

# Implementation of Weighted Fair Queuing including RSVP in a BSD kernel

Ian Marsh\*

October 21, 1997

---

\*Supported by Ericsson Telecom AB.

# Contents

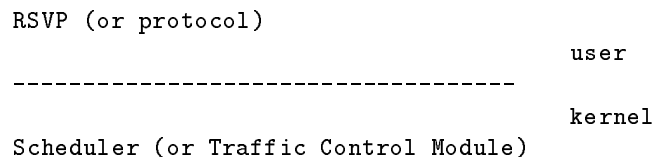
<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Basic Architecture</b>	<b>3</b>
<b>3</b>	<b>Example Session</b>	<b>4</b>
<b>4</b>	<b>Scheduler</b>	<b>4</b>
4.1	Platform . . . . .	4
4.2	Duality . . . . .	5
4.3	User perspective . . . . .	5
4.3.1	Open . . . . .	5
4.3.2	Ioctl . . . . .	5
4.3.3	Read . . . . .	6
<b>5</b>	<b>Kernel perspective</b>	<b>6</b>
5.1	File system Interface . . . . .	6
5.2	Init/Delete . . . . .	7
5.3	Adding Flow Information . . . . .	8
5.4	Deleting Flow Information . . . . .	8
<b>6</b>	<b>Int-serv/RSVP parameter handling</b>	<b>9</b>
6.1	Explanation of Int-serv parameters . . . . .	9
<b>7</b>	<b>Reading data from the kernel</b>	<b>10</b>
<b>8</b>	<b>Stand alone mode</b>	<b>10</b>
<b>9</b>	<b>Weighted Fair Queuing</b>	<b>10</b>
9.1	Theory . . . . .	11
9.2	WFQ Implementation . . . . .	11
9.3	Future work . . . . .	12
<b>10</b>	<b>Practicalities</b>	<b>12</b>
10.1	Installing, compiling and debugging . . . . .	13
10.2	Installation . . . . .	13
10.3	Debugging . . . . .	14
10.3.1	Daemon . . . . .	14
10.3.2	Kernel . . . . .	14
<b>11</b>	<b>Using RSVP</b>	<b>15</b>
11.1	Errors . . . . .	15
<b>12</b>	<b>Internals</b>	<b>15</b>
<b>13</b>	<b>Caveats</b>	<b>16</b>
13.1	Known Bugs . . . . .	16
13.2	Not implemented . . . . .	16

# 1 Introduction

The main goal of this implementation is to make a networking device that can schedule IP traffic on existing network interfaces. Currently scheduling of packets is conducted according to a scheme known as Weighted Fair Queuing [2].

The device understands the RSVP protocol [3], interprets it and uses information from the protocol to set parameters in the WFQ scheduler.

In order to make clear which terms refer to the architecture the following diagram should be used as reference



In this text we use RSVP and protocol interchangeably, likewise so for scheduler and Traffic control module. The boundary between the protocol and the scheduler is also evident in their separation across user and kernel spaces respectively. Interactions between the two modules is discussed later.

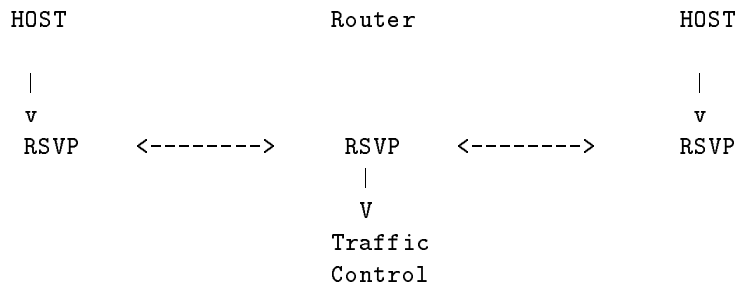
An important secondary goal is to without any modifications to the device driver. Other queuing schedulers [1] have been written but require modifications to the driver or the IP stack in some way.

This report will start with a basic introduction to the system demonstrated with a simple 3 entity system, 2 end systems and a single router. In subsequent sections the internals of the scheduler are explained followed by some details about the protocol. Some examples of how to use the system are given followed by how to extend/change and if need be debug the system.

