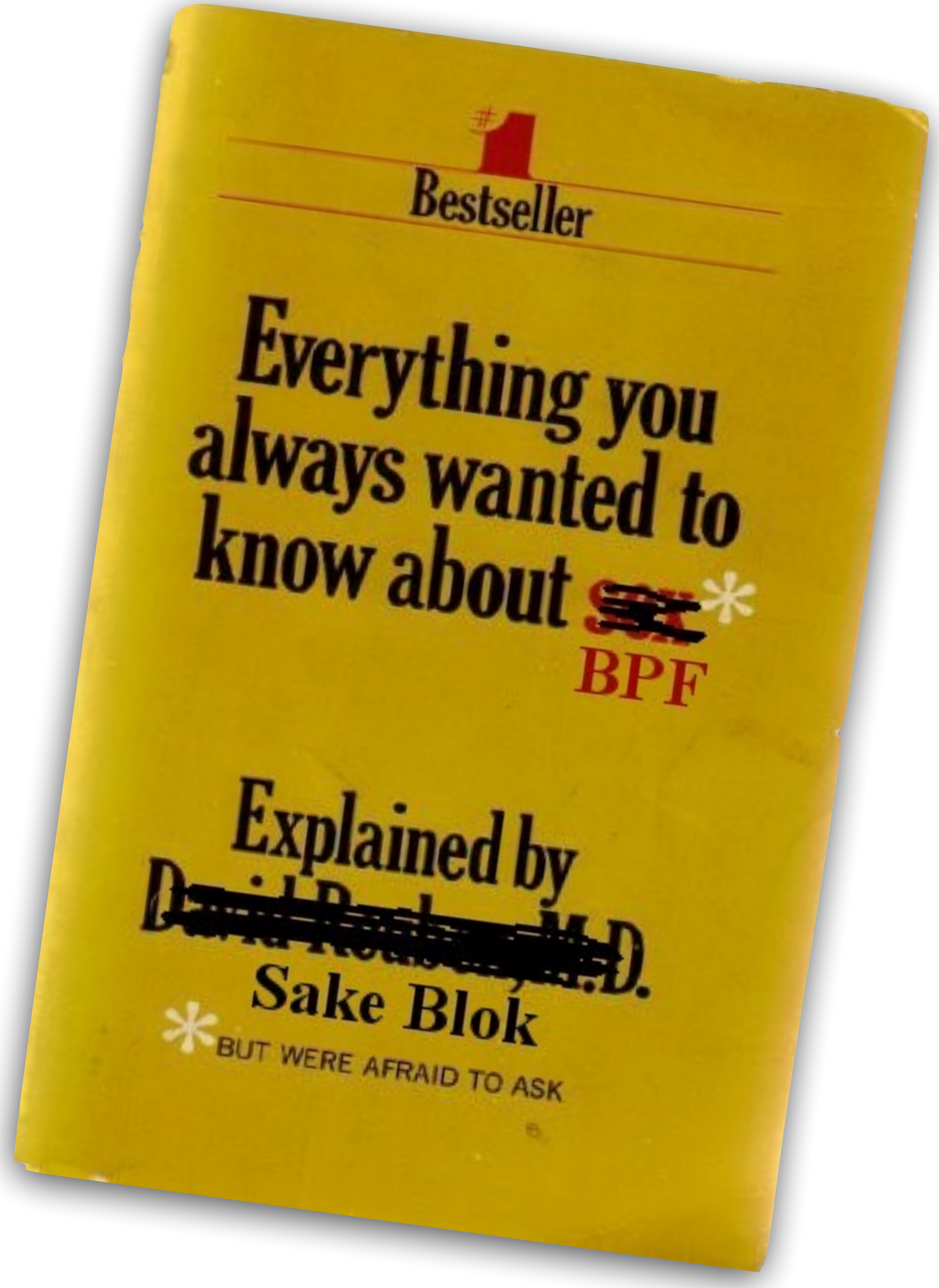
Capture Filter Sorcery

How to Use Complex BPF Capture Filters in Wireshark

June 15th, 2016



Sake Blok

sake.blok@SYN-bit.nl

Relational Therapist for Computer Systems | SYN-bit

About me...

EURONET INTERNET

techno inside

ABN·AMRO

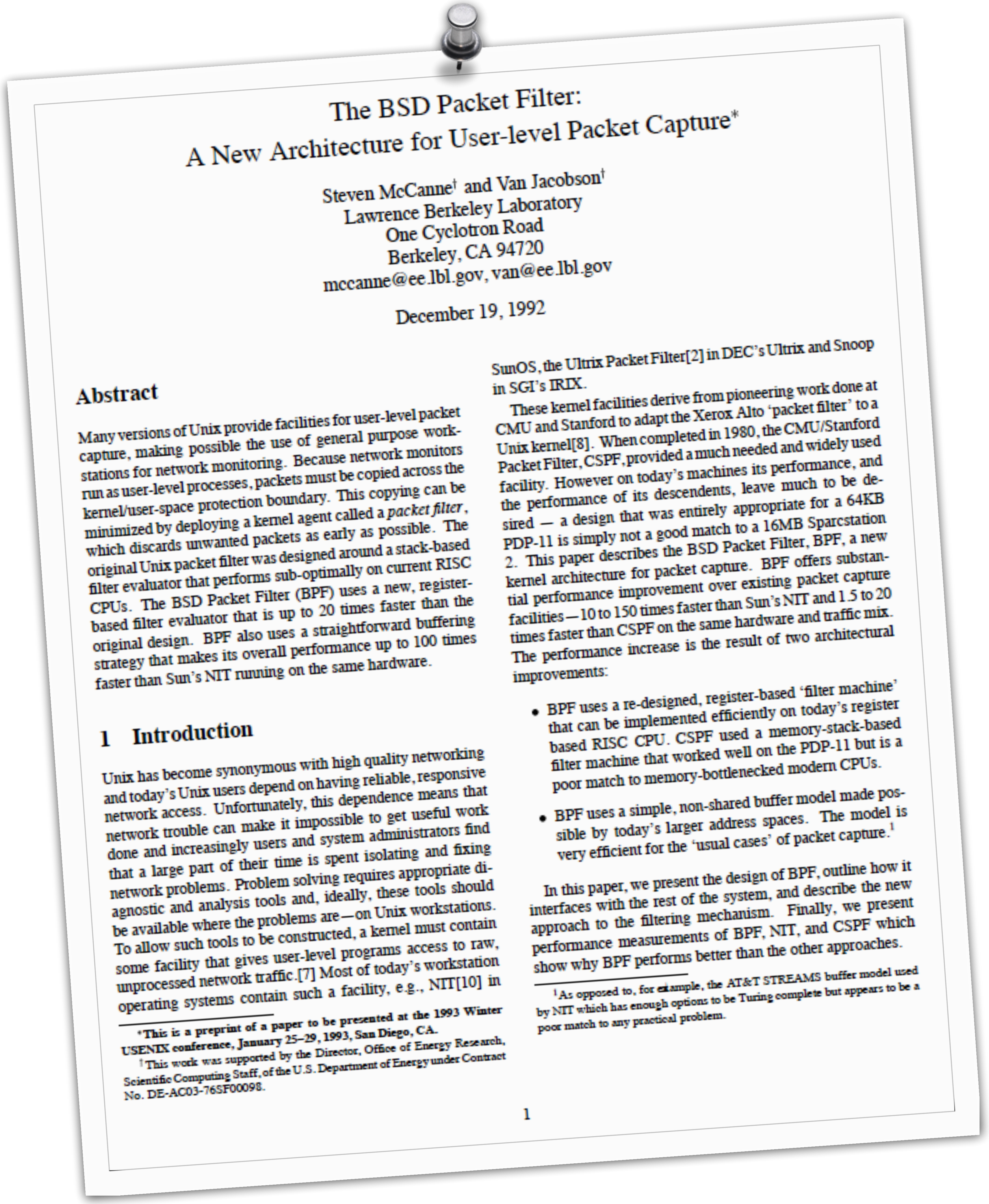
SYN-bit
deep traffic analysis

ion ip

f5

Haarlem

The history of BPF



The BSD Packet Filter:
A New Architecture for User-level Packet Capture*

Steven McCanne† and Van Jacobson†
Lawrence Berkeley Laboratory
One Cyclotron Road
Berkeley, CA 94720
mccanne@ee.lbl.gov, van@ee.lbl.gov

December 19, 1992

Abstract

Many versions of Unix provide facilities for user-level packet capture, making possible the use of general purpose workstations for network monitoring. Because network monitors run as user-level processes, packets must be copied across the kernel/user-space protection boundary. This copying can be minimized by deploying a kernel agent called a *packet filter*, which discards unwanted packets as early as possible. The original Unix packet filter was designed around a stack-based filter evaluator that performs sub-optimally on current RISC CPUs. The BSD Packet Filter (BPF) uses a new, register-based filter evaluator that is up to 20 times faster than the original design. BPF also uses a straightforward buffering strategy that makes its overall performance up to 100 times faster than Sun's NIT running on the same hardware.

