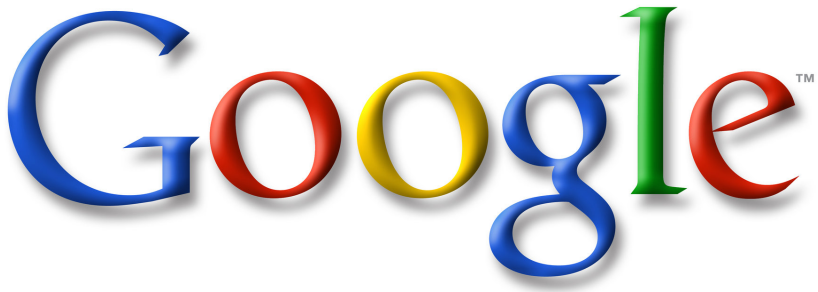
# Byte Queue Limits

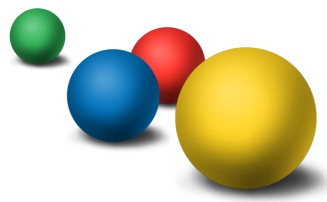
Tomáš Hrubý

August 24, 2012



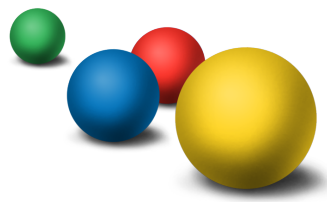
# BQL - Motivation


- Packets spend enough time enqueued within the stack
- When a packet gets to a NIC it is enqueued again
- HW queue length in TX descriptors are decided ad-hoc
- The queue length is not directly related to bytes or time



# What BQL does

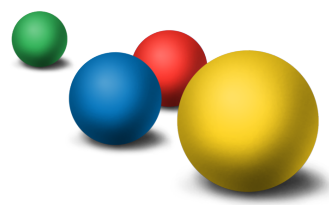

- Reduces unnecessary queuing in the NIC by setting the "right" limit
- Limits queuing to what the NIC is able to TX
- Pushes queuing from NIC to the network stack
- Reduces latency



# The limit

Google

- Based on bytes the NIC dequeued recently
- Changes dynamically
- Soft - allows to be exceeded once (overlimit)
- Grows fast and shrinks slowly



# The algorithm

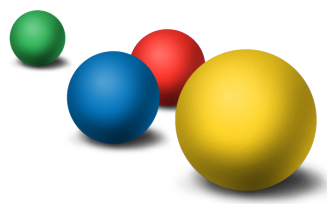
Google

## Generic implementation in dynamic queue limits (DQL) library

- Packets are queued up to a dynamic limit
- Packets should be periodically retired by a completion event
- Limit is increased whenever the NIC starves (dequeues too fast)
- Limit is decreased when the NIC cannot keep up for a while

## Starvation

- When queuing is blocked (limit has been reached)
- All data were consumed
- Completion has not happened yet



# Increasing the limit


Interval - the period between to invocations of `dql_completed()`

`ovlimit && !inprogress`

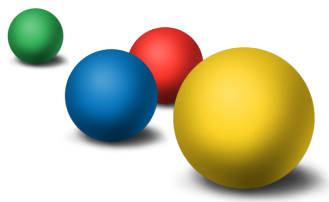
- Queue was certainly over limit in this interval

`dql->prev_ovlimit && all_prev_completed`

- Loose condition, queue might have been starving during the interval

## Increment

```
* When queue is starved increase the limit by the amount
* of bytes both sent and completed in the last interval,
* plus any previous over-limit.
*/
limit += POSDIFF(completed, dql->prev_num_queued) +
         dql->prev_ovlimit;
```



- Limit grows aggressively, can more than double
- Limit stops increasing when it is sufficient

# Decreasing the limit

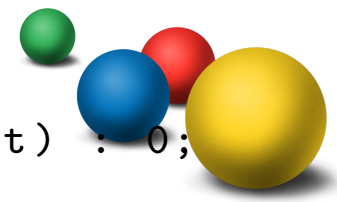

`inprogress && prev_inprogress && !all_prev_completed`

- Slack is only detected when the queue is busy for the entire interval
- Decrease happens only when slacking for a period of time (1s)
- Conservative - decrease by the minimal slack over the period
- Any increase in the limit resets the period

