Beyond the existences of Bufferbloat

Have we found the cure?

DevConf.cz Brno 2013

Jesper Dangaard Brouer
Senior Kernel Engineer, Red Hat
2013-02-23

Who am I

- Name: Jesper Dangaard Brouer
  - Linux Kernel Developer at Red Hat
  - Edu: Computer Science for Uni. Copenhagen
    - Focus on Network, Dist. sys and OS
  - Linux user since 1996, professional since 1998
    - Sysadm, Kernel Developer, Embedded
  - OpenSource projects, author of
      - ADSL-optimizer, CPAN IPTables::libiptc, IPTV-Analyzer
    - Patches accepted into
      - Linux kernel, iproute2, iptables, libpcap and Wireshark
  - Organizer of Netfilter Workshop 2013

What will we learn?

- Learn, artificial benchmarking by industry standard
  - gave us crappy Internet (under-load) with bufferbloat
- Learn, That the car queue analogy is dead
  - and the water-fountain is a better analogy
- Learn, the Linux Kernel is to blame
  - buffer are everywhere, also inside kernel and NICs
- Learn, how we have fixed the Kernel
  - with BQL (Byte Queue Limit) and TSQ (TCP Small Queue)
- Learn, about how CoDel ("coddle") works
  - the new holy-grail of bufferbloat AQM

What is bufferbloat

- Bufferbloat is excess buffering of packets
  - excessive buffer increase, provide *no added value*
  - resulting in high latency

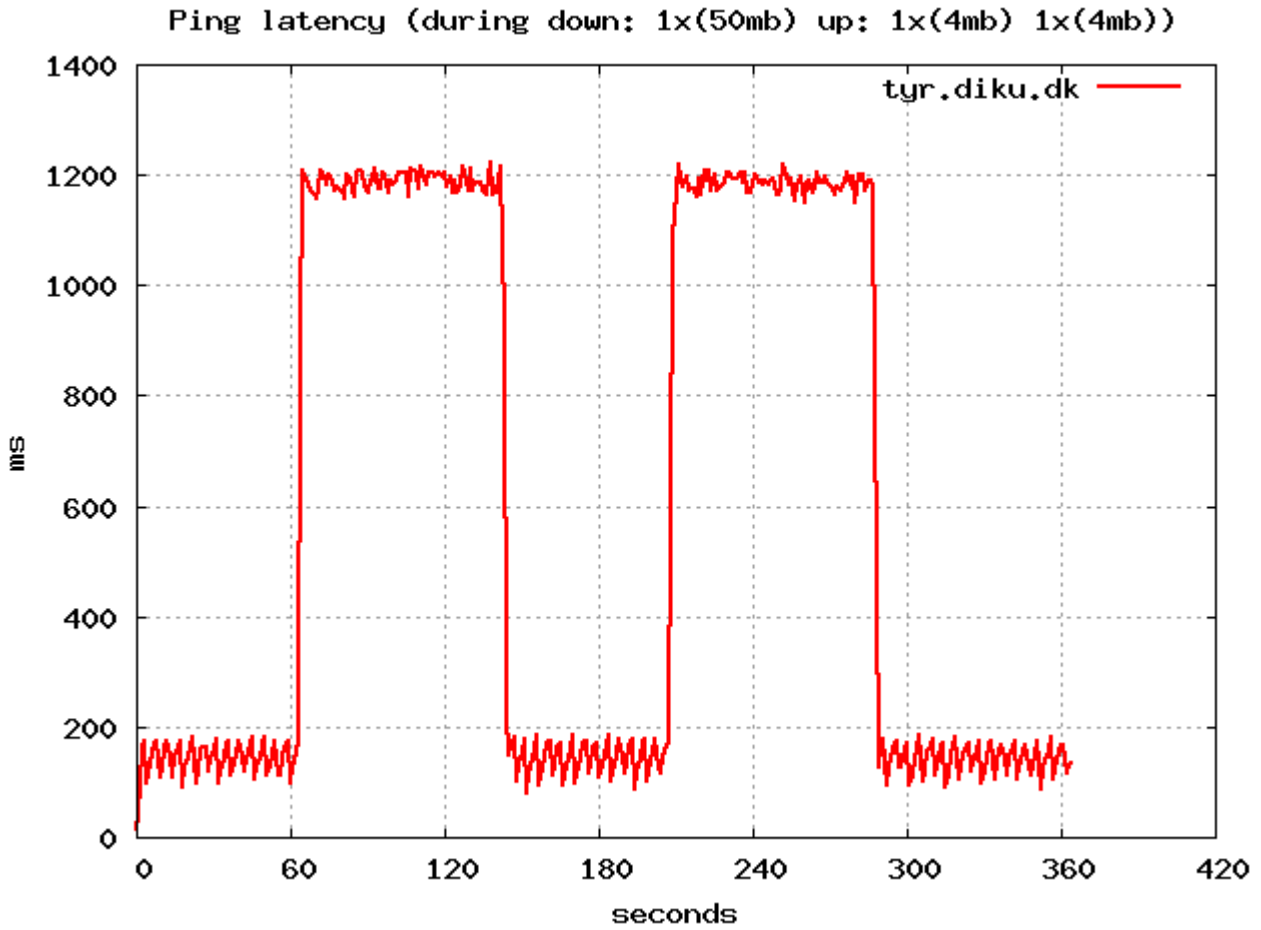
- Misguided attempt to avoid all packet loss?
- Not all packet loss is evil:
  - Packet loss is essential for correct operation
  - Need timely congestion notification (also for ECN)
    - Long queues mess up this feedback