# 2 Basic Architecture

The basic architecture of the system is as shown in the illustration below, all components run the rsvpd daemon, however only the router is responsible for making requests to the Operating System (OS) for resources. The presence of RSVP daemons on the hosts are to enable the communication with the RSVP on the router. The hosts can also use an application program RTAP supplied with the public domain implementation of RSVP from ISI.

1



---

<sup>1</sup>Currently the router must run rsvpd in order to make reservations on the router

### 3 Example Session

The following is given as to show what an RSVP session can look like driven from a user supplied program (RTAP).

Sender

```
T1> session udp 192.168.15.141
T1: rapi_session => sid= 1, fd= 3
T1> sender 192.168.15.155 [t 11111 22 ]
rapi_sender() OK
-----
T1: Resv Event -- Session= 192.168.15.141:0
192.168.15.155:* [G [111(20) p=1K m=200 M=1.5K] R=22 S=1.11K]
-----
```

Receiver

```
T1> session udp 192.168.15.141
T1: rapi_session => sid= 1, fd= 3
-----
T1: Path Event -- Session= 192.168.15.141:0
      sicsgen:* [T [11.1K(22) p=Inf m=0 M=65.5K]]
                [T [11.1K(22) p=Inf m=0 M=65.5K]] G={0 0 0 0}
-----
T1> reserve ff 192.168.15.155 [g 22 1111 111 20 1000 200 1500 ]
```

The user input commands are shown prefixed by T1, and the response by RTAP by T1:. The sender and receiver agree on a “session identifier” which is normally the receivers address and a port number. Thus both enter the session command. Next the sender advertises it’s traffic spec with the sender command and it’s interface address (optionally a port as well if the sender is sending multiple sessions), The receiver should see this as a Path Event, shown between in the receivers session.

The receiver make a reservation to the sender specified thus:

- command reserve
- style Fixed Filter (FF)
- type of service, “g” guaranteed
- traffic spec

Finally the sender may (depends on the network configuration) see a message as a result of this reserve action. More details on how to drive an RSVP session can be found in man 8 rtap.

The traffic specifications are explained in more detail in the Integrated Services section.

### 4 Scheduler

#### 4.1 Platform

The code will as far as is known, will work on all BSD systems. This includes OpenBSD and NetBSD, little effort should be needed to port it to systems which

have TCP/IP implementations based on BSD systems (e.g. SunOS4). As stated no change has been made to the internal structures of the TCP/IP or the driver so constructing the driver to work with systems that use STREAMS for example should not be difficult.

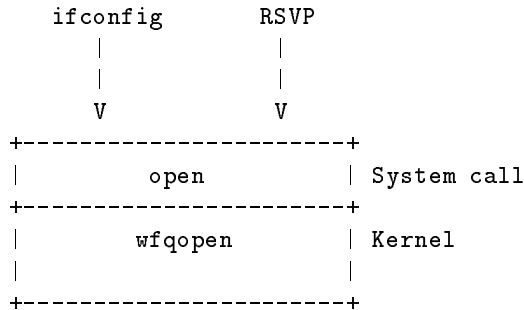
## 4.2 Duality

The objective to use such a device is it looks like a normal character device to the user space programs and a normal networking device to the kernel code. Therefore the device exhibits a “duality” depending on whether the view is from user space or the kernel, each perspective is described in following sections, where developers familiar with devices will be able to comprehend the devices use and likewise for the kernel and driver developers.

## 4.3 User perspective

### 4.3.1 Open

For the user space programmer the scheduler looks like a normal device. There are two ways in which to open/close the device, from the RSVP protocol and manual “ifconfig” command.



In order for the user to open the device a command similar to the one shown below should be issued, how to attach a command like this to RSVP is discussed in the Kernel/RSVP interface section.

```
wfqfd = open('/dev/wfq0', RDWR);
```

A file descriptor is returned by the system as a result of issuing the open call, the path and name of the device should be specified as a string and the flags indicating how the device should be opened. Write is necessary to use the “IO control” (ioctl) system call (man 2 ioctl).

