download the old Diffserv distribution from <ftp://icaftp.epfl.ch/pub/linux/diffserv/dist/ds-8.tar.gz>.

Extract it with

```
tar xzf ds-8.tar.gz
```

Follow the instructions in the file *ds/README* for patching and configuring the kernel.

Instead of doing what is described under “TC installation”, follow the instructions above.

Note: the patch is known to work with the 2.2.14 kernel. It may need manual editing, or may fail completely, with different kernel versions.

2) Install the “iproute2” package

Linux Traffic Control is configured with the utility *tc*. It is part of the iproute2 package. Some Linux distributions already include *tc*, but it may be an old version, without support for Diffserv.

iproute2 is available from many places all over the world, e.g. <http://defiant.coinet.com/iproute2>

To build *tc* with Diffserv support, proceed as follows:

Extract iproute2.

Edit *iproute2/Config* to enable Diffserv support: *TC_CONFIG_DIFFSERV = y*

Edit *iproute2/Makefile* to point *KERNEL_INCLUDE* to your *linux/include* directory, */usr/src/linux-2.4/include* for example.

In *iproute2/*, run *make*

```
cd iproute2
```

```
make
```

The *tc* executable is now in *./tc/tc*.

3) Install our package

Copy *adq.tar.gz* onto your system (into the *installation* directory, for example).

Do

```
tar xzf adq.tar.gz
```

In the directories *installation/jqueue* and *installation/scfque* edit files *Makefile* under both directories so that *IPROUTE* points to your iproute2 installation directory, */iproute2* for example.

```
IPROUTE = /iproute2
```

and point *INCLUDES_KERNEL* to your *linux/include* directory, */usr/src/linux-2.4/include* for example.

```
INCLUDES_KERNEL = -I/usr/src/linux-2.4/include
```

Edit *installation/jqueue/sch_jqueue.c* and *installation/scfque/sch_scfque.c*. specify the path to *kernel.h*, e. g. */usr/src/linux-2.4/include/linux/kernel.h*, so that it is properly included into the two files.

In directory *installation/jqueue* do

```
make
```

```
make install
```

In directory *installation/scfque* do

```
make
```

```
make install
```

You are now ready to use ADQ.

Installation Primers and References can be found at:

- Differentiated Services on Linux: <http://diffserv.sourceforge.net/>
- Linux - Advanced Networking Overview: <http://qos.ittc.ukans.edu/howto/index.html>

III. USAGE

To use these modules requires root privilege, so you need to log in as *root* before you do the following.

A. *jqueue*

To create *jqueue* and add the module into your kernel, do the following:

- 1) First, set *LD_LIBRARY_PATH* to the directory *installation/jqueue*, assuming *installation* is your installation directory.
- 2) Do


```
cd installation/jqueue
```
- 3) Insert the compiled module into kernel by


```
/sbin/insmod jqueue.o
```
- 4) The queuing disciplines should control the output queue of your machine, so they should be associated with the output NIC of your machine. Assume *eth1* is your output NIC, you can create *jqueue* associated with *eth1* with default parameters by using the following command:


```
/sbin/tc qdisc add dev eth1 root jqueue
```

To remove the *jqueue* from the kernel, do the following:

- 1) Delete the *jqueue* associated with *eth1*

```
/sbin/tc qdisc del dev eth1 root jqueue
```
- 2) Remove module


```
/sbin/rmmod jqueue
```

B. *scfque*

The usage of *scfque* is similar to that of *jqueue*.

To create *scfque* and add the module into the kernel, do the following:

- 1) Set *LD_LIBRARY_PATH* to the directory *installation/scfque*, assume *installation* is your installation directory.
- 2) Do


```
cd installation/scfque
```
- 3) Insert the compiled module into kernel by


```
/sbin/insmod scfque.o
```

4) Create *scfque* associated with *eth1* with default parameters by using the following command:

```
/sbin/tc qdisc add dev eth1 root scfque
```

To remove the module from the kernel, do the following:

1) To delete the *scfque* associated with *eth1*, use the following command:

```
/sbin/tc qdisc del dev eth1 root scfque
```

2) Remove module

```
/sbin/rmmod scfque
```

IV. PARAMETER CONFIGURATIONS

In this section, we will explain how to change the configurations of our queuing disciplines, i.e. how to change parameters. We will use *jqueue* as an example. Most of the parameters, if not mentioned specifically, are common to both *jqueue* and *scfque*.

