TcpdumpExamples

`tcpdump (examples)`

Here are some more `tcpdump` examples for some more advanced use cases. For simple usage examples, see the main tcpdump topic.

Filter on protocol (ICMP) and protocol-specific fields (ICMP type)

Capture all ICMP with some exceptions. For example, if a host runs lots of pings (SmokePing for example), it is useful to suppress ICMP echo requests and replies from dumped packets:

```
: root@myhost:~# tcpdump -n icmp and 'icmp[0] != 8 and icmp[0] != 0'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
10:05:02.338826 IP 10.10.33.34 > 10.10.121.2: icmp 36: time exceeded in-transit
10:05:02.587494 IP 10.10.144.66 > 10.10.121.2: icmp 36: host 10.10.144.69 unreachable - admin prohibited filter
10:05:02.699110 IP 10.10.153.118 > 10.10.121.2: icmp 36: host 10.10.153.122 unreachable - admin prohibited
filter
10:05:04.319451 IP 10.10.33.34 > 10.10.121.2: icmp 36: time exceeded in-transit
10:05:07.363278 IP 10.10.148.138 > 10.10.121.2: icmp 36: host 10.10.148.138 unreachable - admin prohibited
filter
10:05:10.220491 IP 10.10.33.34 > 10.10.121.2: icmp 36: time exceeded in-transit
10:05:10.476082 IP 10.10.144.66 > 10.10.121.2: icmp 36: host 10.10.144.69 unreachable - admin prohibited filter
10:05:10.638611 IP 10.10.153.118 > 10.10.121.2: icmp 36: host 10.10.153.122 unreachable - admin prohibited
filter

8 packets captured
8 packets received by filter
0 packets dropped by kernel
```

Same command can be used with predefined header field offset (`icmptype`) and ICMP type field values (`icmp-echo` and `icmp-echoreply`):

```
: root@myhost:~# tcpdump -n icmp and icmp[icmptype] != icmp-echo and icmp[icmptype] != icmp-echoreply
```

Filter on TOS field

Capture all IP packets with a non-zero TOS field
(one byte TOS field is at offset 1 in IP header):

```
: root@myhost:~# tcpdump -v -n ip and ip[1]!=0
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
10:40:03.422052 IP (tos 0x10, ttl  64, id 58534, offset 0, flags [DF], proto 6, length: 100) 10.10.121.2.ssh >
10.10.1.240.33006: P 166745014:16674506
10:40:03.422189 IP (tos 0x10, ttl  64, id 58536, offset 0, flags [DF], proto 6, length: 164) 10.10.121.2.ssh >
10.10.1.240.33006: P 48:160(112) ack 1
10:40:03.422325 IP (tos 0x10, ttl  64, id 58538, offset 0, flags [DF], proto 6, length: 308) 10.10.121.2.ssh >
10.10.1.240.33006: P 160:416(256) ack 1
10:40:03.422906 IP (tos 0x10, ttl  62, id 29167, offset 0, flags [DF], proto 6, length: 52) 10.10.1.240.33006 >
10.10.121.2.ssh: . [tcp sum ok] ack 48
...
```

Filter on TTL field

Capture all IP packets with TTL less than some value (on byte TTL field is at offset 8 in IP header):

```
: root@myhost:~# tcpdump -v ip and 'ip[8]<2'
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
10:30:51.013620 IP (tos 0xc0, ttl   1, id 44119, offset 0, flags [none], proto 2, length: 28) ltest1.arnes.si >
239.255.255.255: igmp v2 report 239.25
10:30:54.035124 IP (tos 0xc0, ttl   1, id 44120, offset 0, flags [none], proto 2, length: 28) ltest1.arnes.si >
CISCO-RP-DISCOVERY.MCAST.NET: igmp v2
10:30:56.049046 IP (tos 0xc0, ttl   1, id 44121, offset 0, flags [none], proto 2, length: 28) ltest1.arnes.si >
SAP.MCAST.NET: igmp v2 report SAP.MCAS
10:30:56.051242 IP (tos 0xc0, ttl   1, id 44122, offset 0, flags [none], proto 103, length: 726) ltest1.arnes.
si > PIM-ROUTERS.MCAST.NET: PIMv2, lengt
        Join / Prune (3), upstream-neighbor: rarnes13-F2-0x200.arnes.si
          1 group(s), holdtime: 3m30s
            group #1: SAP.MCAST.NET, joined sources: 85, pruned sources: 0
              joined source #1: hayakawa.lava.net(S)
              joined source #2: 64.251.62.34(S)
              joined source #3: 64.251.62.35(S)
              joined source #4: 64.251.62.36(S)
              joined source #5: ...)

4 packets captured
4 packets received by filter
0 packets dropped by kernel
```