Thereafter the wfqfd file descriptor can be used as with normal file operations, read/write<sup>2</sup>/close. Should the open fail for some reason, then -1 will be returned, this should be checked for and reported to the application program.

We use the file descriptor particularly for ioctls allowing parameters to set in the kernel called from the RSVP protocol, described next.

### 4.3.2 Ioctl

It is intended that most communication will be done via RSVP, however there might be cases where the user would like to control the interface without issuing RSVP

---

<sup>2</sup>Not needed in this implementation

commands, i.e. to reset the interface for example, for this see the Stand alone section.

The following code shows how to call the scheduler to perform a certain function, in this case “WFQ\_ENABLE” a #defined value that can be matched in the kernel when the ioctl is executed. Additionally a pointer to a structure is passed that contains the data to be passed to the scheduler, for example:

```
struct interface {
    char eth_name[IFNAMSIZ];
    u_int eth_len;
}

bzero((void *)interface, IFNAMSIZ);
strcpy(interface.eth_name, 'ep0');
interface.eth_len = strlen('ep0');

if(ioctl(wfqfd, WFQ_ENABLE, &interface) < 0) {
    log(LOG_DEBUG, 'Error in setting up scheduler');
    return TC_ERR;
}
return TC_OK;
```

### 4.3.3 Read

Reading from the device is accomplished through the read system call. Typically a call would look like :

```
bytes_read = read(wfqfd, buf, sizeof(buf));
```

The device reads the requested number of bytes (sizeof(bytes)) and places them in an array of characters, named buf in this example. A possible use is to return the number of bytes output in a flow.

The following section describes how the kernel implements requests from the user for more background see [4].

## 5 Kernel perspective

Described in the following section is an explanation of the code implemented and rationale behind the implementation. 2 flows of control are possible in the kernel, one for handling the protocol and one for the data stream. The protocol is dealt with first in the file system interface description.

The skeleton file tc\_test.c in the RSVP distribution provides the main interface functions from the user space RSVP daemon to the kernel. The function prototypes are found in rsvp\_var.h. If the RSVP daemon is compiled with the SCHEDULE flag the routines in tc\_test.c will be called automatically from the daemon (all the calls are done from the rsvp\_resv.c file).

### 5.1 File system Interface

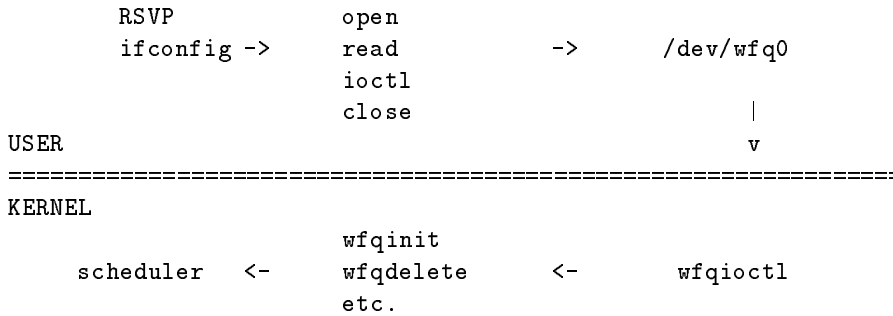
The code is loosely based on a network pseudo device that has a character device interface (cdevsw)<sup>3</sup>. We use the character device interface to pass and obtain in-

---

<sup>3</sup>User space implementations of PPP typically use a similar device device to read and write packets from the kernel

formation to and from the kernel. Packets do not leave the kernel in the flow of data.

The sequence of events from the user to the kernel is shown in the illustration:



The above shows the flow of control from the RSVP daemon or a user “ifconfig” command to the scheduler. Once interpreted by the system call they are, to the kernel, essentially the same.

The dispatcher on the kernel side is a function called wfqioctl. Character device drivers in UNIX are expected to provide an interface to user space programs which implement common UNIX file system commands, read, write, select etc. The ioctl call is used to control a device and manipulate device parameters of the device and it is this system call that interprets the RSVP protocol.

Once a ioctl is called by a user space process the OS will dispatch it to the correct device (via the file descriptor) and hence call the ioctl for the device specified.

