A Look at the Netsniff-NG Toolkit

A Suite of High-Performance Networking Tools

Jon Schipp

E-mail:

jonschipp@gmail.com

Web:

sickbits.net, dclinux.org, jonschipp.com

Midwest Open Source Software Conference
University of Louisville
May 18, 2013

About Me

- Unix Administrator and Network Security Specialist
  1. Small business in Southern Indiana
- Founder
  1. Dubois County Linux User Group
  2. Southern Indiana Computer Klub
- Contributer to Netsniff-NG
  1. Mainly concerned with documentation
  2. Not an expert, still learning
- Enjoys packet analysis and network security monitoring

Background

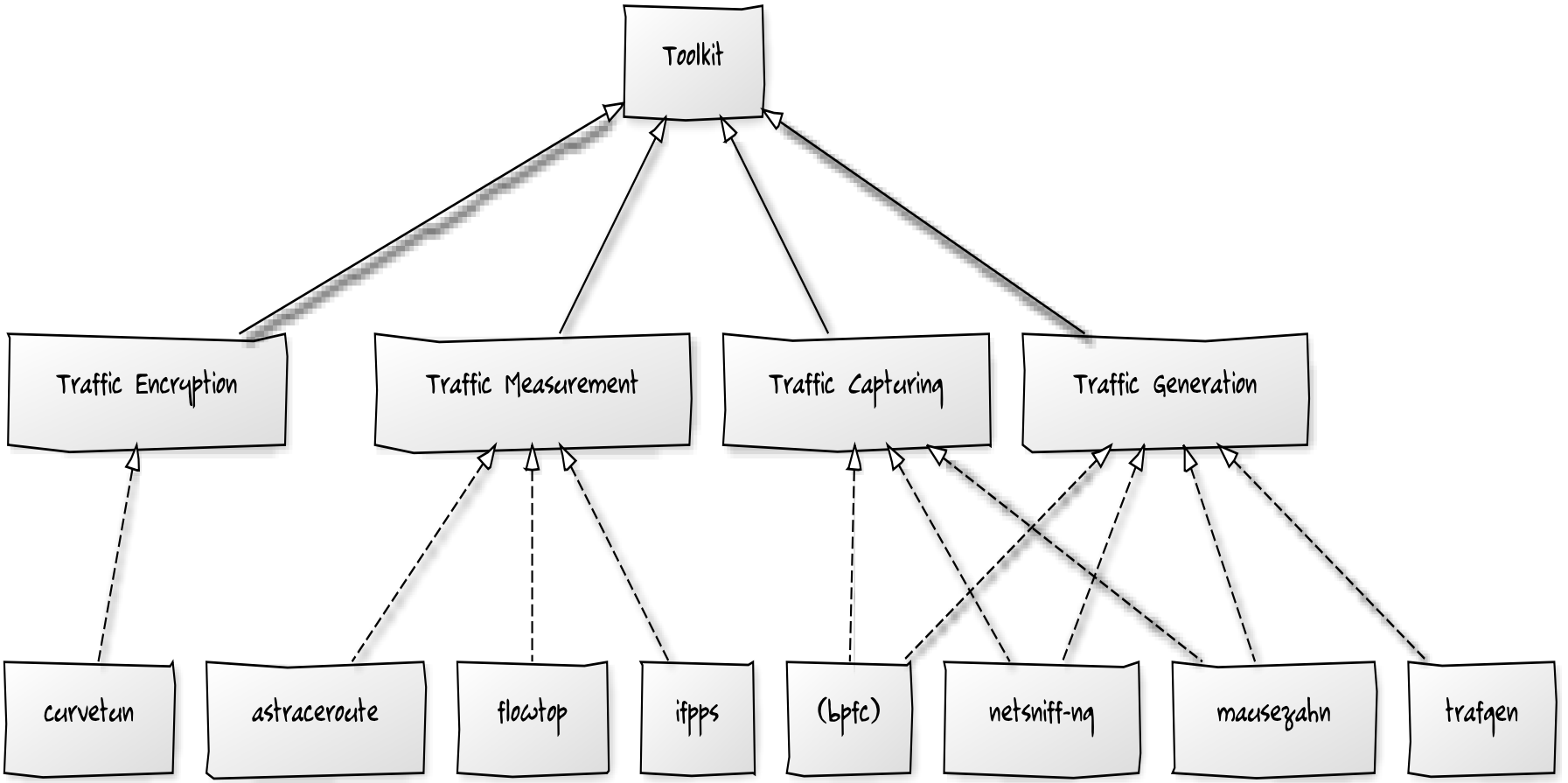
netsniff-ng t  lkit

- Started in 2009 for fun; GNU GPL, version 2.0
  1. Packages available on Debian, Fedora, Red Hat, Gentoo, Arch Linux, Slackware etc.
  2. Included in Security Onion, NST, Xplico's Network Forensic Toolkit, and GRML Linux
- Analysis of network problems
- Debugging tool for network (protocol-)development
- Traffic monitoring, security auditing, stress testing, and more
- Maintainers: Daniel Borkmann (Red Hat) and Tobias Klauser (University of Zurich)

Toolkit Overview

- **netsniff-ng** - a high-performance zero-copy analyzer, pcap capturing and replaying tool
- **trafgen** - a high-performance zero-copy network traffic generator
- **flowtop** - a top-like netfilter connection tracking tool
- **mausezahn** - a packet generator and analyzer for HW/SW appliances with a Cisco-CLI
