tc and cls_bpf: lightweight packet classifying with BPF

Daniel Borkmann
Networking Services Group
<dborkman@redhat.com>

DevConf.CZ, Brno, February 7, 2014

Why QoS?

(c) wikipedia

Terminology in Linux

- **Queueing disciplines (qdisc)**
  - General mechanism to enqueue packets
  - Discipline can be classful or classless
- **Traffic classes (class)**
  - Used in classful qdiscs
  - Tree structured to map different traffic types
  - Own set of attributes e.g., limiters, priorities, some qdiscs allow inter-class bandwidth borrowing
- **Classifiers (cls)**
  - Decision which qdisc/class a packet belongs to
  - Each node can have own filters, but they can also point to subclasses
  - Ematches (extended matches), actions can mostly be attached such as mangling, mirroring, or rerouting

Qdisc types

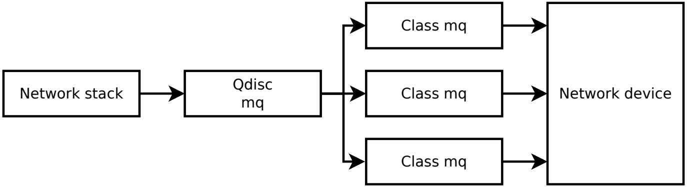
- **Classful qdisc**
  - Qdisc with traffic classes attached to it
  - Allow for user-defined queueing structure and classification
  - **Linux:** atm, cbq, choke, drr, dsmark, fq_codel, hfsc, htb, ingress, mq, mq_prio, prio, qfq, red, sfb, sfq, tbf
- **Classless qdisc**
  - Qdisc without any class attached to it
  - No way to influence structure of its internal queues
  - **Linux:** codel, fifo, fq, hhf, pie, gred, blackhole, teql, netem
- Various combinations of qdiscs possible
- Further reading: `man tc-htb`, `tc-`...

Example 1: pfifo_fast

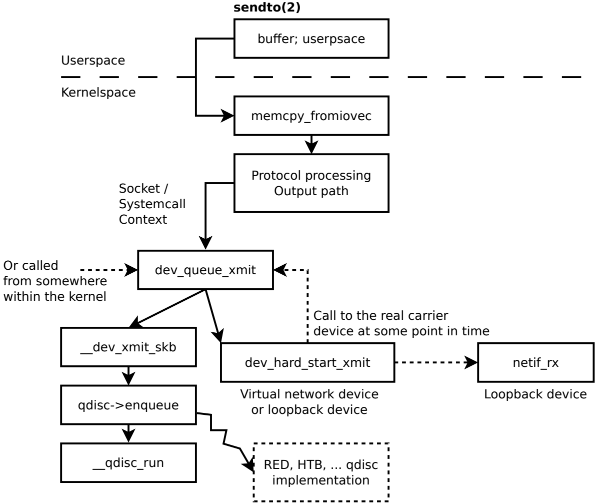
- Default qdisc on Linux
- First-in first-out; 3 bands for priority
- Classification done through packet priority (diffserv)
- Each band can be `txqueuelen` packets long
- Like 3 `pfifo` queues side by side

