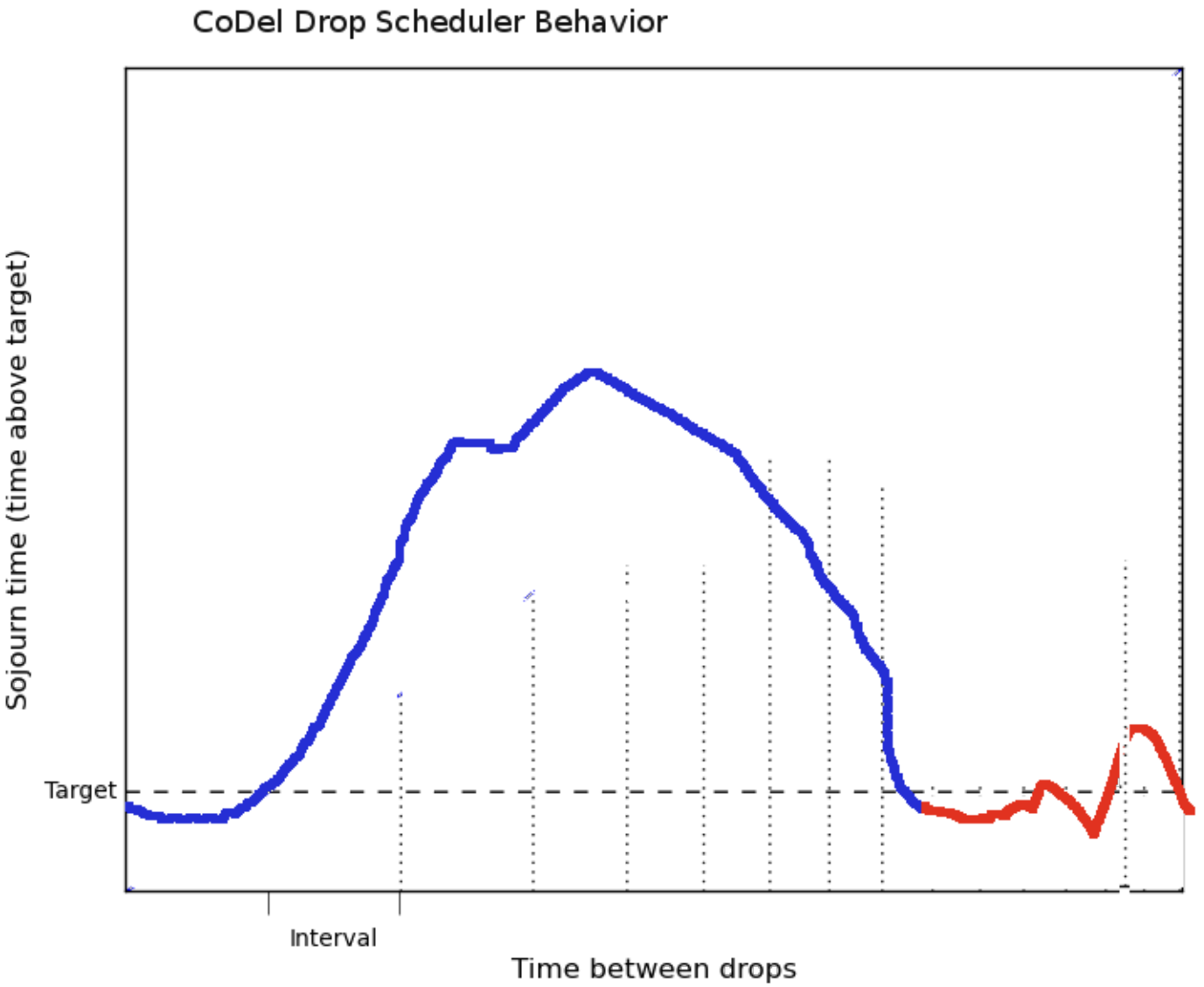
Inside Codel and Fq Codel

http://mirrors.bufferbloat.net/Talks/Stanford2013



Dave Taht <dave.taht@bufferbloat.net>

Codel + SFQ = xFQ_Codel

Two ideas that taste great together

- Successfully controls a single queue's length at a wide range of of potential bandwidths and flows
- Latency spikes shortened
- Good admission control
- Currently has issues at *very,very* high numbers of flows
- Easy to configure
- Better ns2 models exist now...
- Code is deployed in linux 3.5 and later
- Research continues! Multiple variants under development.

- Up to 64k SFQ-like codel queues
- Successfully controls queues' length at a wide range of of potential bandwidths and flows
- Optimizes for short, “sparse” flows (ANTs), and TCP Mice
- Is mostly RTT-fair (not entirely proven)
- Easy to configure
- Has issues at very high numbers of flows and at very, very low bandwidths
- Code is deployed in linux 3.5 and later
- Research continues! Multiple variants under development (nfq_codel, cake)

Web Site and Mailing List

http://bufferbloat.net/projects/codel

Quick Quiz #1
TCP Slow Start:

A) Increases the transmission rate of packets linearly in response to acknowledgments

B) Increases the transmission of packets exponentially in response to acknowledgments

- Extra Credit
  - When does slow start end and congestion avoidance begin?

Bufferbloat

- Wikipedia: “ *a phenomenon in a packet-switched computer network whereby excess buffering of packets inside the network causes high latency and jitter, as well as reducing the overall network throughput.*”

- Bufferbloat is really two things:
  - Excessive buffering at the device, device driver, network queue and tcp/udp queue layers in network stacks on all modern operating systems (windows, mac, Linux, etc)
  - Lack of good packet scheduling and active queue management at ANY layer in all Oses and in common edge devices such as home network gateways, dslams, and cable head ends.
- Without fixing the first, you can't reason about the second.
- You only see the latency spikes when under load.
- Queues are usually either empty, or full.
- All sorts of loads exist, from constant, to transient. Transient spikes exist, but are hard to see. Easy to feel or hear, however. It's easier to create constant loads and measure against those... but not necessarily an accurate representation of reality.

Re: bufferbloat: I don't care about what happens in the data center.

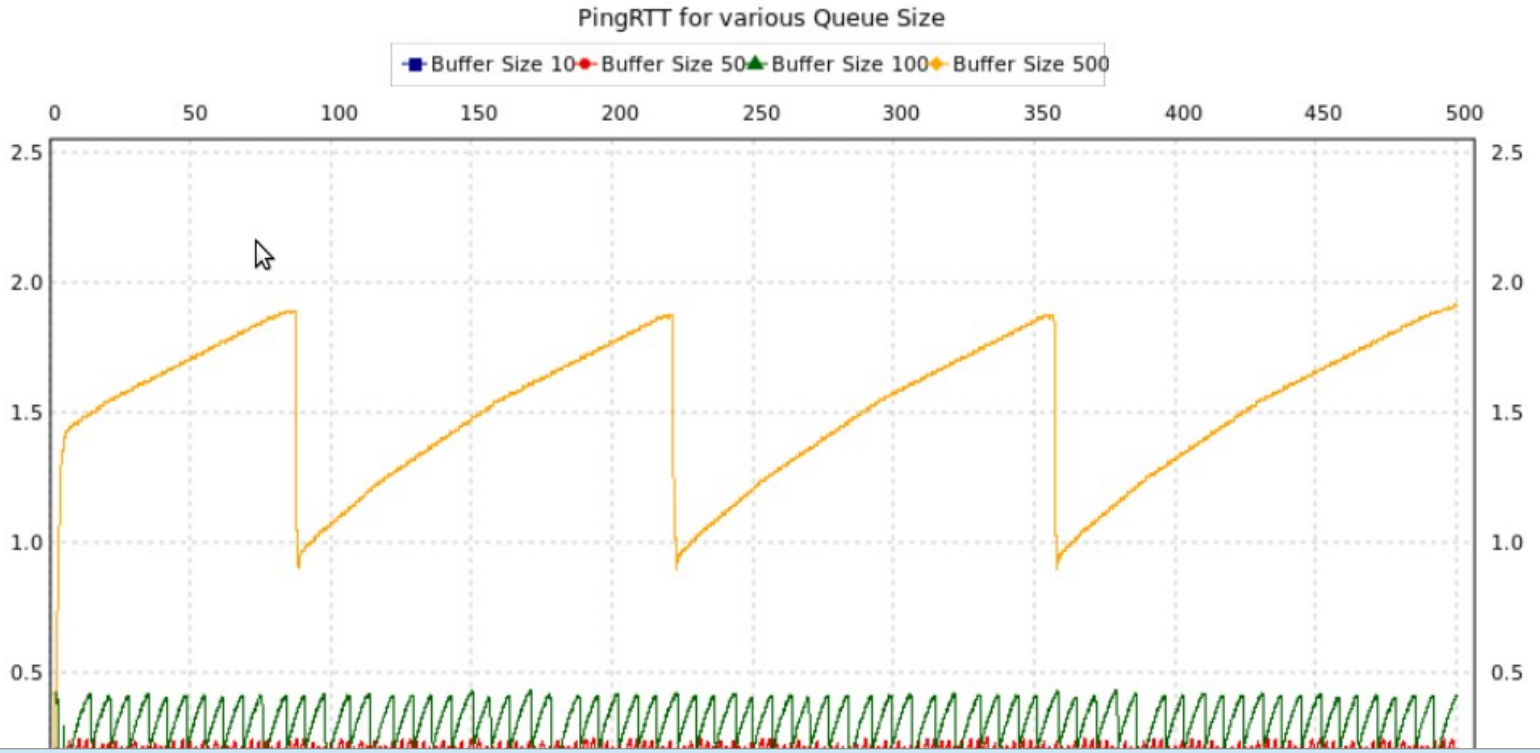
I care about what happens when data exits the data center and enters the real world

Image by: **Stöckli**, Nelson, Hasler
Laboratory for Atmospheres

USGS

Hurricane Linda west of Mexico
September 9, 1997 17:45 UTC

TCP's behavior (TCP 101)



- TCP will always fill the biggest buffer on the path
- See that slow start – over there on the left... it's important. Remember it...