- **bpfc** - a Berkeley Packet Filter (BPF) compiler with Linux extensions
- **ifpps** - a top-like kernel networking and system statistics tool
- **curvetun** - a lightweight multiuser IP tunnel based on elliptic curve cryptography
- **astraceroute** - an autonomous system (AS) trace route utility

Big Picture


Toolkit
Traffic Encryption
Traffic Measurement
Traffic Capturing
Traffic Generation
curvetun
astraceroute
flowtop
ifpps
(bpfc)
netsniff-ng
mausezahn
trafgen

Gain

- In sum, about **as fast you can get** from user space {netsniff-ng, trafgen}
- Use of PF_PACKET's zero-copy RING's (TX and RX)
  1. Fewer context switches and interrupts
  2. Less calls to sendto() and recvfrom()
  3. Bypass the normal packet processing path through the networking stack
- Control over entire packet including data-link layer
- Optimized for Linux
- Small code footprint

Netsniff-NG

- Fast network analyzer, pcap recorder, pcap replayer
  - Speeds matching PF_RING transparent_mode=0 (default)
- Uses PF_PACKET sockets with mmap(2)ed RX and TX RING
- PCAP recording backend for Security Onion, Xplico, NST etc.
- Supports different pcap types and I/O methods, i.e. scatter-gather and mmap(2)

Netsniff-NG, Examples

- Full content, daily packet capture. Excels at this.

```
$ netsniff-ng --in eth1 --out /pcaps/ --silent
--ring-size 2GiB --prio-high --interval 24hrs
```

- Capture multiple raw 802.11 traffic pcap files, each 1GiB

```
$ netsniff-ng -i wlan0 -o raw.pcap --rfraw -s
--interval 1GiB --bind-cpu 0
```