## 5.2 Init/Delete

Within the kernel each flag in the ioctl system call is replicated so user specified command call can be matched:

```

      ioctl(wfqfd, WFQ_ENABLE, &interface)      User
      #define WFQ_ENABLE_IOW('t', 94, struct interface)      Kernel

```

Exact details of how this is done is outside the scope of this report. However in the wfqioctl() function each of requests is matched in a switch statement and a corresponding function is called to carry out this request.

Setting the interface up involves the following steps:

- Check to see if the interface is already up
- Substitute the output of the given interface for a scheduling one
- Globally mark the interface as up
- Initialise a Best Effort flow (flow 0)

Similarly setting the interface down, either by “ifconfig down” or by killing the RSVP daemon with the “kill” command will close the device by as follows:

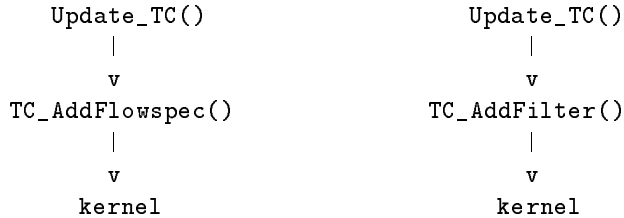
- Check to see if the interface is already down
- Put back the original output handler
- Globally mark the interface as down

- Delete the Best Effort flow (flow 0)

Note: the events above apply only to setting up and down the device. Adding and removing flows of data is the responsibility of the protocol. This process as well as handling the data is described in the next sections.

### 5.3 Adding Flow Information

Referring back to the example session section 2, the reserve command generates 2 messages which result in the following functions being called within the RSVP protocol (arguments not shown). No code in the TC module is called for the session or sender messages.



The TC\_AddFlowspec function passes two structures to the kernel code, a FlowSpec (specified by the reserve command) and the originally specified Sender Traffic spec<sup>4</sup>. The lower of these are used to avoid over-reservations or unnecessary failures taking place.

The Flowspec and Filter specifications are given in the Int-serv spec<sup>5</sup> object numbers 9 and 12 respectively.

The TC\_AddFilter() function is responsible for communicating the the participants to the kernel. The TC\_AddFilter() function first passes the Session to the kernel, so for example, modifications to the FlowSpec can be matched to a Session. Secondly, a FiltSpec is passed to the kernel, which is the sender address. Should extra participants join the session, for example where WF is specified, the router knows becomes aware that new parties have joined.

The Session and Filter specifications are given in the RSVP spec<sup>6</sup> object numbers 1 and 10 respectively.

Filtering based on source address is not used in the current version, however support is implemented.

### 5.4 Deleting Flow Information

As described above deleting a flow, triggered by a “close” command from RTAP, for example, will result in 2 functions being called to delete the flowspec information and the filter spec that was instantiated in the Add calls.

Only a handle is specified by the Update\_TC() function to delete the information needed in the kernel.




---

<sup>4</sup>This is ignored in this implementation, but should be compared to the FlowSpec and the “smaller” of the two passed to the kernel

<sup>5</sup>draft-ietf-intserv-rsvp-use-02.txt

<sup>6</sup>draft-ietf-rsvp-spec-16.txt



Parameters specified from RSVP are stored in statically allocated in the kernel. A parameter (rhandle) is used to index into the table to locate appropriate entries for each session.

## 6 Int-serv/RSVP parameter handling

A document<sup>7</sup> describes these parameters in full detail.

As described the TC module will receive 2 messages with FLOWSPECS in them, one from the sender as a SENDER\_TSPEC (obj class 12) and one from the receiver. In the following the units and meaning as well as some example values.

### 6.1 Explanation of Int-serv parameters

Some words are needed on the units of the parameters. The Sender should advertise what kind rate of traffic they are going to generate (vic 2.8 for example does this) this allows optimal resource reservation in the router, as the receiver could over reserve. The router compares the flowspecs from both the sender and the receiver and selects the lower for the Traffic Control module to reserve.

Tspecs (traffic specs):

- r token bucket rate (bytes/sec)
- b bucket depth (bytes)
- p peak rate (bytes/sec)
- m minimum policed unit (bytes)
- M maximum packet size (bytes)

The r term describes the Rate at which the sender sends, it is normally an “average” rate. Any bursts in the data should be absorbed by b. Both these parameters can be used to set buffers in the scheduler, particularly where multiple senders/receivers share use one Session. These parameters required, if they are not specified the Sender Tspec or Reservation fails. The peak rate can be set as the rate of the fastest rate known between the sender and the receiver, an Ethernet rate could be specified, if unknown (indicated by 0) infinity is assumed. The minimum packet size can also be specified, which may be set as the IP+UDP+RDP packet sizes. If this is not given 0 is taken. The M parameter is maximum packet size, which could, for example be set to the maximum IP packet size 65536 bytes.

RSPEC (desired service):

- R Required rate in (bytes/sec)
- S Slack term (microseconds)

The R in RSpec parameters is the same as the Tspec m parameter and uses the same unit, except it is *user* specified. Normally it should match the Sender’s Tspecs. The R term is essentially the share of a links bandwidth the flow is entitled to.

---

<sup>7</sup> draft-ietf-intserv-rsvp-use-02.txt

The S term, Slack refers to the delay the reserver is willing to incur and is an indicator to the router how much buffer space needs to reserved to support this delay. If not specified, it is taken to be 0. By supplying some Slack it is more likely that a reservation request will be met.

Note: From the R and S terms, p as well as the Senders Tspecs the router is expected to derive buffer requirements. Additionally, it is also the responsibility of each scheduling element in a path to calculate parameters about it's capabilities to pass downstream, known as C and D, this is not done in this implementation. More on this can be found in [5].

Typical examples:

- r 125 Mbytes/sec (1Mbits/sec) CellB/Matrox card
- b 2048 bytes (2k/bytes)
- p 50000 (bytes/sec) Video source
- m 40 IP+UDP+RTP headers no data
- M 1500 Ethernet frame(bytes)
- R 125Mbytes as above (bytes/sec)
- S 10000 (microseconds) 10ms

## 7 Reading data from the kernel

Also implemented is a method to read data from the kernel. As before this is implemented as standard file "reads". Currently the number of bytes per flow is reported on a read request from user space. The frequency of this operation is determined by the timeval structure passed to the select in the user space program. An example is included in the distribution.

## 8 Stand alone mode

Should the user specify parameters to the scheduling device via the shell `ifconfig |iface| |cmd|` command this is possible. In the kernel this is handled by the `wfq-fioctl()` function in the kernel. It is up to the user to specify the `int-serv` parameters correctly when running in stand alone mode.

## 9 Weighted Fair Queuing

Weighted Fair Queuing (WFQ) is a packet scheduling technique allowing guaranteed bandwidth services. The purpose of WFQ is to let several sessions share the same link. WFQ is an approximation of Generalized Processor Sharing (GPS) which, as the name suggest, is a generalization of Processor Sharing (PS) [6]. In PS each session has a separate FIFO queue. At any given time the N active sessions (the ones with non-empty queues) are serviced simultaneously, each at a rate of  $1/N$ th of the link speed. Contrary to PS, GPS allows different sessions to have different service shares. GPS have several nice properties. Since each session has its own queue, an ill-behaved session (who are sending a lot of data) will only punish itself and not other sessions. Further, GPS allows sessions to have different guaranteed bandwidths allocated to them. In [7] Parekh showed that when using a network with GPS switches and a session that is leaky bucket constrained an end-to-end delay bound can be guaranteed.

## 9.1 Theory

GPS is an idealized fluid model not practically realizable. WFQ, first presented in [8], is a packet approximation of GPS. In WFQ a packet at a time is selected and outputted among the active sessions. This works as follows. Each arriving packet is given virtual start and finish times. The virtual start time  $S(k,i)$  and the virtual finish time  $F(k,i)$  of the  $k$ :th packet in session  $i$  are computed as follows:

$$S(k,i) = \max(F(k-1,i), V(a(k,i))) \quad F(k,i) = S(k,i) + L(k,i)/r(i)$$

with  $F(0,i) = 0$ ,  $a(k,i)$  and  $L(k,i)$  are the arrival time and the length of the packet respectively.  $V(t)$  is the virtual time function representing the progression of virtual time in the simulated GPS model and is defined as this:

$$dV(t)/dt = 1/(\text{Sum of active sessions shares at time } t)$$

This means that when there are inactive sessions the virtual time progresses faster. In the corresponding GPS case this can be viewed as that the remaining active sessions get more service.

