Steady state, fairness and transient behaviour of modern AQMs

Toke Høiland-Jørgensen
Karlstad University

Stanford Netseminar, October 30, 2014
Outline

- About me
- Queue management
- Measurement results
- The netperf-wrapper testing tool
- Unsolved problems
- References and questions
Primer: The bufferbloat problem

- Whenever there’s a bottleneck in the network:
  - Packets have to queue: this induces latency
  - ”Bloated” buffers can result in seconds of queueing

- Managing queues can be a way to fix this
Outline

- About me
- Queue management
- Measurement results
- The netperf-wrapper testing tool
- Unsolved problems
- References and questions
About me

- Name: Toke Høiland-Jørgensen
- M.Sc., Comp Sci and Maths from Roskilde University, Denmark
- Currently: PhD student at Karlstad University, Sweden
- Involved in the Bufferbloat community for ~2 years
- Work area: Minimising latency by controlling queueing
Where is Karlstad?
Karlstad University

- About 16,000 students, 1,200 staff
- CS dept ~40 people; organised in three main areas
  - Distributed systems and communication
  - Privacy/security
  - Software Engineering
- National and EU research projects
- CS appointed strategic research area of the university
  - Currently growing quite rapidly
Outline

- About me
- Queue management
- Measurement results
- The netperf-wrapper testing tool
- Unsolved problems
- References and questions
Smart Queue Management

Name for any technology doing something smarter than FIFO. Can comprise:

- AQM (== drop algorithms)
- Fairness queueing
- Shaping
- Prioritisation

Example: The SQM module of the CeroWrt firmware.
The "new" AQMs

- **CoDel: Controlled Delay** (Nichols and Jacobson, 2012)
  - Directly measure queueing delay, drop if it stays high
- **PIE: Proportional Integral Enhanced** (Pan et al, 2013)
  - Periodically update drop probability from queue length and drain rate
- **Adaptive RED** (Floyd et al, 2001 – not so new)
  - Dynamically adjust RED drop probability from average queue size
FQ: Fairness/Flow Queueing

- SFQ: Stochastic Fairness Queueing (McKenney, 1990)
  - Hash flows into queues, round-robin per-packet dequeue
- DRR: Deficit Round-Robin (Shreedhar and Varghese, 1996)
  - Track deficit at dequeue to approximate byte fairness
- fq_codel: Flow Queueing (Dumazet, 2012)
  - Add sparse flow optimisation and the CoDel AQM
Outline

▶ About me

▶ Queue management

▶ Measurement results

▶ The netperf-wrapper testing tool

▶ Unsolved problems

▶ References and questions
My test setup

- Debian Linux – kernel v3.14
- Rate limiting via `tbf`, delay via `dummynet`
The measurements

- Steady-state behaviour
  - The Realtime Response Under Load (RRUL) test
  - VoIP one-way delay
  - Web page retrievals
- Inter-flow fairness
  - Four TCP flows, 10, 50, 200, 500 ms RTTs
- Transient behaviour
  - RRUL latency over time, from when competing flows start

Paper under submission to USENIX NSDI ’15
The scenarios

- Three bandwidth settings
  - 10/1, 10/10, 100/100 Mbps
- 50 ms base latency
- CUBIC TCP (except fairness tests)
- Three AQMs:
  - ARED
  - PIE
  - CoDel
- Three schedulers:
  - SFQ
  - fq_codel
  - fq_nocodel
- And pfifo_fast (Linux default FIFO queue)
## Parameterisation

<table>
<thead>
<tr>
<th></th>
<th>1 Mbps</th>
<th>10 Mbps</th>
<th>100 Mbps</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pfifo_fast</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>txqueuelen</td>
<td>127</td>
<td>127</td>
<td>1000</td>
</tr>
<tr>
<td><strong>ARED</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min</td>
<td>1514</td>
<td>12500</td>
<td>125000</td>
</tr>
<tr>
<td>bandwidth</td>
<td>1 Mbps</td>
<td>10 Mbps</td>
<td>100 Mbps</td>
</tr>
<tr>
<td>max</td>
<td>3028</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>PIE</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>target</td>
<td>20 ms</td>
<td>20 ms</td>
<td>20 ms</td>
</tr>
<tr>
<td>tupdate</td>
<td>30 ms</td>
<td>30 ms</td>
<td>30 ms</td>
</tr>
<tr>
<td>limit</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td><strong>CoDel</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>target</td>
<td>13 ms</td>
<td>5 ms</td>
<td>5 ms</td>
</tr>
<tr>
<td>interval</td>
<td>100 ms</td>
<td>100 ms</td>
<td>100 ms</td>
</tr>
<tr>
<td>limit</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td><strong>SFQ</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>limit</td>
<td>127</td>
<td>127</td>
<td>1000</td>
</tr>
<tr>
<td><strong>fq_codel</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>target</td>
<td>13 ms</td>
<td>5 ms</td>
<td>5 ms</td>
</tr>
<tr>
<td>interval</td>
<td>100 ms</td>
<td>100 ms</td>
<td>100 ms</td>
</tr>
<tr>
<td>limit</td>
<td>10240</td>
<td>10240</td>
<td>10240</td>
</tr>
<tr>
<td><strong>fq_nocodel</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>limit</td>
<td>127</td>
<td>127</td>
<td>1000</td>
</tr>
<tr>
<td>interval</td>
<td>100 s</td>
<td>100 s</td>
<td>100 s</td>
</tr>
</tbody>
</table>
The Good:  
Steady state results
RRUL 10/1 Mbps