1 Introduction

Unix has become synonymous with high quality networking and today's Unix users depend on having reliable, responsive network access. Unfortunately, this dependence means that network trouble can make it impossible to get useful work done and increasingly users and system administrators find that a large part of their time is spent isolating and fixing network problems. Problem solving requires appropriate diagnostic and analysis tools and, ideally, these tools should be available where the problems are—on Unix workstations. To allow such tools to be constructed, a kernel must contain some facility that gives user-level programs access to raw, unprocessed network traffic.[7] Most of today's workstation operating systems contain such a facility, e.g., NIT[10] in SunOS, the Ultrix Packet Filter[2] in DEC's Ultrix and Snoop in SGI's IRIX.

These kernel facilities derive from pioneering work done at CMU and Stanford to adapt the Xerox Alto 'packet filter' to a Unix kernel[8]. When completed in 1980, the CMU/Stanford Packet Filter, CSPF, provided a much needed and widely used facility. However on today's machines its performance, and the performance of its descendents, leave much to be desired — a design that was entirely appropriate for a 64KB PDP-11 is simply not a good match to a 16MB Sparcstation 2. This paper describes the BSD Packet Filter, BPF, a new kernel architecture for packet capture. BPF offers substantial performance improvement over existing packet capture facilities—10 to 150 times faster than Sun's NIT and 1.5 to 20 times faster than CSPF on the same hardware and traffic mix. The performance increase is the result of two architectural improvements:

- BPF uses a re-designed, register-based 'filter machine' that can be implemented efficiently on today's register based RISC CPU. CSPF used a memory-stack-based filter machine that worked well on the PDP-11 but is a poor match to memory-bottlenecked modern CPUs.
- BPF uses a simple, non-shared buffer model made possible by today's larger address spaces. The model is very efficient for the 'usual cases' of packet capture.[1]

In this paper, we present the design of BPF, outline how it interfaces with the rest of the system, and describe the new approach to the filtering mechanism. Finally, we present performance measurements of BPF, NIT, and CSPF which show why BPF performs better than the other approaches.

*This is a preprint of a paper to be presented at the 1993 Winter USENIX conference, January 25–29, 1993, San Diego, CA.
†This work was supported by the Director, Office of Energy Research, Scientific Computing Staff, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

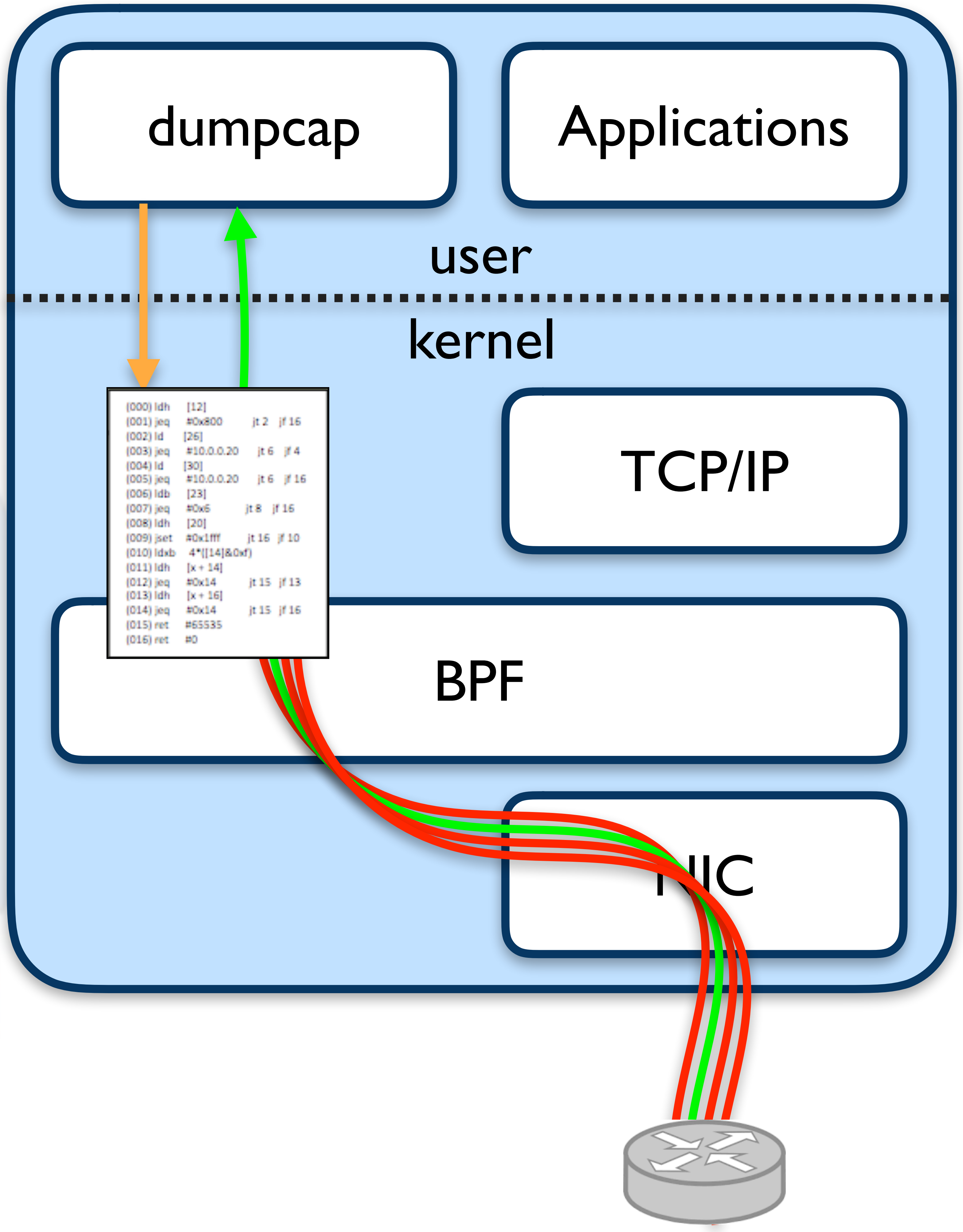
[1] As opposed to, for example, the AT&T STREAMS buffer model used by NIT which has enough options to be Turing complete but appears to be a poor match to any practical problem.

1

https://youtu.be/XHlqIqPvKw8

Architecture…

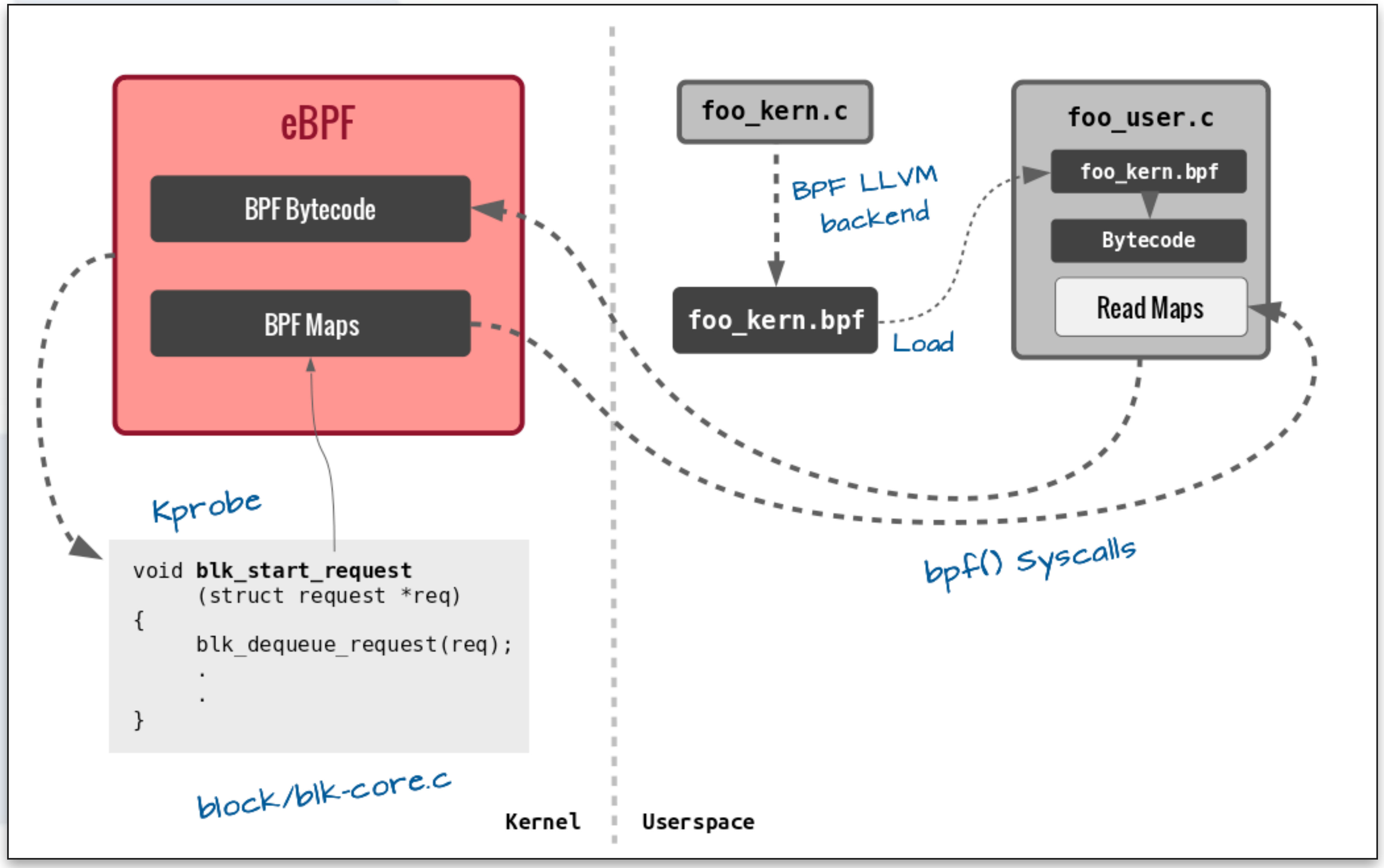
- BPF virtual machine lives inside the kernel
- Userspace program (like dumpcap) compiles a filter into BPF bytecode
- Userspace program loads bytecode into the BPF vm
- BPF vm uses bytecode to filter packets
- No need to copy non-matching packets to user space
- All jumps are forward to prevent kernel loops
- Single BPF program max 4096 instructions