- This is a ns2 **model** from:
  http://staff.science.uva.nl/~delaat/netbuf/bufferbloat_BG-DD.pdf
  (.5Mbit uplink)

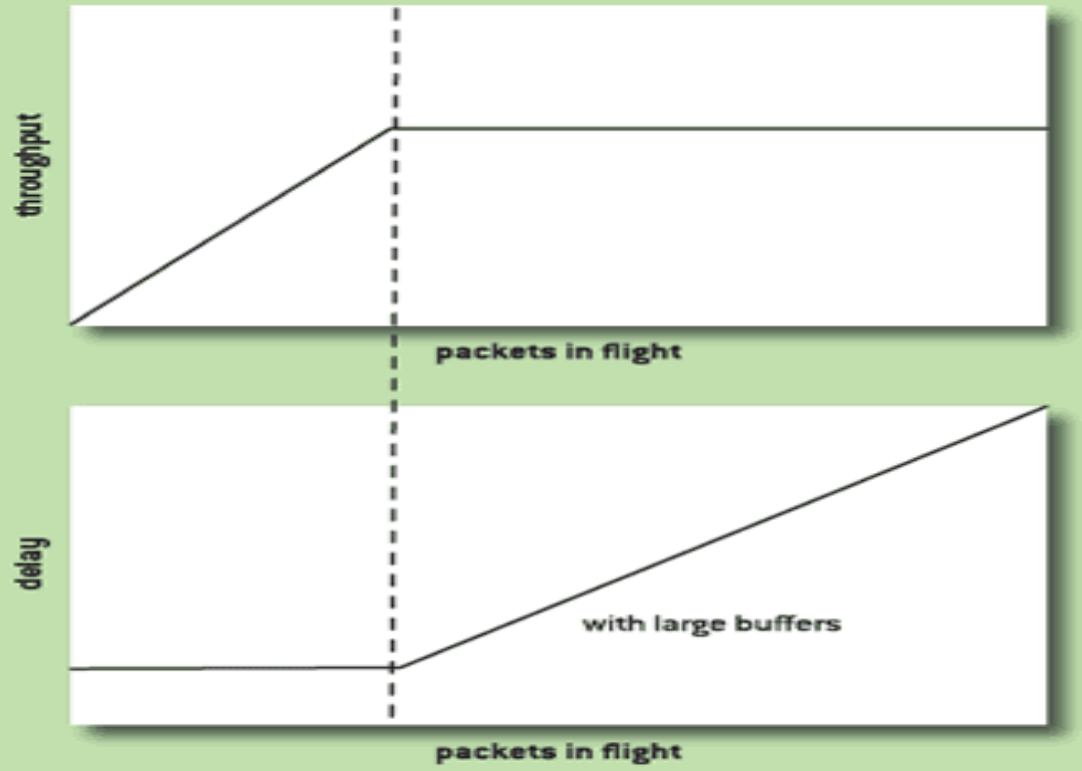
TCP Network Signaling

- Loss (or marking) a packet signals the sender
- The signal indicates the sender should slow down...
- The loss of the packet requires the sender resend the packet (at a later time, at a slower rate), if needed.
- Other protocols use similar mechanisms

Some buffering is needed... but any extra buffering just adds latency.

FIGURE 1

Throughput and Delay



Standard Tail drop Queue Management

- Queues are *necessary* to handle *bursts* from fast into slower bandwidth links.
- A *too small* queue leads to reduced throughput and problems with admission control.
- A *too large* queue leads to higher latency...
- Available bandwidth today varies by 5+ orders of magnitude: 100Kbit – 10GigE.
- We have large bursts (IW10, Network offloads, Wifi) that need to be absorbed and spread out
- *There is no one right size for a drop tail queue – and, lacking an effective AQM, vendors have opted for "too large", rather than too small.*

Of Elephants, Mice, and Ants

- TCP “elephant” flows are long running flows (such as uploads to dropbox, facebook, or youtube, or downloads of isos, etc)
- TCP “mice” are short flows (typical of web traffic), that usually never get out of “slow start”. Google for multiple papers for “TCP Mice elephants”
- ANTs are the short (usually single!) control packets – like TCP syn (session initiation), TCP SYN/ACK, DNS, DHCP, NTP, and to some extent VOIP, videoconferencing, internet radio and gaming packets.
  - Over the Internet the presence of elephants and mice drown out the ants statistically by an enormous margin. However ANTs are important packets! Without them you can't even get an address on a network.
  - A typical web transaction consists of dozens of DNS lookups, each accompanied by dozens of TCP mice, and very rarely, an elephant or two.
  - For web traffic especially: When DNS gets delayed, or a TCP SYN, SYN/ACK, new flow start is also delayed.

- *Note to PETA members: No actual, physical elephants, mice, or ants will be harmed by this preso or algorithms.*

Realtime Response Under Load test (RRUL)

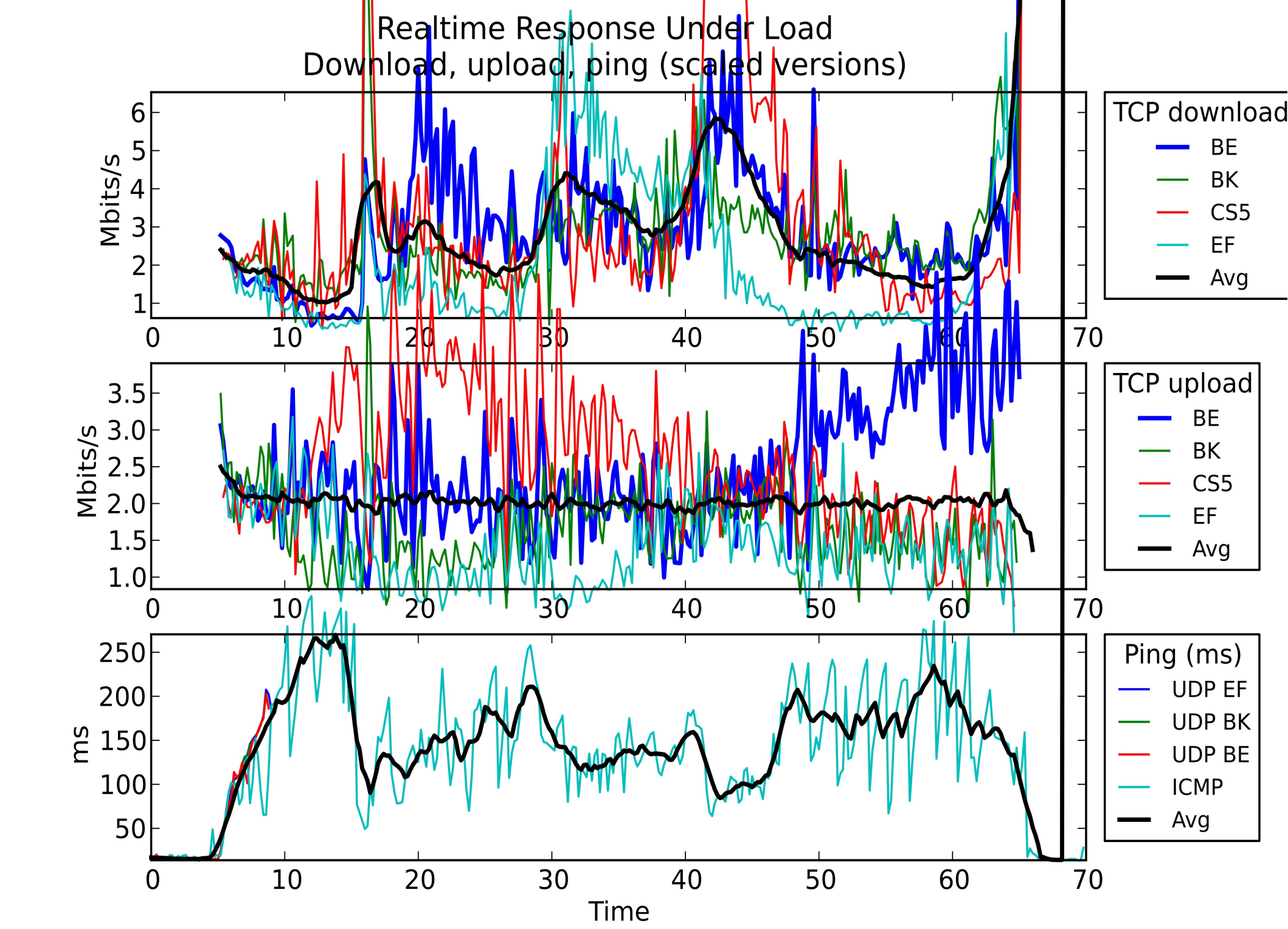
- Tests 4 up, 4 down TCP streams against icmp and udp traffic.
- Also tries diffserv (802.11e) classification
- Json data output
- Native Plots
- Web Interface
- DB backend

Extensions

- rrul46compete: RRUL using ipv4 and ipv6 at the same time.
- rtt_fair: test tcp performance between two or more hosts to see if a system is RTT-fair (meaning that connections to machines at different distances eventually or not get a fair share of the bandwidth)
- reno_cubic_westwood_lp: test performance of different TCPs

Simpler Tests

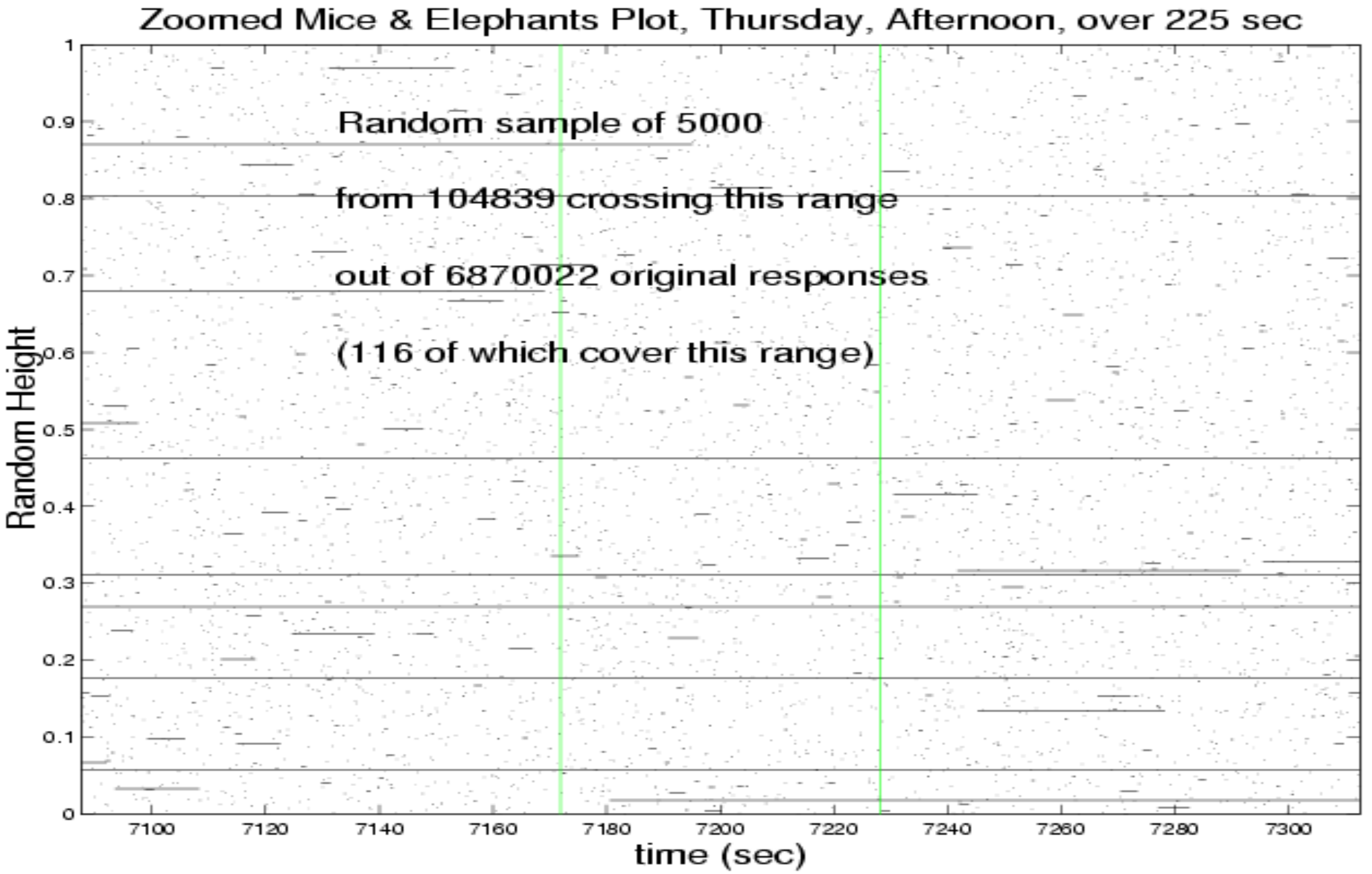
- tcp_bidirectional: a basic test intended to give a "textbook" result of two competing streams against a ping
- tcp_upload: multiple tcp uploads against ping
- tcp_download: multiple tcp downloads against ping


Realtime Response Under Load
Download, upload, ping (scaled versions)
Mbits/s
TCP download
BE
BK
CS5
EF
Avg
Mbits/s
TCP upload
BE
BK
CS5
EF
Avg
ms
Ping (ms)
UDP EF
UDP BK
UDP BE
ICMP
Avg
Time
Local/remote: lupin-gw/snapon.lab.bufferbloat.net - Time: 2013-01-11 11:45:28.437402 - Length/step: 60s/0.20s

Despite RRUL's interesting graphs on latency under load....