Filter on TCP flags (SYN/ACK)

Catch TCP SYN packets:

```
: root@myhost:~# tcpdump -n tcp and port 80 and 'tcp[tcpflags] & tcp-syn == tcp-syn'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
13:15:00.302219 IP 10.10.1.240.33111 > 10.10.121.2.http: S 3284556452:3284556452(0) win 5840 <mss 1460,sackOK,
timestamp 16261279 0,nop,wscale 10>
13:15:00.302272 IP 10.10.121.2.http > 10.10.1.240.33111: S 975107341:975107341(0) ack 3284556453 win 32767 <mss
1460,sackOK,timestamp 2432550825 16261
```

In the example above, all packets with TCP SYN flag set are captured. Other flags (ACK, for example) might be set also. Packets which have only TCP SYN flags set, can be captured like this:

```
: root@myhost:~# tcpdump tcp and port 80 and 'tcp[tcpflags] == tcp-syn'
```

Catch TCP SYN/ACK packets (typically, responses from servers):

```
: root@myhost:~# tcpdump -n tcp and 'tcp[tcpflags] & (tcp-syn|tcp-ack) == (tcp-syn|tcp-ack)'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
13:30:19.501816 IP 10.10.121.2.ssh > 10.10.1.240.33114: S 1940763772:1940763772(0) ack 4250485572 win 32767
<mss 1460,sackOK,timestamp 2433470257 1718
```

Same thing:

```
: root@myhost:~# tcpdump -n tcp and 'tcp[tcpflags] & tcp-syn == tcp-syn' and 'tcp[tcpflags] & tcp-ack == tcp-
ack'
```

Catch ARP packets

```
: root@myhost:~# tcpdump -vv -e -nn ether proto 0x0806
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
08:50:35.842999 00:30:48:27:xx:ff > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806), length 60: arp who-has 193.2.x.y
tell 193.2.x.w
08:50:36.841814 00:30:48:27:xx:ff > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806), length 60: arp who-has 193.2.x.y
tell 193.2.x.w
08:50:37.841396 00:30:48:27:xx:ff > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806), length 60: arp who-has 193.2.x.y
tell 193.2.x.w
...
08:53:18.913968 00:14:38:00:e7:d7 > 00:30:48:27:6a:ff, ethertype ARP (0x0806), length 42: arp reply 193.2.x.z
is-at 00:14:38:00:xx:yy
```

Filter on IP packet length

Catch packets of a specified length (IP packet length (16 bits) is located at offset 2 in IP header):

```
: root@myhost:~# tcpdump -l icmp and '(ip[2:2]>50)' -w - |tcpdump -r - -v ip and '(ip[2:2]<60)'
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
reading from file -, link-type EN10MB (Ethernet)
11:03:59.420856 IP (tos 0x0, ttl 249, id 0, offset 0, flags [none], proto 1, length: 56) lpttlj3-tk.arnes.si >
myhost.arnes.si: icmp 36: time exceeded
11:04:02.274135 IP (tos 0x0, ttl 251, id 17095, offset 0, flags [none], proto 1, length: 56) rsik-orm.arnes.si
> myhost.arnes.si: icmp 36: host rsik-o
11:04:05.452802 IP (tos 0x0, ttl 249, id 34021, offset 0, flags [none], proto 1, length: 56) 10.10.144.66 >
myhost.arnes.si: icmp 36: host ro-mislinja
11:04:05.496384 IP (tos 0x0, ttl 251, id 4071, offset 0, flags [none], proto 1, length: 56) rs-sb-mb.arnes.si >
myhost.arnes.si: icmp 36: host ss-sb-m
<^C>
814 packets captured
814 packets received by filter
0 packets dropped by kernel
tcpdump: pcap_loop: error reading dump file: Interrupted system call
```

Remark: due to some bug in tcpdump, the following command **doesn't** catch packets as expected:

```
: root@myhost:~# tcpdump -v -n icmp and '(ip[2:2]>50)' and '(ip[2:2]<60)'
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
[no output]
```

Because of this, two `tcpdumps` were used in the example above (`tcpdump -l ... -w - |tcpdump -r -...`).
Option `-l` is needed to force first `tcpdump` program to output captured data imeadiately to the second program.