## 2 flavors of slack

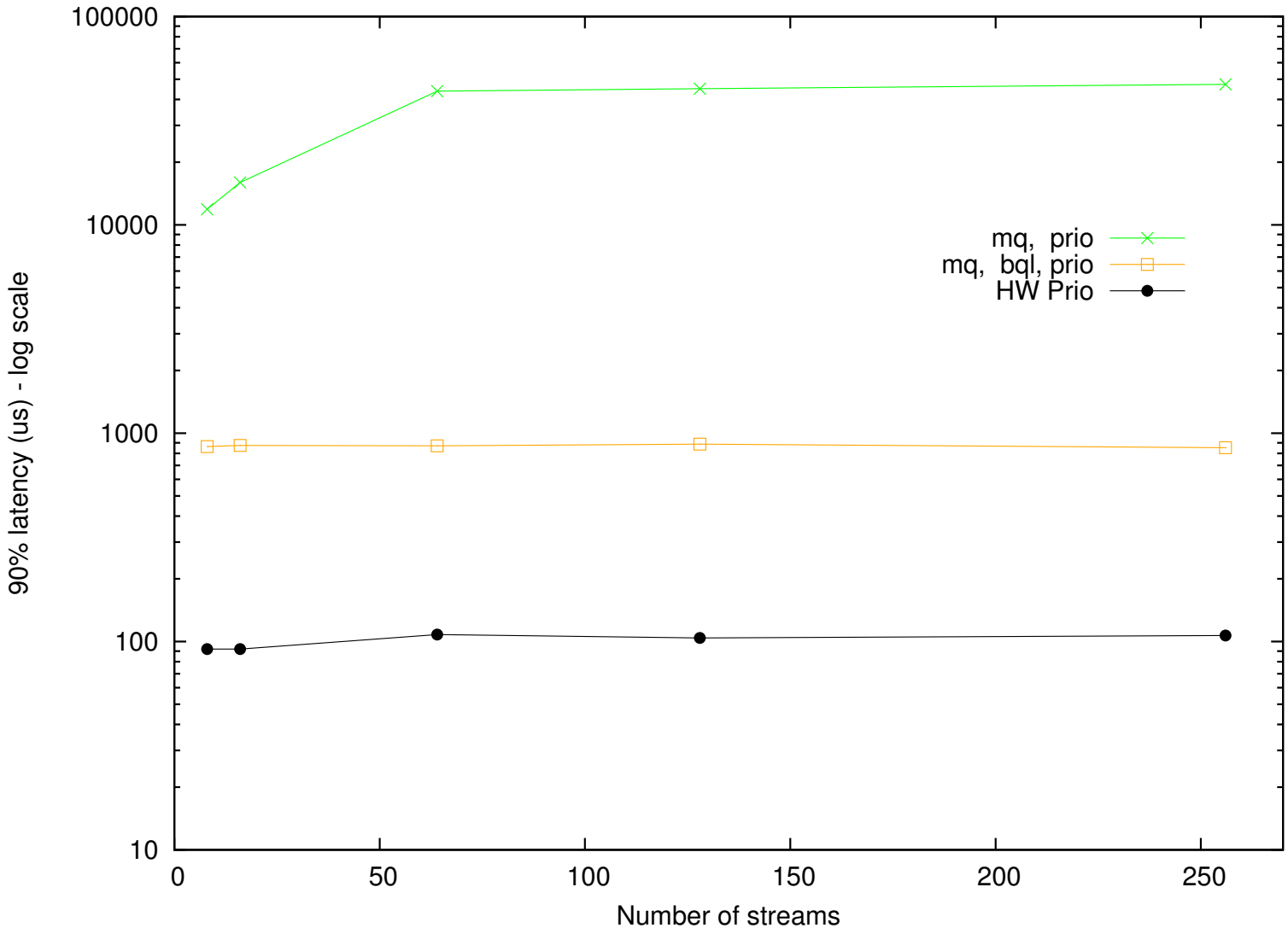
- ```
  slack = POSDIFF(limit + dql->prev_ovlimit,
      2 * (completed - dql->num_completed));
  ```

- ```
  slack_last_objs = dql->prev_ovlimit ?
      POSDIFF(dql->prev_last_obj_cnt, dql->prev_ovlimit) : 0;
  ```