The real point of the RRUL test is to run other kinds of network flows against it.

How do you see the effects of a queuing algorithm, on a loaded network, on admission control, and on the mice and ants... verses the elephants?



Source: TCP Mice and Elephants

Latency via passive measurement

- Mark Allman - "Comments on Bufferbloat"

  "~5% of connections to "residential" hosts exhibit added delays of >=400 milliseconds, a delay that is certainly noticeable and would make interactive applications (gaming, voip etc) pretty much unusable."

- C. Chirichella & Dario Rossi –

  To the Moon and back: are bufferbloat delays really that large?

  "99% of the windows of the median peer have a queuing delay below 60 ms. Moreover, only 10% of the peers experience a 90-th (99-th) percentile above 100 ms (200 ms). In other words, for the 10% of peers that are most affected by bufferbloat, only 10% (1%) of their 1-second windows incur more than 100 ms (200 ms) queuing delay."

Note...

Web traffic is "interactive", too!

The characteristics of web traffic

- A DNS lookup (example: www.google.com) happens (10-40ms)
- A TCP session is initiated (SYN and SYN/ACK exchange (RTT dependent: 0 to 100s of ms)
- A HTTP request is issued over the connection (HTTP GET /index.html) (RTT again)
- The data is transferred. It ramps up with “slow start” into a flow, to the maximum bandwidth available as a function of RTT, TCP acks, and competing traffic, using packet loss/delay for an estimate depending on the congestion control algorithm.
- The flow ends.
- The process repeats until an entire web page is loaded.

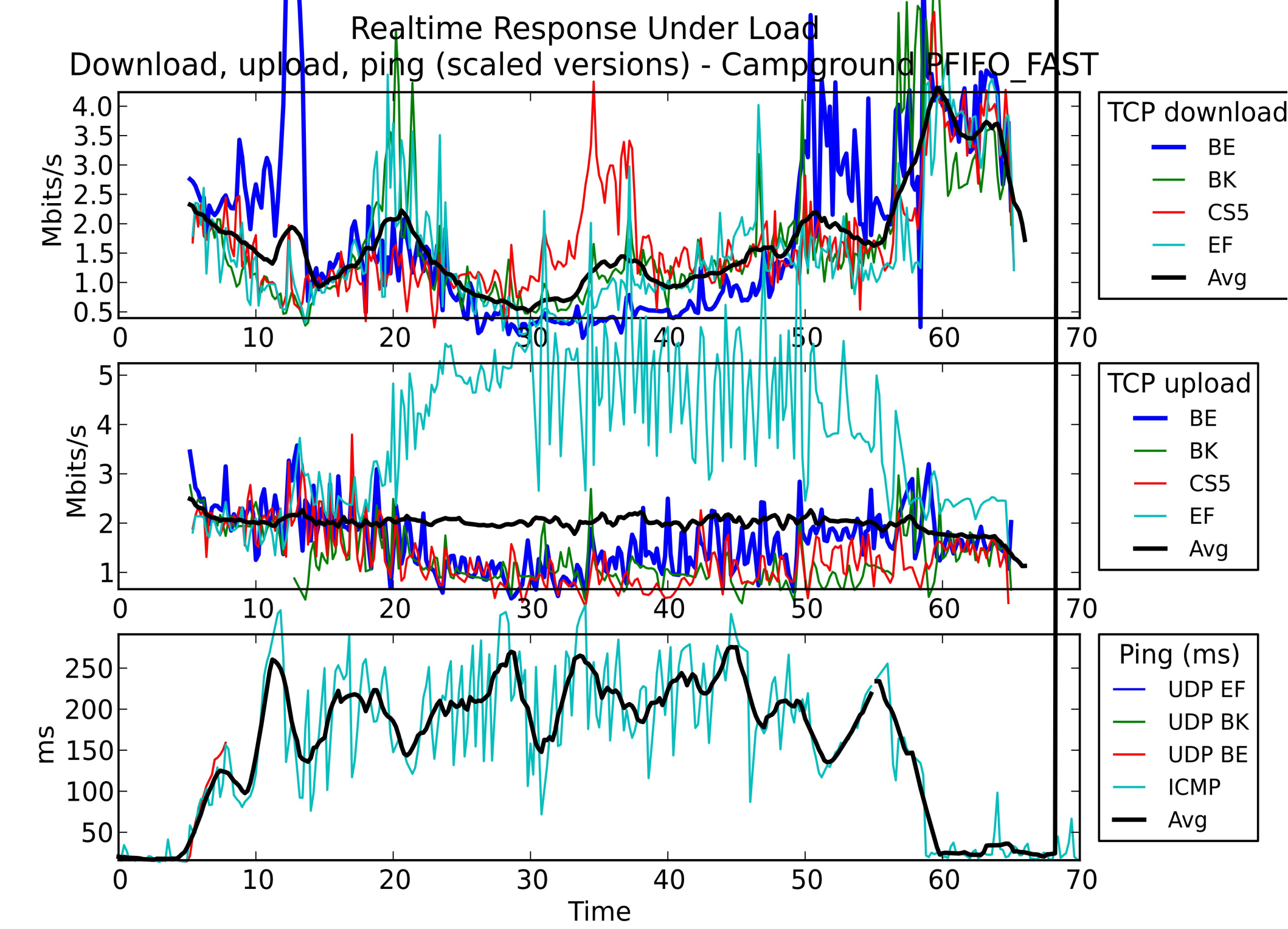
Web browsers optimize to *mask* latency as much as possible

- DNS → TCP syn → Syn-Ack → Flow-Start → Flow → Flow end process is done in parallel, usually 6 or more at the same time, in most web browsers.
- DNS is partially cached, but there might be dozens of DNS lookups, each with varying RTT.
- HTTP pipelining helps, where used, to skip the syn and syn/ack process in establishing new TCP connections
- The (new) “TCP fast open” method, embeds the HTTP GET in the SYN, eliminating one RTT. (but may break some middleboxes)

But: what happens... ...if the queue is full?

- The DNS packets are delayed
- The TCP syn and syn/acks are delayed
- Slow start is delayed
- And finding an optimum flow rate...

  takes a long time.
- What *one* other flow does on non fq_codeled network, can mess up your whole network.


Realtime Response Under Load
Download, upload, ping (scaled versions) - Campground PFIFO_FAST
Mbits/s
TCP download
BE
BK
CS5
EF
Avg
Mbits/s
TCP upload
BE
BK
CS5
EF
Avg
ms
Ping (ms)
UDP EF
UDP BK
UDP BE
ICMP
Avg
Time
Local/remote: lupin-gw/snapon.lab.bufferbloat.net - Time: 2013-01-11 12:50:00.893620 - Length/step: 60s/0.20s

Despite RRUL's interesting graphs on latency under load....

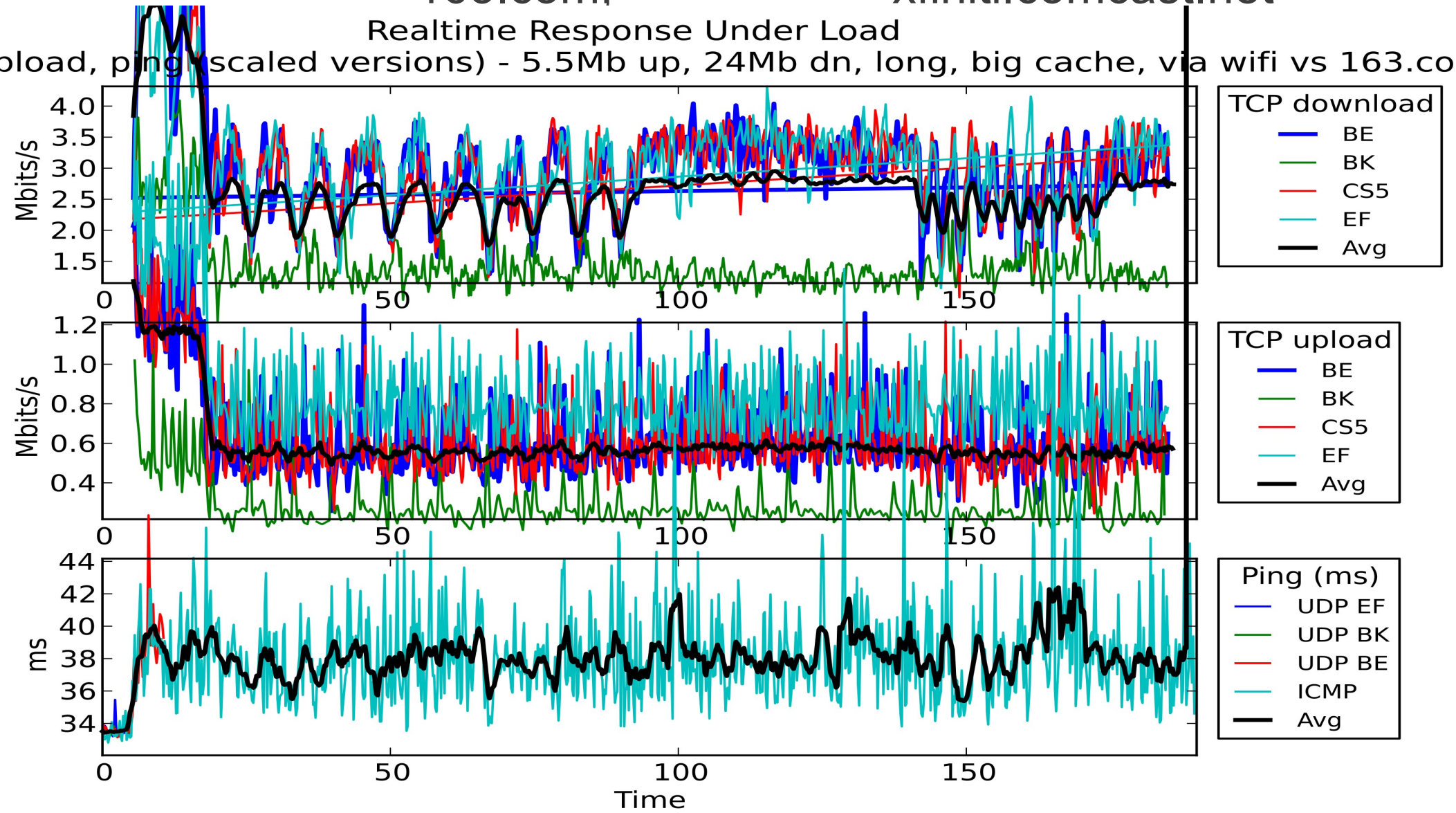
The real point of the RRUL test is to run other kinds of network flows against it.