- Replay interface to interface

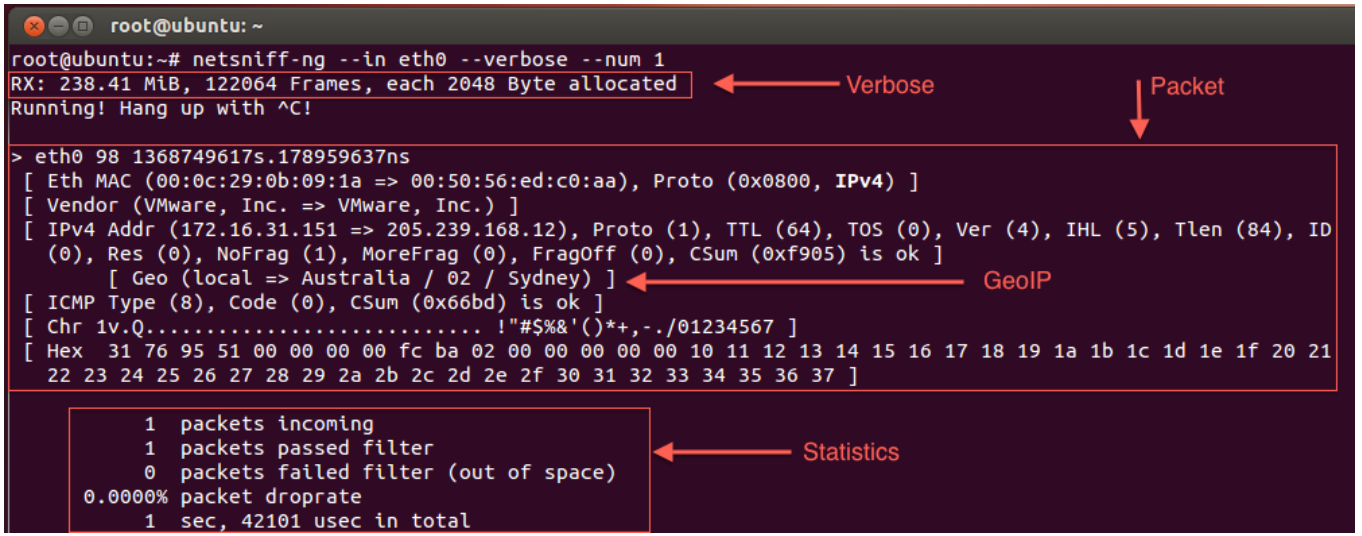
```
$ netsniff-ng --in eth0 --out eth1 --type host
--kernel-pull 100 --mmap -s -b 3
```

- Apply BPF filter

```
$ netsniff-ng -i eth0 -f "tcp and port 22"
$ netsniff-ng -i eth0 -f filter.bpf
$ netsniff-ng -i eth0 icmp
```

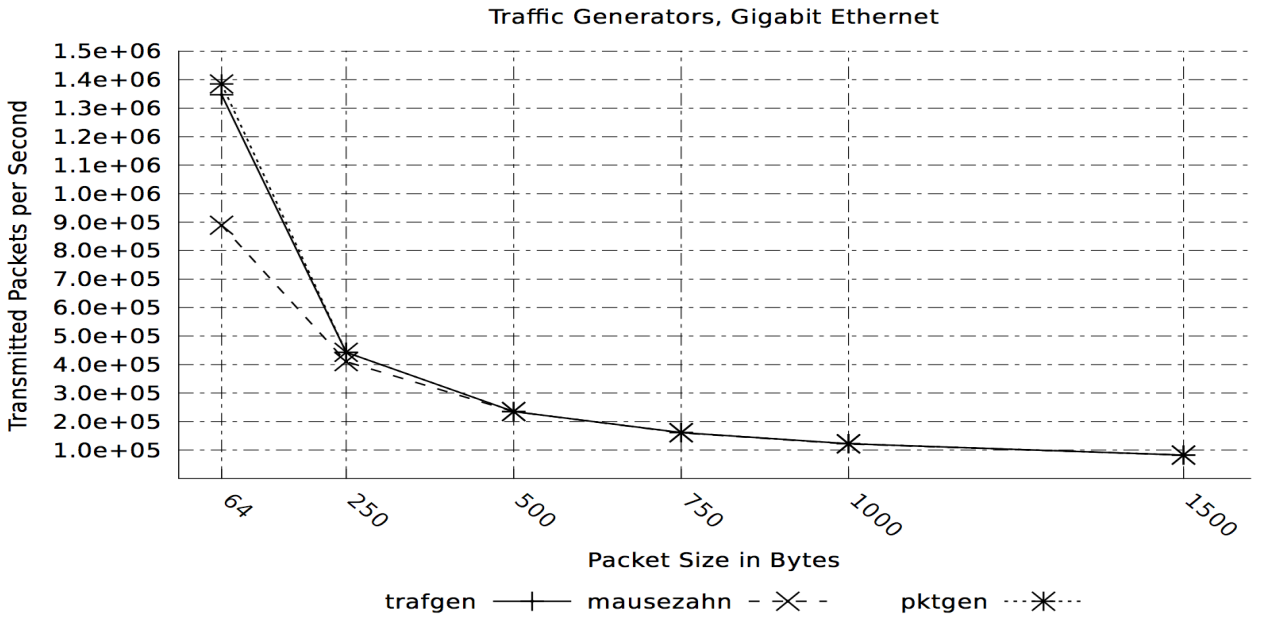
Netsniff-NG, Example Output (ICMP Echo)