Mean induced latency (ms)

Mean TCP goodput (Mbit/s)
RRUL 10/10 Mbps

Mean induced latency (ms)
8.8 8.9 9.0 9.1 9.2 9.3 9.4 9.5 9.6

Mean TCP goodput (Mbit/s)
pfifo_fast ared pie codel sfq fq_nocodel fq_codel

Graph showing the comparison of different queue management algorithms based on RRUL 10/10 Mbps.
RRUL 100/100 Mbps

Mean induced latency (ms)
82 84 86 88 90 92 94 96

Mean TCP goodput (Mbit/s)
pfifo_fast ared pie codel sfq fq_nocodel fq_codel
VoIP 10/1 Mbps

The graph shows the cumulative probability of one-way delay in milliseconds for different queue management algorithms at a VoIP rate of 10/1 Mbps. The x-axis represents the one-way delay in milliseconds, while the y-axis represents the cumulative probability. The graph compares the performance of various algorithms, including pfifo_fast, ared, pie, codel, sfq, fq_nocodel, and fq_codel, highlighting their effectiveness in managing packet delays.
VoIP 100/100 Mbps

![Graph showing cumulative probability versus one-way delay for different queue management algorithms.](image-url)
Web: Huffington Post 10/10 Mbps w/RRUL background
The Bad:
Inter-flow fairness
Fairness New Reno

![Fairness Index Chart]

The chart above illustrates the fairness index for different queue management algorithms at various bit rates: 100Mbit, 10Mbit, and 1Mbit. The algorithms compared include pfifo_fast, arend, pie, codel, sfq, fq_nocodel, and fq_codel. The y-axis represents the fairness index, ranging from 0.0 to 1.2. The x-axis lists the queue management algorithms.
Fairness flow throughput 100 Mbps
Fairness flow throughput 1 Mbps
The Ugly:
Transient behaviour
100/100 Mbps
10/10 Mbps

![Graph showing latency over time for different queue management techniques at 10/10 Mbps.]
10/1 Mbps
Summary of results

- **The Good: Steady state behaviour**
  - AQMs can significantly improve latency under load
  - FQ algorithms even more so
  - Although CoDel does have some issues at 100 Mbps

- **The Bad: Inter-flow fairness**
  - AQMs exacerbate TCP RTT unfairness
  - FQ does achieve almost perfect fairness

- **The Ugly: Transient behaviour**
  - AQMs take up to tens of seconds to contain latency at competing flow startup
  - FQ doesn’t miss a beat
Outline

- About me
- Queue management
- Measurement results
- The netperf-wrapper testing tool
- Unsolved problems
- References and questions
The netperf-wrapper tool

A Python wrapper for running tests; main features:

- Run several tools in concert
  - and parse their output to a common format (JSON)
- Store metadata along with the test results
- Automatic gathering of metadata
- Batch facilities
- Plotting; lots of plotting.
Included tests

- Simple single-flow tests (ping, TCP ul/dl, UDP flood)
- Latency under load tests
  - 1 TCP flow up/down/bidirectional
  - RRUL variants
  - Periodic UDP bursts
  - On/off TCP flows
- RTT fairness tests
- Comparing TCPs (cubic, reno, westwood, ledbat)
- Application-specific (HTTP, VoIP)
Test specifications

DATA_SETS = o([
    ('TCP upload BE',,
    {'command': find_netperf("TCP_STREAM", LENGTH, HOST,
                              marking="CS0,CS0"),
     'delay': DELAY, 'units': 'Mbits/s',
     'runner': 'netperf_demo',}),

    ('TCP upload BK',
    {'command': find_netperf("TCP_STREAM", LENGTH, HOST,
                              marking="CS1,CS1"),
     'delay': DELAY, 'units': 'Mbits/s',
     'runner': 'netperf_demo',}),

    ('TCP upload avg',
    {'apply_to': [glob("TCP upload *"),
                  exclude=["TCP upload sum"]]},
     'units': 'Mbits/s', 'runner': 'average',})

