JIT BPF

- Optimized code (exec time, small size)
- Only on x86_64 [but ports should be easy]
- Might be hard to debug
- To experiment : Avoid function calls to get packet payload -> inline the fast path (when accessing bytes in skb head)
- How about using JIT technique for main iptables loop ?

JIT {ip|ip6|...}tables

- Only one copy of 'code', shared by all cpus
- Might give even smaller 'code' on UP :
- (One iptable entry minimum size being 156 bytes! [no match, no target])
- Packet/bytes counters using percpu data [one instruction on x86_64 with %gs prefix, two instructions on i386 with %fs prefix]

RTNL/(Network BKL) can be bad

- RTNL + synchronize_rcu() is bad
- Device dismantle needs synchronize_rcu() and rcu_barrier()
- Some guys want to create/delete hundred devices per second.
- Expedited rcu not always a solution
- → Work queue to perform the final work, and let the rtnl being released ASAP.
- rtnl_trylock() is horrible

UDP improvements

- Inetpeer scalability improvements (commit 4b9d9be839 inetpeer: remove unused list), thanks to report from Andi Kleen and Tim Chen with their benchs on 40 core machine (80 threads)
- Followed by commit 2b77bdde9 (inetpeer: lower false sharing effect)

UDP transmit,

- Still we bounce on dst refcount badly on xmit path on memcached workload [many threads sending UDP messages to single destination]

  → allow small frames (<= 256 ?) being copied from User to Kernel first, then perform RCU route lookup to get dst, and dont change dst refcount.
- Minimal changes using set_fs(KERNEL_DS) trick, building on stack iov.
- Adds a copy, but cache hot copies are not that bad.

Packet schedulers

- Bufferbloat hype made some bugs surface (ECN related : IP defrag bug, ipv6 bug)
- New packet schedulers (SFB, CHOke, SFB)
- SFQ improvements (IP frag problem with Congestion Notification)
- pfifo_fast default is a problem, and few admins are aware of possible starvation caused by a single tcp flow (with crazy sending window, and SuperPacket { aka TSO})
-

Packet schedulers (cont)

- Remove spinlock limitation on trafic rate limiting (especially on multiqueue devices), using batches
- Use percpu counters for packet/bytes values, for example in tc filters [they should run concurrently on all cpus]
- est_timer() used to compute rate estimations, would run slower and would need to run in process context instead of softirq (workqueue instead of timer)

Try to reduce percpu needs

- More and more cpus → we should use percpu data only where really needed
- MIB now use one table instead of two (USER/BH), finally !
- Some rarely used counters should use plain atomic_long_t (as done with ICMP counters in ipv6, as they are per device)

Adaptative refcounter

- Goal : reduce dst refcount contention. Could be used on netdevice refcounts too.
- At creation time, an object without percpu data
- { atomic_t refcnt ;
  int __percpu *pcpurrefs ;
  int lastcpu ;
  int cpumismatch_count ;
  } ;
- If number of cpu mismatches reaches a limit, expand the refcounter to a full percpu structure.
  As this calls __alloc_percpu(), this can only be done from process context, with no lock held (might be tricky)
- At refcounter dismantle phase, signal that pcpurrefs cannot be used any more (all further changes must be done on shared refcnt atomic)