Extended use…

- Network/Packet related
  - iptables filtering rules (used by cloudflare for instance)
  - Network Function Virtualization (NFV)
  - Myself: DDoS protection on F5 for a customer

- (Linux) system related
  - Internal BPF
  - eBPF



Back to network packets…

- Why do you want to filter?
  - Overloading the CPU
  - Not enough disk I/O
- Why not?
  - You might miss important packets
- Best strategy:
  - Do not filter if you don't need to
  - Use snaplength if you do not need payload
  - Filter out what you don't want to see
  - Filter specifically for what you expect to see

Example of reducing packets

- sctp.chunk_type == 4 || sctp.chunk_type==5 : 22089 frames
  - (vlan and (ip[2:2] - 32 - sctp[14:2]=0 and (sctp[12]=4 or sctp[12]=5))) or
    (vlan and (ip[2:2] - 32 - sctp[14:2]=0 and (sctp[12]=4 or sctp[12]=5)))
- vrrp : 13441 frames
  - ether dst host 01:00:5e:00:00:12 or 33:33:00:00:00:12
- bfd : 2284 frames
  - vlan and vlan and ip and udp dst port 3784 and udp[4:2]=32 and udp[11:1]=24
- ospf : 2251 frames
  - ether dst host 01:00:5e:00:00:05
- arp : 1960 frames
  - arp or (vlan and arp) or (vlan and arp)
- slow : 848 frames
  - ether dst host 01:80:c2:00:00:02
- hsrp : 192 frames
  - ether dst host 01:00:5e:00:00:02
- stp : 126 frames
  - ether dst host 01:00:0c:cc:cc:cd or 01:80:c2:00:00:00

90%
80%
70%
60%

Example of reducing packets

- 6910 packets kept out of 50101 (~86% reduction)
- 1.672.554 bytes kept out of 10.043.072 (~83% reduction)

- Final filter:
  - not (ether dst host 01:00:5e:00:00:12 or 33:33:00:00:00:12 or 01:00:5e:00:00:05 or 01:80:c2:00:00:02 or 01:00:5e:00:00:02 or 01:00:0c:cc:cc:cd or 01:80:c2:00:00:00 or arp or (vlan and (sctp[12]=4 or sctp[12]=5 or arp)) or (vlan and (sctp[12]=4 or sctp[12]=5 or (udp dst port 3784 and udp[4:2]=32 and udp[11:1]=24) or arp)))

Performance in post capture filtering

- display filters will first fully dissect a packet, then filter
- BPF filters will just look at specific offsets for specific values
  - first mismatch will make it go to the next packet
- using BPF in post capture processing not possible (yet?) in wireshark/ tshark
- Use tcpdump or windump instead (for now)

BPF virtual machine

- Created with motorola 6502 in mind
  - Accumulator (A), index register (X)
  - Packet-based memory model
  - Direct/indirect addressing
  - Arithmetic and Conditional logic

- Bytecode
  - list of fixed size instructions
  - opcode (16 bits)
  - jump true (8 bits)
  - jump false (8 bits)
  - 'k' value (32 bits)

```
{ 0x28, 0, 0,  0x0000000c },
{ 0x15, 0, 14, 0x00000800 },
{ 0x20, 0, 0,  0x0000001a },
{ 0x15, 2, 0,  0xc0a80064 },
{ 0x20, 0, 0,  0x0000001e },
{ 0x15, 0, 10, 0xc0a80064 },
{ 0x30, 0, 0,  0x00000017 },
{ 0x15, 0, 8,  0x00000006 },
{ 0x28, 0, 0,  0x00000014 },
{ 0x45, 6, 0,  0x00001fff },
{ 0xb1, 0, 0,  0x0000000e },
{ 0x48, 0, 0,  0x0000000e },
{ 0x15, 2, 0,  0x00000050 },
{ 0x48, 0, 0,  0x00000010 },
{ 0x15, 0, 1,  0x00000050 },
{ 0x6,  0, 0,  0x00040000 },
{ 0x6,  0, 0,  0x00000000 },
```