Filter on encapsulated content (ICMP within PPPoE)

Capturing packets from PPPoE session. For example: we mirror a link that connects xDSL modem and home PC or router.
Mirrored packets are ethernet frames with PPPoE/IP packets encapsulated.
In the following example, we are looking for ICMP packets in PPPoE frames. A simple command like

```
: root@myhost:~# tcpdump -v -n icmp
```

will **not** produce expected results, because packets that we monitor are being encapsulated into a PPPoE frames.
Of course, tcpdump can't locate IP protocol {{}} ICMP at normal offset in an ethernet frame.
We must therefore take into account the additional headers: 14 bytes for ethernet and 8 bytes for PPPoE.
IP protocol is located at offset 9 in the IP header, which gives us offset 31 in the mirrored ethernet
frame. Therefore, ICMP packets (protokol 1) are captured with

```
: root@myhost:~# tcpdump -v -n ether[31] = 1
```

Simultaneous output to dump file and (decoded) standard output

You may want to dump packets to a file, but still see the decoded headers "live" on your terminal. While this is not supported *directly* by `tcpdump` you can use the powerful pipe mechanism to obtain this effect:

```
: leinen@bonadea[leinen]; sudo tcpdump -s 0 -i tun0 -c 10 -w - -U | tee foo.pcap | tcpdump -n -r -
tcpdump: WARNING: arptype 65534 not supported by libpcap - falling back to cooked socket
tcpdump: listening on tun0, link-type LINUX_SLL (Linux cooked), capture size 65535 bytes
reading from file -, link-type LINUX_SLL (Linux cooked)
11:04:19.980160 IP 130.59.28.18 > 130.59.10.36: ICMP echo request, id 32872, seq 65, length 64
11:04:19.984536 IP 130.59.10.36 > 130.59.28.18: ICMP echo reply, id 32872, seq 65, length 64
11:04:20.984104 IP 130.59.28.18 > 130.59.10.36: ICMP echo request, id 32872, seq 66, length 64
11:04:20.987571 IP 130.59.10.36 > 130.59.28.18: ICMP echo reply, id 32872, seq 66, length 64
11:04:21.992102 IP 130.59.28.18 > 130.59.10.36: ICMP echo request, id 32872, seq 67, length 64
11:04:21.995676 IP 130.59.10.36 > 130.59.28.18: ICMP echo reply, id 32872, seq 67, length 64
11:04:22.996109 IP 130.59.28.18 > 130.59.10.36: ICMP echo request, id 32872, seq 68, length 64
11:04:22.999714 IP 130.59.10.36 > 130.59.28.18: ICMP echo reply, id 32872, seq 68, length 64
11:04:24.004176 IP 130.59.28.18 > 130.59.10.36: ICMP echo request, id 32872, seq 69, length 64
10 packets captured
10 packets received by filter
0 packets dropped by kernel
11:04:24.007083 IP 130.59.10.36 > 130.59.28.18: ICMP echo reply, id 32872, seq 69, length 64
: leinen@bonadea[leinen]; ls -l foo.pcap
-rw-r--r-- 1 leinen leinen 1184 2008-11-28 11:04 foo.pcap
```

Explanation: The first `tcpdump` call captures the packets, and dumps the (binary) data to standard output (`-w -=). The =-U` (unbuffered) flag causes each packet to be written out immediately, circumventing the normal output buffering. This preserves the real-time characteristics better. The binary packets are piped to the `tee` command, which writes them to a file (`foo.pcap=) and at the same time outputs them again on standard output. From there, they are decoded using =tcpdump -r -`.

– Main.MatjazStraus - 01 Oct 2007

-- Main.SimonLeinen - 28 Nov 2008