- Verbose (`--verbose`)
  - Default RX ring buffer size, number of frames that can fit in buffer, max frame slot size (increase w/ `--jumbo-support`)
- Packet
  - MAC OUI lookup, Geo IP, protocol dissector, ASCII, Hex
- Statistics
  - Incoming, passed filter, dropped, drop rate, capture duration



Trafgen

- Fast multithreaded low-level network traffic generator
- Uses AF_PACKET sockets with mmap(2)ed TX RING
- Powerful packet configuration syntax, more flexible than pktgen



Trafgen, Examples

- Convert PCAP to Trafgen configuration, then generate

```
$ netsniff-ng --in network.pcap --out network.cfg
$ trafgen --in network.cfg --out eth0 --rand
```

- Generate packets and perform mortality test (ICMP Echo)

```
$ trafgen --dev eth0 --conf fuzzing.cfg
--smoke-test 10.0.0.1
```

- Generate **raw** 802.11 frames

```
$ trafgen --dev wlan0 --rfraw --conf
beacon-test.cfg --verbose --cpus 2
```

- Pass built-in configuration through stdin

```
$ trafgen -e |
trafgen --in - --out eth0 --num 1 --cpp
```

Trafgen Expression Language - Features

- Macros
  1. fill(), rnd(), drnd(), const8()...const64(), seqinc(), ddec() etc.
  2. fill() - fill(x, n) - fill(0xff, 10) - Fill (repeat) data x for n bytes
  3. rnd() - rnd(n) - rnd(582) - Generate n bytes of random data
- Data Elements & Types
  1. Hexidecimal, Decimal, Octal, Binary, Characters, Strings
  2. 0xff, 100, 05607, 0b11111111, 'A', "Sweet!"
  3. ...and shellcode, "\xca \xfe \xba \xbe"
- C Preprocessor
  1. Ability to pass configuration through CPP (`--cpp`)
  2. C `#define`'s can be used via `#include <stddef.h>`
  3. `e.g. #define ETH_P_IP 0x0800`
     `/* Internet Protocol packet */`

Trafgen Packet Configuration Examples

Single packet definition: `{ ..packet data.. }`

Create a 64 byte packet of all 0xff's: `{ fill {0xff, 64} }`

Packet with 512 bytes of random data: `{ drnd(512) }`

```
$ python -c "print '{\n' + ' \" MyPacket\",\n' * 8 + '}\n'"
{
 "MyPacket",
 ...
 "MyPacket",
}
$ netsniff-ng --in eth0
eth0 56 1366850783s.90970749ns
[ Eth MAC (74:79:50:61:63:6b => 79:50:61:63:6b:65), Proto (0x6574) ]
[ Vendor (Unknown => Unknown) ]
[ Chr  etMyPacketMyPacketMyPacketMyPacketMyPacketMyPacket ]
[ Hex 65 74 4d 79 50 61 63 6b 65..]
```

Trafgen Packet Configuration Examples Cont. 1

```
$ cat udp.cfg

/* Note:  dynamic elements make trafgen slower!  */
/* Use CPU 1-4 */

cpu(0:3):  {

  /* Ethernet */
  drnd(12),

  /* IPv4 Protocol */
  c16(0x0800),

  /* IP Header */
  drnd(20),

  /* UDP Header */
  drnd(8),

  /* Data */
  rnd(1472)

 }
```