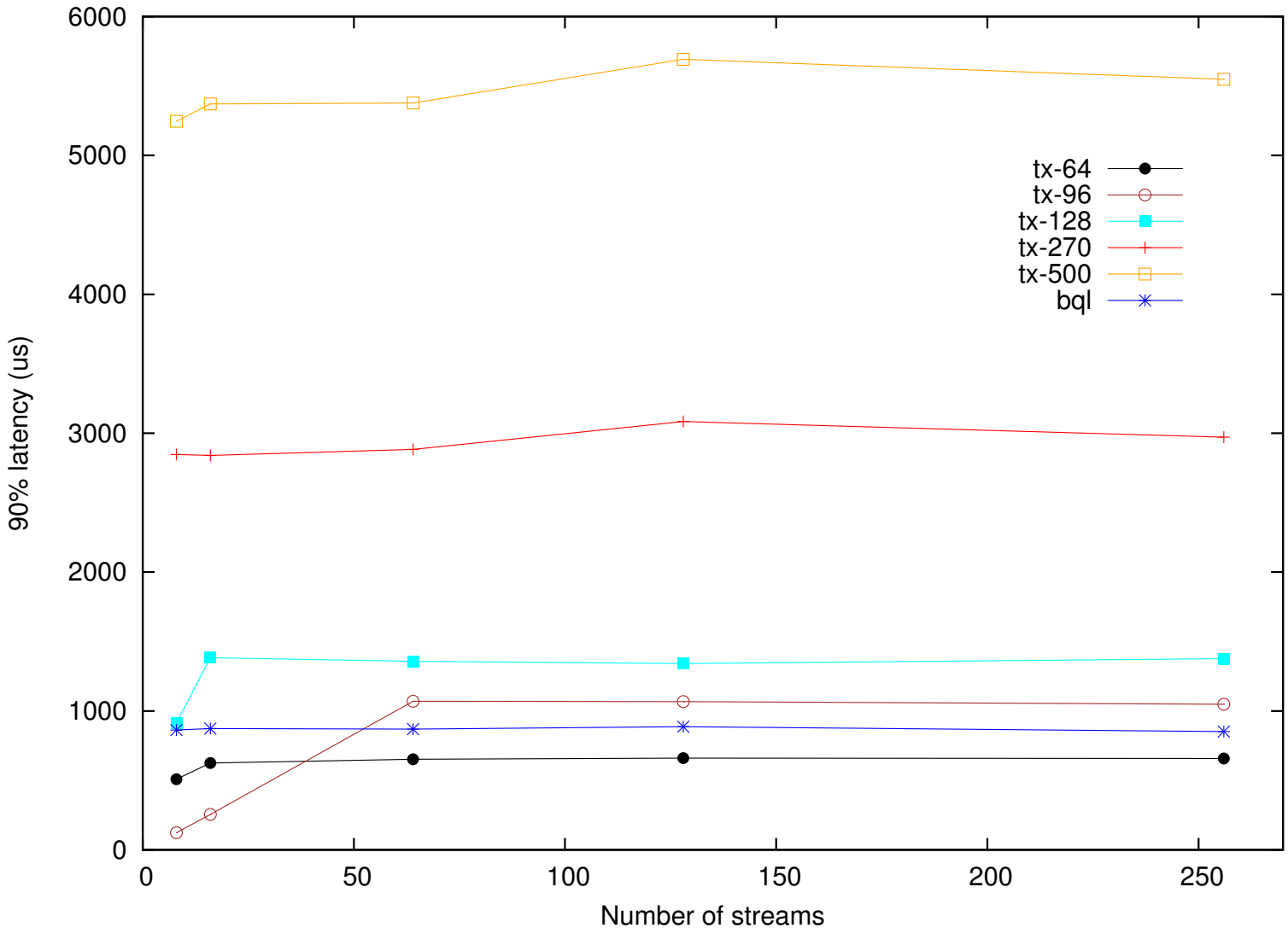
# BQL benefits

## Reduced latency



# BQL benefits

## The "right" TX right size



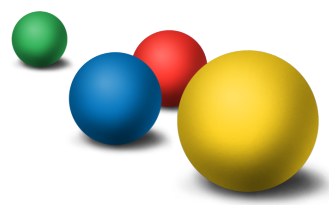
# BQL benefits


## The "right" TX Right size

| Packet size | Bit rate Mbps | Pkt rate kpps | Q size bytes | Q size pkts |
|---|---|---|---|---|
| 66 | ~1025 | ~1490 | ~9577 | ~145 |
| 1514 | ~9995 | ~810 | ~122974 | ~81 |