Dark matter of the Internet

- Bufferbloat - Term now well established
  - By Jim Gettys (starting around 2009)
  - IETF Journal 2011: Bufferbloat: Dark Buffers in the Internet
  - http://www.internetsociety.org/articles/bufferbloat-dark-buffers-internet
- The dark matter/buffers of the Internet
  - Only exposed when queuing occurs
  - Never see these buffers until they start to fill
- As we have experienced, fixing it in one place
  - expose yet another level of buffering

How bad is it?

- Single TCP flow can fill up the queue
  - ADSL Upstream 512Kbit/s -> easy results in 1.2 sec delays

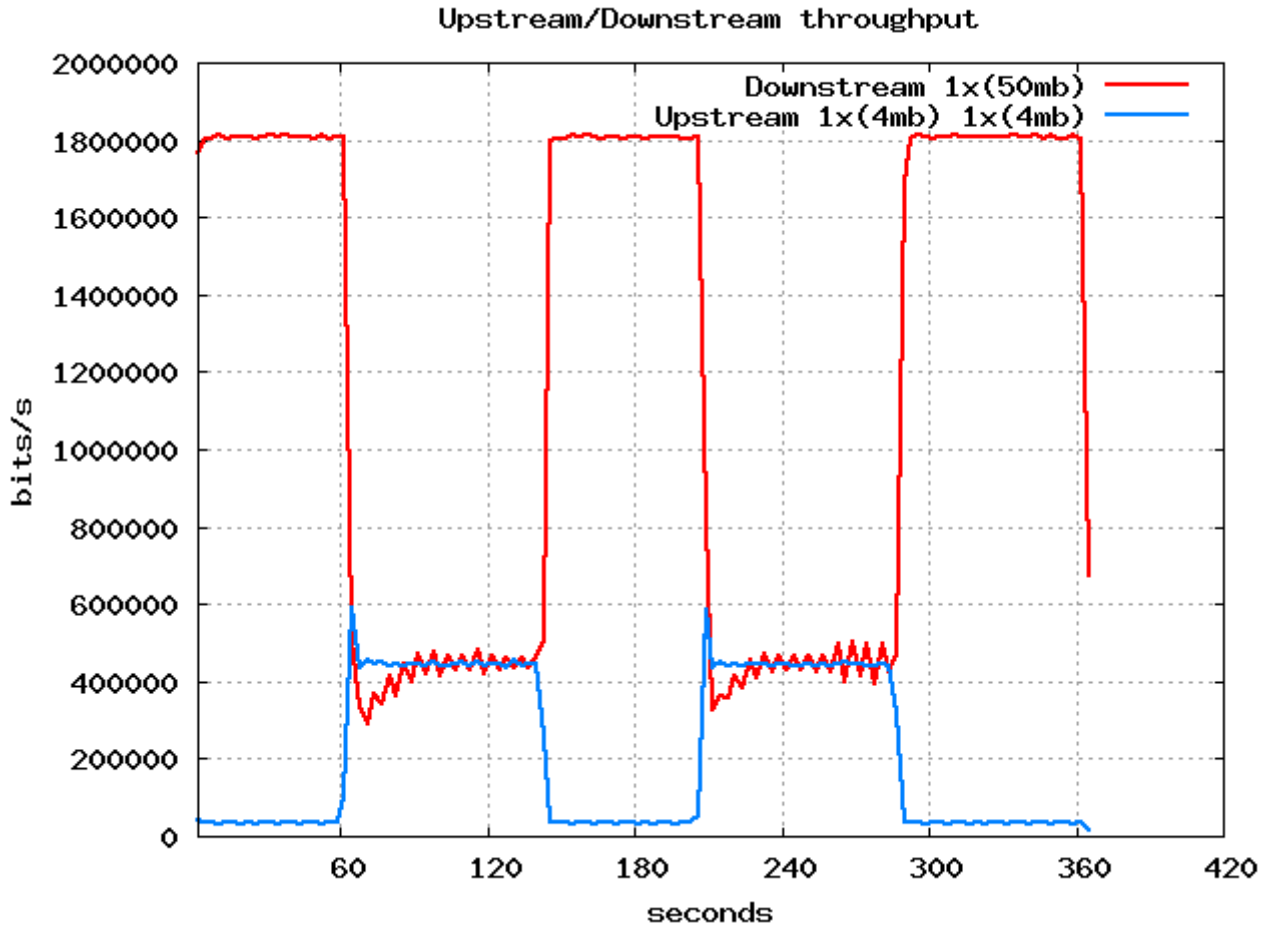


What was the queue size

- Buffer Bloat: The calculations
  - http://netoptimizer.blogspot.dk/2010/12/buffer-bloat-calculations.html
  - Bandwidth is **454 Kbit/s** (ADSL overhead)
  - Measured delay was **1138 ms**
  - **Buffer size**: 454 Kbit/s * 1138 ms = **64581 bytes**
    - **(Comes from TCP window size)**
- Transmission delay of a 1500 bytes (MTU) packet is
  - 1500 bytes / 454 Kbit/s = **26.34 ms**
- Surprise:
  - 64KBytes is a lot of queue on a 512 Kbit/s link

Bidirectional traffic also suffers

- ADSL link 2Mbit/s download 512 Kbit/s upload
  - ACK pkts for download, delayed on upload link



What happened

- Historically
  - Memory was expensive, small queues in HW
  - Memory is cheaper now, large queue in HW
- Nobody noticed bad effects, because
  - Used the wrong measurements
    - artificial lab benchmarks
      - (not representative of real traffic)

Bad industry benchmarking

- The official benchmark is always
  - Bandwidth and perhaps Packets Per Second
- Hey wait a minute
  - Forgot to measure latency
  - Forgot to do bidirectional testing
  - Forgot to look at Latency under load

"It's the Latency, Stupid"

--Stuart Cheshire, May 1996.

New Benchmarking needed

- Dave Taht is working on RRUL
  - Realtime Response Under Load test

- Toke Høiland-Jørgensen, test tools avail on github
  - https://github.com/tohojo/netperf-wrapper
  - http://akira.ruc.dk/~tohojo/bufferbloat/

Prerequisite for AQM (Active Queue Management)

"A modern AQM is just one piece of the solution to bufferbloat"

Cite: http://queue.acm.org/detail.cfm?id=2209336

- Do packet scheduling, at bottleneck
  - Need to be bufferbloat aware
- Active Queue Management (like CoDel) have no effect
  - When you don't control the queue

Good vs. bad queue

- Buffers and queues is a necessary part of the network
- Good queue:
