libpcap: An Architecture and Optimization Methodology for Packet Capture

Sharkfest ’11

Steve McCanne, CTO
Riverbed Technology
My story begins with a U.C. Berkeley course

- Back in spring 1988, I took the compilers course in computer science at U.C. Berkeley
- Taught by a guest lecturer from LBL
  - Van Jacobson
- Learned standard compiler topics
  - scanning, parsing, code generation, optimization
- Took summer job in Van’s group at end of term
LBL

• It was a great time and place in Internet history
  – Summer job evolved into staff scientist position
• “Network Research Group”
  – Van Jacobson
  – Sally Floyd
  – Vern Paxson
  – Steve McCanne
• Lucky to be surrounded by such creative intellect
# LBL Network Research Group

- flex
- TCP congestion control
- VJ header compression (CSLIP)
- BSD packet filter (BPF)
- tcpdump, pcap
- traceroute, pathchar
- BRO
- SDP/SIP

- VoIP (RTP)
- Mbone tools (vic, vat, wb)
- Scalable reliable multicast (SRM)
- ns - network simulator
- Class-based queuing (CBQ)
- Random early drop (RED)
- diffserv
Congestion Control

• When I first joined, Van was wrapping up his work on TCP congestion control
  – He had figured out why the Arpanet kept collapsing...
Packet Capture

• Van needed to look at packet traces
  – to understand the problem
  – to experiment with fixes
  – to see that the solution was working
etherfind

• Frustrated with Sun’s packet capture tool
  – “etherfind” based on Unix “find” command
• Several problems
  – Clumsy filtering syntax
    • How many of you do “find . | grep ...” instead?
  – Protocol decoding was weak and cryptic
  – Horrible performance
The LAN Bottleneck
The LAN Bottleneck
Enter tcpdump

• There must be a better way...
  – Set out to work on a new model in a tool called tcpdump
  – “Filter” packets before they come up the stack
    • Inspired by Jeff Mogul’s prior work on “enet”
  – Compile high-level filter specification into low-level code that filters packets at driver level
  – Kernel module called Berkeley Packet Filter (BPF)
tcpdump

sun workstation

tcp stack

bpf

NIC

file system

disk

ethernet
tcpdump

“ip host ftp.arpa.net and tcp port ftp-data”

sun workstation

tcpdump

tcp stack

bpf

NIC

file system

ethernet

disk

kernel

user
tcpdump

“ip host ftp.arpa.net and tcp port ftp-data”
tcpdump

- Run test program to ftp.arpa.net
- ftp
- tcpdump
- tcp stack
- bpf
- NIC
- file system

sun workstation

packets to/from arpanet
disk

ethernet
tcpdump

Run test program to ftp.arpa.net

sun workstation

ftp
tcpdump
tcp stack
bpf
NIC

user

kernel

file system

arpanet packets copied to tcpdump

LAN traffic blocked by BPF

disk

packets to/from arpanet

ethernet
tcpdump

Run test program to ftp.arpa.net

sun workstation

tcpdump

Packets decoded and displayed

ftp tcpdump xterm

tcp stack

bpf

LAN traffic blocked by BPF

NIC

file system

arpanet packets copied to tcpdump

packets to/from arpanet

disk

file system

packets to/from arpanet

LAN traffic blocked by BPF

tcp stack

bpf

NIC

fiber system
tcpdump

Run test program to ftp.arpa.net

LAN traffic blocked by BPF
The BPF virtual machine

• First thing, I had to design a VM model that would run in the kernel
• Came up with a virtual machine architecture and set of machine instructions
  – Knew Apple II from my junior high days
  – Modeled after Motorola 6502
  – Accumulator (A), index register (X)
  – Packet-based memory model
  – Arithmetic and conditional logic
Example

FTP data packets for host 10.0.0.20

(000) ldh [12]
(001) jeq #0x800 jt 2 jf 16
(002) ld [26]
(003) jeq #10.0.0.20 jt 6 jf 4
(004) ld [30]
(005) jeq #10.0.0.20 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 #0x14 jt 15 jf 13
(013) ldh [x + 16]
(014) jeq #0x14 jt 15 jf 16
(015) ret #65535
(016) ret #0
Example

FTP data packets for host 10.0.0.20

Is ethernet type IP?

Is ethernet type IP?

A: [ ] X: [ ]

packet

ether   IP   TCP   data

(000) ldh [12]
(001) jeq #0x800   jt 2  jf 16
(002) ld [26]
(003) jeq #10.0.0.20   jt 6   jf 4
(004) ld [30]
(005) jeq #10.0.0.20   jt 6   jf 16
(006) ldb [23]
(007) jeq #0x6   jt 8   jf 16
(008) ldh [20]
(009) jset #0x1ff   jt 16   jf 10
(010) ldxb 4*([14]&0xf)
(011) ldh [x + 14]
(012) jeq #0x14   jt 15   jf 13
(013) ldh [x + 16]
(014) jeq #0x14   jt 15   jf 16
(015) ret #65535
(016) ret #0
Example

FTP data packets for host 10.0.0.20

Is ethernet type IP?

A: 0x800

X: 

packet

<table>
<thead>
<tr>
<th>ether</th>
<th>IP</th>
<th>TCP</th>
<th>data</th>
</tr>
</thead>
</table>

(000) ldh    [12]  
(001) jeq    #0x800  jt 2  jf 16  
(002) ld      [26]  
(003) jeq    #10.0.0.20  jt 6  jf 4  
(004) ld      [30]  
(005) jeq    #10.0.0.20  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    #0x14  jt 15  jf 13  
(013) ldh     [x + 16]  
(014) jeq    #0x14  jt 15  jf 16  
(015) ret    #65535  
(016) ret    #0
Example

FTP data packets for host 10.0.0.20

Is IP src address 10.0.0.20?

A:  
X:  

packet

debug level

ether  IP  TCP  data

(000) ldh  [12]
(001) jeq  #0x800  jt 2  jf 16
(002) ld  [26]
(003) jeq  #10.0.0.20  jt 6  jf 4
(004) ld  [30]
(005) jeq  #10.0.0.20  jt 6  jf 16
(006) ldb  [23]
(007) jeq  #0x6  jt 8  jf 16
(008) ldh  [20]
(009) jset  #0x1ff  jt 16  jf 10
(010) ldxb  4*[14]&0xf
(011) ldh  [x + 14]
(012) jeq  #0x14  jt 15  jf 13
(013) ldh  [x + 16]
(014) jeq  #0x14  jt 15  jf 16
(015) ret  #65535
(016) ret  #0
Example

FTP data packets for host 10.0.0.20

Is IP src address 10.0.0.20?

A: 10.0.1.11  X: 

packet

ether  IP  TCP  data
Example

FTP data packets for host 10.0.0.20

Is IP dst address 10.0.0.20?