Trafgen Packet Configuration Examples, Cont. 2

```
$ trafgen -e
/* Note:  dynamic elements make trafgen slower!  */
#include <stddef.h>

{
 /* MAC Destination */
 fill(0xff, ETH_ALEN),

 /* MAC Source */
 0x00, 0x02, 0xb3, drnd(3),

 /* IPv4 Protocol */
 c16(ETH_P_IP),

 /* IPv4 Version, IHL, TOS */
 0b01000101, 0,

 /* IPv4 Total Len */
 c16(59),
```

```
/* IPv4 Ident */
drnd(2),

/* IPv4 Flags, Frag Off */
0b01000000, 0,

/* IPv4 TTL */
64,

/* Proto TCP */
0x06,

/* IPv4 Checksum (IP header from, to) */
csumip(14, 33),

/* Source IP */
drnd(4),

/* Dest IP */
drnd(4),

/* TCP Source Port */
drnd(2),

/* TCP Dest Port */
c16(80),
```

```
  /* TCP Sequence Number */
  drnd(4),

  /* TCP Ackn.  Number */
  c32(0),

  /* TCP Header length + TCP SYN/ECN Flag */
  c16((0x8 << 12) | TCP_FLAG_SYN | TCP_FLAG_ECE)

  /* Window Size */
  c16(16),

  /* TCP Checksum (offset IP, offset TCP) */
  csumtcp(14, 34),

  /* TCP Options */
  0x00, 0x00, 0x01, 0x01, 0x08, 0x0a, 0x06,
  0x91, 0x68, 0x7d, 0x06, 0x91, 0x68, 0x6f,

  /* Data blob */
  "gotcha!",

}
```

In Conjunction: Netsniff-NG & Trafgen

- PCAP-2-Trafgen

```
$ netsniff-ng --in pkt.pcap --out - --num 1 -f tcp
{
 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x00, 0x02, 0xb3, 0xba,
 0x34, 0x93, 0x08, 0x00, 0x45, 0x00, 0x00, 0x3b, 0x97, 0xea,
 0x40, 0x00, 0x40, 0x06, 0xc1, 0x1d, 0x6a, 0x11, 0x43, 0x2e,
 0xa0, 0x1f, 0x94, 0x56, 0x96, 0xa0, 0x00, 0x50, 0xae, 0xd6,
 0x7e, 0x01, 0x00, 0x00, 0x00, 0x00, 0x80, 0x42, 0x00, 0x10,
 0x8d, 0xb3, 0x00, 0x00, 0x01, 0x01, 0x08, 0x0a, 0x06, 0x91,
 0x68, 0x7d, 0x06, 0x91, 0x68, 0x6f, 0x67, 0x6f, 0x74, 0x63,
 0x68, 0x61, 0x21,
}
```

- Together now!

```
$ netsniff-ng --in pkt.pcap --out - --num 1 -f tcp |
trafgen --in - --out eth0 --num 1000000
```

Case Study: Performance Metrics

- Packet capture performance test
- Create and generate packet distributions with Trafgen
  1. RFC 2544 "Benchmarking Methodology for Network Interconnect Devices"
  2. e.g. 64, 128, 256, 512, 1024, 1280, 1518
  3. Generate with scripting language e.g. Python, "0xff" * 64
- Verify traffic on receiving machine with Ifpps
- Script a timer and SIGINT for packet capturing tools
- Calculate average packet loss e.g. Awk
- Next up on my to-do list.

Case Study: Full Content Daily PCAPs

- Daily PCAP capture and rotation with Netsniff-NG
  1. Record packets from 3 NIC's attached to 3 separate networks
  2. Bind to specific CPU and run process as high priority
  3. Run as non-root user and redirect stats to file
- Resulting filenames, $timestamp.pcap e.g 1367028821.pcap

```
$ netsniff-ng --bind-cpu 0 --in dmz --out /pcap/dmz/
--prefix "" --ring-size 4GiB --silent --prio-high --verbose
--interval 24hrs --user 1000 --group 1000 >>dmz.stats 2>&1

$ netsniff-ng --bind-cpu 1 --in wlan --out /pcap/wlan/
--prefix "" --ring-size 512MiB -Q --silent --prio-high
--verbose --interval 24hrs --user 1000 --group 1000
>>wlan.stats 2>&1

$ netsniff-ng --bind-cpu 2 --in lan-bot --out /pcap/lan/
--prefix "" --ring-size 4GiB -Q --silent --prio-high
--verbose--interval 24hrs --user 1000 --group 1000
>>lan.stats 2>&1
```