  - Function as shock absorber, allow and handle burst
- Bad queue:
  - Long standing queue, Only adds delay

- CoDel: first AQM to distinguish
  - between good vs. bad queue

Problem: Buffers are everywhere

- Buffers are everywhere!
  - Also inside the kernel network stack
  - Hidden queues in the NIC
  - Wireless drivers especially bad (do packet aggregates)
- Cannot deploy any AQM e.g. CoDel
  - before we have control of the queue
  - packet queue must form at the qdisc level
- Two recent techniques in the kernel
  - **BQL** - **B**yte **Q**ueue **L**imit (by Tom Herbert/Google)
  - **TSQ** - **T**CP **S**mall **Q**ueue (by Eric Dumazet/Google)

BQL motivation

- Goal of Byte Queue Limit (BQL):
  - reduce latency caused by excessive queuing HW
  - without sacrificing throughput
- BQL essential for CoDel/AQM
  - Don't want queuing in the HW device
  - Need to move queue to the qdisc

What BQL does

- Try to avoid "over" queuing in the NIC
- Dynamic adjust queuing to what NIC is able to TX
  - by tracking TX completion
- Based on number of **bytes** (the NIC dequeued recently)
  - Better than number of packets
    - as bytes correlates with the transmission delay
- Not strict, allow to be exceeded
- Tracking generally, grows fast and shrinks slowly

BQL needs driver modification

- One problem with BQL
  - Need to modify every NIC driver
  - Thus, not all drivers support BQL yet
- Use API (include/linux/netdevice.h):
  - netdev_tx_sent_queue()
    - Called to inform stack when packets are queued
  - netdev_tx_completed_queue()
    - Called at end of transmit completion to inform stack of number of bytes processed
  - netdev_tx_reset_queue()
    - optional to reset state in the stack

BQL kernel details

- Kernel details:
  - It uses the __QUEUE_STATE_STACK_XOFF bit
  - Based on Dynamic Queue Limits (DQL) API
    - include/linux/dynamic_queue_limits.h
  - Maintained per TX HW queue

TSQ - TCP Small Queues

- Queuing also occurred inside TCP stack
  - Eric Dumazet solved, with TCP Small Queues (TSQ)
- Sockets are marked throttled
  - if amount of data waiting to be transmitted (sk_wmem_alloc)
    - is above limit
  - Use sk_buff destructor to "open-up" (needs tasklet tricks)
  - (more info see http://lwn.net/Articles/507065/)
- Default size
  - Minimum allow two packets
  - Due to packet aggregation TSO / GSO
  - The limit is 2x 64K bytes
  - Adjust via: /proc/sys/net/ipv4/tcp_limit_output_bytes

CoDel: The AQM grail for bufferbloat?

- After fixing the Linux stack (with BQL and TSQ)
  - Queuing occur in the right place (in the kernel)
- Need bufferbloat aware AQM algorithm
  - CoDel: Controlling Queue Delay
    - by Kathleen Nichols and Van Jacobson
- Van Jacobson great talk about CoDel
  - at IETF84 (Vancouver 2012)
    - Video/sound and Slides

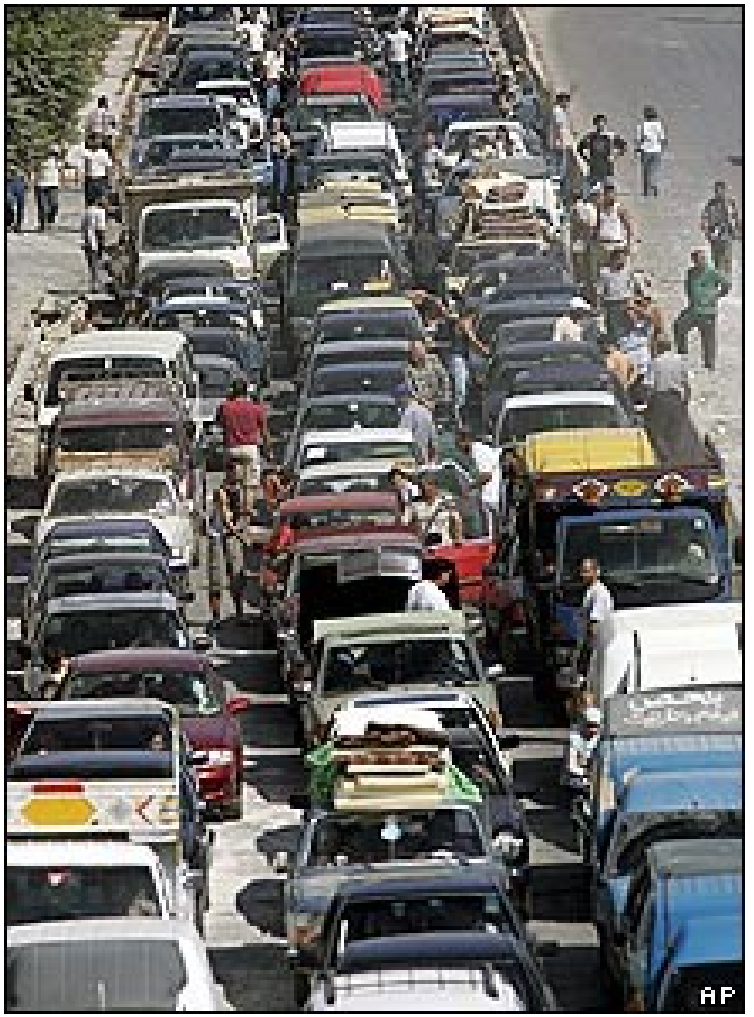
Implemented in Linux 3.5 (by Eric Dumazet and Dave Taht)

and avail for ns2 and ns3

Van Jacobson killed the car analogy

The car queue analogy is broken

- The outside observed delays are the same
- The inside dynamics are completely different



Van Jacobson water-fountain(1/2)