single TX process, single queue only

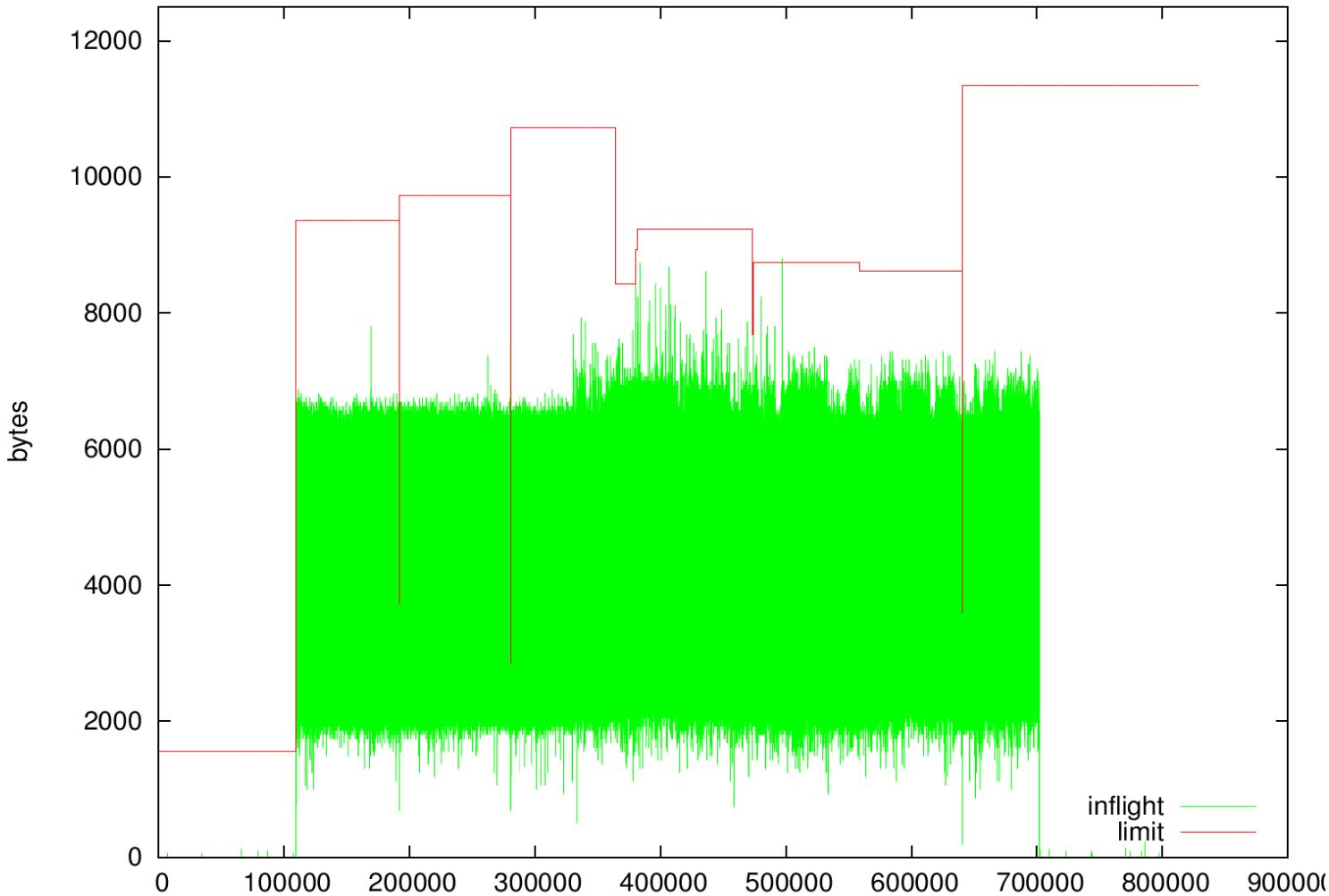
- Default bnx2x TX descriptor queue size is 4078
- Default ixgbe TX descriptor queue size is 4096
- Using BQL, the actually used size is significantly smaller