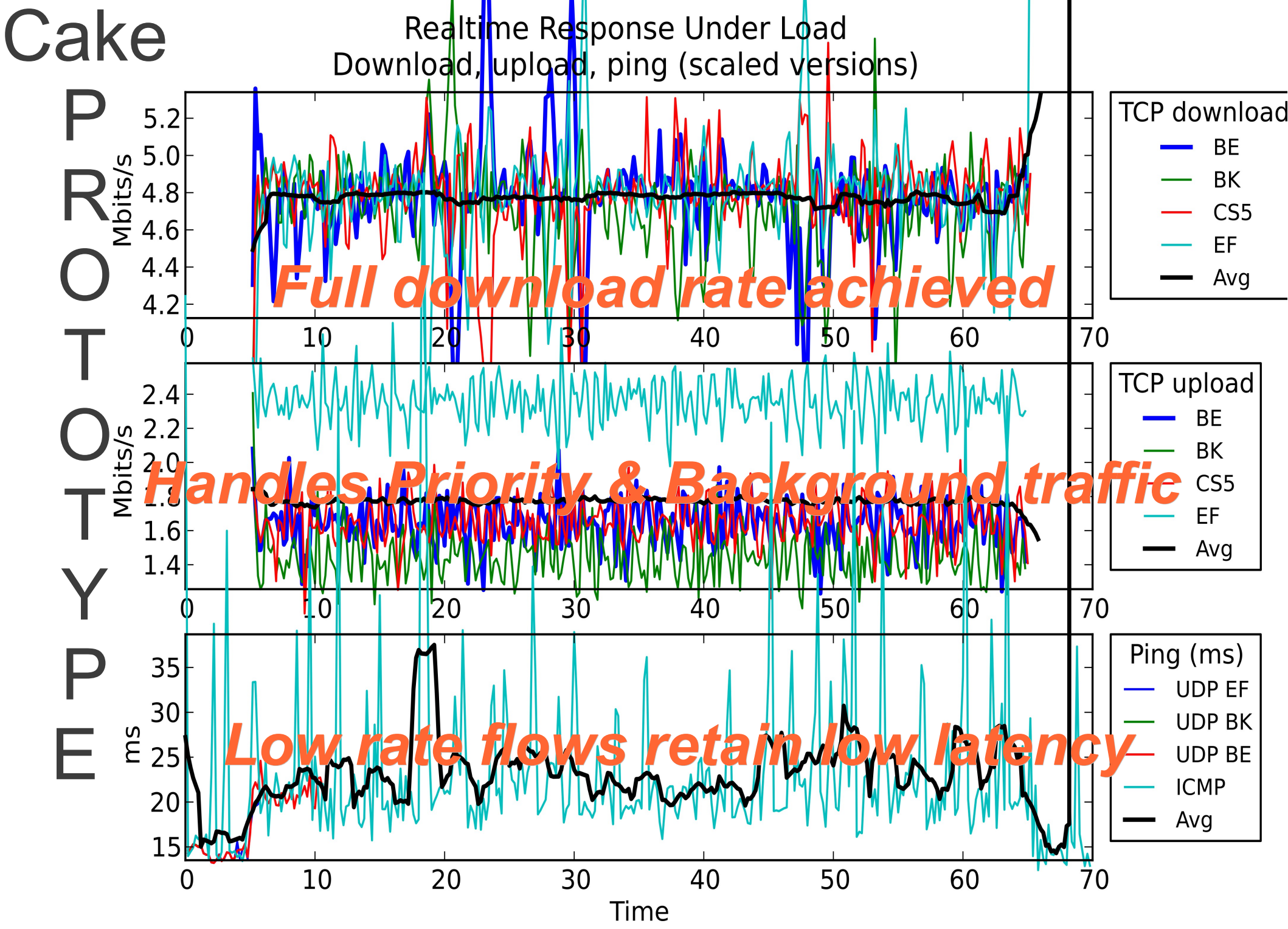
How do you see the effect of a queuing algorithm, on a loaded network, on the mice and elephants?

Answer: observe the effect of ant/mouse traffic on the elephants!

(htb + nfq_codel) RRUL test vs Chrome Web Page Benchmark

163.com, xfiniti.comcast.net




Cake
PROTOTYPE
Realtime Response Under Load
Download, upload, ping (scaled versions)
Mbits/s
TCP download
BE
BK
CS5
EF
Avg
Full download rate achieved
TCP upload
Handles Priority & Background traffic
ms
Ping (ms)
UDP EF
UDP BK
UDP BE
ICMP
Low rate flows retain low latency
Time
Local/remote: lupin-gw/snapon.lab.bufferbloat.net - Time: 2013-01-11 12:01:51.421053 - Length/step: 60s/0.20s

Traffic types in the Home

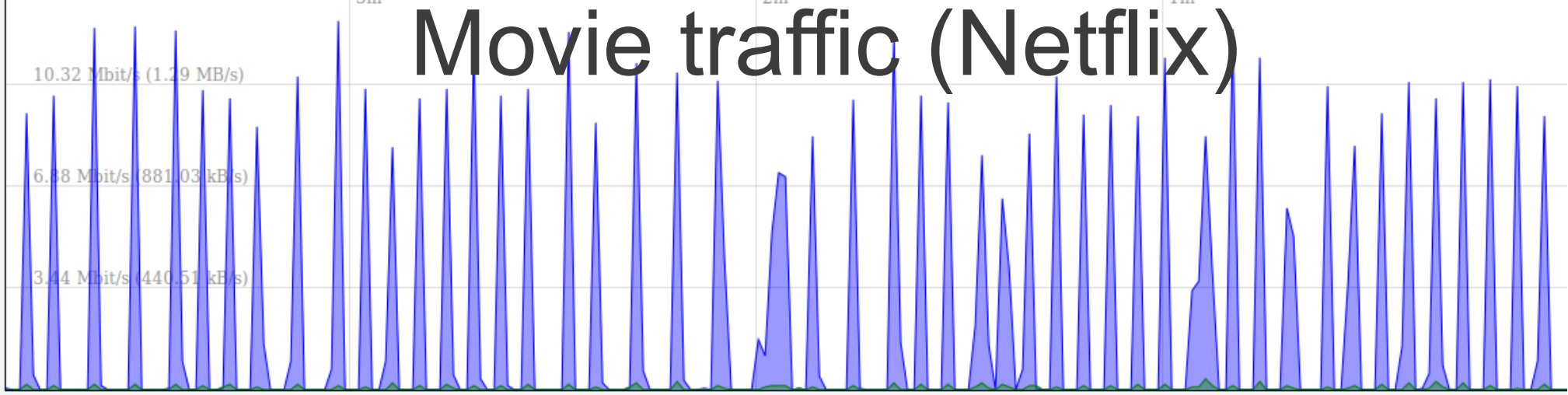
Low Latency

- Web
- Videoconferencing
- Audio streaming
- VOIP
- Gaming

Bulk/Background

- Software Updates
- Dropbox
- Email
- VPN
- Video

Movie traffic (Netflix)

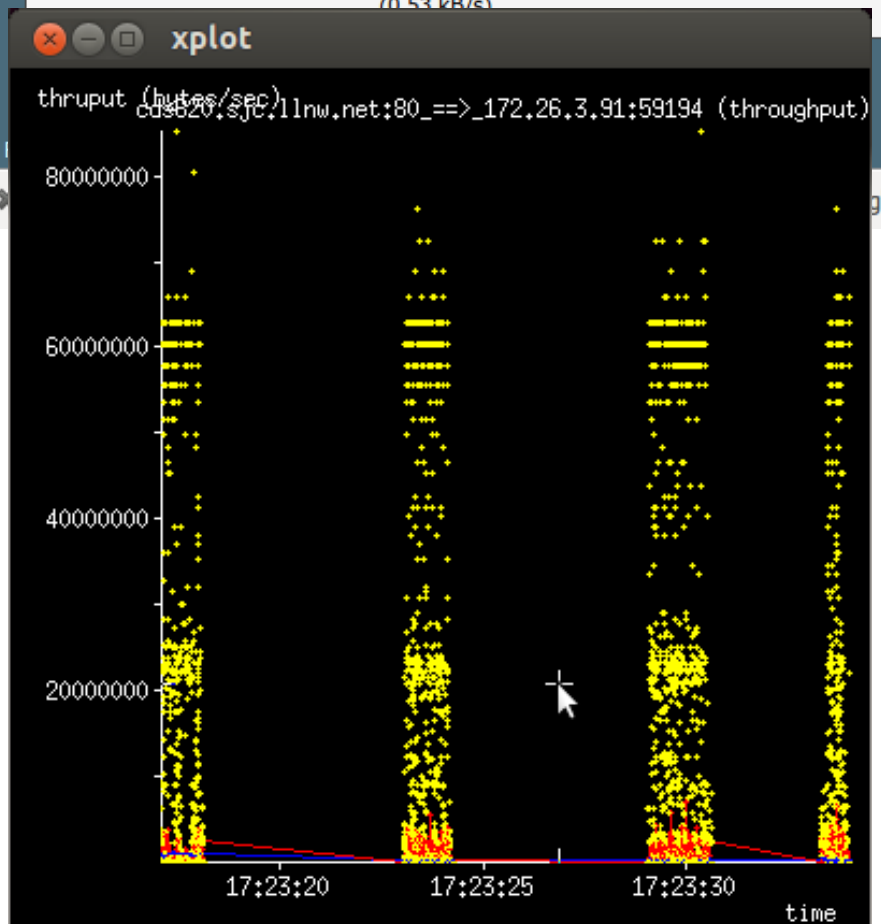


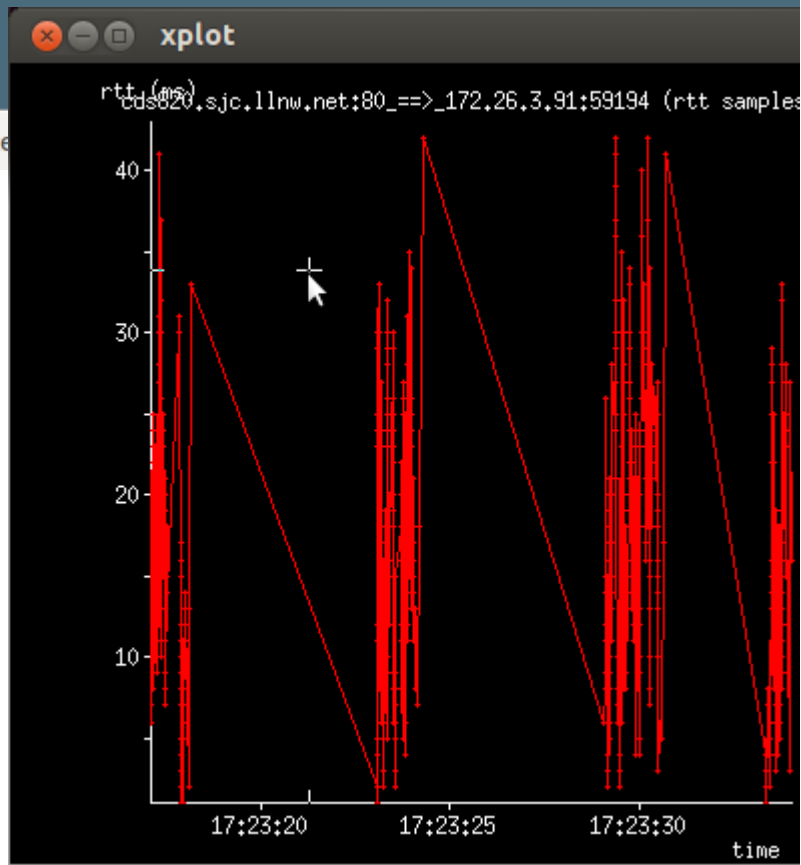
(3 minute window, 3 second interval)

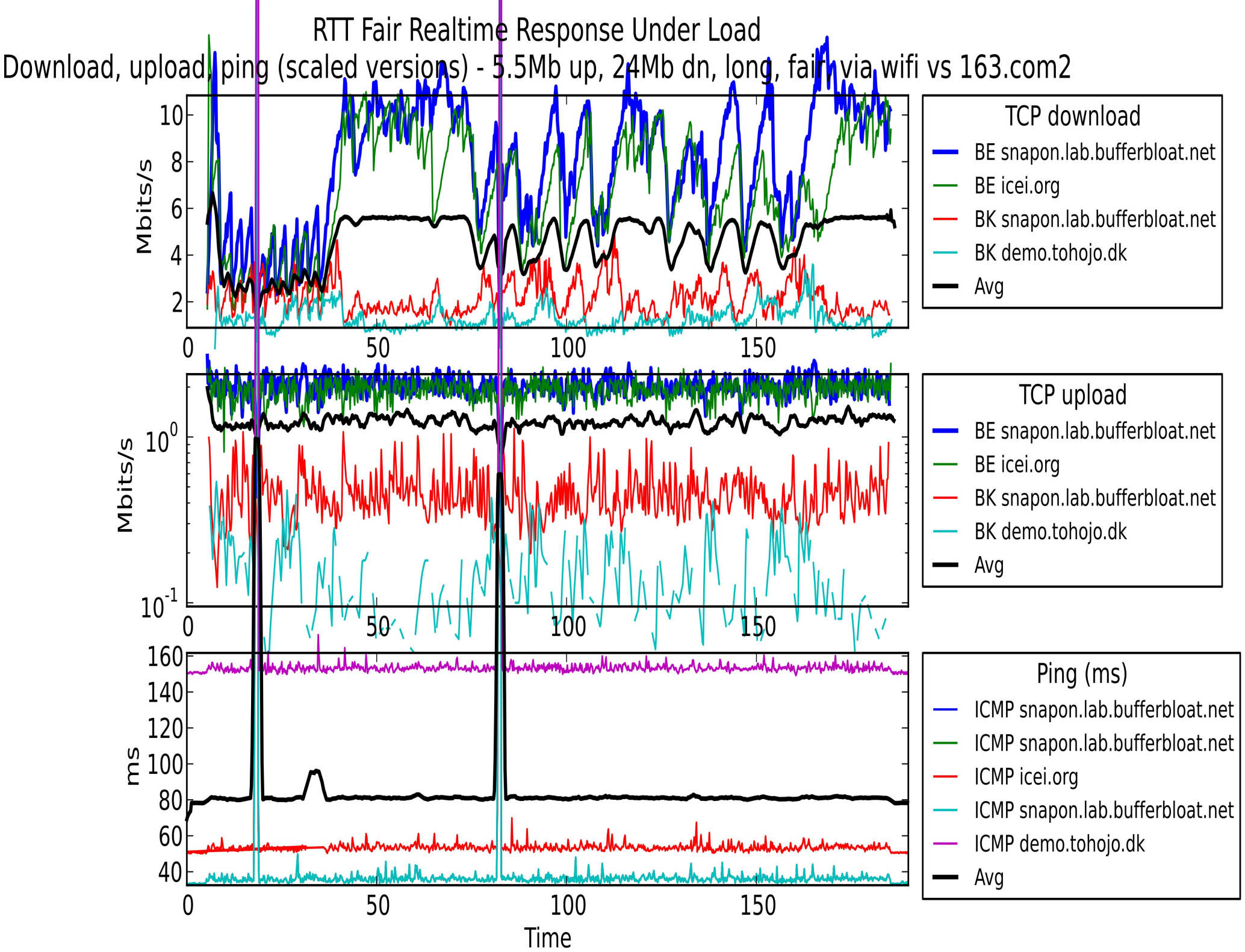
Inbound: 7.55 kbit/s (0.94 kB/s)
Outbound: 4.27 kbit/s (0.53 kB/s)

Average: 677.4 kbit/s (84.68 kB/s)
Average: 18.56 kbit/s (2.32 kB/s)

Peak: 12.51 Mbit/s (1.56 MB/s)
Peak: 412.42 kbit/s (51.55 kB/s)






RTT Fair Realtime Response Under Load
Download, upload, ping (scaled versions) - 5.5Mb up, 24Mb dn, long, fair, via wifi vs 163.com2
Mbits/s
10
8
6
4
2
0
50
100
150
TCP download
BE snapon.lab.bufferbloat.net
BE icei.org
BK snapon.lab.bufferbloat.net
BK demo.tohojo.dk
Avg
Mbits/s
10^0
10^{-1}
0
50
100
150
TCP upload
BE snapon.lab.bufferbloat.net
BE icei.org
BK snapon.lab.bufferbloat.net
BK demo.tohojo.dk
Avg
ms
160
140
120
100
80
60
40
0
50
100
150
Ping (ms)
ICMP snapon.lab.bufferbloat.net
ICMP snapon.lab.bufferbloat.net
ICMP icei.org
ICMP snapon.lab.bufferbloat.net
ICMP demo.tohojo.dk
Avg
Time

Apologies

While I have data sets of google hangouts, audio streaming, voip, gaming and bittorrent against the RRUL test... against various combinations of pfifo_fast (drop tail), codel, ns2 codel, fq_codel, nfq_codel, cake...I didn't have time to plot them all.

Prototypes of the RRUL test suite are available at:

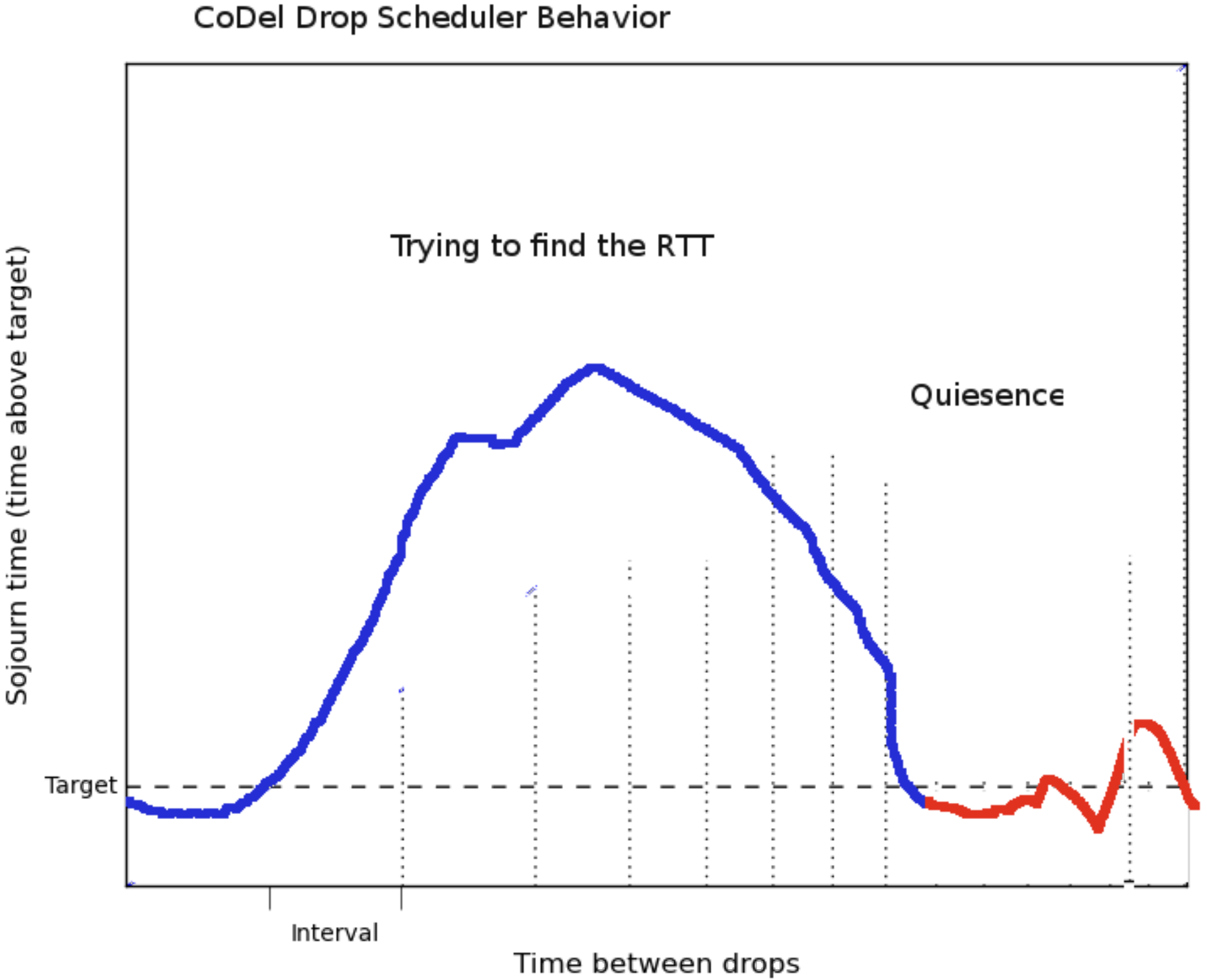
- https://github.com/tohojo/netperf-wrapper
- Give it a try yourself! The CDF plots are great, too! Please upload your interesting rrul plots to the RRUL Rogues Gallery
- Paper: http://akira.ruc.dk/~tohojo/bufferbloat/bufferbloat-final.pdf
- Major server/test expansion is in the works
- Huge thanks to Toke Høiland-Jørgensen <toke@toke.dk> for turning 1/3 of the RRUL specification into code in 2 months flat.

Inside Codel

- Introduction
- Timestamp based latency Monitoring
- The Drop Scheduler
- Remission
- Resumption

Introducing Kathie Nichols and Van Jacobson's Codel algorithm

- Measure the latency in the queue, from ingress to egress, via timestamping on entry and checking the timestamp on exit.
- When latency exceeds target, think about dropping a packet
- After latency exceeds target for an interval, drop a packet at the HEAD of the queue (not the tail!)
- If that doesn't fix it, after a shorter interval (inverse sqrt), drop the next packet sooner, again, at the HEAD.
- Keep decreasing the interval between drops per the control law until the latency in the queue drops below target.
- We start with 100ms as the interval for the estimate, and 5ms as the target. This is good on the world wide internet
- Data centers need much smaller values.


CoDel Drop Scheduler Behavior
Trying to find the RTT
Quiesence
Sojourn time (time above target)
Target
Interval
Time between drops

Any Questions so far?

- CoDel attempts to find the RTT of TCP friendly flows and drop the minimum number of packets needed to retain low latency and high throughput.

- It's a pretty nice pony.

Issues with Codel

- Single queue works well with mixtures of "TCP friendly" streams, but not with floods
- Multiple variants – linux kernel version currently different from ns2 version
- Doesn't do a lot for ants and mice
- Turns delay based TCP into loss based TCP.

Quick Quiz #2

What does Codel do against a stream in slow start?