A. Differentiating Packets

Packets are filtered in function *jqueue_classify* in file *sch_jqueue.c* based on TOS fields. Packets with TOS fields equal to 10 are considered conformed packets. Excess packets have TOS fields equal to 8. The default value for TOS field is 0. So packets with TOS fields equal to 0 are considered as best-effort packets. *jqueue_classify* returns 2 if a conformed packet is identified, 1 for excess packets and 0 for best-effort packets.

Four specific non-zero TOS values are commonly used in linux diffserv implementation for applications such as ftp and ssh. Correspond to the four different TOS values, there're four different types of services. They are: minimum delay service with TOS value 0x10, maximum throughput service with TOS value 0x08, maximum reliability service with TOS value 0x04 and minimum cost service with TOS value 0x02. Our conformed traffic TOS matches minimum delay service, and our excess traffic matches maximum throughput service. If desired, user can change the marking by modifying function *jqueue_classify*.

B. Parameters changable by command line

Four parameters of the queuing discipline can be specified by user in command line when creating the queuing disciplines. They are the two token bucket parameters, delay and ordering.

1) *Token Bucket Parameters*: We assume the a token bucket modeled traffic contract for the conformed traffic. You can change it to any other model, but you need to change the way that the conformed queue size is computed so that there'll be enough buffer for the conformed packets to guarantee zero loss. Guaranteed queue size is computed in function *jqueue_change* in file *sch_jqueue.c*.

Token Bucket model has two parameters: token rate and token bucket depth. They are stored in variable *token_rate_* and *delta_*. And their values are assigned in function *jqueue_change* in file *sch_jqueue.c*. The default token rate is 100 K bytes per second and the default token bucket depth is 2000 bytes.

2) *Delay*: The maximum delay that the real-time packets will experience in the buffer is stored in variable *delay_* and assigned its value in function *jqueue_change* in file *sch_jqueue.c*. The default value is 10 ms.

3) *Ordering*: Ordering is the last parameter that can be specified by user in command line when creating the queuing discipline. It is stored in variable *ordering_* and assigned in function *jqueue_change* in file *sch_jqueue.c*. Its default value is 0, i.e., the ordering of real-time packets will not be preserved. The ordering of real-time packets will be preserved if this variable equals 1.

In summary, if you need to create a *jqueue* with maximum delay 5 ms, token rate 300 K bytes per second, token bucket depth 3000 bytes and ordering needs to be preserved, you can use the following command.

```
/sbin/tc qdisc add dev eth0 root jqueue order 1 delay 5ms
rate 300kbps delta 3000
```

If you need to make other parameters changable by command lines, you need to declare of them both in file *tc_jqueue.h* and in *sch_jqueue.c* and add interpretation scripts in file *q_jqueue.c*.

C. constants. h

Other parameters are defined as constants in file *constants. h*.

- **BANDWIDTH_OUT**
This value is the output link bandwidth of the scheduler. The default value is 10 Mbps.
- **EXCESS_RATIO** and **BESTEFFORT_RATIO**
This is the ratio of sharing the residue bandwidth between excess and best-effort traffic. The residue bandwidth is shared between excess and best-effort traffic at a ratio *EXCESS_RATIO* : *BESTEFFORT_RATIO*. The default ratio is 1 : 9, i. e., the excess traffic takes 10% of the residue bandwidth.
- **RQ_BUFFER**
This is the total amount of buffer in bytes allocated to the real-time traffic. The conforming queue size is computed by the module according to the token bucket parameters. And after conforming queue takes the amount of buffer it needed, the excess queue takes the remaining. The default value is 30000 bytes.
- **BQ_BUFFER**
This is the maximum buffer size for best-effort queue in bytes. The default value is 20000 bytes.
- **AVG_SIZE**
This is the average packet size used by the module. The default value is 500 bytes.
- **PEAKRATE**
This is the peak rate of the input link rate. It is used in *scfque* only. The default value is 100 Mbps.
- **MAXPACKET**
This is the maximum size of real-time packets. It is used in *scfque* only. The default value is 1500 bytes.