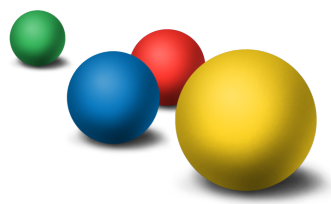


# BQL in action

Google

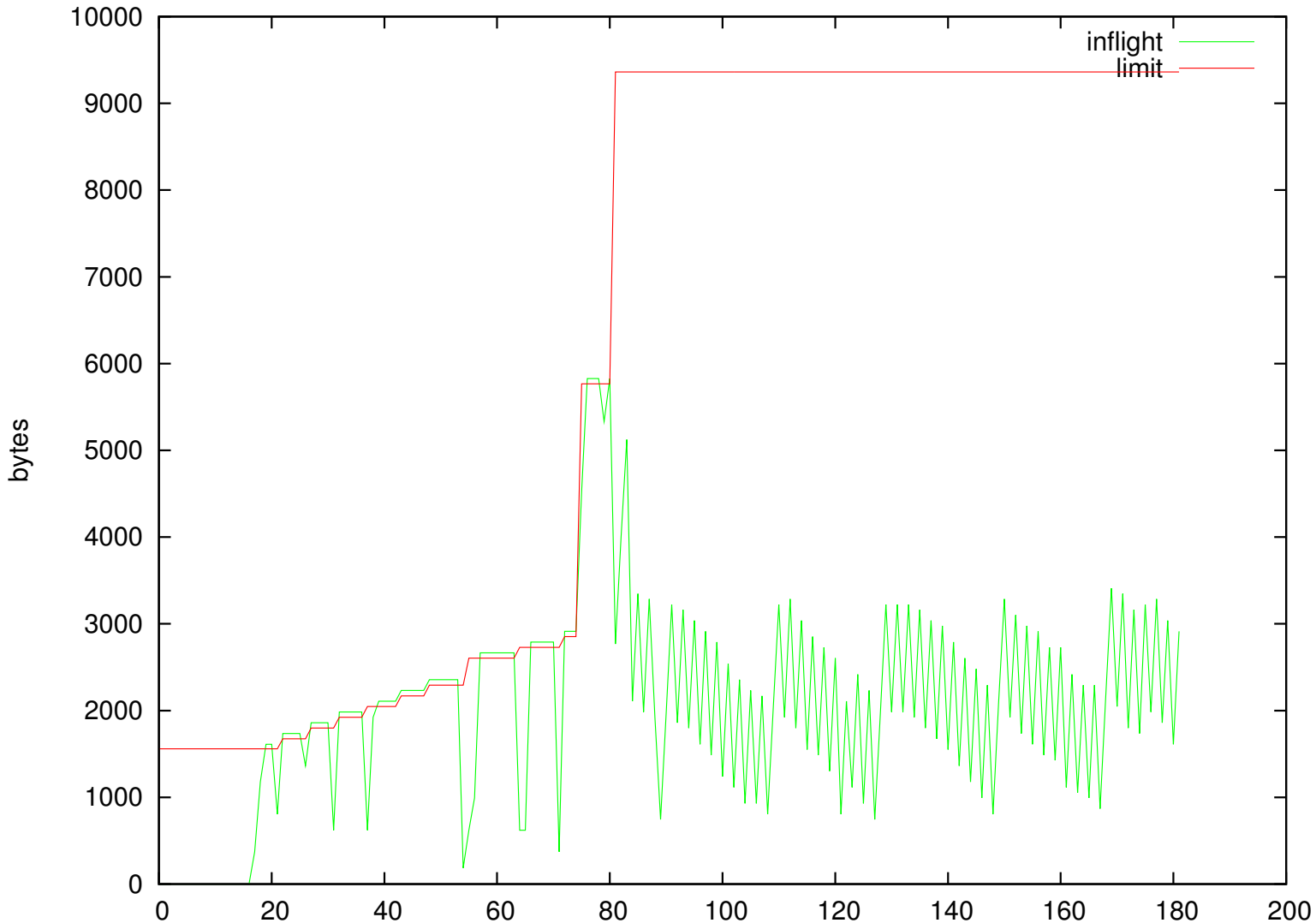
