

Netfilter updates since last NetDev

NetDev 2.2, Seoul, Korea (Nov 2017)

<pablo@netfilter.org>

Pablo Neira Ayuso

What does this cover?

- Not a tutorial...
 - Incremental updates already upstream
 - Ongoing development efforts
 - Highlights of the NFWS'17 in Faro, Portugal
 - A bit of performance numbers
 - Using new netfilter ingress hook available since 4.3
 - `pktgen_bench_xmit_mode_netif_receive.sh`
 - One single CPU
 - Smallest packet size (worst case)

What does this cover? (2)

- For those that are new to nftables...
 - nftables replaces {ip,ip6,eb,arp}tables
 - It's well documented:
 - <https://wiki.nftables.org>
 - man nft(8)
 - nftables 0.8 release (Oct 13th,2017)
 - 306 commits since last release
 - 26 unique contributors (x2 compared to 2016)

nftables performance numbers

- Dropping packets, with 4.14.0-rc+patch
- iptables from prerouting/raw:
iptables -I PREROUTING -t raw -p udp -dport 9 -j DROP
5999928pps
- nftables from ingress (x2 faster):
nft add rule netdev ingress udp dport 9 drop
12356983pps
nft add rule netdev ingress udp dport { 1, 2, ..., 384 } drop
11844615pps

Faster nftables sets: Overview

- Selects backend based on description from userspace
 - Number of elements (if known), eg. `nft add rule x y tcp dport { 22, 80 }`
 - Key length, eg. `tcp dport` (16 bits), `ip6 saddr` (128 bits)
 - Intervals, eg. `ip saddr { 192.168.1.10-192.168.1.100 }`
- Sets come with big O notation to indicate scalability
 - lookup
 - space
- User doesn't need to know need to learn about datastructures and play tuning games
- Two policies:
 - Performance, select the faster implementation (default behaviour)
 - Memory, selects the one that consumes less memory

Faster nftables sets: Overview (2)

- Existing set backend implementations
 - Hashtable
 - Two variants
 - fixed size (number of elements is known)
 - resizable
 - With timeout implementation.
 - Bitmap, up to 16 bit keys, hence it consumes:
 - 64 bytes for 8 bits keys.
 - 16 Kbytes for 16 bits keys.
 - Red-black tree, for intervals

Faster nf_tables sets: hashtable

- Performance evaluation from nft ingress
 - one rule with constant sets, default policy drop
- Resizable hashtable: 11076337pps, **5316Mb/sec**
- Fixed size hashtable (just 150 more LOC)
 - Selected if userspace indicates size:
 - Constant sets (number of elements is known)
 - nft add element x y { 192.168.1.2, 192.168.1.10, 192.168.1.20 }
 - User specifies 'size' statement in set definition, eg.
 - nft add set x y { type ipv4_addr; size 128; }
 - 16-bit or 32-bit key: 13109944pps **6292Mb/sec**
 - > 32-bit key: 12670233pps **6081Mb/sec**

Faster nf_tables sets: bitmap

- Keeps a list of existing dummy objects
 - Keeps element comments, only used for dumping
 - Increases memory consumption
 - No timeout support yet
- From lookup path, uses bitmap representation
 - Two bits to represent current and next/previous generation
- 16-bit key: 16755207pps **8042Mb/sec**
- Selected from keys ≤ 16 bits
 - If default policy is performance

Faster nf_tables sets: rbtree

- Red-black tree for ranges
 - No timeout support yet
- Faster lockless since 4.13 (Howells' RPC idea)
- With 3 ranges: **9952520pps**
- With 12 ranges: **9130579pps**

What's new in nftables since last NetDev

- fib expression from netdev for early reverse path filter and RTBH (Pablo M. Bermudo)

```
# nft add rule netdev filter ingress \  
    fib saddr . iif oif missing drop  
# nft add rule netdev filter ingress meta mark set 0xdead \  
    fib daddr . mark type vmap { \  
        blackhole : drop, \  
        prohibit : jump prohibited, \  
        unreachable : drop }
```

- TCP options and route path mtu (Florian Westphal)

```
# nft add rule inet mangle forward \  
    tcp option maxseg set rt mss
```

What's new in nftables since last NetDev (2)

- Rise nf_tables objects name size up to 255 chars for DNS names as per RFC1035 (Phil Sutter)

```
# nft add set filter server1.pool.badguy.com { \  
    type ipv4_addr\; }
```

- Display generation ID and process (Phil Sutter)

```
# nft monitor  
add table netdev test  
add chain netdev test test { \  
    type filter hook ingress priority 0; policy accept; }  
add rule netdev test test udp dport 9  
# new generation 18 by process 22900 (nft)
```

What's new in nftables since last NetDev (3)

- Limit stateful object (Pablo M. Bermudo)

```
# nft add limit filter lim1 rate 512 kbytes/second
# nft add limit filter lim2 rate 1024 kbytes/second \
  burst 512 bytes
# nft add rule filter prerouting \
  limit name tcp dport map {
  443 : "lim1", \
  80 : "lim2", \
  22 : "lim1"}
```

 - No rate limit update command yet.
- Add NLM_F_NONREC to netlink: Bail out if user requests non-recursive deletion for tables and sets.