Real-Life Examples

- Used trafgen to create a UDP fragmentation DoS attack
  1. by Jesper Dangaard Brouer, Linux kernel network developer
  2. [net-next PATCH V2 0/6 net]: frag performance tuning cachelines for NUMA/SMP systems
     http://lists.openwall.net/netdev/2013/01/29/44
- Used bpfc to prove/exploit a Linux BPF x86 JIT compiler bug
  1. by Markus Kotter
  2. net: bpf jit: fix an off-one bug in x86 64 cond jump target
     http://carnivore.it/2011/12/27/linux_3.0_bpf_jit_x86_64_exploit

Bpfc

- Is a Berkely Packet Filter compiler
- Supports internal Linux extensions
- Filter opcodes can be passed to netsniff-ng:
  `$ bpfc arp > arp.bpf && netsniff-ng -f arp.bpf`
- Useful for:
  1. Complex filters that cannot be expressed with the high-level syntax
  2. Low-level kernel BPF machine/JIT debugging
  3. Learn how to read, write, and debug BPF (pedagogical)

Bpfc, Examples

- High-level BPF (Mnemonics)
- libpcap equiv 'ether src aa:bb:cc:dd:ee:ff'

```
$ cat ethernet.bpfc
ld [8]
jne #0xccddeeff, Drop
ldh [6]
jne #0xaabb, Drop
ret #1514
Drop:  ret #0
```

Bpfc, Examples Cont.

- Low-level BPF (Op Codes)
- Compile source file to BPF machine language:

```
$ bpfc ethernet.bpfc
{ 0x20, 0, 0, 0x00000008 },
{ 0x15, 0, 3, 0xccddeeff },
{ 0x28, 0, 0, 0x00000006 },
{ 0x15, 0, 1, 0x0000aabb },
{ 0x6, 0, 0, 0x000005ea },
{ 0x6, 0, 0, 0x00000000 },
```

Bpfc, Examples Cont.

- Use of linux kernel extension, filter by CPU

```
$ cat cpu.bpfc
ld #cpu
jeq #0,L1,L2
L1:  ret #-1
L2:  ret #0
```

- Compile filter

```
$ bpfc cpu.bpfc > cpu.bpf
{ 0x20, 0, 0, 0xfffff024 },
{ 0x15, 0, 1, 0x00000000 },
{ 0x6, 0, 0, 0xffffffff },
{ 0x6, 0, 0, 0x00000000 },
```