Example 2: mq

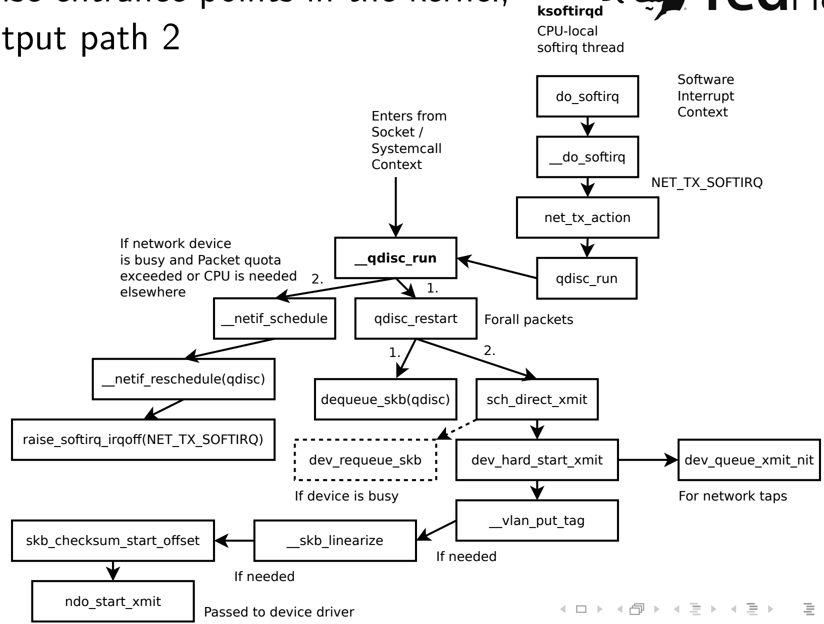
- Multiqueue scheduler
- If NIC provides multiple TX queues, mq used by default
- Creation of `dev->num_tx_queues` classes
- Replaces single per device TX lock with a per queue lock
- Classes can contain other qdiscs again e.g. codel, ...



Qdisc entrance points in the kernel, Output path 1



Qdisc entrance points in the kernel, Output path 2



Classification

(c) constructionweekonline

Linux classifier and ematches

- Various classifiers available in the kernel e.g.,
  - `cls_u32`, 32 bit key classifier
  - `cls_fw`, classification through `skb->mark`
  - `cls_cgroup`, application cgroup classification
  - `cls_basic`, ematch trees classification
- Ematches
  - Small classifier submodules not worth writing a full classifier for
  - Can be interconnected to form a logic expression
  - Can get attached to extend classifier's functionality
- Example:
  - `tc filter add .. basic match ... 'meta(nfmark gt 24)' and 'meta(tcindex mask 0xf0 eq 0xf0)'`

BPF-based classifier, idea

- BPF itself used in `packet(7)` sockets for filtering, i.e. `libpcap(3)`
- Used for early drop in kernel for uninteresting packets
- Minimal, lightweight register machine, interpreted
- Transparently JITed in the kernel for x86_64, sparc, ppc, arm, s390
  - `echo 1 >/proc/sys/net/core/bpf_jit_enable`
- Verdicts in BPF via `packet(7)`
  - Drop packet (not interesting for pcap trace)
  - Truncate packet at a particular offset
  - Keep whole packet for user space
- Verdicts repurposed in `cls_bpf` for qdisc classid
- Multiple possible exit points in BPF program for various classids

BPF-based classifier, idea

- BPF itself used in `packet(7)` sockets for filtering, i.e. `libpcap(3)`
- Used for early drop in kernel for uninteresting packets
- Minimal, lightweight register machine, interpreted
- Transparently JITed in the kernel for x86_64, sparc, ppc, arm, s390
  - `echo 1 >/proc/sys/net/core/bpf_jit_enable`
- Verdicts in BPF via `packet(7)`
  - Drop packet (not interesting for pcap trace)
  - Truncate packet at a particular offset
  - Keep whole packet for user space
- Verdicts repurposed in `cls_bpf` for qdisc classid
- Multiple possible exit points in BPF program for various classids

BPF architecture (net/core/filter.c)

- Registers, volatile: `A:32, X:32, M[16]:32, (pc)`
- Program layout: sequence of (`I:16, JT:8, JF:8, K:32`) tuple
- Instruction categories:
  - **load:** `ld, ldi, ldh, ldb, ldx, ldxi, ldxb`
  - **store:** `st, stx`
  - **branch:** `jmp, ja, jeq, jneq, jlt, jle, jgt, jge, jset`
  - **alu:** `add, sub, mul, div, mod, neg, and, or, xor, lsh, rsh`
  - **return:** `ret`
  - **misc:** `tax, txa`

BPF extensions

- Linux has a couple of BPF extensions for loading into A
- Examples: ifindex, queue mapping, cpu, vlan tag, rxhash, mark
- Invoked through 'overloading' load instructions in offset mode
  - K interval: `[0xfffff000, 0xffffffff]`
  - Extensions: `K := 0xfffff000 + <x>`
  - Compilers: `getsockopt(2)` for `SO_BPF_EXTENSIONS` (3.8/3.14)

BPF examples (bpf_asm syntax)

- IPv4 TCP packets

```
ldh [12]
jne #0x800, drop
ldb [23]
jneq #6, drop
ret #-1
drop: ret #0
```

- Accelerated VLAN, ID 10

```
ld vlan_tci
jneq #10, drop
ret #-1
drop: ret #0
```

BPF JIT compiler (2.6.39)

- If BPF code includes unsupported instructions, fallback to interpreter
- First stage rough opcode image size estimation
- Executable memory obtained from `module_{alloc,free}()` helpers
- Compiler passes for opcode emitters; generation of prologue, epilogue
- Eventually icache flush and setting of entry point `fp->bpf_func`
- Raw opcode image dump:
  - `echo 2 >/proc/sys/net/core/bpf_jit_enable`

BPF toolchain (3.9/3.14, tools/net/)

- `bpf_asm`
  - For low-level BPF asm filter translation
  - For debugging, development, auditing, high-end purposes
  - I.e. `libpcap(3)` compiler workarounds, code optimization and fast adaption for BPF extensions
- `bpf_jit_disasm`
  - JIT emitted opcode image disassembler
  - For low-level optimization, verification, filter development
- `bpf_dbg`
  - Runs BPF filter in user space on a given pcap file
  - Forward/backward single stepping, breakpoints, register dumps, etc

BPF toolchain, bpf_asm

```
$ cat foo
ldh [12]
jne #0x806, drop
ret #-1
drop: ret #0

$ ./bpf_asm foo
4,40 0 0 12,21 0 1 2054,6 0 0 4294967295,6 0 0 0,

$ ./bpf_asm -c foo
{ 0x28,  0,  0, 0x0000000c },
{ 0x15,  0,  1, 0x00000806 },
{ 0x06,  0,  0, 0xffffffff },
{ 0x06,  0,  0, 0000000000 },
```

BPF toolchain, bpf_jit_disasm

```
$ ./bpf_jit_disasm -o
    94 bytes emitted from JIT compiler (pass:3, flen:9)
    ffffffffa0356000 + <x>:
       0:        push   %rbp
        55
       1:        mov    %rsp,%rbp
        48 89 e5
       4:        sub    $0x60,%rsp
        48 83 ec 60
       8:        mov    %rbx,-0x8(%rbp)
        48 89 5d f8
      [...]
      5c:        leaveq
        c9
      5d:        retq
        c3
```

BPF toolchain, bpf_dbg

```
$ ./bpf_dbg
[...]
> breakpoint 1
breakpoint at: l1: jeq #0x800, l2, l5
> run
-- register dump --
pc:       [0]
code:     [40] jt[0] jf[0] k[12]
curr:     l0: ldh [12]
A:        [00000000][0]
X:        [00000000][0]
M[0,15]:  [00000000][0]
-- packet dump --
len: 42
  0: 00 19 cb 55 55 a4 00 14 a4 43 78 69 08 06 00 01
 16: 08 00 06 04 00 01 00 14 a4 43 78 69 0a 3b 01 26
 32: 00 00 00 00 00 00 0a 3b 01 01
(breakpoint)
>
```

cls_bpf tc frontend

- cls_bpf configuration through tc's f_bpf frontend
- 2 modes: bytecode, bytecode-file, (3rd in progress)
  - `tc filter add dev em1 parent 1: bpf run <mode> [...]`
- Also police and action can be attached, no ematches though
- Bytecode can be gathered from bpf_asm or libpcap(3)