What's new in nftables since last NetDev (4)

- Dry run mode (Pablo M. Bermudo)

```
# nft --check add rule x y ip protocol vmap { \  
    tcp : jump tcp_chain, \  
    udp : jump udp_chain }  
# nft --check add element x z { 192.168.2.1 }
```

- Wildcards to include files from scripts (Ismo Puustinen):
Include "/etc/ruleset/*.nft"

- --echo option (Phil Sutter):

```
# nft --echo --handle add rule ip x y \  
    tcp dport {22, 80} accept  
add rule ip t c tcp dport { ssh, http } accept # handle 2
```

ferm ideas for nftables

- ferm is around since 2001:
 - <http://ferm.foo-projects.org>
 - People seem to ♥ this...
 - nftables syntax is clearly inspired by this: Expands to iptables commands.
- Features we can add from there:
 - Define variable from command line call:
`ferm --def '$name=value' ...`
 - Test the rules without fearing to lock yourself out.
`--interactive ... --timeout`
 - External command invocations
`@def $DNSSERVERS = `grep nameserver /etc/resolv.conf | awk '{print $2}'`;
chain INPUT proto tcp saddr $DNSSERVER ACCEPT;`

libnftables: high level library

- Joint work by Eric Leblond and Phil Sutter.

- Simple API, for those in the rush.

```
nft = nft_ctx_new(NFT_CTX_DEFAULT);  
nft_run_cmd_from_buffer(nft, cmd, sizeof(cmd));  
nft_ctx_free(nft);
```

- Still to be done:

- Allow to select output to display errors.
- Batch commands.

- More advanced API to control Netlink IO.

Conntrack updates since last NetDev

- Mostly work done by Florian Westphal.
- Speed up netns removal by selective calls of `synchronize_net()`
- Speed up conntrack by simplifying ct extension infrastructure: No expensive runtime time calculation of extension area.
- Reduce memory footprint by using smaller arrays.
- Conntrack hooks registered once there's rule using `-m` state.
- Allow to get rid of unassured flows under stress for DCCP, SCTP and TCP protocols.
- No more fake conntrack object for nottracking: better cache efficiency.
- Conntrack hashtable resizing bugfixes (Liping Zhang)

Flow offload infrastructure

- Idea: Add generic software flow table from netfilter ingress hook.
 - For each packet, extract tuple and look up at the flow table.
 - Miss: Let the packet follow the classic forwarding path.
 - Hit: Packet is pushed out to the destination and interface.
 - NAT mangling, if any.
 - Decrement TTL.
 - Send packet via `neigh_xmit(...)`.
 - Expire flows if we see no more packets.

Flow offload infrastructure (2)

- Add entry to software flow table from conntrack object in established state.
- Configure flow offload through rule:

```
table ip x {
    chain y {
        type filter hook forward priority 0;
        ip protocol tcp flow offload counter
    }
}
```

- Print flows that are offloaded:

```
# cat /proc/net/nf_conntrack
ipv4  2 tcp    6 src=10.141.10.2 dst=147.75.205.195 sport=36392
dport=443 src=147.75.205.195 dst=192.168.2.195 sport=443
dport=36392 [OFFLOAD] mark=0 zone=0 use=2
```

Flow offload infrastructure (3)

- Flow offload forward PoC in software is ~2.75 faster:
 - Rule in forward chain to select what to offload
 - `pktgen_bench_xmit_mode_netif_receive.sh` to dummy device to exercise the forwarding path
 - One single CPU
 - Smallest packet size (worst case)
- Performance numbers:
 - Classic forwarding path (baseline):
1848888pps 887Mb/sec (887466240bps)
 - Flow offload forwarding:
5155382pps 2474Mb/sec (2474583360bps)

Flow offload infrastructure (4)

- Switches/NICs come with built-in flow table.
- Observing out of tree patches to support hardware flow table from Netfilter in OpenWRT.
- Hardware flow table configuration need to hold mdio mutex:
 - Queue configuration to kernel thread.
 - Few packets follow the software flow table until configuration is done.
- Pass struct `flow_offload` as parameter to `ndo` to offload to hardware:
 - `int (*ndo_flow_add)(struct flow_offload *flow);`
 - `int (*ndo_flow_del)(struct flow_offload *flow);`
- ... otherwise, fall back to the software flow table.

Netfilter updates since last NetDev

NetDev 2.2, Seoul, Korea (Nov 2017)

Pablo Neira Ayuso

<pablo@netfilter.org>