## Minimal packets ~1Gbps



# BQL in action

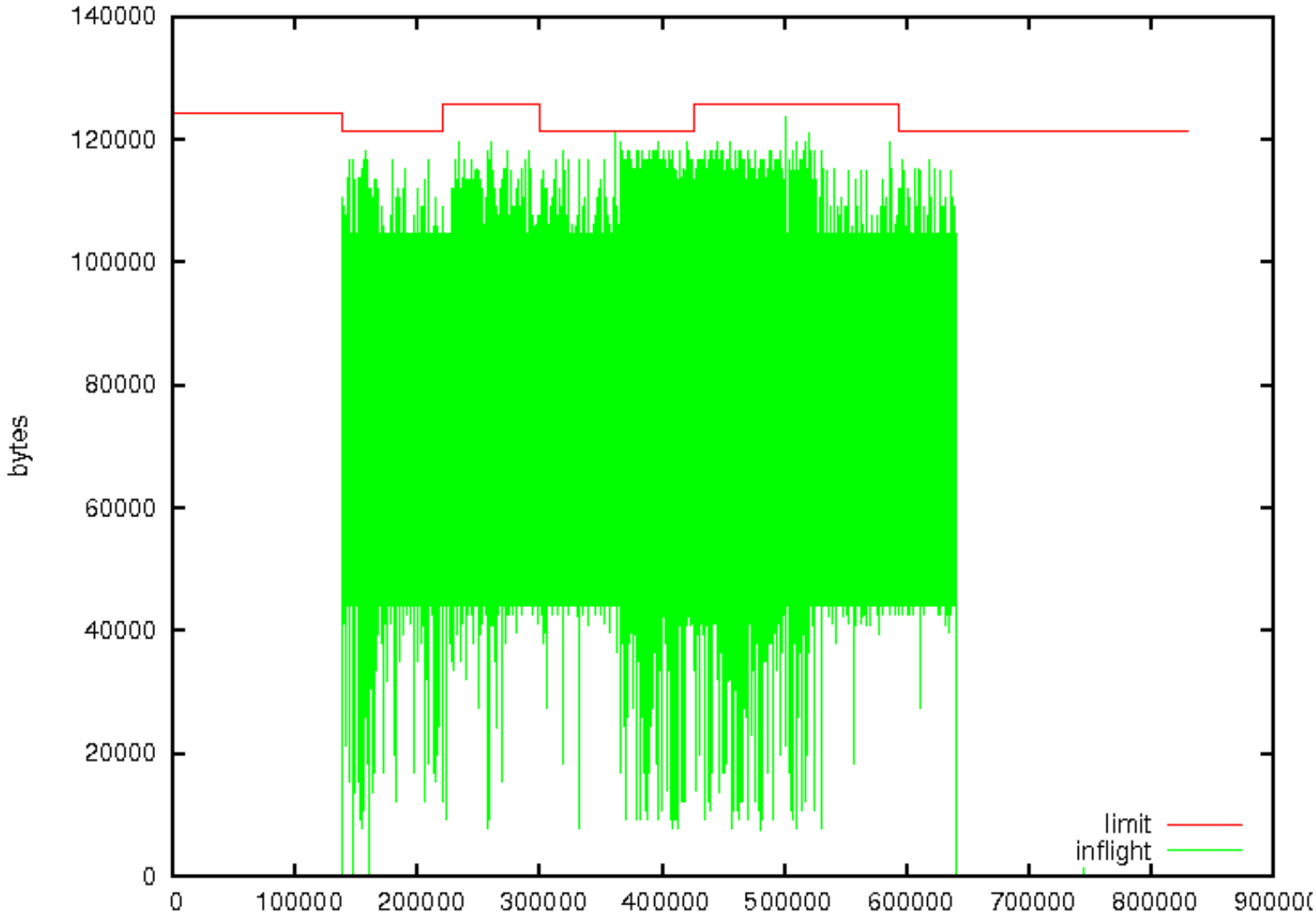
## Minimal packets ~1Gbps - initial ramp up