V. ERROR MESSAGES

Most of errors are due to improper configuration parameters.

1) What is “xyz”?

This error message shows up when user entered some unrecognizable configuration parameters, such as “xyz”, when creating the qdisc. The only recognizable command line configuration parameters are: “rate”, “delta”, “delay” and “order”. After the error message, proper usage and example of creating qdisc will be showed.

2) Illegal “rate”/“delta”/“delay”/“order”.

This error message shows up when user entered some unrecognizable unit for the parameter in the error message. Legal units are defined in file *iproute2/tc/tc_util.c* comes with the *iproute2* package.

The valid units for “rate” are: “kbps”, “mbps”, “mbit”, “kbit” and “bps”. “bps” is the default unit if no unit is specified. The meaning of these units are:

```
1 kbps = 1024 byte per second.
1 mbps = 1024*1024 byte per second.
1 mbit = 1024*1024 bit per second.
1 kbit = 1024 bit per second.
1 bps  = 1 byte per second.
```

The valid units for “delta” are: “kb”, “mb”, “kbit”, “mbit” and “b”. “b” is the default unit if no unit is specified.

The meaning of these units are:

```
1 kb   = 1024 byte.
1 mb   = 1024*1024 byte.
1 kbit = 1024 bit.
1 mbit = 1024*1024 bit.
1 b    = 1 byte.
```

The valid units for “delay” are: “s”, “sec”, “secs”, “ms”, “msec”, “msecs”, “us”, “usec” and “usecs”. “us” is the default unit if no unit is specified. The meaning of these units are:

```
1 s = 1 sec = 1 secs = 1 second = 1000000 usec.
1 ms = 1 msec = 1msecs = 1000 usec.
1 us = 1 usec = 1usec.
```

“order” doesn’t take any unit. Any non-zero value will be interpreted as 1.

- 3) *error: Best-effort queue creation failure.* This error message will show up when creation of best-effort queue as a FIFO qdisc failed. This shouldn't happen normally. If it does happens, check whether you have installed *iproute2* package improperly and whether the QoS option in the kernel is turned on or not.
- 4) *error: Required conformed queue size larger than real-time traffic buffer.*
This error message shows up when the computed conformed queue size needed to guarantee the lossless and require delay bound of the conformed traffic is larger than the total buffer assigned to real-time packets. You need to increase *RQ_BUFFER* in file *constants.h*. See [3] for more detail.
- 5) *error: Required conformed transmission rate faster than output link bandwidth.*
This is error message is for *scfque* only. It shows up when the computed transmission rate needed to serve the conformed traffic, based on the delay requirement and token bucket parameters, is larger than the output link bandwidth, thus such requirement can't be satisfied.
- 6) *error: Excess queue creation failure.*
This error message shows up when the program failed to allocate enough buffer for the excess queue. It happens when there's not enough kernel buffer to allocate to the excess queue.
- 7) *error: Excess queue initialization failure.*
This error message shows up when the init function of excess queue failed. This happens when there's not enough kernel buffer to allocate to some other data structures associated with the excess queue. See [3] for more detail.

VI. REPORTING BUGS

Reporting bugs to yaqing@seas.upenn.edu.

ACKNOWLEDGEMENTS

This work is support in part by NSF grant ANI-9902943 and a grant from Nortel Networks.

REFERENCES

- [1] Y. Huang, R. Guerin, P. Gupta, "Supporting excess real-time traffic with active drop queue," Technical report under submission, Dept. of ESE, Univ. of Pennsylvania, July 2002.
- [2] "iproute2," <http://defiant.coinet.com/iproute2>.
- [3] Y. Huang, P. Gupta, R. Guerin, "Supporting excess real-time traffic with active drop queue: Implementation overview," Technical report, Dept. of ESE, Univ. of Pennsylvania, July 2002.
- [4] "Differentiated Services on Linux," <http://diffserv.sourceforge.net/>.
- [5] "The Linux Kernel HOWTO," <http://www.tldp.org/HOWTO/Kernel-HOWTO.html>.