A: [ ] X: [ ]

packet

ether  IP  TCP  data

(000) ldh [12]
(001) jeq #0x800  jt 2  jf 16
(002) ld [26]
(003) jeq #10.0.0.20  jt 6  jf 4
(004) ld [30]
(005) jeq #10.0.0.20  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 #0x14  jt 15  jf 13
(013) ldh [x + 16]
(014) jeq #0x14  jt 15  jf 16
(015) ret #65535
(016) ret #0
FTP data packets for host 10.0.0.20

Is IP dst address 10.0.0.20?

A: 10.0.0.20  X:  

Packet

ether | IP | TCP | data

Example:

(000) ldh [12]
(001) jeq #0x800 jr 2 jf 16
(002) ld [26]
(003) jeq #10.0.0.20 jr 6 jf 4
(004) ld [30]
(005) jeq #10.0.0.20 jr 6 jf 16
(006) ldb [23]
(007) jeq #0x6 jr 8 jf 16
(008) ldh [20]
(009) jset #0x1ff jr 16 jf 10
(010) ldxb 4*([14]&0xf)
(011) ldh [x + 14]
(012) jeq #0x14 jr 15 jf 13
(013) ldh [x + 16]
(014) jeq #0x14 jr 15 jf 16
(015) ret #65535
(016) ret #0
Example

FTP data packets for host 10.0.0.20

Is IP protocol TCP?

A: [ ] X: [ ]
Example

FTP data packets for host 10.0.0.20

Is IP protocol TCP?

A: 6  X: 

```
(000) ldh [12]
(001) jeq #0x800  jt 2  jf 16
(002) ld [26]
(003) jeq #10.0.0.20  jt 6  jf 4
(004) ld [30]
(005) jeq #10.0.0.20  jt 6  jf 16
(006) ldb [23]
(007) jeq #0x6  jt 8  jf 16
(008) ldh [20]
(009) jset #0x1ff  jt 16  jf 10
(010) ldxb 4*[14]&0xf)
(011) ldh [x + 14]
(012) jeq #0x14  jt 15  jf 13
(013) ldh [x + 16]
(014) jeq #0x14  jt 15  jf 16
(015) ret #65535
(016) ret #0
```
Example

FTP data packets for host 10.0.0.20

Is it first or only frag?

A:  
X:  

packet

table:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ether</td>
<td>IP</td>
<td>TCP</td>
<td>data</td>
</tr>
</tbody>
</table>

Code:

(000) ldh [12]
(001) jeq #0x800 jt 2 jf 16
(002) ld [26]
(003) jeq #10.0.0.20 jt 6 jf 4
(004) ld [30]
(005) jeq #10.0.0.20 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 #0x14 jt 15 jf 13
(013) ldh [x + 16]
(014) jeq #0x14 jt 15 jf 16
(015) ret #65535
(016) ret #0
FTP data packets for host 10.0.0.20

Is it first or only frag?

A: 0  X: 

packet

(000) ldh [12]
(001) jeq #0x800 jt 2 jf 16
(002) ld [26]
(003) jeq #10.0.0.20 jt 6 jf 4
(004) ld [30]
(005) jeq #10.0.0.20 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 #0x14 jt 15 jf 13
(013) ldh [x + 16]
(014) jeq #0x14 jt 15 jf 16
(015) ret #65535
(016) ret #0
Example

FTP data packets for host 10.0.0.20

Is TCP src port FTP?

A:  X:  

packet

<table>
<thead>
<tr>
<th>ether</th>
<th>IP</th>
<th>TCP</th>
<th>data</th>
</tr>
</thead>
</table>

(000) ldh [12]
(001) jeq #0x800 jt 2 jf 16
(002) ld [26]
(003) jeq #10.0.0.20 jt 6 jf 4
(004) ld [30]
(005) jeq #10.0.0.20 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 #0x14 jt 15 jf 13
(013) ldh [x + 16]
(014) jeq #0x14 jt 15 jf 16
(015) ret #65535
(016) ret #0
Example

FTP data packets for host 10.0.0.20

Is TCP src port FTP?

A: [ ]

X: 20

packet
FTP data packets for host 10.0.0.20

Is TCP src port FTP?

A: 8377  
X: 20

Example

```
(000) ldh [12]
(001) jeq #0x800      jt 2     jf 16
(002) ld [26]
(003) jeq #10.0.0.20   jt 6     jf 4
(004) ld [30]
(005) jeq #10.0.0.20   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 #0x14        jt 15    jf 13
(013) ldh [x + 16]
(014) jeq #0x14        jt 15    jf 16
(015) ret #65535
(016) ret #0
```
Example

FTP data packets for host 10.0.0.20

Is TCP dest port FTP?

<table>
<thead>
<tr>
<th>ether</th>
<th>IP</th>
<th>TCP</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A: 8377  X: 20

packet

(000) ldh [12]
(001) jeq #0x800  jt 2  jf 16
(002) ld [26]
(003) jeq #10.0.0.20  jt 6  jf 4
(004) ld [30]
(005) jeq #10.0.0.20  jt 6  jf 16
(006) ldb [23]
(007) jeq #0x6  jt 8  jf 16
(008) ldh [20]
(009) jset #0x1ff  jt 16  jf 10
(010) ldxb 4*([14]&0xf)
(011) ldh [x + 14]
(012) jeq #0x14  jt 15  jf 13
(013) ldh [x + 16]
(014) jeq #0x14  jt 15  jf 16
(015) ret #65535
(016) ret #0
Example

FTP data packets for host 10.0.0.20

Is TCP dest port FTP?

A: 0x14
X: 20

packet

ether  IP  TCP  data

(000) ldh [12]
(001) jeq #0x800  jt 2  jf 16
(002) ld [26]
(003) jeq #10.0.0.20  jt 6  jf 4
(004) ld [30]
(005) jeq #10.0.0.20  jt 6  jf 16
(006) ldb [23]
(007) jeq #0x6  jt 8  jf 16
(008) ldh [20]
(009) jset #0x1ff  jt 16  jf 10
(010) ldxb 4*([14]&0xf)
(011) ldh [x + 14]
(012) jeq #0x14  jt 15  jf 13
(013) ldh [x + 16]
(014) jeq #0x14  jt 15  jf 16
(015) ret #65535
(016) ret #0
### Example

FTP data packets for host 10.0.0.20

```plaintext
return TRUE
```

<table>
<thead>
<tr>
<th>ether</th>
<th>IP</th>
<th>TCP</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: 0x14</td>
<td>X: 20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
(000) ldh [20]
(001) jeq #0x800  jt 2  jf 16
(002) ld [26]
(003) jeq #10.0.0.20  jt 6  jf 4
(004) ld [30]
(005) jeq #10.0.0.20  jt 6  jf 16
(006) ldb [23]
(007) jeq #0x6  jt 8  jf 16
(008) ldh [20]
(009) jset #0x1ff  jt 16  jf 10
(010) ldxb 4*[14]&0xf
(011) ldh [x + 14]
(012) jeq #0x14  jt 15  jf 13
(013) ldh [x + 16]
(014) jeq #0x14  jt 15  jf 16
(015) ret #65535
(016) ret #0
```
The Challenge

• The BPF virtual machine model is a very flexible and efficient model for packet filtering
• But, you would never want to write low-level BPF programs every time you wanted to filter packets
• So, we needed a higher level model...
A Filter Language

Instead of writing this...

```plaintext
(000) ldh [12]
(001) jeq #0x800      jt 2    jf 16
(002) ld [26]
(003) jeq #0x100.0.20  jt 6    jf 4
(004) ld [30]
(005) jeq #0x100.0.20  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 #0x14          jt 15   jf 13
(013) ldh [x + 16]
(014) jeq #0x14          jt 15   jf 16
(015) ret #65535
(016) ret #0
```
A Filter Language

ip host ftp.arpa.net and tcp port ftp-data

Just say this...

```
(000) ldh [12]
(001) jeq #0x800       jt 2   jf 16
(002) ld [26]
(003) jeq #10.0.0.20   jt 6   jf 4
(004) ld [30]
(005) jeq #10.0.0.20   jt 6   jf 16
(006) ldb [23]
(007) jeq #0x6         jt 8   jf 16
(008) ldh [20]
(009) jset #0x1ff      jt 16  jf 10
(010) ldxb 4*([14]&0xf)
(011) ldh [x + 14]
(012) jeq #0x14        jt 15  jf 13
(013) ldh [x + 16]
(014) jeq #0x14        jt 15  jf 16
(015) ret #65535
(016) ret #0
```
A Filter Language

ip host ftp.arpa.net and tcp port ftp-data

And let a compiler translate...

(000) ldh [12]
(001) jeq #0x800 jt 2 jf 16
(002) ld [26]
(003) jeq #10.0.0.20 jt 6 jf 4
(004) ld [30]
(005) jeq #10.0.0.20 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) ldx [4*(14)&0xf]
(011) ldh [x + 14]
(012) jeq #0x14 jt 15 jf 13
(013) ldh [x + 16]
(014) jeq #0x14 jt 15 jf 16
(015) ret #65535
(016) ret #0
The Challenge

• This is where things got a bit tricky
• Designing the language and parser so it was easy on users turned out to be hard
• Learned an important life lesson from Van
  – It’s easy to make things hard
  – It’s hard to make things easy
  – It’s usually better to do the latter
BPF Language

• The BPF filter language starts from a basic predicate, which is true iff the specified packet field equals the indicated value

   pred: field val

   field: protocol dir selector
BPF Language

• The BPF filter language starts from a basic predicate, which is true iff the specified packet field equals the indicated value

\[
\text{pred: field val}
\]

\[
\text{field: protocol dir selector}
\]

\[
\begin{align*}
\text{ether} & \quad \text{ip} & \quad \text{tcp} & \quad \text{icmp} & \quad \text{udp} & \quad <\text{none}> \\
\text{src} & \quad \text{dst} & \quad \text{host} & \quad \text{net} & \quad \text{port} & \quad <\text{none}>
\end{align*}
\]
BPF Language

- The BPF filter language starts from a basic predicate, which is true iff the specified packet field equals the indicated value.

```
pred: field val
```

```
field: protocol dir selector
```

```
ether
  ip
  tcp
  icmp
  udp
  <none>
```

```
src
dst
<none>
```

```
host
net
port
```

```
examples
```

```
<table>
<thead>
<tr>
<th>ip</th>
<th>src</th>
<th>host</th>
<th>10.0.0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp</td>
<td>dst</td>
<td>port</td>
<td>80</td>
</tr>
</tbody>
</table>
```

BPF Logic

- Language includes logic to stitch together predicates into complex logic operations
  - `pred or pred`
  - `pred and pred`
  - `not pred`
  - `'(' pred ')'`

ip src host X and not port 80
My First Attempt

expr: pred
  | expr AND pred
  | expr OR pred
  | NOT expr
  | ‘(‘ expr‘)’

pred: field val

field: protocol dir selector

etc...
A Problem

• But Van didn’t like it... too clunky...

ip src host X or ip src host Y or ip src host Z
A Problem

• But Van didn’t like it... too clunky...

ip src host X or ip src host Y or ip src host Z
A Problem

• But Van didn’t like it... too clunky...
  
ip src host X or ip src host Y or ip src host Z

• Why not just say...
  
ip src host X or Y or Z

• This should be easy enough to fix...
My Second Attempt

• Introduce two layers of logic
  – Lower layer would handle predicates with multiple values
    • ip host x or y
    • tcp port 80 or 1024
  – Upper layer would handle the combinations of the lower-layer expressions
    • (ip host x or y) and (tcp port 80 or 1024)
My Second Attempt

expr: term
   | expr AND term
   | expr OR term
   | NOT expr
   | ‘(‘ expr‘)’

term: pred
   | term AND val
   | term OR val
   | NOT term
   | ‘(‘ term ‘)’

pred: field val

field: protocol dir selector

etc...
The Second Problem

• But this didn’t work at all
  – the parser needs to decide to parse as a *term*
    \[ \text{ip src host x or y and z} \]
  – or parse input as an *expr*
    \[ \text{ip src host x or y and tcp port z} \]
  – when the partial input didn’t provide enough info
    \[ \text{ip src host x or y and unknown} \]
    \[
    \text{parsed input} \quad \text{look-ahead}
    \]
It’s easy to make things hard

• Some easy ways out...
  – require parens or another grouping symbol
    • ip host ( x or y ) and tcp port z
    • ip host { x or y } and tcp port z
  – have different families of logic symbols
    • e.g., “and”, “AND”, “or”, “OR”
    • ip host x or y AND tcp port z
  – introduce terminator symbol
    • ip host x or y . and tcp port z
It’s hard to make things easy

• But all those solutions made things harder on the user, even though they were easy outs
  – So, Van challenged me
  – “There must be a way. Figure it out.”

• I spent a week or two frustratingly thinking about it and finally the light bulb came on
  – Turns out this was a novel language construct
The Solution

• Have a single level of logic, not two
• Allow predicates or values to be tacked onto an expression
• i.e., an \textit{expr} can be both
  – \textit{expr} AND \textit{pred}
  – \textit{expr} AND \textit{val}
Third Time’s a Charm

expr: pred
   | expr AND pred
   | expr AND val
   | expr OR pred
   | expr OR val
   | NOT expr
   | ‘(‘ expr‘)’

pred: field val
Not so fast...

• Ok, this grammar worked out fine, but now code generation became tricky
• Fortunately, this problem while tricky, had a solution...
My Third Attempt

expr: pred
  | expr AND pred
  | expr AND val
  | expr OR pred
  | expr OR val
  | NOT expr
  | ('(expr)')

pred: field val

{ $$ = gen_cmp($1, $2); }
My Third Attempt

expr: pred
  | expr AND pred { $$ = gen_and($1,$3); }
  | expr AND val { ??? }
  | expr OR pred { $$ = gen_or($1, $3); }
  | expr OR val { ??? }
  | NOT expr { $$ = gen_not($2); }
  | ‘(‘ expr‘)’ { $$ = $2; }

pred: field val { $$ = gen_cmp($1, $2); }
The Solution

LA:

| sym | fla | code |

expr: pred
| expr AND pred |
| expr AND val |
| expr OR pred |
| expr OR val |
| NOT expr |
| ‘( expr )’ |

pred: field val
My Third Attempt

\[
\begin{align*}
expr: & \text{ pred} \\
| \text{ expr AND pred} \\
| \text{ expr AND val} \\
| \text{ expr OR pred} \\
| \text{ expr OR val} \\
| \text{ NOT expr} \\
| \text{ ‘(' expr‘)}
\end{align*}
\]

\[
\begin{align*}
pred: & \text{ field val} \\
\text{ { } t = \text{ gen\_cmp($1.fld, $3);} } \\
\text{ { } $$\$.code = \text{ gen\_and($1, t)} } \\
\text{ { } $$\$.fld = $1.fld; } \\
\text{ { } $$$.code = \text{ gen\_cmp($1, $2); } \\
\text{ { } $$\$.fld = $1; } \\
\end{align*}
\]
My Third Attempt

expr: pred
  | expr AND pred
  | expr AND val
  | expr OR pred
  | expr OR val
  | NOT expr
  | ‘(‘ expr‘)’

pred: field val

{ $$.code = gen_cmp($1, $2); $$.fld = $1; }

{ t = gen_eq($1.fld, $3); $$.code = gen_and($1, t) $$.fld = $1.fld; }
My Third Attempt

expr: pred
  | expr AND pred
  | expr AND val
  | expr OR pred
  | expr OR val
  | NOT expr
  | ‘(‘ expr‘)’

pred: field val

{ $$ = \text{gen\_vand($1, $3)}; }$$
{ $$ = \text{gen\_cmp($1, $2)}; }$$
My Third Attempt

expr: pred

<table>
<thead>
<tr>
<th>expr AND pred</th>
<th>{$$ = gen_and($1,$3); }</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr AND val</td>
<td>{$$ = gen_vand($1, $3); }</td>
</tr>
<tr>
<td>expr OR pred</td>
<td>{$$ = gen_or($1, $3); }</td>
</tr>
<tr>
<td>expr OR val</td>
<td>{$$ = gen_vor($1, $3); }</td>
</tr>
<tr>
<td>NOT expr</td>
<td>{$$ = gen_not($2); }</td>
</tr>
<tr>
<td>‘(‘ expr’)’</td>
<td>{$$ = $2; }</td>
</tr>
</tbody>
</table>

pred: field val

{ $$ = gen_cmp($1, $2); }
Example

LA: IP

<table>
<thead>
<tr>
<th>sym</th>
<th>fld</th>
<th>code</th>
</tr>
</thead>
</table>

ip src host x or y and tcp dst port z

expr: pred

| expr AND pred | { $$ = gen_and($1,$3); } |
| expr AND val  | { $$ = gen_vand($1, $3); } |
| expr OR pred  | { $$ = gen_or($1, $3); } |
| expr OR val   | { $$ = gen_vor($1, $3); } |

pred: field val

field: proto dir selector

pred: field val

| pred: field val | { $$ = gen_cmp($1, $2); } |

field: proto dir selector
### Example

**LA: SRC**

```latex
src host x or y and tcp dst port z
```

<table>
<thead>
<tr>
<th>IP</th>
<th>sym</th>
<th>fld</th>
<th>code</th>
</tr>
</thead>
</table>

**expr: pred**

- `expr AND pred`  
  
  ```latex
  \{ \$$ = \text{gen\_and}($1, $3); \}
  ```

- `expr AND val`  
  
  ```latex
  \{ \$$ = \text{gen\_vand}($1, $3); \}
  ```

- `expr OR pred`  
  
  ```latex
  \{ \$$ = \text{gen\_or}($1, $3); \}
  ```

- `expr OR val`  
  
  ```latex
  \{ \$$ = \text{gen\_vor}($1, $3); \}
  ```

**pred: field val**  

```latex
\{ \$$ = \text{gen\_cmp}($1, $2); \}
```
Example

<table>
<thead>
<tr>
<th>SRC</th>
<th>IP</th>
</tr>
</thead>
</table>

expr: pred

<table>
<thead>
<tr>
<th>expr AND pred</th>
<th>expr AND val</th>
<th>expr OR pred</th>
<th>expr OR val</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ $$$ = gen_and($1,$3); }</td>
<td>{ $$$ = gen_vand($1, $3); }</td>
<td>{ $$$ = gen_or($1, $3); }</td>
<td>{ $$$ = gen_vor($1, $3); }</td>
</tr>
</tbody>
</table>

pred: field val

\{ $$$ = gen_cmp($1, $2); \}

field: proto dir selector
Example

<table>
<thead>
<tr>
<th>LA: X</th>
<th>x or y and tcp dst port z</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST</td>
<td></td>
</tr>
<tr>
<td>SRC</td>
<td></td>
</tr>
<tr>
<td>IP</td>
<td></td>
</tr>
</tbody>
</table>

**expr:** `pred`
- `expr AND pred`  `{ $\$ = gen_and($1,$3); }`
- `expr AND val`  `{ $\$ = gen_vand($1,$3); }`
- `expr OR pred`  `{ $\$ = gen_or($1,$3); }`
- `expr OR val`  `{ $\$ = gen_vor($1,$3); }

**pred:** `field val`
- `{ $\$ = gen_cmp($1,$2); }

**field:** `proto dir selector`

<table>
<thead>
<tr>
<th>sym</th>
<th>fld</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

LA: X

\[ x \text{ or } y \text{ and tcp dst port } z \]

<table>
<thead>
<tr>
<th>field</th>
<th>ISH</th>
</tr>
</thead>
</table>

expr: pred

- \( \text{expr AND pred} \)  \{ \$$ = \text{gen_and($1,$3);} \}
- \( \text{expr AND val} \)  \{ \$$ = \text{gen_vand($1,$3);} \}
- \( \text{expr OR pred} \)  \{ \$$ = \text{gen_or($1,$3);} \}
- \( \text{expr OR val} \)  \{ \$$ = \text{gen_vor($1,$3);} \}

pred: field val  \{ \$$ = \text{gen_cmp($1,$2);} \}

field: proto dir selector

sym     fld  code
**Example**

LA: X

---

### LA: X

**x or y and tcp dst port z**

### Table

<table>
<thead>
<tr>
<th>field</th>
<th>ISH</th>
</tr>
</thead>
<tbody>
<tr>
<td>sym</td>
<td>fld</td>
</tr>
</tbody>
</table>

### Expr: pred

- `expr AND pred`  
  ```
  $$ \text{gen_and($1,$3);} 
  $$
  ```

- `expr AND val`  
  ```
  $$ \text{gen_vand($1,$3);} 
  $$
  ```

- `expr OR pred`  
  ```
  $$ \text{gen_or($1,$3);} 
  $$
  ```

- `expr OR val`  
  ```
  $$ \text{gen_vor($1,$3);} 
  $$
  ```

### Pred: field val

- `field: proto dir selector`  
  ```
  $$ \text{gen_cmp($1,$2);} 
  $$
  ```
**Example**

```
<table>
<thead>
<tr>
<th>field</th>
<th>ISH</th>
</tr>
</thead>
<tbody>
<tr>
<td>val(x)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**expr:** `pred`

- `expr AND pred`  
  ```
  $$\text{\texttt{\$\$ = gen\_and($1, $3);}}$$
  ```
- `expr AND val`  
  ```
  $$\text{\texttt{\$\$ = gen\_vand($1, $3);}}$$
  ```
- `expr OR pred`  
  ```
  $$\text{\texttt{\$\$ = gen\_or($1, $3);}}$$
  ```
- `expr OR val`  
  ```
  $$\text{\texttt{\$\$ = gen\_vor($1, $3);}}$$
  ```

**pred:** `field val`

```
$$\text{\texttt{\$\$ = gen\_cmp($1, $2);}}$$
```

**field:** `proto dir selector`

```
<table>
<thead>
<tr>
<th>sym</th>
<th>fld</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LA: OR

or y and tcp dst port z
```
Example

<table>
<thead>
<tr>
<th>field</th>
<th>ISH</th>
</tr>
</thead>
<tbody>
<tr>
<td>val(x)</td>
<td></td>
</tr>
</tbody>
</table>

expr. pred

- `expr AND pred`  \[ \{ \$$ = \text{gen\_and}(\$$1,\$$3); \} \]
- `expr AND val`  \[ \{ \$$ = \text{gen\_vand}(\$$1, \$$3); \} \]
- `expr OR pred`  \[ \{ \$$ = \text{gen\_or}(\$$1, \$$3); \} \]
- `expr OR val`  \[ \{ \$$ = \text{gen\_vor}(\$$1, \$$3); \} \]

pred: `field val`  \[ \{ \$$ = \text{gen\_cmp}(\$$1, \$$2); \} \]

field: proto dir selector
**Example**

```
LA: OR

<table>
<thead>
<tr>
<th>pred</th>
<th>ISH</th>
<th>C1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sym</th>
<th>fld</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- **expr:** pred
  - expr AND pred:
    - expr AND val:
      - expr OR pred:
        - expr OR val:
          - pred: field val:
            - ```
              $$$ = gen_cmp($1, $2);
            ```

- **field:** proto dir selector
Example

```
expr: pred

| expr AND pred | {$$ = gen_and($1, $3); } |
| expr AND val  | {$$ = gen_vand($1, $3); } |
| expr OR pred  | {$$ = gen_or($1, $3); }   |
| expr OR val   | {$$ = gen_vor($1, $3); }   |

pred: field val  { $$ = gen_cmp($1, $2); } 

field: proto dir selector

LA: OR

or y and tcp dst port z
```

```
<table>
<thead>
<tr>
<th>pred</th>
<th>ISH</th>
<th>C1</th>
</tr>
</thead>
<tbody>
<tr>
<td>sym</td>
<td>fld</td>
<td>code</td>
</tr>
</tbody>
</table>
```
### Example

**LA:** OR

<table>
<thead>
<tr>
<th>expr</th>
<th>ISH</th>
<th>C1</th>
</tr>
</thead>
</table>

**expr:** pred

- `expr AND pred`  
  { $\$ = gen_and($1, $3); }
- `expr AND val`  
  { $\$ = gen_vand($1, $3); }
- `expr OR pred`  
  { $\$ = gen_or($1, $3); }
- `expr OR val`  
  { $\$ = gen_vor($1, $3); }

**pred:** field val  
{ $\$ = gen_cmp($1, $2); }

**field:** proto dir selector

<table>
<thead>
<tr>
<th>sym</th>
<th>fld</th>
<th>code</th>
</tr>
</thead>
</table>
**Example**

```
LA: Y

expr | ISH | C1
-----|-----|----
expr |     |    
expr |     |    
expr |     |    
expr |     |    
expr |     |    
expr |     |    

**expr: pred**

| expr AND pred | $$ = gen_and($1,$3); |
| expr AND val  | $$ = gen_vand($1, $3); |
| expr OR pred  | $$ = gen_or($1, $3); |
| expr OR val   | $$ = gen_vor($1, $3); |

**pred: field val**

<table>
<thead>
<tr>
<th>field: proto dir selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>field: proto dir selector</td>
</tr>
<tr>
<td>field: proto dir selector</td>
</tr>
<tr>
<td>field: proto dir selector</td>
</tr>
</tbody>
</table>

**field: proto dir selector**
```
Example

<table>
<thead>
<tr>
<th>LA: AND</th>
<th>and tcp dst port z</th>
</tr>
</thead>
<tbody>
<tr>
<td>val(y)</td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>expr</td>
<td>ISH</td>
</tr>
<tr>
<td></td>
<td>C1</td>
</tr>
<tr>
<td>sym</td>
<td>fld</td>
</tr>
<tr>
<td>code</td>
<td></td>
</tr>
</tbody>
</table>

**expr: pred**

- `expr AND pred`:
  ```
  $$ = gen_and($1,$3);
  $$
  ```
- `expr AND val`:
  ```
  $$ = gen_vand($1, $3);
  $$
  ```
- `expr OR pred`:
  ```
  $$ = gen_or($1, $3);
  $$
  ```
- `expr OR val`:
  ```
  $$ = gen_vor($1, $3);
  $$
  ```

**pred: field val**

```
$$ = gen_cmp($1, $2);
$$
```
Example

<table>
<thead>
<tr>
<th>val(y)</th>
<th>OR</th>
<th>expr</th>
<th>ISH</th>
<th>C1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

expr: pred

<table>
<thead>
<tr>
<th>expr AND pred</th>
<th>expr AND val</th>
<th>expr OR pred</th>
<th>expr OR val</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ $$ = gen_and($1, $3); }</td>
<td>{ $$ = gen_vand($1, $3); }</td>
<td>{ $$ = gen_or($1, $3); }</td>
<td>{ $$ = gen_vor($1, $3); }</td>
</tr>
</tbody>
</table>

pred: field val

{ $$ = gen_cmp($1, $2); }

field: proto dir selector
Example

and tcp dst port z

expr: pred
  | expr AND pred   { $$ = gen_and($1,$3); }
  | expr AND val    { $$ = gen_vand($1,$3); }
  | expr OR pred    { $$ = gen_or($1,$3); }
  | expr OR val     { $$ = gen_vor($1,$3); }

pred: field val     { $$ = gen_cmp($1,$2); }
field: proto dir selector
## Example

### LA: AND

\[ \text{and tcp dst port z} \]

<table>
<thead>
<tr>
<th>expr</th>
<th>ISH</th>
<th>C2</th>
</tr>
</thead>
</table>

**expr: pred**

- \( \text{expr AND pred} \)
  \{ \text{\$\$ = gen\_and($1, $3); } \}
- \( \text{expr AND val} \)
  \{ \text{\$\$ = gen\_vand($1, $3); } \}
- \( \text{expr OR pred} \)
  \{ \text{\$\$ = gen\_or($1, $3); } \}
- \( \text{expr OR val} \)
  \{ \text{\$\$ = gen\_vor($1, $3); } \}

**pred: field val**

\{ \text{\$\$ = gen\_cmp($1, $2); } \}

**field:**

- proto
- dir
- selector
Example

<table>
<thead>
<tr>
<th>LA: TCP</th>
<th>tcp dst port z</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>AND</th>
<th>expr</th>
<th>ISH</th>
<th>C2</th>
</tr>
</thead>
</table>

expr: pred
- expr AND pred
  - $$ = gen_and($1,$3);$$
- expr AND val
  - $$ = gen_vand($1, $3);$$
- expr OR pred
  - $$ = gen_or($1, $3);$$
- expr OR val
  - $$ = gen_vor($1, $3);$$

pred: field val
- $$ = gen_cmp($1, $2);$$

field: proto dir selector

<table>
<thead>
<tr>
<th>sym</th>
<th>fld</th>
<th>code</th>
</tr>
</thead>
</table>
### Example

**LA: DST**

<table>
<thead>
<tr>
<th>TCP</th>
<th>AND</th>
<th>expr</th>
<th>ISH</th>
<th>C2</th>
</tr>
</thead>
</table>

**expr:** `pred`

- `expr AND pred`: $\{ \text{\$\$ = gen\_and($1,$3);} \} $
- `expr AND val`: $\{ \text{\$\$ = gen\_vand($1, $3);} \} $
- `expr OR pred`: $\{ \text{\$\$ = gen\_or($1, $3);} \} $
- `expr OR val`: $\{ \text{\$\$ = gen\_vor($1, $3);} \} $

**pred:** `field val`

- `field: proto dir selector`

- `sym: f1d code`
Example

LA: PORT

<table>
<thead>
<tr>
<th>DST</th>
<th>TCP</th>
<th>AND</th>
<th>expr</th>
<th>ISH</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>sym</td>
<td>fld</td>
<td>code</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **expr**: `pred`
  - `expr AND pred`  
    - `{ $\$ = \text{gen\_and}($1, $3); }`
  - `expr AND val`  
    - `{ $\$ = \text{gen\_vand}($1, $3); }`
  - `expr OR pred`  
    - `{ $\$ = \text{gen\_or}($1, $3); }`
  - `expr OR val`  
    - `{ $\$ = \text{gen\_vor}($1, $3); }`

- **pred**: `field val`
  - `{ $\$ = \text{gen\_cmp}($1, $2); }`

- **field**: `proto dir selector`
### Example

**LA: Z**

<table>
<thead>
<tr>
<th>PORT</th>
<th>DST</th>
<th>TCP</th>
<th>AND</th>
<th><strong>expr</strong></th>
<th>ISH</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sym</strong></td>
<td>fld</td>
<td>code</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### pred

- `expr AND pred`:
  ```
  \$
  = \text{gen\_and}(1, 3);
  \$
  ```

- `expr AND val`:
  ```
  \$
  = \text{gen\_vand}(1, 3);
  \$
  ```

- `expr OR pred`:
  ```
  \$
  = \text{gen\_or}(1, 3);
  \$
  ```

- `expr OR val`:
  ```
  \$
  = \text{gen\_vor}(1, 3);
  \$
  ```

#### field

- `field: proto`:
- `field: dir`:
- `field: selector`

```
\$
= \text{gen\_cmp}(1, 2);
\$
```
Example

<table>
<thead>
<tr>
<th>field</th>
<th>TDP</th>
<th>expr</th>
<th>ISH</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td></td>
<td>pred</td>
<td></td>
<td></td>
</tr>
<tr>
<td>expr</td>
<td>ISH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sym</td>
<td>fld</td>
<td>code</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**expr: pred**

- `expr AND pred`:
  ```
  $$ \begin{align*}
  \text{gen_and}(\$1, \$3); \end{align*}
  $$
  ```
- `expr AND val`:
  ```
  $$ \begin{align*}
  \text{gen_vand}(\$1, \$3); \end{align*}
  $$
  ```
- `expr OR pred`:
  ```
  $$ \begin{align*}
  \text{gen_or}(\$1, \$3); \end{align*}
  $$
  ```
- `expr OR val`:
  ```
  $$ \begin{align*}
  \text{gen_vor}(\$1, \$3); \end{align*}
  $$
  ```

**pred: field val**:

```
 $$ \begin{align*}
 \text{gen_cmp}(\$1, \$2); \end{align*}
 $$
 ```

**field: proto dir selector**
### Example

#### LA: `<eof>`

<table>
<thead>
<tr>
<th><code>val(z)</code></th>
<th>TDP</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>field</code></td>
<td>ISH</td>
<td><code>C2</code></td>
</tr>
<tr>
<td><code>AND</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>expr</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### `expr`: `pred`

- `expr AND pred`: 
  ```
  $$
  $$$ = \text{gen\_and}($1, $3);
  $$
  ```
- `expr AND `val``: 
  ```
  $$
  $$$ = \text{gen\_vand}($1, $3);
  $$
  ```
- `expr OR pred`: 
  ```
  $$
  $$$ = \text{gen\_or}($1, $3);
  $$
  ```
- `expr OR `val`: 
  ```
  $$
  $$$ = \text{gen\_vor}($1, $3);
  $$
  ```

#### `pred`: `field` `val`

- `field`: `proto` `dir` `selector`
Example

LA: <eof>

<table>
<thead>
<tr>
<th>Field</th>
<th>TDP</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>val(z)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>field</td>
<td>ISH</td>
<td>C2</td>
</tr>
<tr>
<td>AND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>expr</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

pred: field val

expr: pred

- expr AND pred
  - $$ = \text{gen\_and}(\$1, \$3);$$
- expr AND val
  - $$ = \text{gen\_vand}(\$1, \$3);$$
- expr OR pred
  - $$ = \text{gen\_or}(\$1, \$3);$$
- expr OR val
  - $$ = \text{gen\_vor}(\$1, \$3);$$

field: proto dir selector

sym | fld | code |
|-----|-----|------|

- pred: field val
  - $$ = \text{gen\_cmp}(\$1, \$2);$$
Example

LA: <eof>

<table>
<thead>
<tr>
<th>pred</th>
<th>TDP</th>
<th>C3</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>expr</td>
<td>ISH</td>
<td>C2</td>
</tr>
</tbody>
</table>

expr: pred
| expr AND pred | { $$$ = gen_and($1,$3); } |
| expr AND val  | { $$$ = gen_vand($1, $3); } |
| expr OR pred  | { $$$ = gen_or($1, $3); } |
| expr OR val   | { $$$ = gen_vor($1, $3); } |

pred: field val
{ $$$ = gen_cmp($1, $2); }

field: proto dir selector
### Example

<table>
<thead>
<tr>
<th>LA: &lt;eof&gt;</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>pred</th>
<th>TDP</th>
<th>C3</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>expr</td>
<td>ISH</td>
<td>C2</td>
</tr>
<tr>
<td>sym</td>
<td>fld</td>
<td>code</td>
</tr>
</tbody>
</table>

#### expr: pred

- \( \text{expr AND pred} \)
  
  \[
  \text{gen\_and($1, $3);}
  \]

- \( \text{expr AND val} \)
  
  \[
  \text{gen\_vand($1, $3);}
  \]

- \( \text{expr OR pred} \)
  
  \[
  \text{gen\_or($1, $3);}
  \]

- \( \text{expr OR val} \)
  
  \[
  \text{gen\_vor($1, $3);}
  \]

#### pred: field val

\[
\text{gen\_cmp($1, $2);}
\]

#### field: proto dir selector
### Example

**LA:** `<eof>`

<table>
<thead>
<tr>
<th>expr</th>
<th>ISH</th>
<th><code>C_4</code></th>
</tr>
</thead>
</table>

**output BPF code**

- **expr:** `pred`

  - `expr AND pred`: \{ ``` = gen_and($1,$3); ``` \}
  - `expr AND val`: \{ ``` = gen_vand($1,$3); ``` \}
  - `expr OR pred`: \{ ``` = gen_or($1,$3); ``` \}
  - `expr OR val`: \{ ``` = gen_vor($1,$3); ``` \}

- **pred:** `field val`
  - \{ ``` = gen_cmp($1,$2); ``` \}

- **field:** `proto dir selector`
Code Generation

• Now that we have a language and a parser to translate it, how do those gen() functions actually work?
  – gen_cmp() generates code to compare a packet field to a value
  – Ex: tcp src port 100
    • “tcp src port” is the field
    • 100 is the value
tcp src port 100

IP?

frag 0?

TCP?

sport 100?
Compound Logic

• Now, what if want traffic in either direction for port 100?
  – tcp port 100
    • tcp src port 100
    • OR
    • tcp dst port 100
tcp src port 100

IP?

frag 0?

TCP?

sport 100?

FALSE

TRUE
tcp port 100

- **tcp src port 100**
  - IP?
  - frag 0?
  - TCP?
  - sport 100?
  - FALSE
  - TRUE

- **tcp dst port 100**
  - IP?
  - frag 0?
  - TCP?
  - dport 100?
  - FALSE
  - TRUE

OR

- FALSE
- TRUE
tcp port 100

tcp src port 100

OR

tcp dst port 100
Compound Logic

• What if I want packets *between* port 100 and some other specific port 200?
  – tcp port 100
  – AND
  – tcp port 200

• The output gets ever more complex
tcp port 100 and 200?

tcp port 100
tcp port 100 and 200

tcp port 100

FALSE

tcp port 200

TRUE
tcp port 100 and 200

true
false

AND

tcp port 100

false
true

FALSE

true
false

TRUE
tcp port 100 and 200
### The Raw Code

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>ldh</td>
<td>[16]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>jeq</td>
<td>#0x800</td>
<td>jt 2</td>
<td>jf 43</td>
</tr>
<tr>
<td>002</td>
<td>ldh</td>
<td>[16]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>003</td>
<td>jeq</td>
<td>#0x86dd</td>
<td>jt 4</td>
<td>jf 10</td>
</tr>
<tr>
<td>004</td>
<td>ldb</td>
<td>[24]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>005</td>
<td>jeq</td>
<td>#0x6</td>
<td>jt 6</td>
<td>jf 10</td>
</tr>
<tr>
<td>006</td>
<td>ldh</td>
<td>[58]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>007</td>
<td>jeq</td>
<td>#0x64</td>
<td>jt 22</td>
<td>jf 8</td>
</tr>
<tr>
<td>008</td>
<td>ldh</td>
<td>[60]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>009</td>
<td>jeq</td>
<td>#0x64</td>
<td>jt 22</td>
<td>jf 10</td>
</tr>
<tr>
<td>010</td>
<td>ldh</td>
<td>[16]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>jeq</td>
<td>#0x800</td>
<td>jt 12</td>
<td>jf 43</td>
</tr>
<tr>
<td>012</td>
<td>ldb</td>
<td>[27]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>013</td>
<td>jeq</td>
<td>#0x6</td>
<td>jt 14</td>
<td>jf 43</td>
</tr>
<tr>
<td>014</td>
<td>ldh</td>
<td>[24]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>015</td>
<td>jset</td>
<td>#0x1fff</td>
<td>jt 43</td>
<td>jf 43</td>
</tr>
<tr>
<td>016</td>
<td>ldxb</td>
<td>4*([18]&amp;0xf)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>017</td>
<td>ldh</td>
<td>[x + 18]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>018</td>
<td>jeq</td>
<td>#0x64</td>
<td>jt 22</td>
<td>jf 19</td>
</tr>
<tr>
<td>019</td>
<td>ldxb</td>
<td>4*([18]&amp;0xf)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>020</td>
<td>ldh</td>
<td>[x + 20]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>021</td>
<td>jeq</td>
<td>#0x64</td>
<td>jt 22</td>
<td>jf 43</td>
</tr>
<tr>
<td>022</td>
<td>ldh</td>
<td>[16]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>023</td>
<td>jeq</td>
<td>#0x86dd</td>
<td>jt 24</td>
<td>jf 30</td>
</tr>
<tr>
<td>024</td>
<td>ldb</td>
<td>[24]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>025</td>
<td>jeq</td>
<td>#0x6</td>
<td>jt 26</td>
<td>jf 30</td>
</tr>
<tr>
<td>026</td>
<td>ldh</td>
<td>[58]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>027</td>
<td>jeq</td>
<td>#0xc8</td>
<td>jt 42</td>
<td>jf 28</td>
</tr>
<tr>
<td>028</td>
<td>ldh</td>
<td>[60]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>029</td>
<td>jeq</td>
<td>#0xc8</td>
<td>jt 42</td>
<td>jf 30</td>
</tr>
<tr>
<td>030</td>
<td>ldh</td>
<td>[16]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>031</td>
<td>jeq</td>
<td>#0x800</td>
<td>jt 32</td>
<td>jf 43</td>
</tr>
<tr>
<td>032</td>
<td>ldb</td>
<td>[27]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>033</td>
<td>jeq</td>
<td>#0x6</td>
<td>jt 34</td>
<td>jf 43</td>
</tr>
<tr>
<td>034</td>
<td>ldh</td>
<td>[24]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>035</td>
<td>jset</td>
<td>#0x1fff</td>
<td>jt 43</td>
<td>jf 36</td>
</tr>
<tr>
<td>036</td>
<td>ldxb</td>
<td>4*([18]&amp;0xf)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>037</td>
<td>ldh</td>
<td>[x + 18]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>038</td>
<td>jeq</td>
<td>#0xc8</td>
<td>jt 42</td>
<td>jf 39</td>
</tr>
<tr>
<td>039</td>
<td>ldxb</td>
<td>4*([18]&amp;0xf)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>040</td>
<td>ldh</td>
<td>[x + 20]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>041</td>
<td>jeq</td>
<td>#0xc8</td>
<td>jt 42</td>
<td>jf 43</td>
</tr>
<tr>
<td>042</td>
<td>ret</td>
<td>#65535</td>
<td></td>
<td></td>
</tr>
<tr>
<td>043</td>
<td>ret</td>
<td>#0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Redundant and Inefficient
Enter Optimization

• Post-process generated code with optimization techniques
  – libpcap/optimize.c

• Leveraged a bunch of well known techniques from my compilers course
The Dominator Concept

- A well-known technique global data flow optimization at the time used dominators

\[ \text{DOM}(1) = \{ 2, 3, 4, 5 \} \]
\[ \text{DOM}(2) = \{ \} \]
\[ \text{DOM}(3) = \{ 4, 5 \} \]
DOM Example

- Global common sub-expression elimination

\[ a = b + c \]

1 <-> 2

\[ x = b + c \]
DOM Example

• Global common sub-expression elimination

\[
a = b + c
\]

• 2 in DOM(1) => variables on entry to 2 same as on exit to 1, so we can replace \( b + c \) with \( a \)
Not Quite Enough...

• While these traditional techniques are used by the BPF optimizer, they weren’t enough...

Knowing the top node dominates yellow node doesn’t let us eliminate the redundant test for A at the yellow node because either the red or blue path could happen
Edge Dominators

• But if we look at edge relationships instead of node relationships, we can solve the problem

```plaintext
EDOM(1) = { 3, 4 }
EDOM(2) = { }    
EDOM(3) = { }
```
Edge Optimization

• With this knowledge, we can safely move edges to optimize the code...

EDOM(1) = { 3, 4 }
EDOM(2) = { }
EDOM(3) = { }

3 in EDOM(1) =>
know A is true at 3 =>
we can move 3 past second check
Edge Optimization

• With this knowledge, we can safely move edges to optimize the code...

EDOM(1) = \{ 3, 4 \}
EDOM(2) = \{ \} 
EDOM(3) = \{ \}

3 in EDOM(1) =>
know A is true at 3 =>
we can move 3 past second check
Edge Optimization

- Movements create new opportunities. Update EDOM and repeat...

EDOM(1) = \{ 3, 4 \}
EDOM(2) = \{ \}
EDOM(3) = \{ \}
Edge Optimization

- Movements create new opportunities. Update EDOM and repeat...

EDOM(1) = { 3, 4 }
EDOM(2) = { 5, 6 }
EDOM(3) = { }

5 in EDOM(2) =>
know A is false at 5 =>
we can move 2
Edge Optimization

• Movements create new opportunities. Update EDOM and repeat...

EDOM(1) = { 3, 4 }
EDOM(2) = { 5, 6 }
EDOM(3) = { }
Edge Optimization

• Now we can delete unreachable code....
Edge Optimization

• Now we can delete unreachable code....
Edge Optimization

• and simplify the graph...
tcp port 100 and 200

IP?

frag0?

TCP?

sp 100?

dp 100?

IP?

frag0?

TCP?

sp 200?

dp 200?
tcp port 100 and 200
tcp port 100 and 200


FALSE

TRUE
tcp port 100 and 200

- IP?
- frag0?
- TCP?
  - sp 100?
  - dp 100?
- TRUE

- IP?
- frag0?
- TCP?
  - sp 200?
  - dp 200?
- FALSE
tcp port 100 and 200
tcp port 100 and 200
tcp port 100 and 200

FALSE

TRUE
tcp port 100 and 200
tcp port 100 and 200
tcp port 100 and 200
tcp port 100 and 200
tcp port 100 and 200
tcp port 100 and 200

- IP?
- frag0?
- TCP?
- sp 100?
- dp 100?
- sp 200?
- dp 200?

FALSE

TRUE
tcp port 100 and 200

IP?

frag0?
TCP?
sp 100?

dp 100?

FALSE

TRUE

sp 200?
dp 200?
tcp port 100 and 200
tcp port 100 and 200

IP?

frag0?

TCP?

sp 100?

dp 100?

sp 200?

dp 200?

FALSE

TRUE
Before

(000) ldh [16]  
(001) jeq #0x800 jt 2 jf 43  
(002) ldh [16]  
(003) jeq #0x86dd jt 4 jf 10  
(004) ldb [24]  
(005) jeq #0x6 jt 6 jf 10  
(006) ldh [58]  
(007) jeq #0x64 jt 22jf 8  
(008) ldh [60]  
(009) jeq #0x64 jt 22jf 10  
(010) ldh [16]  
(011) jeq #0x800 jt 12jf 43  
(012) ldb [27]  
(013) jeq #0x6 jt 14 jf 43  
(014) ldh [24]  
(015) jset #0x1fff jt 43 jf 16  
(016) ldxb 4*([18]&0xf)  
(017) ldh [x + 18]  
(018) jeq #0x64 jt 22jf 19  
(019) ldxb 4*([18]&0xf)  
(020) ldh [x + 20]  
(021) jeq #0x64 jt 22jf 43

(022) ldh [16]  
(023) jeq #0x86dd jt 24 jf 30  
(024) ldb [24]  
(025) jeq #0x6 jt 26 jf 30  
(026) ldh [58]  
(027) jeq #0xc8 jt 42 jf 28  
(028) ldb [60]  
(029) jeq #0xc8 jt 42 jf 30  
(030) ldh [16]  
(031) jeq #0x800 jt 32 jf 43  
(032) ldb [27]  
(033) jeq #0x6 jt 34 jf 43  
(034) ldh [24]  
(035) jset #0x1fff jt 43 jf 36  
(036) ldxb 4*([18]&0xf)  
(037) ldh [x + 18]  
(038) jeq #0xc8 jt 42 jf 39  
(039) ldxb 4*([18]&0xf)  
(040) ldh [x + 20]  
(041) jeq #0xc8 jt 42 jf 43  
(042) ret #65535  
(043) ret #0
After

(000) ldh [12]
(001) jeq #0x800 jt 2 jf 15
(002) ldb [23]
(003) jeq #0x6  jt 4  jf 15
(004) ldh [20]
(005) jset #0x1fff  jt 15 jf 6
(006) ldxb 4*[14]&0xf)
(007) ldh [x + 14]
(008) jeq #0x64  jt 9  jf 11
(009) ldh [x + 16]
(010) jeq #0xc8  jt 14 jf 15
(011) jeq #0xc8  jt 12 jf 15
(012) ldh [x + 16]
(013) jeq #0x64  jt 14 jf 15
(014) ret #65535
(015) ret  #0
libpcap

• We realized we wanted to build other packet capture applications beyond tcpdump
  – Pulled compiler system and filtering engine out of tcpdump
  – Created an “API” and reusable library
  – Released as “libpcap”

• If different apps were going to be built around this common library, we should have an interchangeable file format for packets traces
pcap File Format

- Elaboration of the “-w” flag to tcpdump
  - tcpdump -w http.pcap port 80
  - Bypass protocol decoding logic in tcpdump
  - Write packets straight to disk
  - Run as fast as possible to minimize drops

<table>
<thead>
<tr>
<th>HDR</th>
<th>TS,LEN</th>
<th>PKT 1</th>
<th>TS,LEN</th>
<th>PKT 2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>version#</td>
<td>timezone</td>
<td>snaplen</td>
<td>link type...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An Open Approach

• Released tcpdump, BPF, libpcap as open source
  – Ported to various operating systems
  – Berkeley Unix (BSD), SunOS, HP, SGI, DEC

• