TNE30019/TNE80014 – Unix for Telecommunications

Traffic Shaping and Emulating Networks with DummyNet

Dr. Jason But

Swinburne University

Outline

Basic Functions

We have ability to change how bridge/switch/router behaves

Network delay

Packets can be delayed before being forwarded

Packet loss

Can randomly drop packets based on configured probability

Rate limiting

Packets are transmitted at line rate but delays are inserted between packet transmissions

Queuing discipline

Default queues are simple FIFOs but can use other types of queues to improve quality of service

FreeBSD DummyNet

- Part of kernel
- Integrated with **ipfw** firewall

Normal firewall operation

- Select packets based on list of rules
- Either allow packets to pass or drop them

With DummyNet

- After packet matches rule, redirect packet to **DummyNet** pipe/queue
- **DummyNet** implements delay, loss, rate limiting, ...

Enabling DummyNet

- Kernel Configuration file
  `options DUMMYNET`
- Loading kernel module
  `kldload dummynet`

/boot/loader.conf

```
bridge_load=''YES''
ipfw_load=''YES''
```

- Now we need to configure our **ipfw** rules

Using DummyNet

- **ipfw** rules are used to divide packets into flows (groups)
- Rule re-directs flow into DummyNet **pipe** or **queue**
- After packet exits pipe/queue it re-enters **ipfw** and the next rule is applied, unless
  - `sysctl net.inet.ip.fw.one_pass=1`
  - Bridged packets

Pipes

- Emulate link with bandwidth, delay and packet loss characteristics

Queues

- Implements Weighted Fair Queueing (WFQ)
- Multiple flows share pipe's bandwidth proportional to their configured weight

Configuring DummyNet Pipes

Creating pipes

```
ipfw pipe <pipe_num> config <config_options>
```

Examples

```
ipfw pipe 1 config bw 512Kbit/s
ipfw pipe 2 config delay 100ms plr 0.005
ipfw pipe 3 config bw 1536Kbit/s delay 10ms plr 0.001
ipfw pipe 4 config bw 56Kbit/s queue 30
```

`bw`	Specify bandwidth of pipe
`delay`	Set packet delay (ms)
`plr`	Packet loss rate (0...1 where 1 = 100%)
`queue`	Number of packets queue can hold (default is 50)

Redirecting packets into pipes

```
ipfw add <rule_num> pipe <pipe_num> <matching rules>
```


Configuring DummyNet Queues

Creating queues

```
ipfw queue <queue_num> config <config_options>
```

Examples

```
ipfw queue 1 config pipe 1 weight 75
ipfw queue 2 config pipe 1 weight 25
```

`pipe`	Connects queue to pipe
`weight`	Weight for flows in queue (1...100)

Redirecting packets into queues

```
ipfw add <rule_num> queue <queue_num> <matching rules>
```

Can use `ipfw sched <pipe_num> config type <sched_type>` to specify scheduling algorithm

Example – Bridge With DummyNet Pipes

Simple example

```
ipfw pipe 1 config bw 1536Kbit/s delay 10ms
ipfw add 1 pipe 1 ip from any to any bridged
```

- Allows all packets through bridge
- All packets are passed to pipe **1** prior to being output

More complex example

```
ipfw pipe 1 config bw 1536Kbit/s delay 10ms
ipfw pipe 2 config bw 256Kbit/s delay 10ms
ipfw add 1 pipe 1 ip from if0 to if1 bridged
ipfw add 2 pipe 2 ip from if1 to if0 bridged
```


Example – Router With DummyNet Queues

Bandwidth restrictions on Internet link

```
 % the classify rules assume net.inet.ip.fw.one\_pass=1
# setup pipes/queues
ipfw pipe 1 config bw 950kbits/s # 95% of 1Mbits/s uplink
ipfw sched 1 config type wf2q+
ipfw queue 1 config pipe 1 weight 75 # engineering
ipfw queue 2 config pipe 1 weight 5 # accounting
ipfw queue 3 config pipe 1 weight 20 # others

# classify the traffic
# only outgoing traffic is limited, incoming is not affected.
ipfw add 10 allow ip from any to any in via fxp0
ipfw add 100 queue 1 ip from 192.168.1.0/24 to any out via fxp0
ipfw add 200 queue 2 ip from 192.168.2.0/24 to any out via fxp0
ipfw add 300 queue 3 ip from any to any out via fxp0
```


Uses for DummyNet

- Emulating network conditions
- Restricting bandwidth for some network traffic
  - Guaranteeing bandwidth for some applications
  - Giving priority to some network applications
- Testing new protocols or applications over different network conditions

DummyNet Alternative on FreeBSD – pf/ALTQ

- Provides many features available to **DummyNet**
  - Bandwidth management
  - Prioritisation of traffic
- Supports more queuing disciplines
- Management of delays and packet loss is lacking
- More attuned to real deployment than an experimental environment

DummyNet Alternative on Linux – iproute2/tc + netem

- **iproute2/tc** used for setting up bandwidth management and queuing disciplines
- **netem** is special queuing discipline for emulating delay, loss
- **iproute2/tc** has its own traffic matching functions to redirect traffic into queues, but can also redirect based on **netfilter** packet matching
- Provides all features available to DummyNet plus
  - More flexible
  - Many more queuing disciplines (including recent developments)
  - Better integrated with TCP Explicit Congestion Control (ECN)
  - More emulation capabilities, e.g. variable delay and loss, packet corruption, ...
- More complicated to configure