Answer:

Not a lot! It has hopefully achieved a drop rate against all previously entered streams that holds latency low, and buffers relatively empty, but it isn't prescient, it doesn't use tachyons, it's can't predict what's going to happen next...

CoDel is a piece of the solution for bufferbloat.

- Another piece, in mitigating slow start, is some form of fair Queueing...

The “Fair Queuing” scheduling algorithms (WFQ, SFQ, QFQ, etc)

Core ideas:

- Break up streams into identifiable flows.
- Offer roughly equal service to each flow.
- Perfect fair queuing is impossible, so various hashing techniques are used. These (usually) divide flows up by a hash value based on a 5 tuple of information (usually src ip, dst ip, src port, dst port, protocol, but can be other things (mac address, etc)
- Stochastic Fair Queuing as described in June IEEE Infocom '90

  http://www2.rdrop.com/users/paulmck/.../paper/sfq.2002.06.04.pdf
  - See also: QFQ, WFQ, etc
- Solves the Mice and Ant problem by giving separate flows their own queue.
- SFQ Widely deployed in “wondershaper” and related shaping tools
- Used in some provider networks (free.fr)
- SFQ Still used tail drop – and had small hard coded limits on queue size - and even an infinitely sized buffer will eventually lose packets (RFC970), So TCP Elephants interact with the size of the queue, often badly.

Improvements to SFQ (linux 3.3)

- Vastly increased the number of hash buckets (making it work at much higher ranges of flows)
- Vastly increased the potential queue depths (higher bandwidths)
- Permutation of the hash induced packet re-ordering, which killed throughput every 10 seconds (fixed by re-permutation, delivering all old packets, and then resuming deliveries)
- Result worked well on Mice and Ant traffic
- Along the way we learned that timestamping was incredibly cheap in linux now
- But:

  At higher speeds and queue lengths, it didn't scale, increasing the size of the control loops, reintroducing latency control problems for TCP elephants (think: stuttery youtube videos) with tail drop...
- Requires configuration to work at nearly any bandwidth
- So, it didn't work nearly well enough...

SFQRED: A FQ + AQM hybrid (linux 3.4 (jan, 2012))

- By looking at the interaction of QFQ + RED, and desiring something that did both fair queuing, and active queue length management, Eric Dumazet developed “SFQRED”, which:
- Utilized a better version of RED (“ARED”) from Sally Floyd in 2002
- SFQRED Introduced head drop for the first time in any AQM that I know of...
- Astoundingly successful in controlling queue length in 4-200Mbit bandwidths. Head drop vastly improved reaction time in elephant flows, Mice worked great, Ants even better. It was highly encouraging!
- Flaws:
  - Unbelievably hard to configure at any bandwidth
  - Didn't handle variable bandwidth at all (unsuitable for cable modems, wifi, or any shared network)
- But: *It took 1 week to convert SFQRED into FQ_Codel*

“FQ_Codel provides great isolation... if you've got low-rate videoconferencing and low rate web traffic they never get dropped. A lot of issues with IW10 go away, because all the other traffic sees is the front of the queue. You don't know how big its window is, but you don't care because you are not affected by it. FQ_Codel increases utilization across your entire networking fabric, especially for bidirectional traffic...”

“If we're sticking code into boxes to deploy codel, *don't do that*.

Deploy fq_codel. It's just an across the board win.”

- ***Van Jacobson***

IETF 84 Talk

Obligatory FQ_Codel Math

The number of buckets filled by n elephant sessions given

T hash buckets (default: 1024 for FQ-CoDel) is given by:

$$k = T * (1 - (1 - 1/T)^n)$$

Given k, the probability of a thin session colliding with a

Mouse/ant session is just $k/T = 1 - (1 – 1/T)^n$. When n is small, the thick sessions tend not to collide (as one would expect). So at 100 thick sessions, there is a 9.3% chance of any given thin session colliding.

At 1024 thick sessions, there is enough collision between these sessions that there is "only" a 63% chance of a given thin flow colliding with a thick flow.

CONCLUSION: As thick sessions are rare, in home use, it's more than good enough. It MIGHT be good enough for head-ends, hosts, and servers, too. And elsewhere.

FQ_Codel Principles

- HEAD DROP, not tail drop.
- Queues are shock absorbers
- What matters is the delay within a flow.
- Shoots packets in elephant flows after they start accumulating delay. Don't shoot anything else!
- See Van Jacobson's preso at: http://recordings.conf.meetecho.com/Recordings/wa
- Or Eric Dumazet's preso at: http://linuxplumbers.ubicast.tv/videos/codel-and-fq_

```
static int fq_codel_enqueue(struct sk_buff *skb, struct Qdisc *sch)
{
	struct fq_codel_sched_data *q = qdisc_priv(sch);
	unsigned int idx;
	struct fq_codel_flow *flow;
	int uninitialized_var(ret);

	idx = fq_codel_classify(skb, sch, &ret);
	if (idx == 0) {
		if (ret & __NET_XMIT_BYPASS)
			sch->qstats.drops++;
		kfree_skb(skb);
		return ret;
	}
	idx--;

	codel_set_enqueue_time(skb);
	flow = &q->flows[idx];
	flow_queue_add(flow, skb);
	q->backlogs[idx] += qdisc_pkt_len(skb);
	sch->qstats.backlog += qdisc_pkt_len(skb);

	if (list_empty(&flow->flowchain)) {
		list_add_tail(&flow->flowchain, &q->new_flows);
		q->new_flow_count++;
		flow->deficit = q->quantum;
	}
	if (++sch->q.qlen < sch->limit)
		return NET_XMIT_SUCCESS;
```

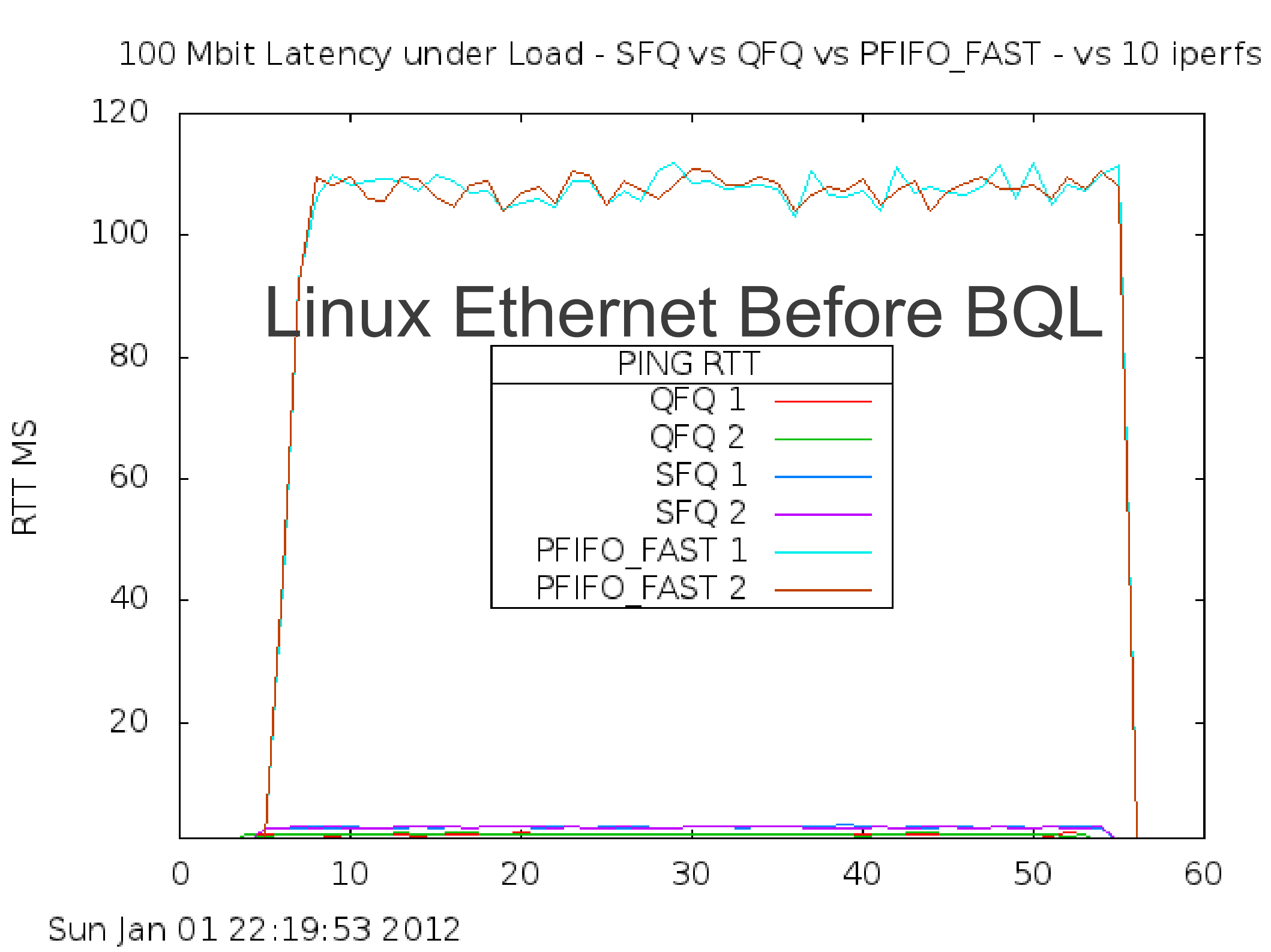
```
q->drop_overlimit++;
        /* Return Congestion Notification only if we dropped a packet
         * from this flow.
         */
        if (fq_codel_drop(sch) == idx)
                return NET_XMIT_CN;

        /* As we dropped a packet, better let upper stack know this */
        qdisc_tree_decrease_qlen(sch, 1);
        return NET_XMIT_SUCCESS;
}
```

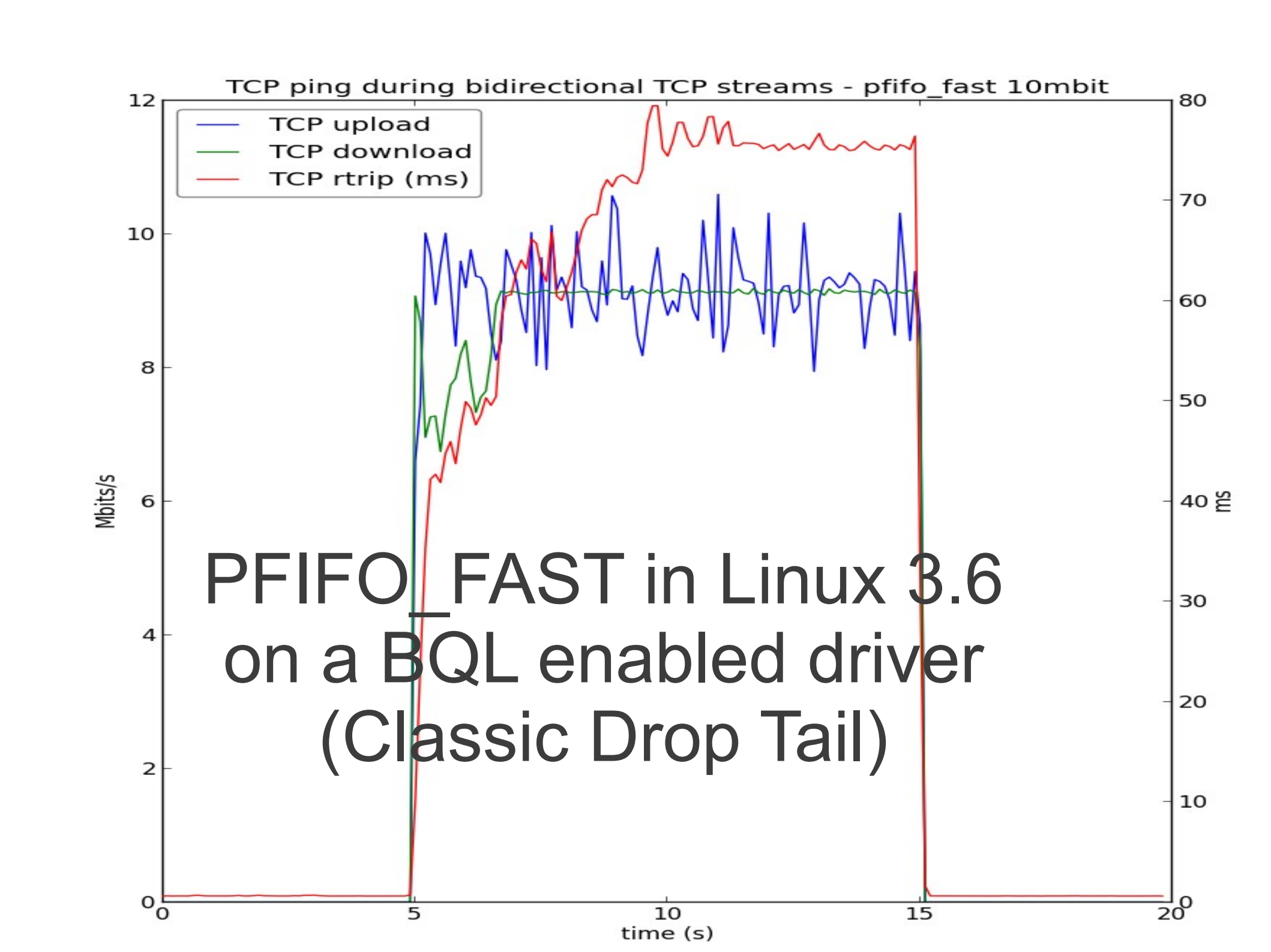
```
static struct sk_buff *fq_codel_dequeue(struct Qdisc *sch)
{
	struct fq_codel_sched_data *q = qdisc_priv(sch);
	struct sk_buff *skb;
	struct fq_codel_flow *flow;
	struct list_head *head;
	u32 prev_drop_count, prev_ecn_mark;

begin:
	head = &q->new_flows;
	if (list_empty(head)) {
		head = &q->old_flows;
		if (list_empty(head))
			return NULL;
	}
	flow = list_first_entry(head, struct fq_codel_flow, flowchain);

	if (flow->deficit <= 0) {
		flow->deficit += q->quantum;
		list_move_tail(&flow->flowchain, &q->old_flows);
		goto begin;
	}

	prev_drop_count = q->cstats.drop_count;
	prev_ecn_mark = q->cstats.ecn_mark;
```

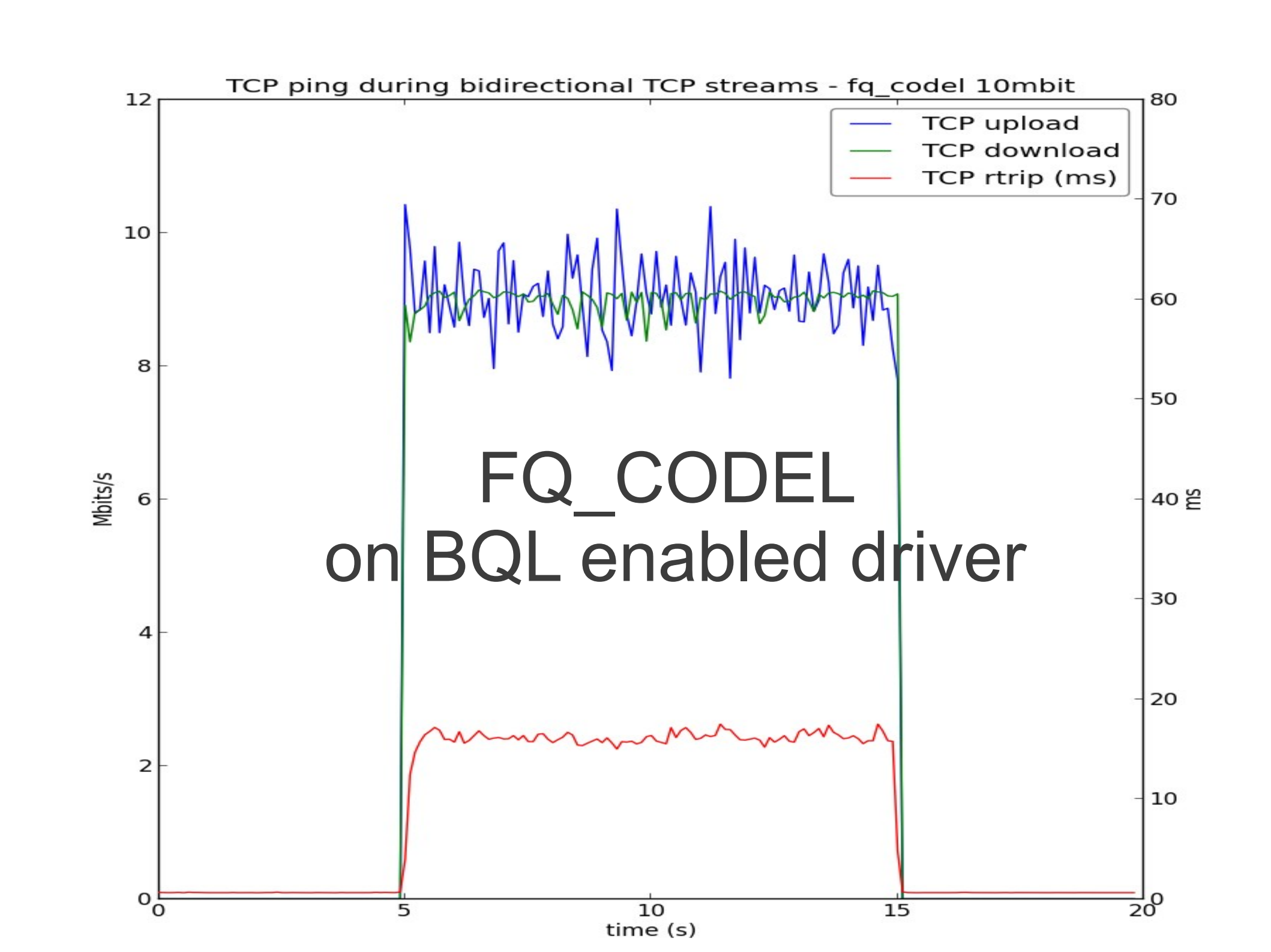
```
skb = codel_dequeue(sch, &q->cparams, &flow->cvars, &q->cstats,
                    dequeue);

        flow->dropped += q->cstats.drop_count - prev_drop_count;
        flow->dropped += q->cstats.ecn_mark - prev_ecn_mark;

        if (!skb) {
                /* force a pass through old_flows to prevent starvation */
                if ((head == &q->new_flows) && !list_empty(&q-
>old_flows))
                        list_move_tail(&flow->flowchain, &q->old_flows);
                else
                        list_del_init(&flow->flowchain);
                goto begin;
        }
        qdisc_bstats_update(sch, skb);
        flow->deficit -= qdisc_pkt_len(skb);
        /* We cant call qdisc_tree_decrease_qlen() if our qlen is 0,
         * or HTB crashes. Defer it for next round.
         */
        if (q->cstats.drop_count && sch->q.qlen) {
                qdisc_tree_decrease_qlen(sch, q->cstats.drop_count);
                q->cstats.drop_count = 0;
        }
        return skb;
}
```

DEQUE

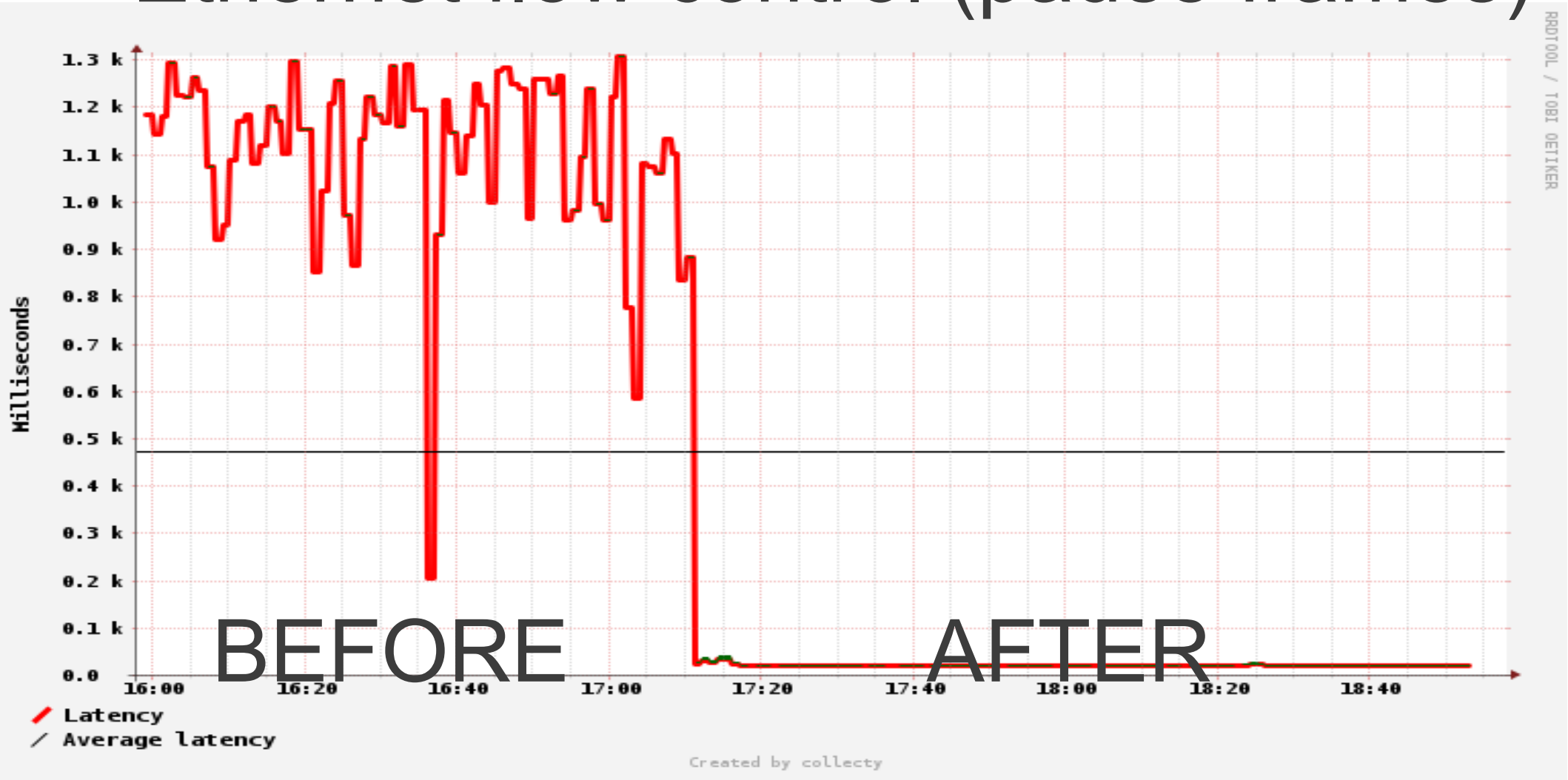
Sources


100 Mbit Latency under Load - SFQ vs QFQ vs PFIFO_FAST - vs 10 iperfs
Linux Ethernet Before BQL
PING RTT
QFQ 1
QFQ 2
SFQ 1
SFQ 2
PFIFO_FAST 1
PFIFO_FAST 2
RTT MS
120
100
80
60
40
20
0
10
20
30
40
50
60
Sun Jan 01 22:19:53 2012