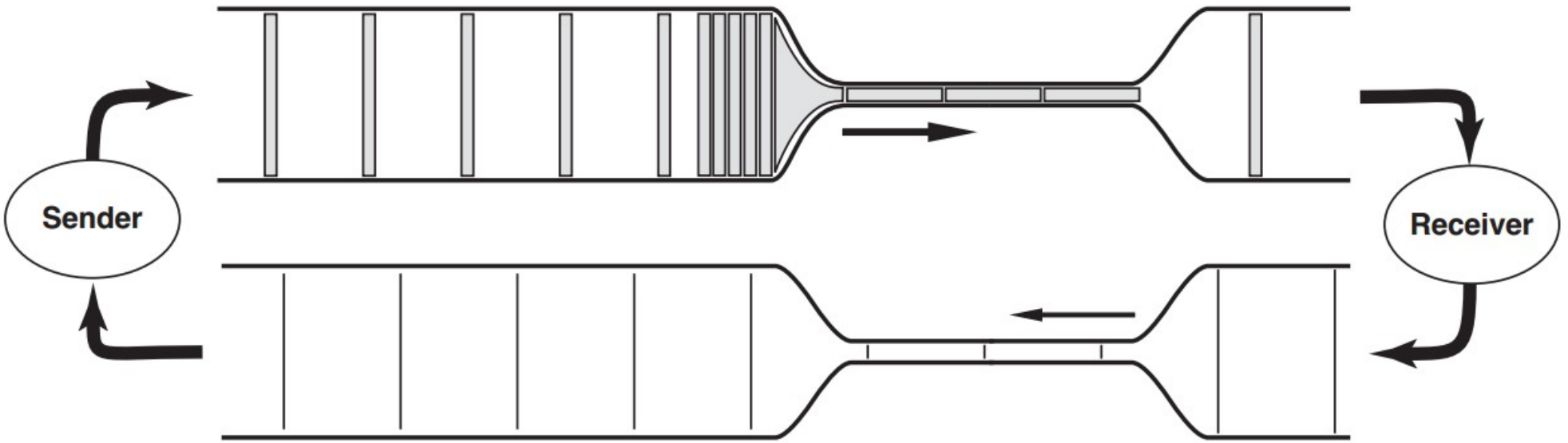
Better analogy is a water-fountain with a pump

- It is a closed loop servo system
- This is how TCP works
  - due TCP-ACK to flow balance

Van Jacobson water-fountain(2/2)

- The water-level in the pond is NOT affected by:
  - Flow rate, pump pressure or bigger pipes
- Can change the water-level by
  - adding or removing water
    - till the overflow drain
- The pond is the queue (in this closed loop servo system)
  - its the backlog for the pump to process
- Don't need a huge pond to run the fountain
  - just minimum to keep the pump from running dry

TCP self-clocking Window Flow Control



- Packets are streached out in time, after bottleneck time-space is maintained
- The ACK feedback maintains a steady state, queue formed stays constant
- The 5 packet queue is the “pond” in the water-fountain / bufferbloat issue
- For a flow, only a single bottleneck on path, due to time spacing

CoDel design goal: Do no harm

- Top level design goal: **Do no harm**
  - Only turns it self "on" when there is a problem
  - either does nothing
  - or reduces delay without affecting throughput
- Makes it perfect for wide deployment

CoDel: Good vs. bad queue

- CoDel basically solves determining
  - the difference between good vs. bad queue
- Good queue: Function as shock absorber, allow and handle burst
- Bad queue: Long standing queue, only adds delay

- Queuing viewed as a servo loop feedback system
  - Observe that (TCP) bursts go away in a RTT
  - *Queue that does not go away in a RTT* is **bad queue**
  - Good queue is min queue size over a sliding window

CoDel: Queue size in bytes is wrong

- Measure queue size in bytes (is the traditional method)
  - It is bad because:
    - we really just care about the delay queue causes
    - to compute delay, need to know (output) bandwidth
    - (and bandwidth can change over time)
- Instead look at time-in-queue rather than bytes
  - Easy to directly measure (delay of a single packet)
  - Termed: *Sojourn Time* ("a temporary stay")

Byte queue does not scale

- Classical byte counting
  - Have coupling to a shared state (bytes in queue)
  - between enqueue and dequeue
  - requiring locking (bad scaling)

CoDel: Sojourn Time (time-in-queue)

- The beauty of measuring time-in-queue (sojourn time)
  - No locking required
    - Simply timestamp SKB/packet on enqueue
    - Calc time-in-queue (sojourn time) at dequeue time
    - Be smart at dequeue time
  - Basically allow unlimited enqueue
    - Not a problem, memory was cheap right
  - Works for time-varying bandwidth
    - e.g. wireless and shared links

Sojourn Multi-Queue behavior

- Surprising good Multi-Queue behavior
  - MQ HW does not affect time-in-queue measurement
  - Packets will arrive at the same rate with MQ HW
  - Output rate/bandwidth is the same
  - Thus, time-in-queue is same measurement
- Simply measured time used by the entire system
  - which is better than byte-measuring what happened in a specific queue
- Plus no-locking also gives very good MQ behavior

CoDel: Min queue size needed

- Cannot let link go idle
  - Min 1 MTU packet size time over bottleneck link
    - Due to TCP self-clocking MTU delay at bottleneck
  - Also need 2 packets, as might not arrive well spaced
- How much more queue will increase throughput?
  - and not cause too much in delays
- TCP control law affect us
  - Packet drop cuts TCP window in half
    - Cutting too small window hurts throughput
    - as it takes too long (RTTs) to ramp up

CoDel: The trade-off setpoint target

- Need a trade-off between bandwidth-and-delay
  - Van Jacobson quantifies this trade-off
  - See his talk explaining this (28 min slide 17,18,19,20)
- Minimum sojourn time (setpoint target)
  - must be 5% of the (nominal) target RTT
  - which in CoDel is 100 ms, giving 5 ms
  - This yields substantial utilization improvement
    - for small added delay.
- Nominal RTT target should be bigger than any real RTT
  - of connections going through the box.

CoDel: Simplified algorithm

- Oversimplified version of the basic algorithm
  - If sojourn time > 5 ms (setpoint target)
    - for 100 ms nominal target RTT
  - Then begin to drop packets
    - increasing according to a control law
      - that is TCP friendly
    - basically: drop more and more packets
      - if the queue stays congested
- Real algo see:
  - http://queue.acm.org/detail.cfm?id=2209336

CoDel: Please use **fq_codel**

- fq_codel: Fair Queue + CoDel
  - Almost: SFQ + CoDel
  - But smarter, distinguish "new" vs. "old" flows
  - Result: favors interactive flows

- Try it! - one-liner enable via:

```
tc qdisc add dev ethX root fq_codel
```


CoDel: Deployment issue

- Home gateway is the bottleneck
  - queue occur *inside* the ADSL or Cable modem
- Move the queue by introducing another bottleneck
  - Another router box in front
  - With ratelimiting to push-back/obtain queue control
  - Sacrifice bandwidth to become the bottleneck link
  - Also looses dynamic adjust, e.g. "boost" products
- REMEMBER: Take ADSL link layer into account
  - Simply use tc “linklayer” option
    - I implemented that back in 2005

ADSL linklayer overhead

How much bandwidth to sacrifice?

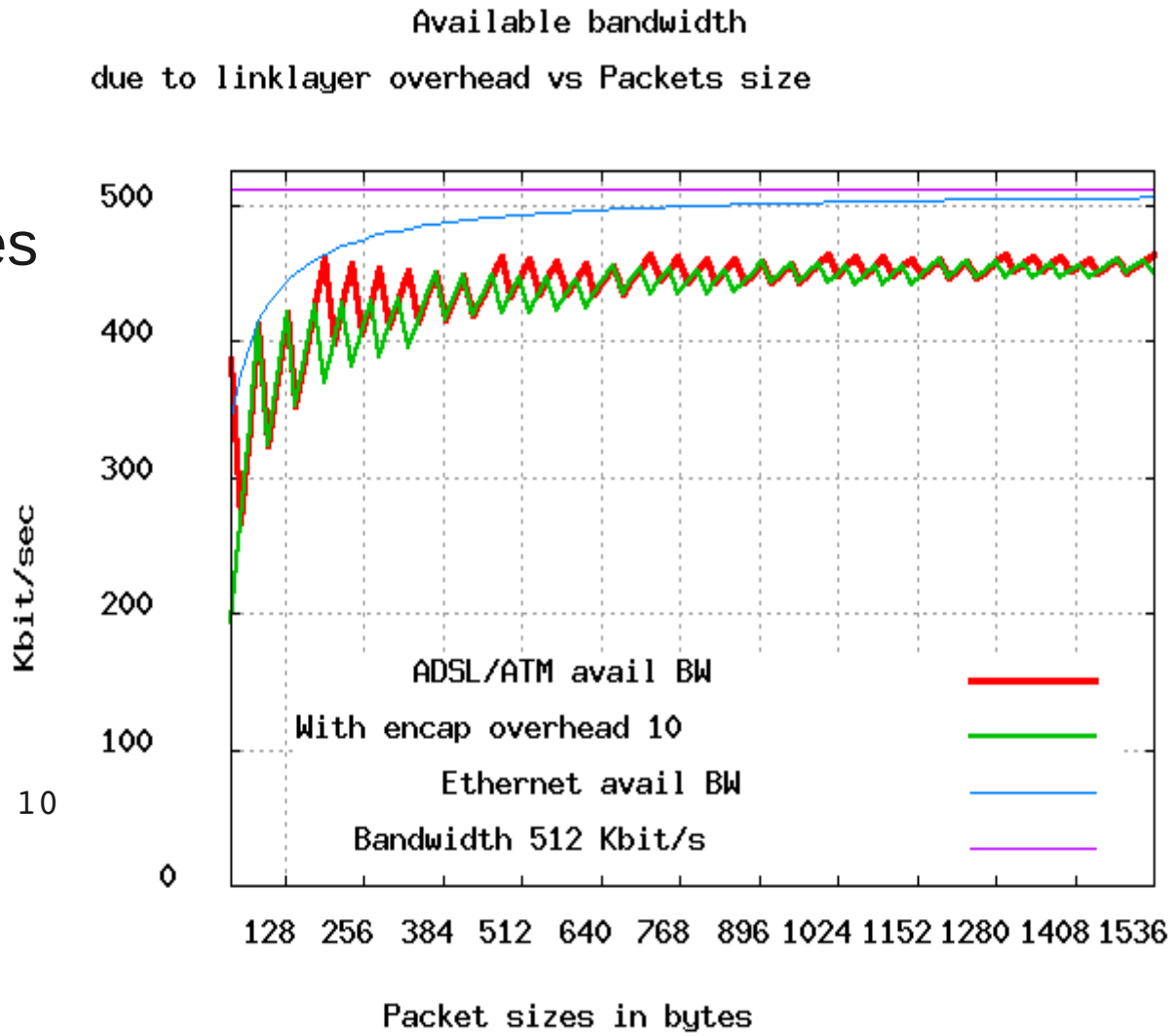
ACK packets on 40 bytes

- uses 106 bytes on wire

  (2x ATM 53 bytes frames)
- Due to ADSL encap overhead (40+10 > 48)
- An 62% overhead, for a very common packet

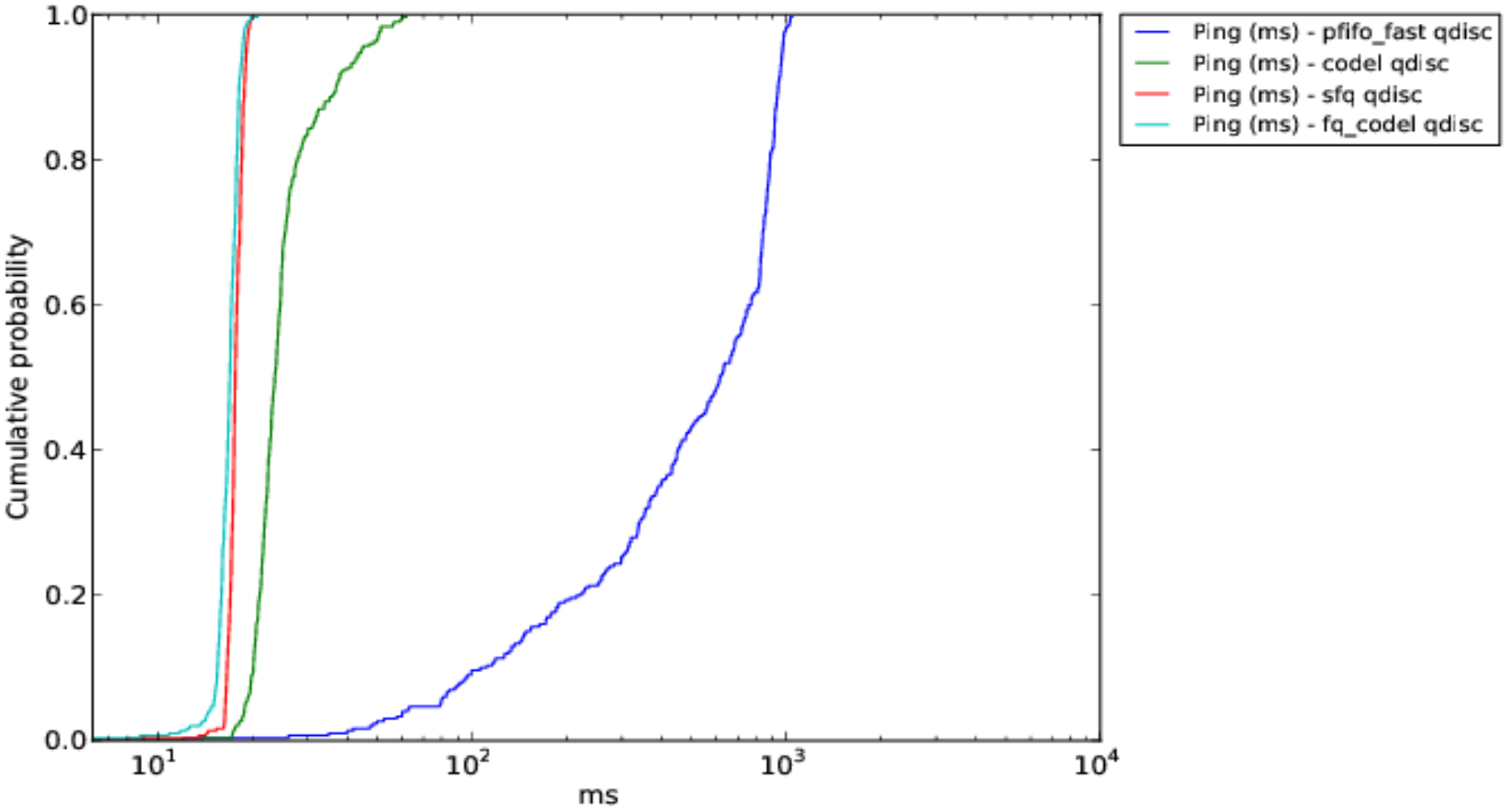
Fix, just add to your "`tc`" command:

```
tc ... linklayer ADSL overhead 10
```