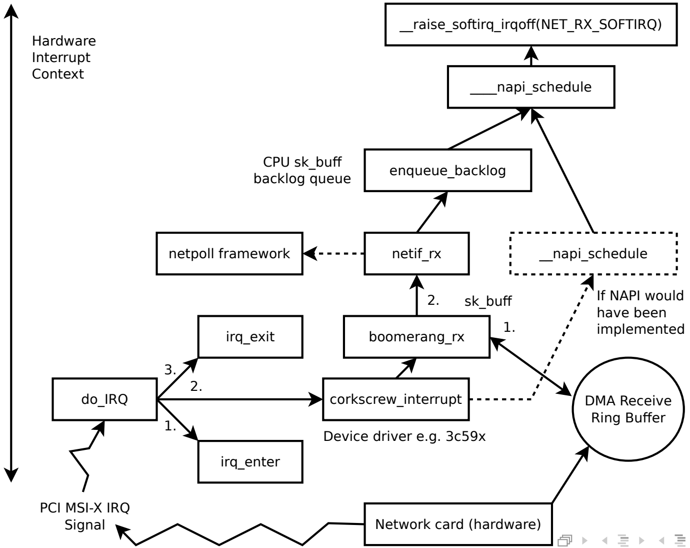
Matching performance

- Filter execution time until verdict, in cycles
- Machine: x86_64/Core2 U9400, 4GB RAM, 3.13+ kernel
- Filter: IPv4 (no frag) and TCP ACK; average over 1024 runs
- `cls_bpf`, JITed:
  - Best case: ~26 cyc (miss)
  - Worst case: ~45 cyc (hit)
- `cls_u32`:
  - Best case: ~64 cyc (miss)
  - Worst case: ~113 cyc (hit)

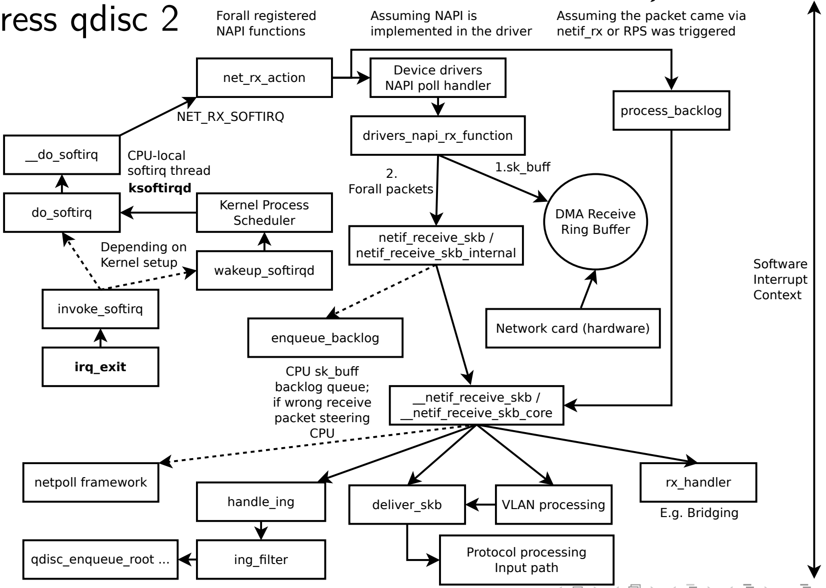
What about qdiscs and ingress traffic?

- Main use case for qdiscs is output path e.g. for shaping
- Tied to socket's wmem buffer accouting; adjusted on skb destructor
- Ingress abilities very limited, only policing possible via filter
- Qdisc for this purpose: `sch_ingress`, example:
  - `tc qdisc add dev em1 handle 1: ingress`
  - `tc filter add dev em1 parent 1: bpf run bytecode [...] police rate 256kbit burst 10k drop flowid 1:1`
- Entrance point is form `__netif_receive_skb_core()` ...

Qdisc entrance points in the kernel, Ingress qdisc 1



Qdisc entrance points in the kernel, Ingress qdisc 2



Work In Progress

- High level to BPF compiler for tc
  - Now we have opened up BPF and tc for power users
  - Next step is to give users a choice for cli filters
  - Challenges:
    - Lexing, parsing of high-level DSL for ≥ 1 classids
    - Low-level code generation, merging, optimization
    - Exploitation of BPF extensions

Thanks! Questions?

- torvalds/linux.git, since 3.13/3.14
  - bpf_asm, bpf_dbg, bpf_jit_disasm: **tools/net/**
  - cls_bpf: **net/sched/**
  - docs: **Documentation/networking/filter.txt**
- shemminger/iproute2.git
  - tc and f_bpf: **tc/**
- Sources, talk:
  - Kernel tree, Git log
  - http://www.infradead.org/~tgr/libnl/doc/route.html