)
Metadata collected automatically

```
"metadata": {
  "BATCH_NAME": "rrul",
  "BATCH_TIME": "2014-10-02T15:31:11.616664",
  "DATA_FILENAME": "batch-rrul-2014-10-02T153111-50ms-10Mbit-ared-cubic-01.json.gz",
  "EGRESS_INFO": {
    "bql": {
      "tx-0": "1879048192"},
    "classes": null,
    "driver": "e1000e",
    "iface": "eth2",
    "link_params": {
      "ether": "e8:39:35:14:03:31",
      "qlen": "1000"},
    "nexthop": "10.60.1.2",
    "offloads": {
      "generic-receive-offload": false,
      "generic-segmentation-offload": false,
      "large-receive-offload": false,
      "tcp-segmentation-offload": false,
      "udp-fragmentation-offload": false},
    "qdiscs": [{
      "id": "0",
      "name": "pfifo_fast",
      "params": {
        "0": "1", "1": "", "2": "0", "bands": "3", "priomap": "1", "refcnt": "2"},
      "parent": "root"},
    "src": "10.60.1.1",
    "target": "10.60.4.2"},
  "GATEWAYS": [{
    "iface": "eth0", "ip": "192.168.60.1"}],
  "HOST": "testserv-05",
  "HOSTS": [{"testserv-05"},
    "IP_ADDRS": [{
      "eth0": ["192.168.60.91", "fe80::21e:4ff:fee6:3884"],
      "eth2": ["10.60.1.1", "10.60.1.5", "fe80::ea39:35ff:fe14:331"],
      "lo": ["127.0.0.1", "]"}],
  "IP_VERSION": 4,
  "KERNEL_NAME": "Linux",
  "KERNEL_RELEASE": "3.14.4-tohojo-1",
  "LENTH": 140,
  "LOCAL_HOST": "tohojo-testbed-01",
  "NAME": "rrul_be",
  "NETPERF_WRAPPER_VERSION": "0.7.0-git-cbbab94",
  "NOTE": "",
  "REMOTE_METADATA": {
    "testbed-02": {
      "EGRESS_INFO": {
        "bql": {
          "tx-0": "1879048192"},
        "classes": [{
          "id": "1:1", "name": "tbf", "params": {
            "leaf": "2:", "parent": "1:"},
          "id": "2:1", "name": "red", "params": {}, "parent": "2:"}]
      }
    }
  }
}
```

Batch facilities

```
[Batch::global]
# set options
ip_version = 4
length = 140
# build values from variable expansions
title = qdisc:{$qdisc_label} rep:{$repetition} rtt:{$rtt} rate:{$rate_down}/{$rate_up} cc:{$cc}
filename_extra = {$rtt}-${rate_up}-${qdisc_label}-${cc}-${repetition}
output_path = batch-{$batch_time}/{$batch_name}/${rate_up}-${repetition}
# run pre/post commands
commands = clear_caches, setup_qdiscs, tcpdump_client, tcpdump_egress, tcpdump_ingress, tcpdump_server
# iterate over arguments
for_qdiscs = ared, fq_codel, fq_nocodel, codel, pie, pfifo_fast, pfifo_fast_1000, sfq
for_bandwidths = 100mbit, 10mbit, 1mbit

# arguments
[Arg::pie]
inherits = global
qdisc_name = pie
qdisc_args =

# commands
[Command::tcpdump]
filename = ${data_filename}
exec = ssh ${hostname} "python tcpdump-wrapper.py start ${filename} -i ${interface} -s ${capsize}"
type = pre
essential = yes```
GUI for exploring data sets
Installing netperf-wrapper

- Ubuntu/Debian: Go to http://goo.gl/ysYJ7r
- Arch Linux: Install from AUR.
- Others (including OSX w/macports):

$ sudo pip install netperf-wrapper  
$ wget ftp://ftp.netperf.org/netperf/netperf-2.6.0.tar.gz  
$ tar -xzf netperf-2.6.0.tar.gz  
$ cd netperf-2.6.0  
$ ./configure --enable-demo  
$ make  
$ sudo make install
Running the RRUL test

# Running the test
$ netperf-wrapper rrul netperf-west.bufferbloat.net \ 
   -t "Stanford wifi test"

# Viewing the result -- PyQt4 installed
$ netperf-wrapper --gui <filename>.json.gz

# Viewing the result -- otherwise
$ netperf-wrapper -f plot <filename>.json.gz
Outline

▶ About me
▶ Queue management
▶ Measurement results
▶ The netperf-wrapper testing tool
▶ Unsolved problems
▶ References and questions
Unsolved problems – research

- "Parameterless" – not so much
- Transient behaviour
  - Varying bandwidths, esp. wireless
- FQ – for everything?
Unsolved problems – operative

- Storage and indexing of test data
- A better "speedtest" or similar
- Deployment – convincing the world
- Standardisation
Outline

- About me
- Queue management
- Measurement results
- The netperf-wrapper testing tool
- Unsolved problems
- References and questions
References

- **Software and websites:**
  - The Bufferbloat project: http://www.bufferbloat.net
  - The CeroWrt router firmware: http://www.bufferbloat.net/projects/cerowrt
  - Netperf-wrapper: https://github.com/tohojo/netperf-wrapper
  - Test results dataset: https://kau.toke.dk/modern-aqms/
  - My email: toke@toke.dk

- **AQM algorithms:**

- **FQ algorithms:**
Questions?