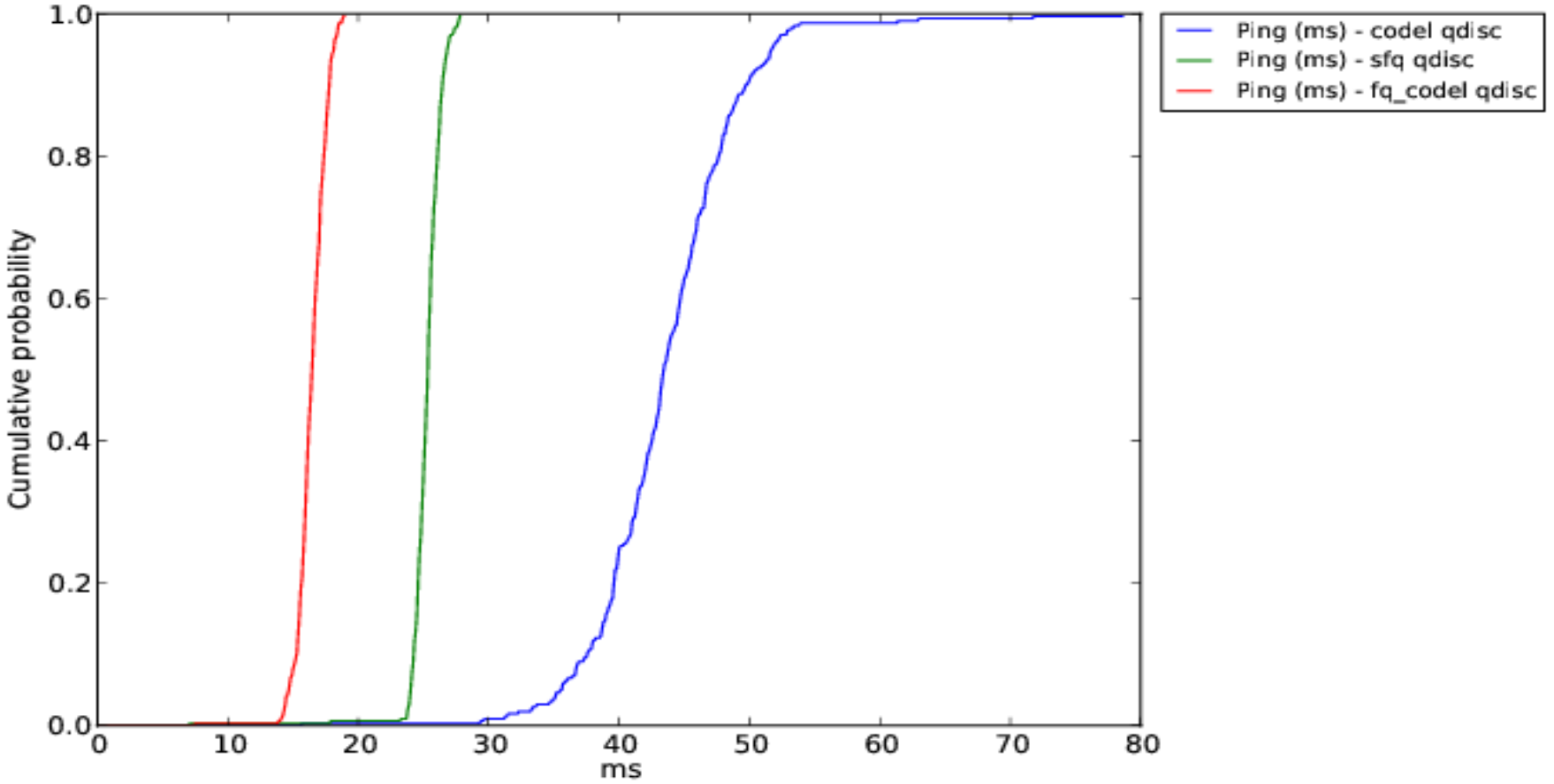
AQM comparison results (1/2)

- Toke's CDF (Cumulative Distribution Function) results
  - CDF plot of ping time distributions for bidirectional TCP test



AQM comparison results (2/2)

- Why to choose fq_codel
  - CDF plot of ping time distributions for the RRUL test
    - fq_codel is a clear winner



Conclusions

- Have we found the cure?
  - Yes, Have fixed internal kernel stack buffering
  - Yes, CoDel is bufferbloat aware
  - Yes, **but** getting it deployed is the challenge

Future work

- What is missing
  - Update every home router on the planet...
  - Change fq_codel to be default qdisc
  - Minor stuff
    - More work on wireless (works well, but HW problems)
    - More work on slow link
    - Tuning CoDel drop restart point
  - 3G deployment/fixing also needed

The End

- Questions?