The packet selected for output is the packet with the smallest virtual finish time. In [7] Parekh describes a Packet GPS (PGPS) algorithm which is identical to the WFQ algorithm described here. He derives several relationships between a fluid GPS system and a corresponding packet WFQ system:

1. finish time of a packet in the WFQ system compared to the GPS system will not be later than at most the transmission time of a maximum sized packet.
2. The number of bits serviced in a session by the WFQ system will not fall behind the GPS system by more than a maximum sized packet.

In [9] an improved algorithm, called Worst-case Fair Weighted Fair Queuing (WF2Q), is presented. In WF2Q only packets who have a virtual start time that has been passed are considered for output. WF2Q is a better approximation of GPS and has lower delay bounds but increases implementation complexity.

Both WFQ and WF2Q must simulate the virtual time which is computationally expensive. In [10] an algorithm, dubbed WF2Q+ with a new virtual time function is described.

$$V(t+T) = \max(V(t) + T, \min(S(h(i,t),i)))$$

where  $h(i,t)$  is the packet number of the first packet in sessions  $i$ 's queue at time  $t$ . Also WF2Q+ can guarantee delay bounds given a session that is constrained with a leaky bucket.

## 9.2 WFQ Implementation

The WFQ implementation primarily consists of three routines: `getSessionIdentity`, `enqueue` and `dequeue`. The data structures are one FIFO queue per session in form of a circular buffer as well as header information for each queue. There is also an sorted array with session identities. The `getSessionIdentity` function performs a binary search in the array.

Only the virtual finish time ( $F$ ) of the first packet in each queue is of interest at any given time. So, instead of storing  $F$  for each packet a  $F$  for the first packet in each queue is stored in the queue header. When a packet is dequeued the corresponding  $F$  is updated accordingly.

Time is increased in steps each time a packet is dequeued. The arrival time of packets are set to the depart time of the packet currently being outputted, which is a deviation from true WFQ. Virtual time is updated every time a packet is dequeued since it is then time is altered and since arrival time of packets are set to these times too. Virtual time progresses at different speeds depending on the amount of active bandwidth, therefore we need to find out at which times sessions become inactive.

The virtual time a session becomes inactive is the  $F$  of the last packet in the session, the session virtual finish time ( $SF$ ). The smallest  $SF$  among active sessions is found and the corresponding time is calculated. If this time is less than the current time then this session is deactivated and the active bandwidth is decreased accordingly. This is done iteratively until the corresponding time is more than the current time. When this happens the session in question is not inactive yet and instead the virtual time that corresponds to the current time is calculated.

```

enqueue(packet, i)
1  if not active(i)
2    activate(i)
3    active_r += r(i)
4  if queue(i) is empty
5    F(i) = SF(i) = max(F(i), V(t)) + L / weight
6  else
7    SF(i) += L / weight
8  put(packet, queue(i))

dequeue()
1  i = min(active queues F(i))
2  packet = get(queue(i))
3  t += L / r
4  if active(i)
5    F(i) += Lnext / r(i)
6  for ever
7    j = min(active queues SF(j))
8    tmp_t = prev_t + (SF(j) - V(t)) * active_r / r
9    if tmp_t > t
10     V(t) += (t - prev_t) * r / active_r
11     prev_t = t
12     return packet
13  prev_t = tmp_t
14  V(t) = SF(j)
15  deactivate(j)
16  active_r -= r(j)

```

### 9.3 Future work

Future improvements may include:

- Better data structures and optimized code like calendar queues and hash tables.
- Better data structure for time representation, right now a floating point value is used.
- An SCFQ (Self Clocked Fair Queuing) implementation. SCFQ is a simple but not so good approximation of GPS.

## 10 Practicalities

The next few subsections explain how to re-build the kernel with WFQ/RSVP support, also explained is how to make changes and some hints on debugging the code.

