Packet Sockets, BPF and Netsniff-NG

(Brief intro into finding the needle in the network haystack.)

Daniel Borkmann
<borkmann@redhat.com>

Open Source Days, Copenhagen, March 9, 2013

Motivation and Users of PF_PACKET

- Useful to have raw access to network packet data in user space
  - Analysis of network problems
  - Debugging tool for network (protocol-)development
  - Traffic monitoring, security auditing and more
- **libpcap** and all tools that use this library
  - Used only for packet reception in user space
  - tcpdump, Wireshark, nmap, Snort, Bro, Ettercap, EtherApe, dSniff, hping3, p0f, kismet, ngrep, aircrack-ng, and many many more
- **netsniff-ng** toolkit (later on in this talk)
- Suricata and other projects, also in the proprietary industry
- Thus, this concerns a huge user base that PF_PACKET is serving!

Motivation and Users of PF_PACKET

- Useful to have raw access to network packet data in user space
  - Analysis of network problems
  - Debugging tool for network (protocol-)development
  - Traffic monitoring, security auditing and more
- **libpcap** and all tools that use this library
  - Used only for packet reception in user space
  - tcpdump, Wireshark, nmap, Snort, Bro, Ettercap, EtherApe, dSniff, hping3, p0f, kismet, ngrep, aircrack-ng, and many many more
- **netsniff-ng** toolkit (later on in this talk)
- Suricata and other projects, also in the proprietary industry
- Thus, this concerns a huge user base that PF_PACKET is serving!

Features of PF_PACKET

- Normally `sendto(2)`, `recvfrom(2)` calls for each packet
  - Buffer copies between address spaces, context switches
- How can this be further improved (`PF_PACKET` features)?[1]
  - **Zero-copy** RX/TX ring buffer ("packet `mmap(2)`")
    - "Avoid obvious waste" principle
  - Socket **clustering** ("packet fanout") with e.g. CPU pinning
    - "Leverage off system components" principle (i.e. exploit locality)
  - Linux socket **filtering** (Berkeley Packet Filter)
    - "Shift computation in time" principle

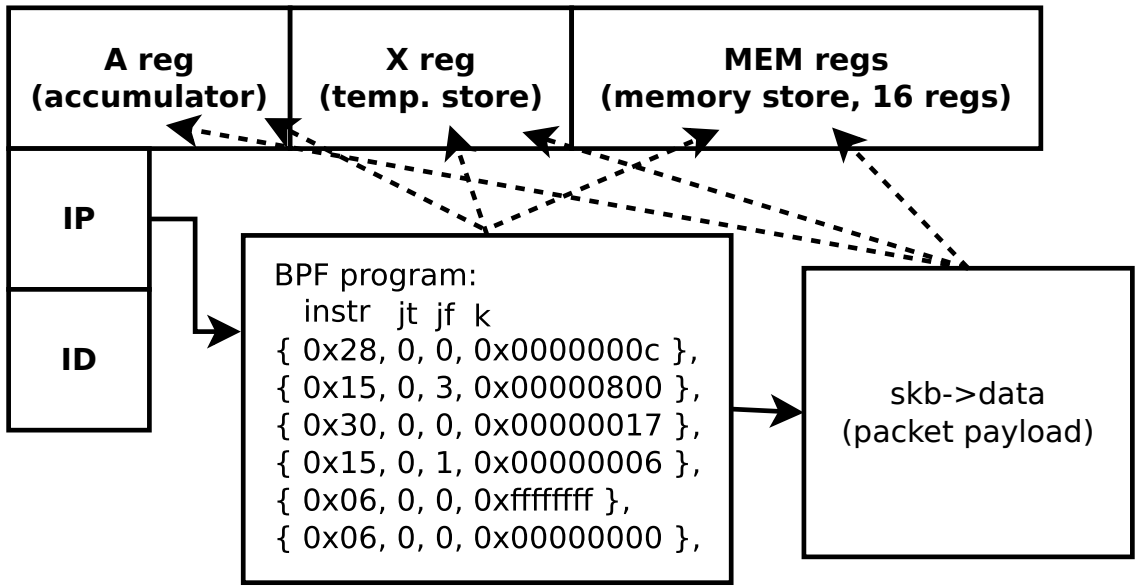
[1] Principle names from: "G. Varghese, Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices."

Features of PF_PACKET

- Normally `sendto(2)`, `recvfrom(2)` calls for each packet
  - Buffer copies between address spaces, context switches
- How can this be further improved (`PF_PACKET` features)?[1]
  - **Zero-copy** RX/TX ring buffer ("packet `mmap(2)`")
    - "Avoid obvious waste" principle
  - Socket **clustering** ("packet fanout") with e.g. CPU pinning
    - "Leverage off system components" principle (i.e. exploit locality)
  - Linux socket **filtering** (Berkeley Packet Filter)
    - "Shift computation in time" principle

[1] Principle names from: "G. Varghese, Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices."

BPF architecture ('92)



- Van Jacobson, Steven McCanne, *the* filter system for Linux, BSD
- Kernel "virtual machine", invoked by `PF_PACKET` for filtering
- JIT compilers for: x86/x86_64, SPARC, PowerPC, ARM, s390
- Instruction categories: load, store, branch, alu, return, misc
- Own kernel extensions, e.g. access cpu number, vlan tag, ...

Netsniff-NG Toolkit

- Useful networking toolkit for daily kernel plumbing, security auditing, system monitoring or administration
- Set of minimal tools: **netsniff-ng**, **trafgen**, **astraceroute**, **curvetun**, **ifpps**, **bpfc**, **flowtop**, **mausezahn**
- Core developers: Daniel Borkmann[2], Tobias Klauser[2], Markus Amend, Emmanuel Roullit, Christoph Jäger, Jon Schipp (documentation)
- `git clone git://github.com/borkmann/netsniff-ng.git`
- Project since 2009, started just for fun; GNU GPL, version 2.0
- Here, short intro into netsniff-ng, trafgen

[2] Project Maintainer

Netsniff-NG Toolkit

- Useful networking toolkit for daily kernel plumbing, security auditing, system monitoring or administration
- Set of minimal tools: **netsniff-ng**, **trafgen**, **astraceroute**, **curvetun**, **ifpps**, **bpfc**, **flowtop**, **mausezahn**
- Core developers: Daniel Borkmann[2], Tobias Klauser[2], Markus Amend, Emmanuel Roullit, Christoph Jäger, Jon Schipp (documentation)
- `git clone git://github.com/borkmann/netsniff-ng.git`
- Project since 2009, started just for fun; GNU GPL, version 2.0
- Here, short intro into netsniff-ng, trafgen

[2]Project Maintainer

netsniff-ng, Examples

- Usual work mode, with high-level, tcpdump-like filter:
  - `netsniff-ng --in eth0 tcp or udp`
- Capture pcap files of Alexey Kuznetzov's format, with low-level filter:
  - `netsniff-ng --in eth0 --out dump.pcap -b 0 -s -T 0xa1b2cd34 -f bpfops`
- Capture multiple raw 802.11 traffic pcap files, each 1GiB, mmap(2)ed:
  - `netsniff-ng --in wlan0 --rfraw --out /probe/ -s -m --interval 1GiB -b 0`
- Replay a pcap file in scatter-gather, also `tc(8)` can be used again:
  - `netsniff-ng --in dump.pcap -k 100 --out eth0 -s -G -b 0`

netsniff-ng, Examples

- Usual work mode, with high-level, tcpdump-like filter:
  - `netsniff-ng --in eth0 tcp or udp`
- Capture pcap files of Alexey Kuznetzov's format, with low-level filter:
  - `netsniff-ng --in eth0 --out dump.pcap -b 0 -s -T 0xa1b2cd34 -f bpfops`
- Capture multiple raw 802.11 traffic pcap files, each 1GiB, mmap(2)ed:
  - `netsniff-ng --in wlan0 --rfraw --out /probe/ -s -m --interval 1GiB -b 0`
- Replay a pcap file in scatter-gather, also `tc(8)` can be used again:
  - `netsniff-ng --in dump.pcap -k 100 --out eth0 -s -G -b 0`

trafgen, Examples

- Usual work mode (all CPUs, send conf through C preprocessor[3]):
  - `trafgen --dev eth0 --conf tcp_syn_test --cpp`
- Injection of raw 802.11 frames (yes, also works with TX_RING):
  - `trafgen --dev wlan0 --rfraw --conf beacon_test --cpus 2`
- Device smoke/fuzz testing with ICMP probes:
  - `trafgen --dev eth0 --conf stack_fuzzing \ --smoke-test 10.0.0.2`
  - Machine_a (trafgen, 10.0.0.1) $\longleftrightarrow$ Machine_b (victim, 10.0.0.2)
  - Will print last packet, seed, iteration if machine gets unresponsive
- Plus, you can combine trafgen with `tc(8)`, e.g. `netem`

[3] `trafgen -e` for a built-in example

trafgen, Examples

- Usual work mode (all CPUs, send conf through C preprocessor[3]):
  - `trafgen --dev eth0 --conf tcp_syn_test --cpp`
- Injection of raw 802.11 frames (yes, also works with TX_RING):
  - `trafgen --dev wlan0 --rfraw --conf beacon_test --cpus 2`
- Device smoke/fuzz testing with ICMP probes:
  - `trafgen --dev eth0 --conf stack_fuzzing \ --smoke-test 10.0.0.2`
  - Machine$_a$ (trafgen, 10.0.0.1) $\longleftrightarrow$ Machine$_b$ (victim, 10.0.0.2)
  - Will print last packet, seed, iteration if machine gets unresponsive
- Plus, you can combine trafgen with `tc(8)`, e.g. `netem`

[3] `trafgen -e` for a built-in example

What might be next in Netsniff-NG?

- **astraceroute:**
  - DNS traceroute to detect malicious DNS injections on transit traffic (reported by anonymous researchers at SIGCOMM 2012 paper)
- **mausezahn:**
  - Improve its imported code and integrate it into the main repository
- **netsniff-ng, mausezahn:**
  - New protocol dissectors/generators like SCTP, DCCP, BGP, etc
- **netsniff-ng:**
  - Compressed on-the-fly bitmap indexing for large PCAP files
  - Try to find a sane way to utilize multicore with packet_fanout
- **netsniff-ng, trafgen, mausezahn:**
  - Performance benchmark on 10Gbit/s
  - Optimize capturing/transmission performance (`PF_PACKET` plumbing)

Thanks! Questions?

- **Web:** `http://netsniff-ng.org`
- **Fellow hackers, clone and submit patches:** :-)
  - git clone `git://github.com/borkmann/netsniff-ng.git`