- Run with netsniff-ng

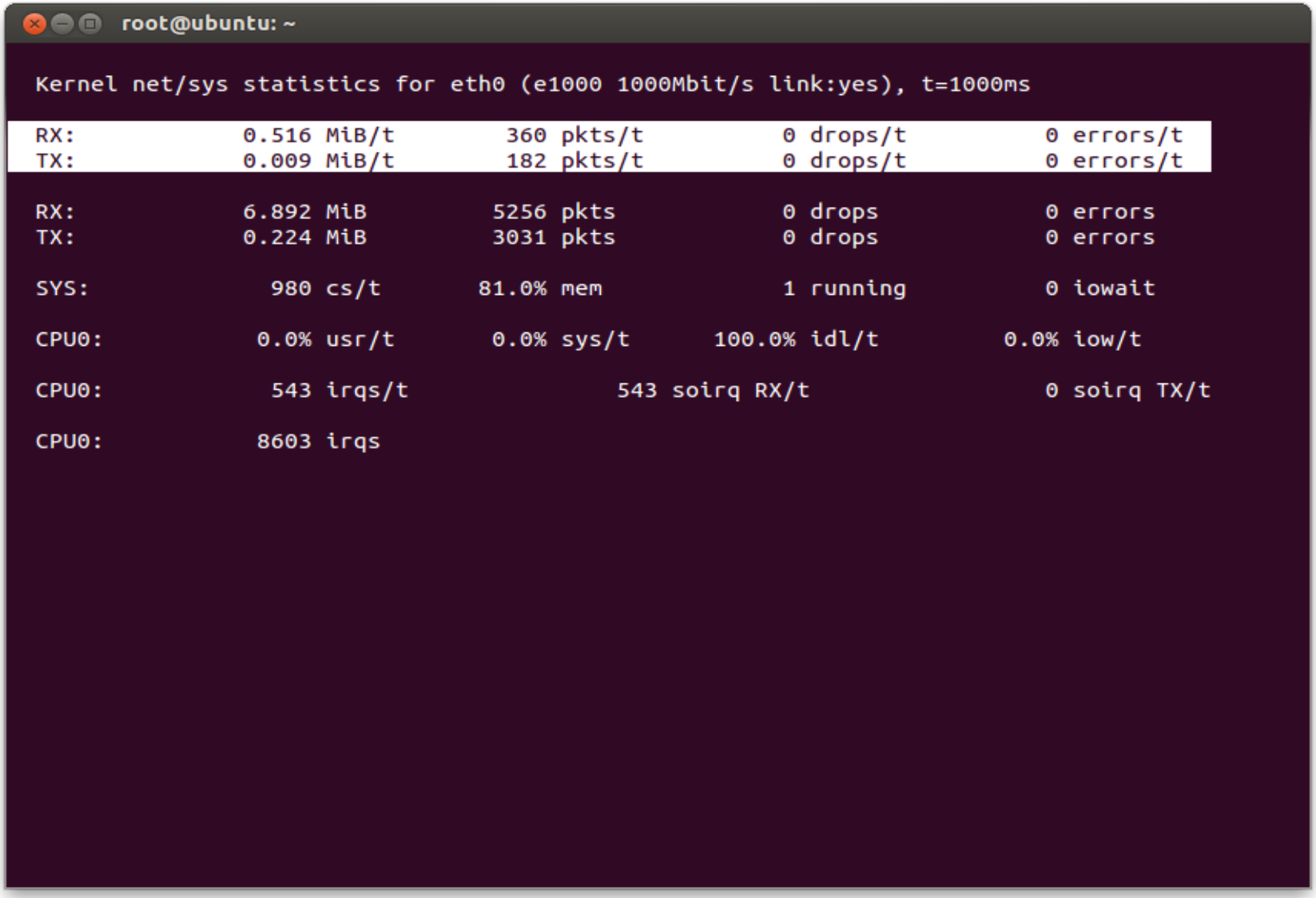
```
$ netsniff-ng --in eth0 -f cpu.bpf
```

Ifpps

- Reads kernel stats from **procfs**, e.g.
  `$ watch -n 1 "cat /proc/net/dev | column -t"`
- Does **not** use user space monitoring libraries which lead to inaccurate statistics under high load.
  1. What some people do: iptraf (libpcap): 246,000 pps
  2. What the system actually sees: ifpps: 1,378,000 pps
- Replace use of vmstat and ifstat (in conjunction) with a single ifpps window. PPS, BPS, CPU, IRQ, I/O Wait etc.

```
$ ifpps eth0
$ ifpps --promisc --dev eth0
$ ifpps --loop -p --csv -d wlan0 > gnuplot.dat
```

Ifpps, Cont. 1


root@ubuntu: ~
Kernel net/sys statistics for eth0 (e1000 1000Mbit/s link:yes), t=1000ms
RX: 0.516 MiB/t 360 pkts/t 0 drops/t 0 errors/t
TX: 0.009 MiB/t 182 pkts/t 0 drops/t 0 errors/t
RX: 6.892 MiB 5256 pkts 0 drops 0 errors
TX: 0.224 MiB 3031 pkts 0 drops 0 errors
SYS: 980 cs/t 81.0% mem 1 running 0 iowait
CPU0: 0.0% usr/t 0.0% sys/t 100.0% idl/t 0.0% iow/t
CPU0: 543 irqs/t 543 soirq RX/t 0 soirq TX/t
CPU0: 8603 irqs

Ifpps, Cont. 2

- Pull data out of the CSV format (not technically CSV, uses spaces, not commas)

```
$ awk '$2 ~ !/[a-zA-Z]/ { print "Packets Per Second:
" $3 }' ifpps.csv | head -3

Packets Per Second:  225810
Packets Per Second:  256062
Packets Per Second:  240789
```

- Convert timestamp to a human readable format with gawk

```
$ gawk '$4 > 0 || $5 > 0 { print strftime("%x-%X",$1)
" | Drops:  "$4 " Errors:  "$5 }'

01/20/2013-07:53:45 AM | Drops:  900 Errors:  0
01/20/2013-07:54:02 AM | Drops:  100 Errors:  0
01/20/2013-07:54:58 AM | Drops:  0 Errors:  3
```