## 10.1 Installing, compiling and debugging

The following files are needed in addition to the kernel sources:

- if\_wfq.c
- if\_wfq.h
- rsvp.h
- rsvp\_types.h
- rsvp\_intserv.h
- rapi\_lib.h

Also the following user-space programs are needed:

- tc\_test.c
- flow\_read.c

## 10.2 Installation

in sysi386conf there should be a kernel build file, originally called GENERIC, here a pseudo-device should be added with the name wfq:

```
pseudo-device wfq 1
```

Add the number of devices you would like to have running on the machine simultaneously. If you intend to have scheduling on multiple interfaces then add the number you need.

Also the files.i386 needs to include the file:

```
net/if\_wfq.c          optional          wfq          device-driver
```

The header file is not needed, this is found with the make depend command.

Finally a major device number is needed for the character device:

```
81          wfq          Scheduling pseudo device driver
```

The major device should be the same as the one specified in the code itself (if\_wfq.c), in order to build the config file type:

```
config -g
```

this will give you a kernel with symbols for debugging. Change dir to the ../../compile/BUILDNAME. Type “make depend”, this will create a .depend file.

Should you want to debug the kernel code it is recommended that you edit the Makefile and remove the -O flag as this can give some odd behavior on debugging the code. Trivial variables are optimised away making GDB step oddly.

Having the symbols available does not mean that there has to be a large executable (taking time to boot and link) as separate symbol tables can be generated with gdb, described in the GDB manual.

This should generate a kernel file to be installed with “make install”. The modified kernel is ready to be run, sync the system and type “reboot”.

## 10.3 Debugging

As the software runs in both user space and in the kernel different methods of debugging may be needed.

### 10.3.1 Daemon

Debugging a program that is running involves “attaching” to the process. Ensure the daemon has been compiled with the `-g` option and load the symbols with the “file” command.

Attach to the process “attach `pid`” from within `gdb`, this will stop the process, where you can add breakpoints/watchpoints and then continue. Asynchronous events will stop the process and then use `gdb` as normal.

### 10.3.2 Kernel

Some kernel debugging can be done from user space with the “`gdb -k` command” access to the `/kernel` file and `/dev/mem` is needed. Global variables can be examined/changed (ifnet for example).

A kernel debugger `DDB` is also included in the FreeBSD distribution (enable by `DDB` in the config file). This can be invoked by a “hot key” sequence `Ctrl-Alt-Esc` which will give a `ddb` prompt, breakpoints can be set, however no source listing is available.

Finally the kernel code can be examined remotely by `DDB` running on the host being debugged and `GDB` running on a remote host. Attaching the two machines with a serial cable is possible and then issue commands from the remote terminal. The remote machine is then able to both instruct the debugged machine to step whilst it looks up the symbols on the remote machine. Therefore it is necessary to have a copy of the kernel on the remote machine and the same source files (if accurate symbol lookup is required).

Start a debugger session on the debugging machine:

```
gdb -k <KERNEL>
set the target remote /dev/cuaa0
```

On the machine being debugged enter the sequence

```
ctrl-alt-esc      /* break to DDB */
>gdb              /* give control to remote gdb */
>s                /* break into debugger */
```

Now the remote machine has control over the machine being debugged. From the prompt it is possible to use all of `GDB`'s functionality as if it were local, set break points, examine variables etc.

```
> break wfqioctl  /* Make the machine stop here */
> c               /* start it running */
```

using `.gdbinit` to automate multiple breakpoints, commands, display of variable makes life easier:

```
file /home/ianm/sys/compile/SICSRTR/kernel
directory /home/ianm/sys/net
directory /home/ianm/sys/netinet
directory /home/ianm/sys/i386/isa
directory /home/ianm/sys/kern
target remote /dev/cuaa0
```

## 11 Using RSVP

Make sure the rsvpd daemon is running on both end hosts and the router. (It can be added to the rc.local file) These hosts do not need to have the -DSCHEDULE flag.

When the daemon starts it creates 2 files:

- /var/log/rsvpd.log /\* logging pertained to rsvpd \*/
- /var/run/rsvpd.pid /\* it's current PID \*/

The log file contains information output by the daemon itself, if extra debug info is added (for example to the tc\_test.c file) it will appear in the log file.

The rsvpd.pid enables the the daemon to be manipulated in scripts. For example kill 'cat /var/run/rsvpd.pid'. It is also useful for attaching the debugger to the daemon in a .gdbinit script.

It is necessary to have the MROUTED option compiled into the kernel as well as to have a mrouted running to forward RSVP Path messages between different sub-nets, one especially built for RSVP on BSD can be found at ftp:ftp.parc.xerox.compub/net-researchipmulti.

### 11.1 Errors

```
1> session udp 192.168.15.141
connect: Connection refused
Could not connect to rsvp server
RAPI: rsvp\_session() err 9 : RSVP daemon doesn't respond
```

Then the daemon is probably not running, start it with:

```
rsvpd
```

if you get:

```
T1> -----
T1: sid=1  Session= 192.168.15.141:0 -- RSVP error:
                               Sender addr not my interface
      Code=20  Val=14  Node= 0.0.0.0
      ( )      [T [0(0) p=0 m=0 M=0]]
-----
RAPI rapi\_dispatch(): err 13 : (Unused)
```

Then the sender's specification does not match its interface. A further situation could be the sender has switched interfaces, after the RSVP daemon has been started, hence rtap and rsvpd differ.

Also you are not allowed to have both port 0 (wildcard) and port X on the same host. This confuses RSVP as it does not know which thread to deliver a particular message when it's destined for the named port, (error number is 7) Two different ports are allowed.

## 12 Internals

Clock timer (ticks) setting can be determined by sending data as fast as is needed (or full speed). I use ping -f with max. packet size and when the software interrupt

routine is called but there is nothing to output then the routine is being uncalled unnecessarily.

Rationale: Send as fast as needed and still timer being used.

## 13 Caveats

### 13.1 Known Bugs

Deleting a flow - RSVP rhandle sometimes wrong ?

### 13.2 Not implemented

No attempt has been made to calculate the C and D values for our node. Selecting on src address (filtering) WF and SE filters

## References

- [1] *Kenjiro Cho. Alternate Queueing for BSD Unix* From the altq-0.2 distribution. Available from <http://www.csl.sony.co.jp/person/kjc/programs.html>
- [2] *A. Demers, S. Keshav, and S. Shenker, Analysis and Simulation of a Fair Queueing Algorithm.* Proc. SIGCOMM '89, 19(4):1-12, September 1989.
- [3] *L. Zhang, S. Deering, S. Estrin, S. Shenker, and D. Zappala, RSVP: A New Resource ReSerVation Protocol* IEEE Network, September 1993.
- [4] *M. McKusick et. al The Design and Implementation of the 4.4 BSD Operating System.* Addison Wesley 1996.
- [5] *S. Shenker, C. Partridge, R. Guerin. draft-ietf-intserv-guaranteed-svc-08.txt* IETF Draft, 1997. available from <ftp://ds.internic.net>
- [6] *L. Kleinrock. "Queueing Systems, Volume 2: Computer Applications.* Wiley, 1976.
- [7] *A. Parekh. "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks.* PhD dissertation, Massachusetts Institute of Technology, February 1992.
- [8] *A. Demers, S. Keshav, and S. Shenker. "Analysis and simulation of a fair queueing algorithm.* In Journal of Internetworking Research and Experience, pages 3-26, October 1990. Also in Proceedings of ACM SIGCOMM89, pp 3-12.
- [9] *J.C.R. Bennet and H. Zhang. "WF2Q: Worst-case Fair Weighted Fair Queueing.* In IEEE INFOCOM96, San Fransisco, March 1996. <ftp://ftp.cs.cmu.edu/user/hzhang/INFOCOM96.ps.Z>
- [10] *J.C.R. Bennet and H. Zhang. "Hierarchical packet fair queueing algorithms.* In Proceedings of the ACM SIGCOMM96 pages 143-156, Palo Alto, CA, August 1996. <ftp://ftp.cs.cmu.edu/user/hzhang/TON-97-Oct.ps.Z>