TCP ping during bidirectional TCP streams - pfifo_fast 10mbit
TCP upload
TCP download
TCP rtrip (ms)
Mbits/s
ms
time (s)
0
2
4
6
8
10
12
0
10
20
30
40
50
60
70
80
0
5
10
15
20
PFIFO_FAST in Linux 3.6
on a BQL enabled driver
(Classic Drop Tail)


TCP ping during bidirectional TCP streams - fq_codel 10mbit
TCP upload
TCP download
TCP rtrip (ms)
Mbits/s
ms
time (s)
0
2
4
6
8
10
12
0
10
20
30
40
50
60
70
80
0
5
10
15
20
FQ_CODEL
on BQL enabled driver

ADSL modem latency w/FQ_Codel with: Ethernet flow control (pause frames)



- http://planet.ipfire.org/post/ipfire-2-13-tech-preview-fighting-bufferbloat

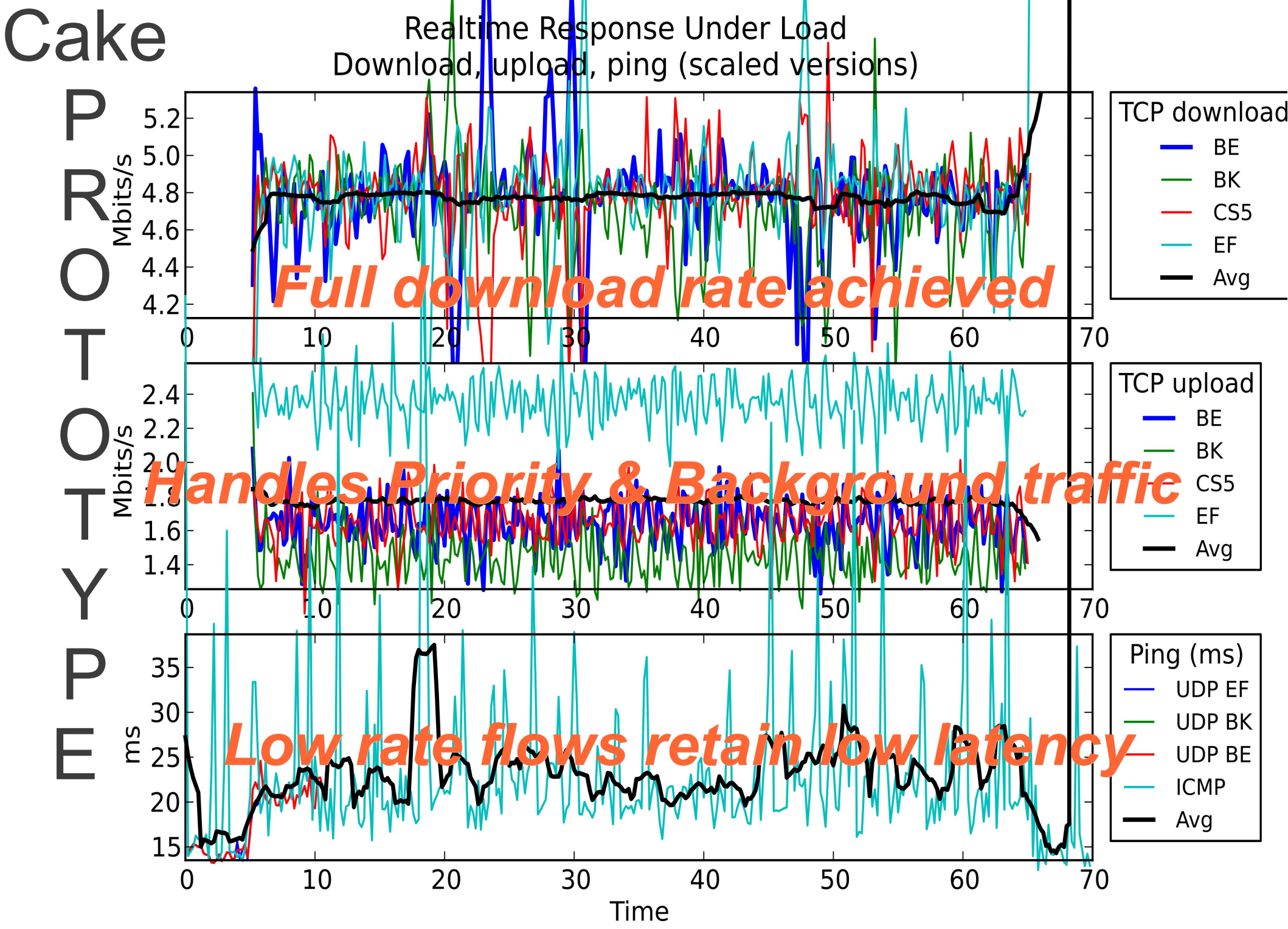
Ongoing Work

- Quest for a full replacement for PFIFO_Fast
- Adding support for software rate limiting
- Enhancements to codel/fq_codel
- Other delay based AQM systems (PIE, etc)
- Further research into the interrelationship of drop mechanisms and fair queuing
- Developing better tests
- Multiple papers
- Pouring codel and fq_codel derivatives into hardware
- Coaxing the universe to try it and deploy it

Cake

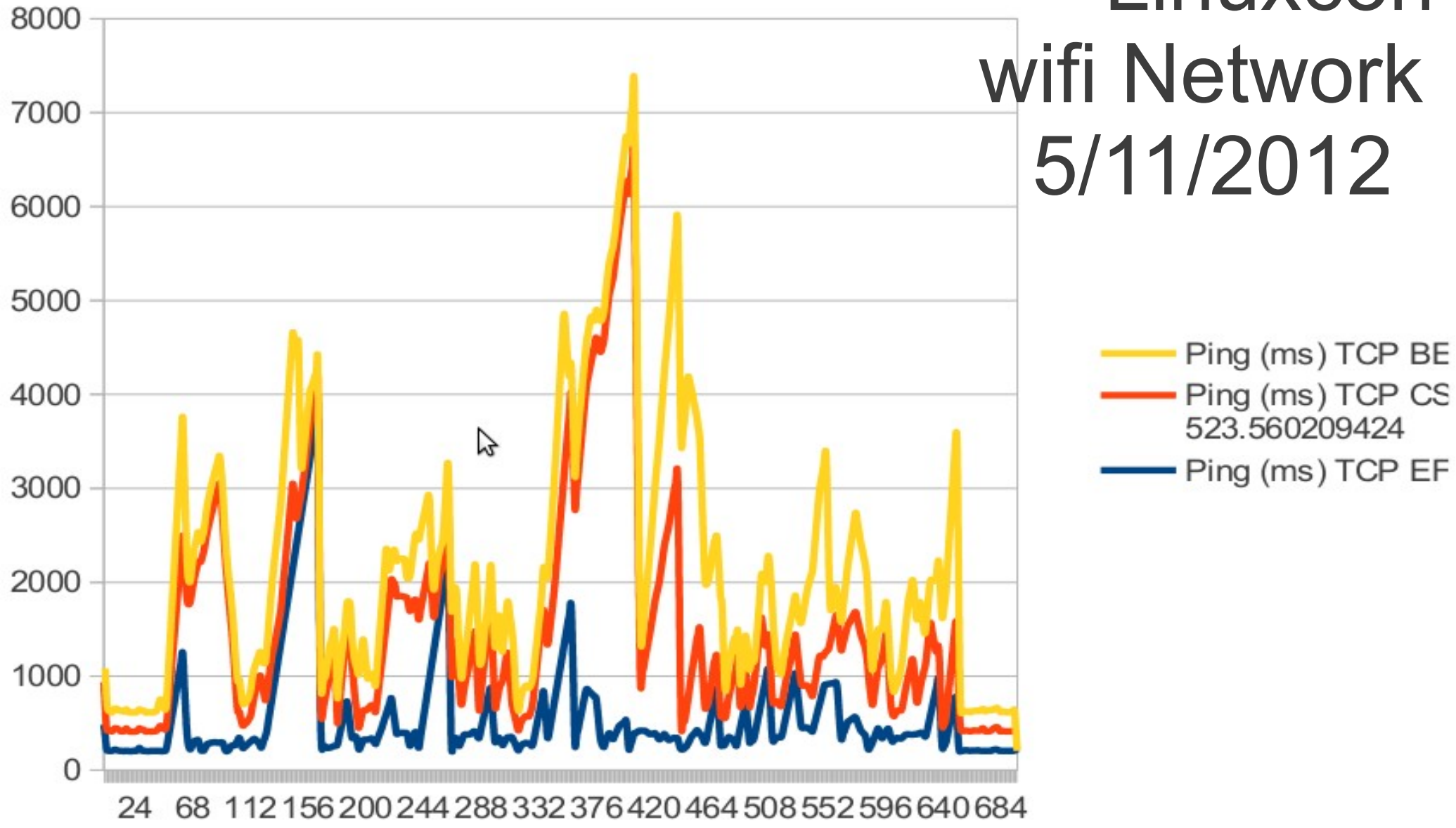
(Common Applications Kept Enhanced)

- Targetted at 32 bit embedded routers
- Chief characteristic: delay neutral classification/ prioritization on top of sfq_codel.
- Probably not a replacement for PFIFO_Fast (doesn't currently deal with offloads very well)
- Definitely will be insufficient for wifi...
- HTB + nfq_codel based prototypes deployed
- I'll release it when it's *baked*
- *Deployed in the Yurtlab Testbed*
- *Others are working on similar stuff*


Cake
PROTOTYPE
Realtime Response Under Load
Download, upload, ping (scaled versions)
Mbits/s
TCP download
BE
BK
CS5
EF
Avg
Full download rate achieved
TCP upload
Handles Priority & Background traffic
ms
Ping (ms)
UDP EF
UDP BK
UDP BE
ICMP
Low rate flows retain low latency
Time
Local/remote: lupin-gw/snapon.lab.bufferbloat.net - Time: 2013-01-11 12:01:51.421053 - Length/step: 60s/0.20s

Fixing bufferbloat on wireless has a *long* way to go.

Linuxcon
wifi Network
5/11/2012



Questions?

Bufferbloat.net Resources

Bufferbloat.net: http://bufferbloat.net
Email Lists: http://lists.bufferbloat.net
IRC Channel: #bufferbloat on chat.freenode.net
CeroWrt: http://www.bufferbloat.net/projects/cerowrt
Other talks: http://mirrors.bufferbloat.net/Talks
Jim Gettys Blog – http://gettys.wordpress.com

A big thanks to the bloat mailing list, Kathie, Van, and Eric,ISC, the CeroWrt contributors, OpenWrt, Google, and the Comcast Technology Research and Development Fund for their interest and support in the work!