# BQL in action

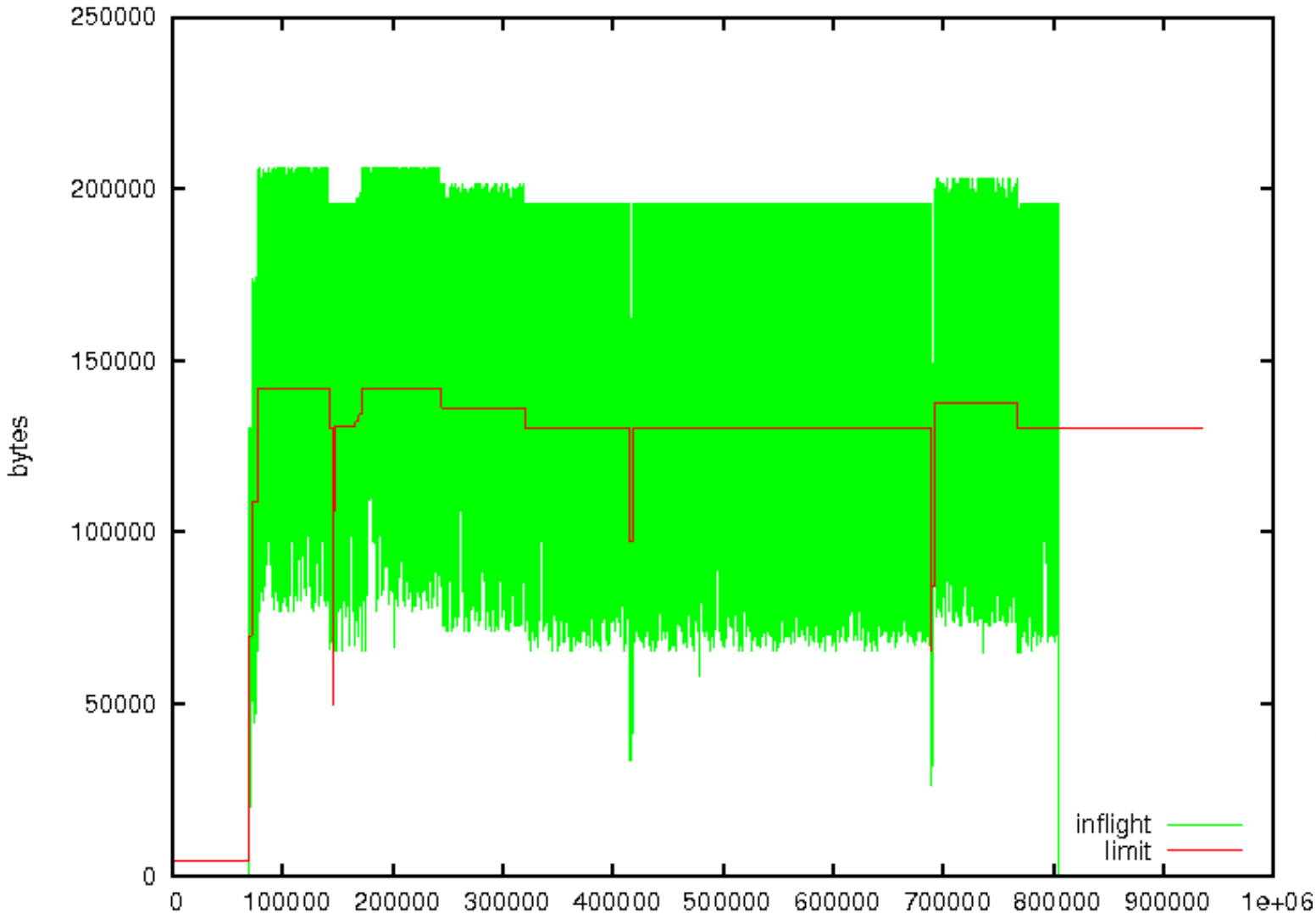
Google

## Maximal packets ~10Gbps



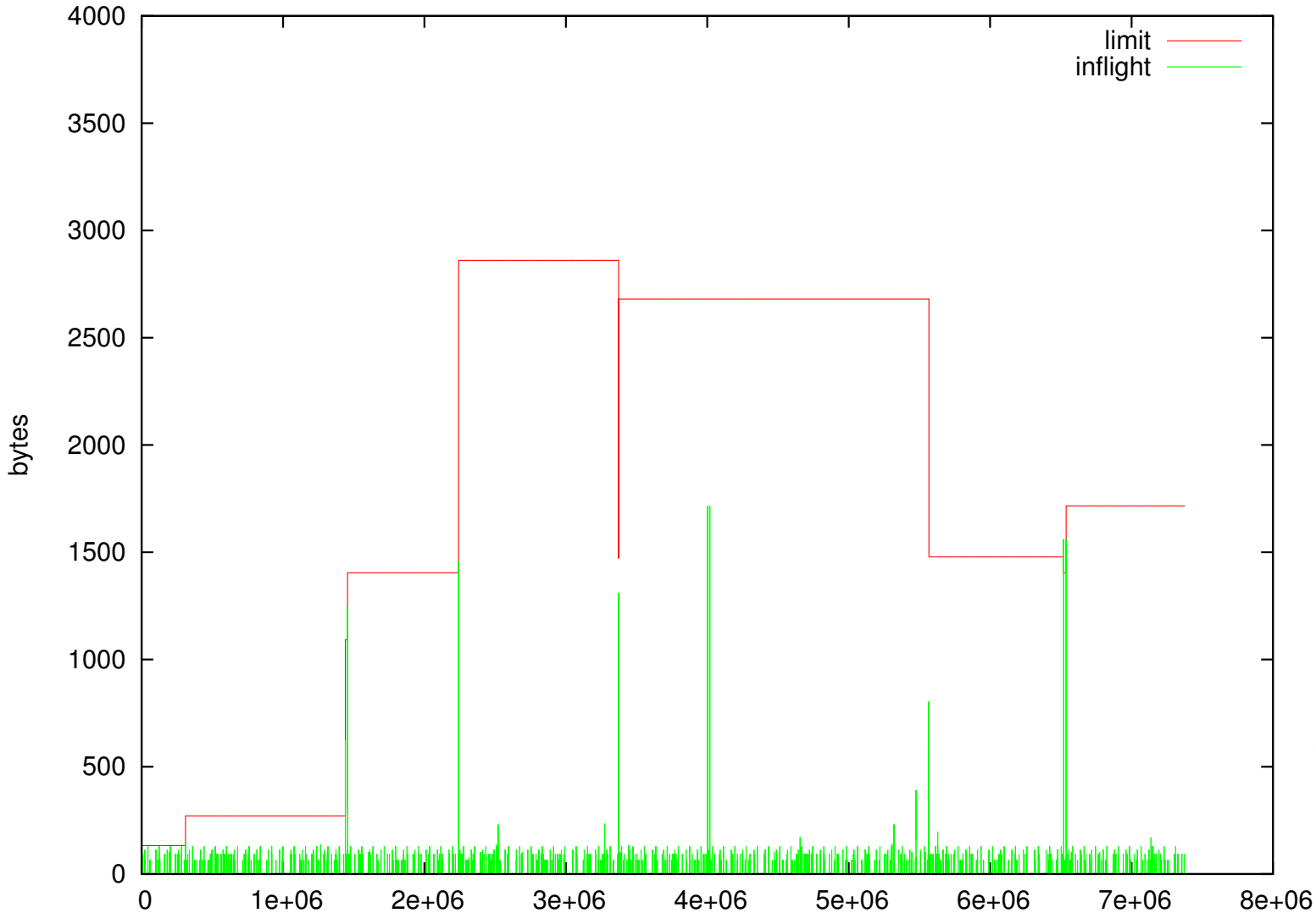
# BQL in action

## 1 TCP stream ~10Gbps (TSO on)



# BQL in action

## Idle noise



# BQL problems

Google

## The good news

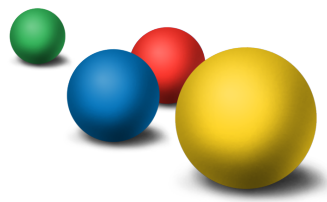
No case where BQL on would work worse than BQL off
(except non-cooperative hw)

## The not so bad news

In some situations, BQL can be tricked to virtually turning itself off by setting the limit to high.

## What is the trick?

Kernel does not guarantee that completion is called periodically



# Completion not periodical

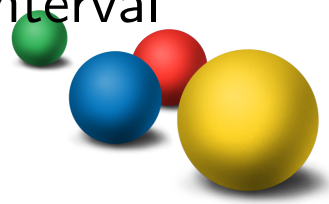
Google

## Periodicity required

- Impossible to say whether the interval was smaller or longer
- Shorter interval looks like congestion $\rightarrow$ slack
- Longer interval looks like the NIC can dequeue more $\rightarrow$ limit too low

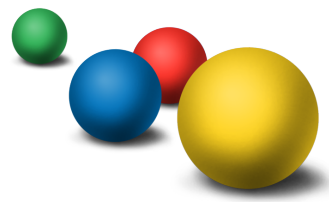
## Completion called in NAPI loop

- Some driver (e.g., bnx2x, gianfar) retire as much as they can
- Last iteration usually retires much less
- Retiring when exiting NAPI may lead to excessively long interval
- Different interval between iterations and new polling start



# Excessively Large limit

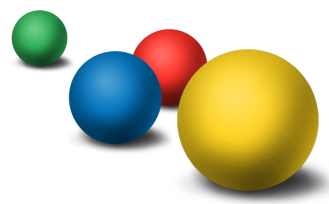

- Occasional large interval leads to large limits
- Large limit leads to large increments, which leads to larger limit ...