Mnemonics

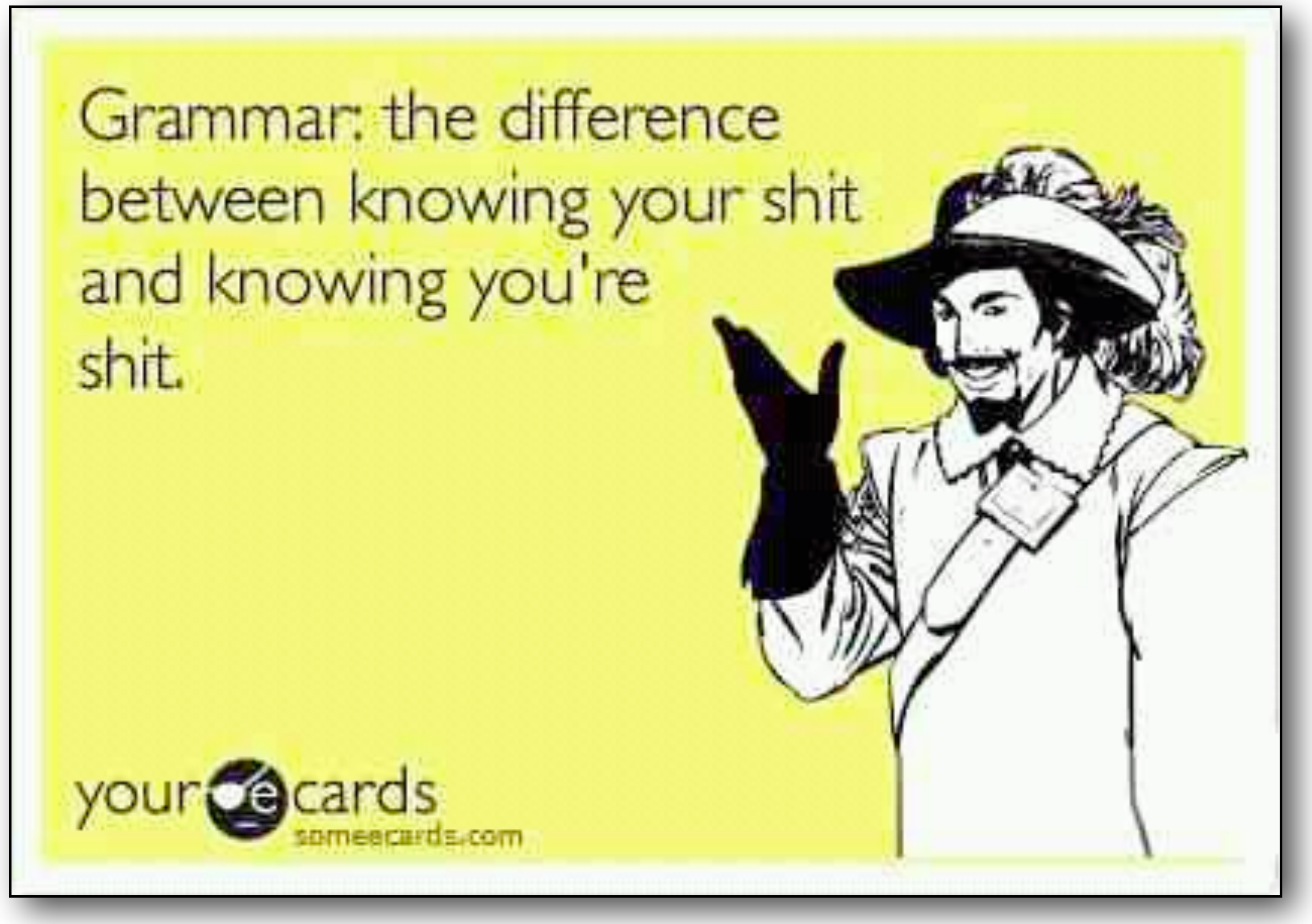
```
Instruction        Addressing mode      Description

ld                 1, 2, 3, 4           Load word into A
ldi                4                    Load word into A
ldh                1, 2                 Load half-word into A
ldb                1, 2                 Load byte into A
ldx                3, 4, 5              Load word into X
ldxi               4                    Load word into X
ldxb               5                    Load byte into X

st                 3                    Copy A into M[]
stx                3                    Copy X into M[]

jmp                6                    Jump to label
ja                 6                    Jump to label
jeq                7, 8                 Jump on k == A
jneq               8                    Jump on k != A
jne                8                    Jump on k != A
jlt                8                    Jump on k < A
jle                8                    Jump on k <= A
jgt                7, 8                 Jump on k > A
jge                7, 8                 Jump on k >= A
jset               7, 8                 Jump on k & A

add                0, 4                 A + <x>
sub                0, 4                 A - <x>
mul                0, 4                 A * <x>
div                0, 4                 A / <x>
mod                0, 4                 A % <x>
neg                0, 4                 !A
and                0, 4                 A & <x>
or                 0, 4                 A | <x>
xor                0, 4                 A ^ <x>
lsh                0, 4                 A << <x>
rsh                0, 4                 A >> <x>

tax                                     Copy A into X
txa                                     Copy X into A

ret                4, 9                 Return
```

```
Addressing mode Syntax               Description

0               x/%x                 Register X
1               [k]                  BHW at byte offset k in the packet
2               [x + k]              BHW at the offset X + k in the packet
3               M[k]                 Word at offset k in M[]
4               #k                   Literal value stored in k
5               4*([k]&0xf)          Lower nibble * 4 at byte offset k in the packet
6               L                    Jump label L
7               #k,Lt,Lf             Jump to Lt if true, otherwise jump to Lf
8               #k,Lt                Jump to Lt if predicate is true
9               a/%a                 Accumulator A
```

```
(000) ldh      [12]
(001) jeq      #0x800          jt 2     jf 16
(002) ld       [26]
(003) jeq      #0xc0a80064     jt 6     jf 4
(004) ld       [30]
(005) jeq      #0xc0a80064     jt 6     jf 16
(006) ldb      [23]
(007) jeq      #0x6            jt 8     jf 16
(008) ldh      [20]
(009) jset     #0x1fff         jt 16    jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x50           jt 15    jf 13
(013) ldh      [x + 16]
(014) jeq      #0x50           jt 15    jf 16
(015) ret      #262144
(016) ret      #0
```

BPF filter language

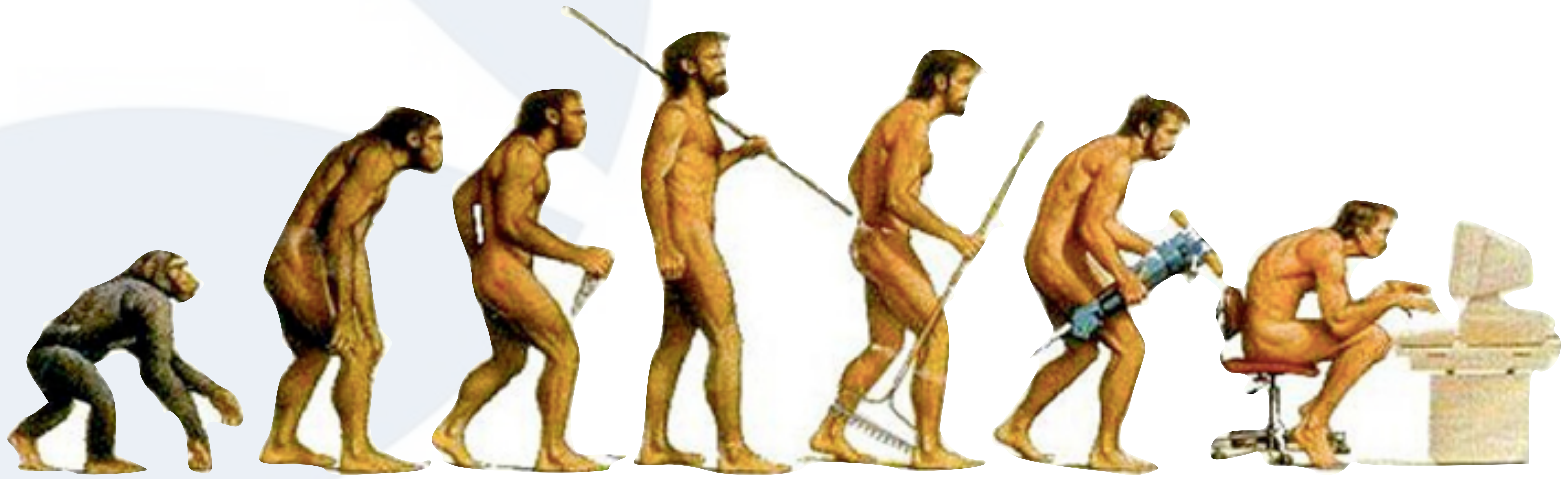
- The filter expression consists of one or more primitives.
- Primitives usually consist of an id preceded by one or more qualifiers.
- There are three different kinds of qualifier:
  - **type** qualifiers say what kind of thing the id name or number refers to.
  - **dir** qualifiers specify a particular transfer direction to and/or from id.
  - **proto** qualifiers restrict the match to a particular protocol.
- Use **and**, **or** and **not** to combine primitives and create complex filters
- To save typing, identical qualifier lists can be omitted
- Filter arithmetics



Primitives

- Primitives usually consist of an id preceded by one or more qualifiers.
- There are three different kinds of qualifier:
  - type qualifiers say what kind of thing the id name or number refers to.
  - dir qualifiers specify a particular transfer direction to and/or from id.
  - proto qualifiers restrict the match to a particular protocol.

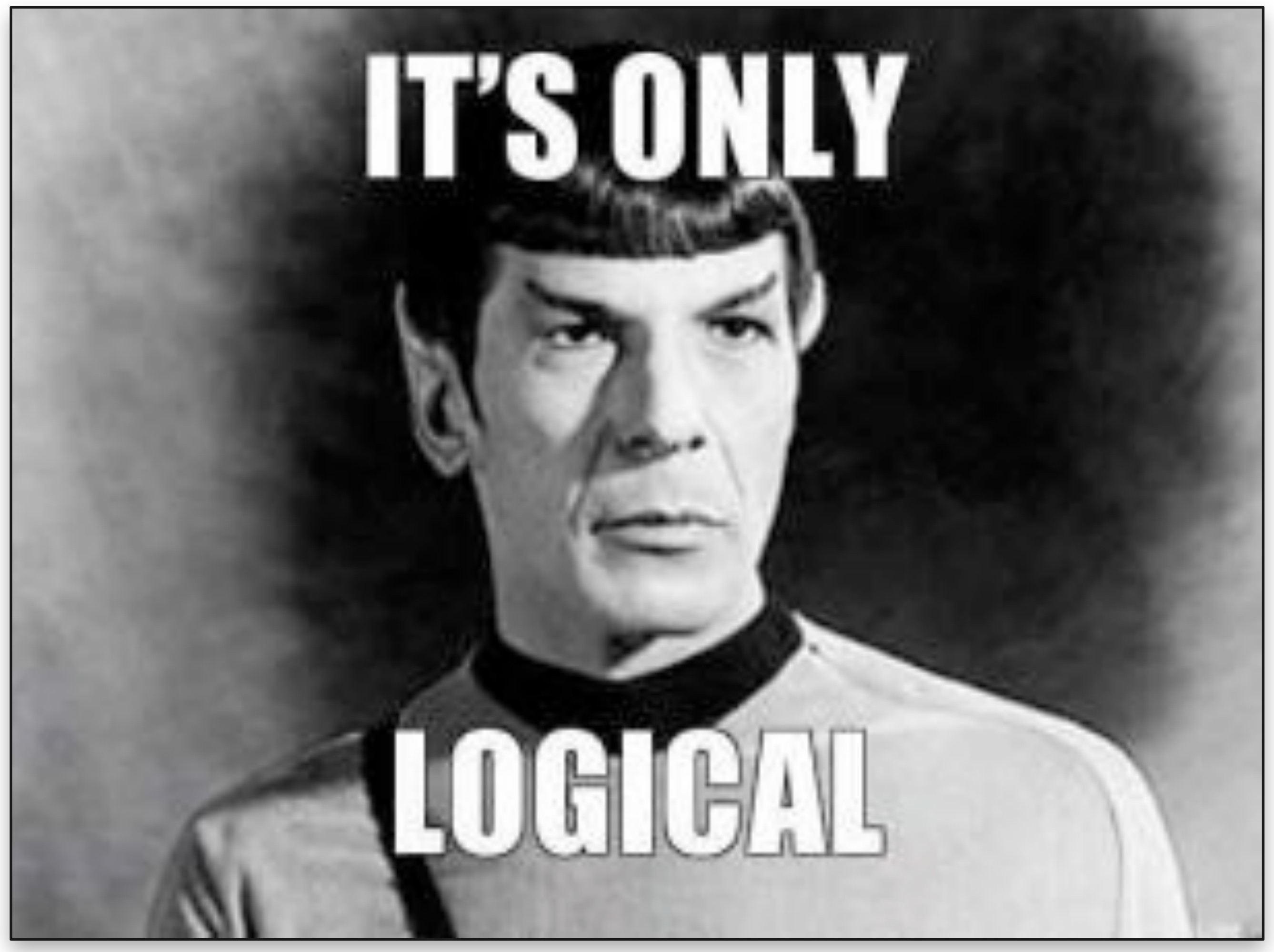
- Examples
  - host sharkfest.wireshark.org
  - src host 10.0.1.217
  - ether src host de:ad:be:ef:ca:fe



It's Logic

- Combine primitives to create complex filters
  - use 'and' or '&&' to concatenate
  - use 'or' or '||' to alternate
  - use 'not' or '!' to negate
  - use parentices to group

- Examples:
  - host 10.0.0.1 and host 10.0.0.2
  - host 10.0.0.1 or host 10.0.0.2
  - host 10.0.0.1 and host 10.1.1.1 and udp port 53 or host 10.1.2.1 and tcp port 80
  - host 10.0.0.1 and ((host 10.1.1.1 and udp port 53) or (host 10.1.2.1 and tcp port 80))



Omit duplicate qualifiers

- To save typing, identical qualifier lists can be omitted.

- Examples:
  - tcp port 80 or tcp port 8080
    - tcp port 80 or 8080
  - host 10.0.0.1 and host 10.0.0.2
    - host 10.0.0.1 and 10.0.0.2
  - src host 10.0.0.1 and dst host 10.0.0.2
    - not identical qualifiers

Getting to specific bytes

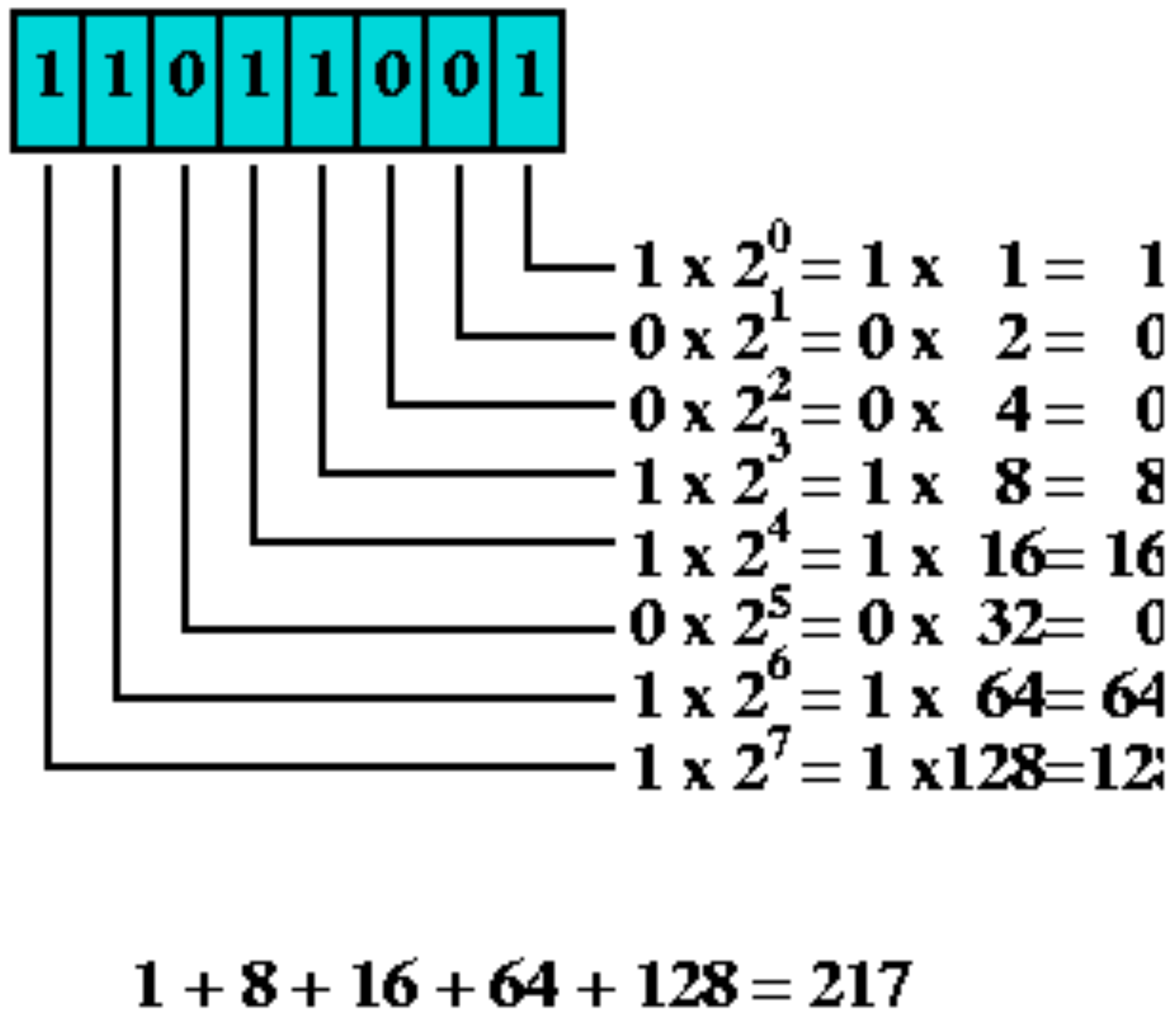
- Use …[x] to address certain byte
  - tcp[13] = 2
  - ip[8] = 64 or ip[8] = 128 or ip[8] = 255
  - ip[8] = 64 or 128 or 255 does not work!

- Can get two bytes with …[x:2] or four bytes with …[x:4]
  - ether[0:4] = 0x109add00 and ether[4:2] = 0xd796
  - ether[0:6] = 0x109add00d796 is not allowed

Two bytes meet. First byte asks, "Are you ill?" Second byte replies, "No, just feeling a bit off."

Getting to specific bits

- Use the logical and (&) to extract one or more bits
  - eth[0:4] & 0xffffff00 = 0x109add00
- Use the logical or (|) to combine bits
- Some fields have names for the offset and specific bits:
  - tcp[tcpflags] & (tcp-syn | tcp-ack) = tcp-syn
  - icmp[icmptype] = icmp-unreach



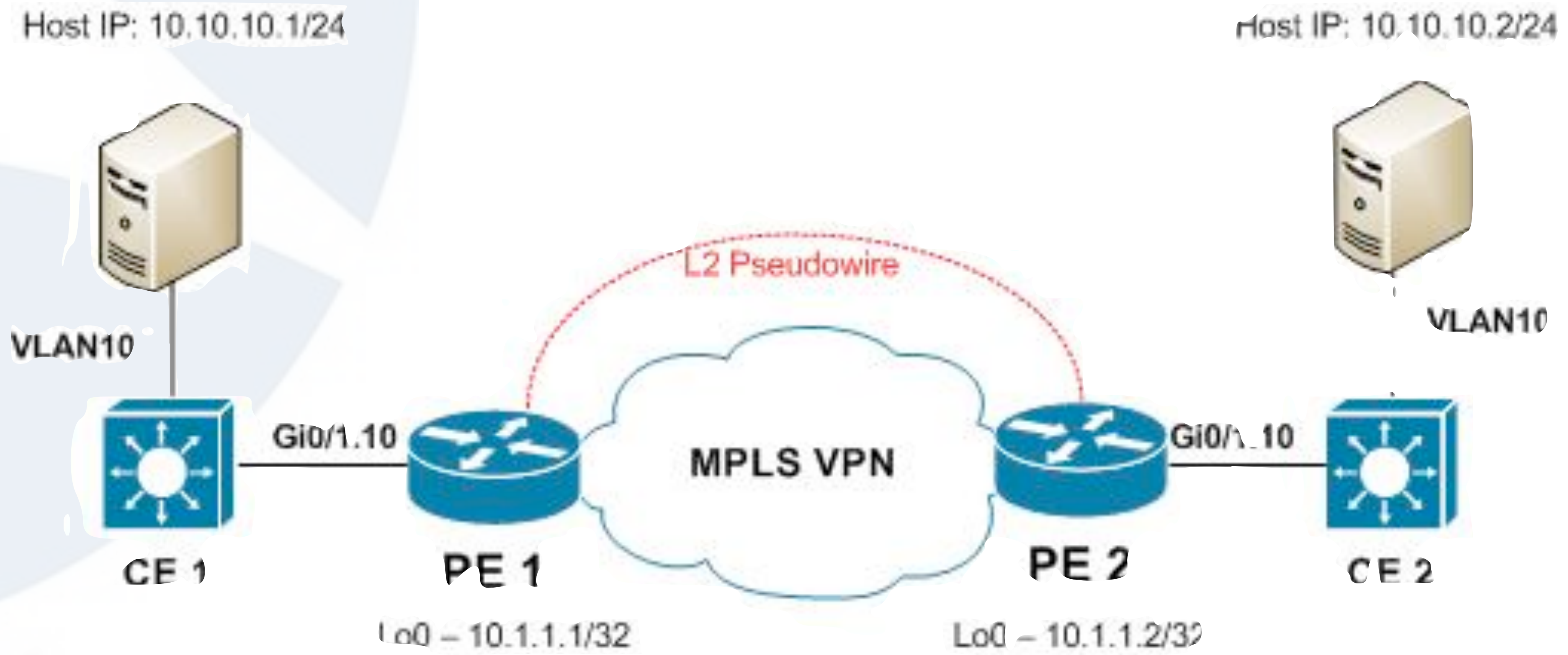
Arithmetics

- Calculations, both on values as on indices
  - Standard: +, -, * and /
  - Right shift: >>
  - Left shift: <<

- Examples:
  - tcp[((tcp[12:1] & 0xf0) >> 2):4] = 0x47455420
  - (ip[2:2] - ((ip[0]&0x0f)<<2) - ((tcp[12]&0xf0)>>2)) > 0

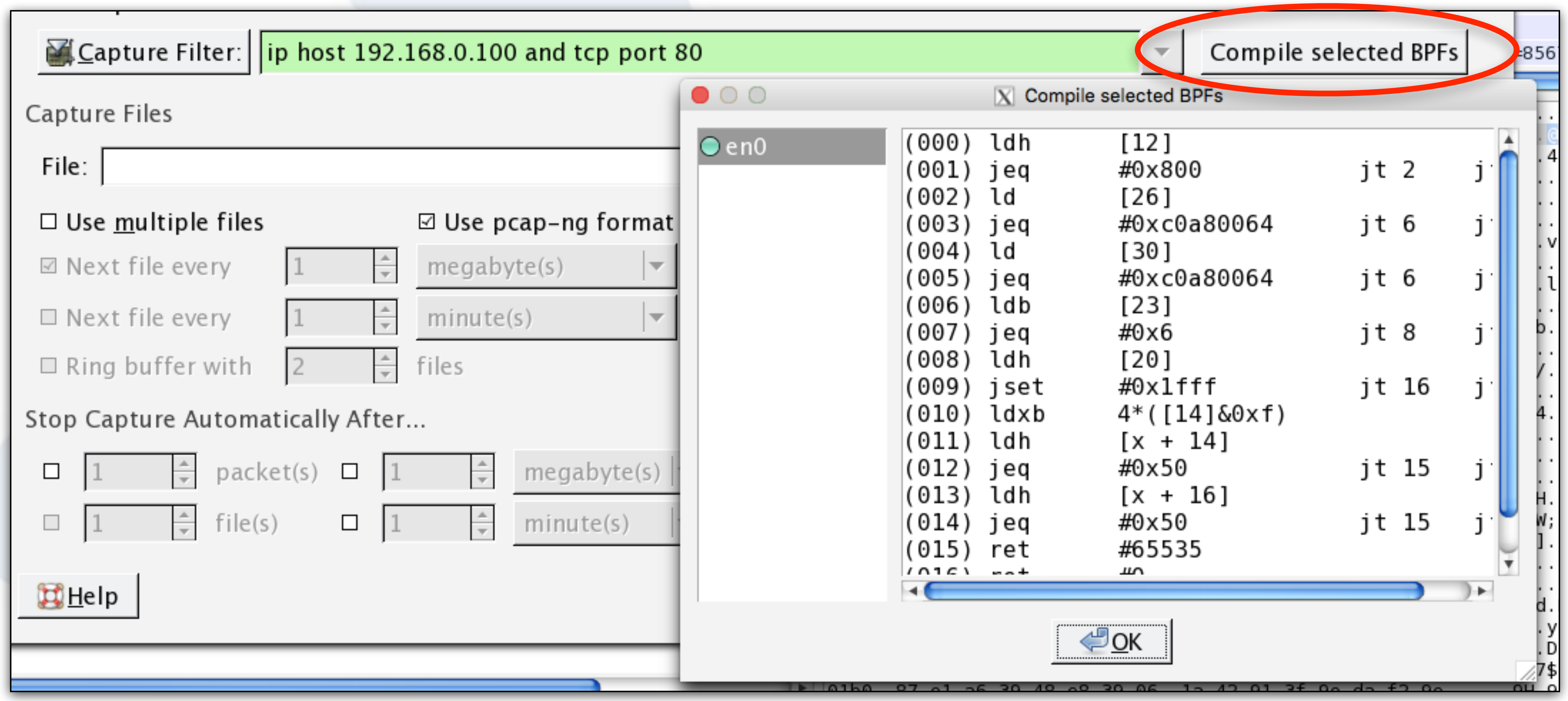
Offset alterations

- vlan keyword increases all offsets by 4
  - vlan 10 and host 10.0.0.1
  - vlan 10 or vlan 20 ==> unexpected result?
- mpls keyword increases all offsets by 4
- pppoes keyword increases all offsets by 8



Back to mnemonics

- Check whether a filter is correct
- Verify whether the filter will actually do what you intend
- Use wireshark's "Compile BPF filter"
- Use 'tcpdump -d' (or 'tcpdump -dd' if you're into bytecode ;-))



BPF machine code

```
$ tcpdump -d -s 128 "ip and host 1.1.1.1 and tcp port 80"
(000) ldh      [12]
(001) jeq      #0x800           jt 2 jf 16
(002) ld       [26]
(003) jeq      #0x1010101       jt 6 jf 4
(004) ld       [30]
(005) jeq      #0x1010101       jt 6 jf 16
(006) ldb      [23]
(007) jeq      #0x6             jt 8 jf 16
(008) ldh      [20]
(009) jset     #0x1fff          jt 16jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x50            jt 15jf 13
(013) ldh      [x + 16]
(014) jeq      #0x50            jt 15jf 16
(015) ret      #128
(016) ret      #0
$
```

Check for ip

```
0000   00 12 1e bb d1 3b f8 1e df d8 87 48 08 00 45 00   .....;.....H..E.
0010   02 bd 28 fb 40 00 40 06 fd 9f c0 a8 01 2b 01 01   ..(.@.@......+Bf
0020   01 01 c3 f6 00 50 f1 b8 8d 85 db 07 fd 9e 80 18   .g...P..........
0030   ff ff ce b2 00 00 01 01 08 0a 2e b9 c5 24 03 63   .............$.c
0040   c5 41 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31   .AGET / HTTP/1.1
0050   0d 0a 48 6f 73 74 3a 20 77 77 77 2e 67 6f 6f 67   ..Host: www.goog
0060   6c 65 2e 6e 6c 0d 0a 55 73 65 72 2d 41 67 65 6e   le.nl..User-Agen
0070   74 3a 20 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28   t: Mozilla/5.0 (
```

```
(000) ldh       [12]
(001) jeq       #0x800          jt 2 jf 16
...
(016) ret       #0
```

Check for source address

```
0000  00 12 1e bb d1 3b f8 1e df d8 87 48 08 00 45 00   .....;.....H..E.
0010  02 bd 28 fb 40 00 40 06 fd 9f c0 a8 01 2b 01 01   ..(.@.@......+Bf
0020  01 01 c3 f6 00 50 f1 b8 8d 85 db 07 fd 9e 80 18   .g...P..........
0030  ff ff ce b2 00 00 01 01 08 0a 2e b9 c5 24 03 63   .............$.c
0040  c5 41 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31   .AGET / HTTP/1.1
0050  0d 0a 48 6f 73 74 3a 20 77 77 77 2e 67 6f 6f 67   ..Host: www.goog
0060  6c 65 2e 6e 6c 0d 0a 55 73 65 72 2d 41 67 65 6e   le.nl..User-Agen
0070  74 3a 20 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28   t: Mozilla/5.0 (
```

```
(002) ld        [26]
(003) jeq       #0x1010101        jt 6 jf 4
```

Check for destination address

```
0000  00 12 1e bb d1 3b f8 1e df d8 87 48 08 00 45 00   .....;.....H..E.
0010  02 bd 28 fb 40 00 40 06 fd 9f c0 a8 01 2b 01 01   ..(.@.@......+Bf
0020  01 01 c3 f6 00 50 f1 b8 8d 85 db 07 fd 9e 80 18   .g...P..........
0030  ff ff ce b2 00 00 01 01 08 0a 2e b9 c5 24 03 63   .............$.c
0040  c5 41 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31   .AGET / HTTP/1.1
0050  0d 0a 48 6f 73 74 3a 20 77 77 77 2e 67 6f 6f 67   ..Host: www.goog
0060  6c 65 2e 6e 6c 0d 0a 55 73 65 72 2d 41 67 65 6e   le.nl..User-Agen
0070  74 3a 20 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28   t: Mozilla/5.0 (
```

```
(004) ld        [30]
(005) jeq       #0x1010101        jt 6 jf 16
...
(016) ret       #0
```

Check for tcp

```
0000  00 12 1e bb d1 3b f8 1e df d8 87 48 08 00 45 00   .....;.....H..E.
0010  02 bd 28 fb 40 00 40 06 fd 9f c0 a8 01 2b 01 01   ..(.@.@......+Bf
0020  01 01 c3 f6 00 50 f1 b8 8d 85 db 07 fd 9e 80 18   .g...P..........
0030  ff ff ce b2 00 00 01 01 08 0a 2e b9 c5 24 03 63   .............$.c
0040  c5 41 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31   .AGET / HTTP/1.1
0050  0d 0a 48 6f 73 74 3a 20 77 77 77 2e 67 6f 6f 67   ..Host: www.goog
0060  6c 65 2e 6e 6c 0d 0a 55 73 65 72 2d 41 67 65 6e   le.nl..User-Agen
0070  74 3a 20 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28   t: Mozilla/5.0 (
```

```
(006) ldb      [23]
(007) jeq      #0x6            jt 8 jf 16
...
(016) ret      #0
```

Check for ip fragmenting

```
0000  00 12 1e bb d1 3b f8 1e df d8 87 48 08 00 45 00   .....;.....H..E.
0010  02 bd 28 fb 40 00 40 06 fd 9f c0 a8 01 2b 01 01   ..(.@.@......+Bf
0020  01 01 c3 f6 00 50 f1 b8 8d 85 db 07 fd 9e 80 18   .g...P..........
0030  ff ff ce b2 00 00 01 01 08 0a 2e b9 c5 24 03 63   .............$.c
0040  c5 41 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31   .AGET / HTTP/1.1
0050  0d 0a 48 6f 73 74 3a 20 77 77 77 2e 67 6f 6f 67   ..Host: www.goog
0060  6c 65 2e 6e 6c 0d 0a 55 73 65 72 2d 41 67 65 6e   le.nl..User-Agen
0070  74 3a 20 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28   t: Mozilla/5.0 (
```

```
(008) ldh      [20]
(009) jset     #0x1fff          jt 16   jf 10
...
(016) ret      #0
```

Check for port number

```
0000   00 12 1e bb d1 3b f8 1e df d8 87 48 08 00 45 00    .....;.....H..E.
0010   02 bd 28 fb 40 00 40 06 fd 9f c0 a8 01 2b 01 01    ..(.@.@......+Bf
0020   01 01 c3 f6 00 50 f1 b8 8d 85 db 07 fd 9e 80 18    .g...P..........
0030   ff ff ce b2 00 00 01 01 08 0a 2e b9 c5 24 03 63    .............$.c
0040   c5 41 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31    .AGET / HTTP/1.1
0050   0d 0a 48 6f 73 74 3a 20 77 77 77 2e 67 6f 6f 67    ..Host: www.goog
0060   6c 65 2e 6e 6c 0d 0a 55 73 65 72 2d 41 67 65 6e    le.nl..User-Agen
0070   74 3a 20 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28    t: Mozilla/5.0 (
```

```
(010)  ldxb      4*([14]&0xf)
(011)  ldh       [x + 14]
(012)  jeq       #0x50              jt 15jf 13
(013)  ldh       [x + 16]
(014)  jeq       #0x50              jt 15jf 16
(015)  ret       #128
(016)  ret       #0
```

offsets in BPF

(ip and host 2.2.2.2) or
(vlan and ip and host 1.1.1.1)

```
(000) ldh      [12]
(001) jeq      #0x800        jt 2 jf 6
(002) ld       [26]
(003) jeq      #0x2020202    jt 13        jf 4
(004) ld       [30]
(005) jeq      #0x2020202    jt 13        jf 14
(006) jeq      #0x8100       jt 7 jf 14
(007) ldh      [16]
(008) jeq      #0x800        jt 9 jf 14
(009) ld       [30]
(010) jeq      #0x1010101    jt 13        jf 11
(011) ld       [34]
(012) jeq      #0x1010101    jt 13        jf 14
(013) ret      #96
(014) ret      #0
```

(vlan and ip and host 1.1.1.1) or
(ip and host 2.2.2.2)

```
(000) ldh      [12]
(001) jeq      #0x8100       jt 2 jf 8
(002) ldh      [16]
(003) jeq      #0x800        jt 4 jf 15
(004) ld       [30]
(005) jeq      #0x1010101    jt 14        jf 6
(006) ld       [34]
(007) jeq      #0x1010101    jt 14        jf 10
(008) ldh      [16]
(009) jeq      #0x800        jt 10        jf 15
(010) ld       [30]
(011) jeq      #0x2020202    jt 14        jf 12
(012) ld       [34]
(013) jeq      #0x2020202    jt 14        jf 15
(014) ret      #96
(015) ret      #0
```

offsets in BPF

```
vlan 10 or vlan 20
```

```
(000) ldh        [12]
(001) jeq        #0x8100          jt 2 jf 5
(002) ldh        [14]
(003) and        #0xfff
(004) jeq        #0xa             jt 10jf 5
(005) ldh        [16]
(006) jeq        #0x8100          jt 7 jf 11
(007) ldh        [18]
(008) and        #0xfff
(009) jeq        #0x14            jt 10jf 11
(010) ret        #96
(011) ret        #0
```

offsets in BPF

```
vlan and (ether[14:2]&0x0fff=10 or ether[14:2]&0x0fff=11)
```

```
(000) ldh        [12]
(001) jeq        #0x8100           jt 2 jf 7
(002) ldh        [14]
(003) and        #0xfff
(004) jeq        #0xa              jt 6 jf 5
(005) jeq        #0xb              jt 6 jf 7
(006) ret        #96
(007) ret        #0
```

TCP length > 0

```
0000   00 12 1e bb d1 3b f8 1e df d8 87 48 08 00 45 00   .....;.....H..E.
0010   00 40 62 92 40 00 40 06 c6 85 c0 a8 01 2b 42 66   .@b.@.@......+Bf
0020   0d 67 c3 f6 00 50 f1 b8 8d 84 00 00 00 00 b0 02   .g...P..........
0030   ff ff ed fc 00 00 02 04 05 b4 01 03 03 03 01 01   ................
0040   08 0a 2e b9 c5 24 00 00 00 00 04 02 00 00         .....$........
```

```
TCP length > 0
ip and tcp and
 (ip[2:2]
  - ((ip[0]&0x0f)<<2)
  - ((tcp[12:1]&0xf0)>>2)
 ) > 0
```

```
(000) ldh      [12]
(001) jeq      #0x800           jt 2  jf 22
(002) ldb      [23]
(003) jeq      #0x6             jt 4  jf 22
(004) ldh      [16]
(005) st       M[1]
(006) ldb      [14]
(007) and      #0xf
(008) lsh      #2
(009) tax
(010) ld       M[1]
(011) sub      x
(012) st       M[5]
(013) ldxb     4*([14]&0xf)
(014) ldb      [x + 26]
(015) and      #0xf0
(016) rsh      #2
(017) tax
(018) ld       M[5]
(019) sub      x
(020) jgt      #0x0             jt 21  jf 22
(021) ret      #262144
(022) ret      #0
```

Capture filter on steroids : SIP or SIP over IPIP

```
0000   c2 01 57 75 00 00 c2 00 57 75 00 00 08 00 45 00    ..Wu....Wu....E.
0010   00 78 00 14 00 00 ff 04 a7 6b 0a 00 00 01 0a 00    .x.......k......
0020   00 02 45 00 00 64 00 14 00 00 ff 11 b5 7f 01 01    ..E..d..........
0030   01 01 02 02 02 02 13 c4 13 c4 00 50 d3 85 52 45    ...........P..RE
...
```

```
port 5060 or
(ip proto 4 and
   (
     ip[20+9]=17 or ip[20+9]=6
   ) and (
     ip[20+20+0:2]=5060
     or
     ip[20+20+2:2]=5060
   )
)
```

```
port 5060 or
(ip proto 4 and
   (
     ip[((ip[0]&0x0f)<<2)+9]=17 or ip[((ip[0]&0x0f)<<2)+9]=6
   ) and (
     ip[((ip[0]&0x0f)<<2)+((ip[((ip[0]&0x0f)<<2)]&0x0f)<<2)+0:2]=5060
     or
     ip[((ip[0]&0x0f)<<2)+((ip[((ip[0]&0x0f)<<2)]&0x0f)<<2)+2:2]=5060
   )
)
```

Capture filter on steroids : SIP or SIP over IPIP

```
(000) ldh      [12]
(001) jeq      #0x800          jt 2    jf 58
(002) ldb      [23]
(003) jeq      #0x84           jt 6    jf 4
(004) jeq      #0x6            jt 6    jf 5
(005) jeq      #0x11           jt 6    jf 13
(006) ldh      [20]
(007) jset     #0x1fff         jt 58   jf 8
(008) ldxb     4*([14]&0xf)
(009) ldh      [x + 14]
(010) jeq      #0x13c4         jt 57   jf 11
(011) ldh      [x + 16]
(012) jeq      #0x13c4         jt 57   jf 58
(013) jeq      #0x4            jt 14   jf 58
(014) ldb      [14]
(015) and      #0xf
(016) lsh      #2
(017) add      #9
(018) tax
(019) ldb      [x + 14]
(020) jeq      #0x11           jt 22   jf 21
(021) jeq      #0x6            jt 22   jf 58
(022) ldb      [14]
(023) and      #0xf
(024) lsh      #2
(025) st       M[15]
(026) ldb      [14]
(027) and      #0xf
(028) lsh      #2
(029) tax
```

```
(030) ldb      [x + 14]
(031) and      #0xf
(032) lsh      #2
(033) tax
(034) ld       M[15]
(035) add      x
(036) tax
(037) ldh      [x + 14]
(038) jeq      #0x13c4         jt 57   jf 39
(039) ldb      [14]
(040) and      #0xf
(041) lsh      #2
(042) st       M[12]
(043) ldb      [14]
(044) and      #0xf
(045) lsh      #2
(046) tax
(047) ldb      [x + 14]
(048) and      #0xf
(049) lsh      #2
(050) tax
(051) ld       M[12]
(052) add      x
(053) add      #2
(054) tax
(055) ldh      [x + 14]
(056) jeq      #0x13c4         jt 57   jf 58
(057) ret      #262144
(058) ret      #0
```

Taking it one step further

- Write BPF program in mnemonics
- Compile to BPF bytecode
- Use as a filter in capturing
- Why?
  - optimize filters (as the parser is good, but not perfect)
  - do stuff that is not possible with the filter parser
    (like save up to layer 4 for udp and tcp independent of option length)
- Currently not possible to use bytecode as filter in wireshark/tshark
- Use netsniff-ng tools (linux only)

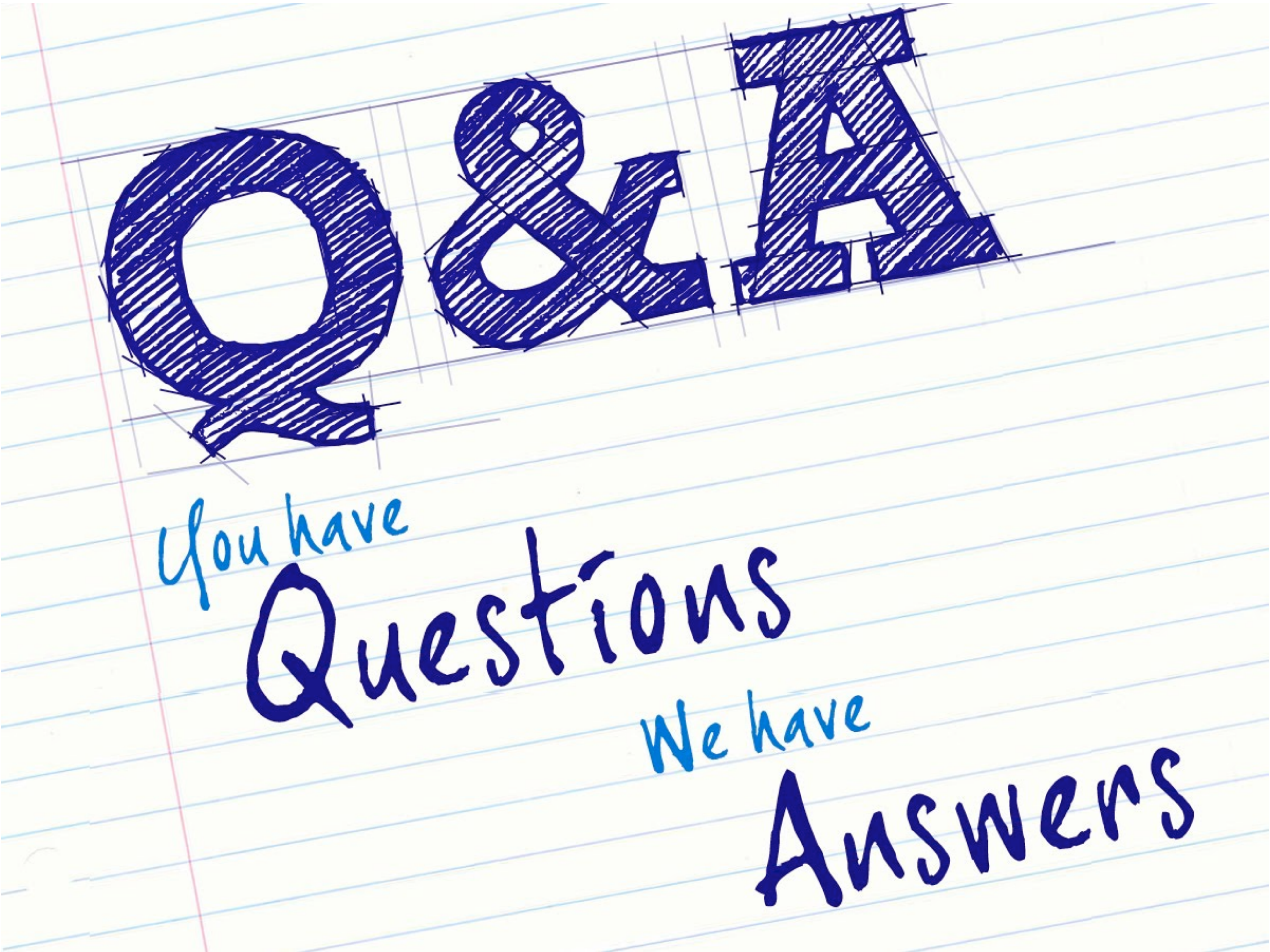
snap.bpf

```
        ; Check for ethertype 802.1Q or IP
        ld [12]
        jeq #0x8100,dot1q
        jne #0x0800,pass14
        ldb [23]
        st M[0]
        ldxb 4*([14]&0xf)
        txa
        add #14
        jmp l4

        ; Parse the 802.1Q header
dot1q:  ld [16]
        jne #0x0800,pass18
        ldb [27]
        st M[0]
        ldxb 4*([18]&0xf)
        txa
        add #18

        ; Check L4
l4:     ld M[0]
        jeq #6,tcp
        jeq #17,udp
        txa
        ret a
```

```
        ; UDP packet
udp:    txa
        add #8
        ret a

        ; TCP packet
tcp:    ldh [x + 0]
        jeq #80,pass
        ldh [x + 2]
        jeq #80,pass

        ; Non HTTP
        ldb [x + 12]
        and #0xf0
        rsh #2
        add x
        ret a

pass14: ret #14
pass18: ret #18
pass:   ret #65536
```

Summary

- BPF is a quick and powerfull filter engine in the kernel
- Bytecode is pushed to BPF from userspace programs
- Very flexibel filter language, learn the language!
- Can be used for post processing (much faster than display filters)
- Check compiled BPF program to verify working
- Write BPF program yourself for even more flexability
- Can filter on anything (as long as you can calculate the offset of the data that you want to filter on)
- No 'searching' for data (as no loops are allowed)


Q&A
You have
Questions
We have
Answers

FIN/ACK, ACK, FIN/ACK, ACK

Thank You!

sake.blok@SYN-bit.nl

SYN-bit

deep traffic analysis