Flowtop

- Top-like ncurses connection tracker
- Built on libnetfilter_conntrack library (requires iptables)
- GeoIP, process name, PID, port/service, reverse DNS

```
Kernel netfilter TCP/UDP flow statistics, [+0]

[wget(4002)]:ipv4:tcp[ESTABLISHED]:ftp ->
      dst: 174.137.42.65:21 (United States, Sacramento)
[wget(4005)]:ipv4:tcp[TIME_WAIT]:http ->
      dst: ir1.fp.vip.ne1.yahoo.com:80 (United States, Sunnyvale)
ipv4:tcp[TIME_WAIT]:http ->
      dst: ve-in-f147.1e100.net:80 (United States, Mountain View)
ipv4:tcp[TIME_WAIT]:http ->
      dst: ord08s07-in-f6.1e100.net:80 (United States, Mountain View)
[wget(4002)]:ipv4:tcp[TIME_WAIT] ->
      dst: 174.137.42.65:36255 (United States, Sacramento)
ipv4:tcp[TIME_WAIT] ->
      dst: 174.137.42.65:62155 (United States, Sacramento)
ipv4:tcp[TIME_WAIT] ->
      dst: 174.137.42.65:13281 (United States, Sacramento)
ipv4:tcp[CLOSE]:http ->
      dst: ir1.fp.vip.gq1.yahoo.com:80 (United States, Sunnyvale)
```

ASTraceroute

- Traceroute utility
- Supports IPv6
- Displays Autonomous System information
- GeoIP location

```
$ astraceroute --ipv6 -i eth0 --numeric --show-packet
--psh --totlen 1400 --payload "abcdefg123456" -H
2607:f8b0:4009:803::100e
...
12:  * * * 202.97.52.205 in AS4134 (CN, China,
Beijing, 39.928902, 116.388298), 202.97.32.0/19 apnic
(1998-08-17), CHINANET-BACKBONE No.31,Jin-rong Street
```

Installation

- Always recommend cloning from the main repository
  1. Actively developed
  2. Latest features
  3. Latest fixes
- Debian/Ubuntu installation example

```
$ apt-get install git build-essential flex bison
ccache libnl-3-dev libnl-genl-3-dev libgeoip-dev
libnetfilter-conntrack-dev libncurses5-dev
liburcu-dev libnet1-dev libpcap-dev zlib1g-dev
$ git clone https://github.com/borkmann/netsniff-ng
$ cd netsniff-ng
$ make && make install
```

Simple!

What Could Be Next?

- **astraceroute**
  - DNS traceroute to detect malicious DNS injections on transit traffic (SIGCOMM 2012 paper)
- **mausezahn**
  - Improve imported code and integrate into the main repo
- **netsniff-ng, mausezahn**
  - New proto dissectors/generators like SCTP, DCCP, BGP, etc.
- **netsniff-ng**
  - Compressed on-the-fly bitmap indexing for large PCAP files
  - Try to find sane way to utilize multicore with packet_fanout
- **netsniff-ng, trafgen, mausezahn**
  - Optimize performance (`AF_PACKET plumbing`)
  - Performance benchmark on 10Gbit/s
- **flowtop**
  - Byte and packet counters
- Toolkit integration into RHEL!

Notes & References

- With permission reused parts of Daniel's Red Hat talk
  http://pub.netsniff-ng.org/paper/devconf_2013.pdf
- Trafgen Expression Language article
  https://sickbits.net/trafgen-expression-language/
- Please read manuals & documentation for more information
  github.com/borkmann/netsniff-ng/tree/master/man
  http://pub.netsniff-ng.org/docs/
- Linux Distribution specific documentation
  https://help.ubuntu.com/community/Netsniff-NG
  https://fedoraproject.org/wiki/Help:Netsniff-NG
- Find configurations via Google
  `ext:txf | ext:cfg | ext:bpf netsniff`

Conclusion

Thanks!!!

netsniff-ng@googlegroups.com
github.com/borkmann/netsniff-ng