- Limit decreases only after fairly long period
- Zero or negative slack cancels decreasing (minimum!)
- Large limit tends to stay



# Excessive decrease

Google

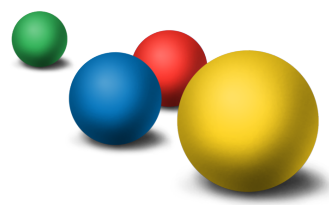
- Due to occasionally shorter intervals on a loaded link
- Sudden bursts on a lightly loaded link
- Much less of a problem than large limit
- Instantly fixed by immediate limit increase
  - Although the limit is low, the NIC dequeues a lot since the queue is long due to the previously large limit
  - The large dequeue is usually complete



# Observation

Google

- Limit tends to change when timing changes for the same load

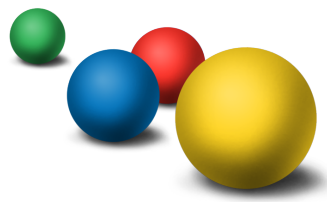


# Possible cure

Google

Execute completion only when fixed amount of time elapsed

- What should be the period?
- Needs a timer to deal with corner cases
- Allows calling a completion wrapper for each packet

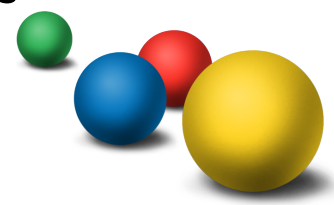


Google

## 128 Streams vs 1 RR

| Feedback | RR rate rps | Qlen pkt |
|---|---|---|
| no BQL (DROP/ECN) | ~94 | ~28 |
| DROP | ~140 | ~113 |
| ECN | ~749 | ~82 |

- Codel has little effect for RR without BQL
- DROP results in a huge tail latency due to retransmissions



# Conclusions

Google

- BQL works fine in vast majority of cases
- BQL may set excessive limit
- Never worse than not having BQL
- Essential for CoDel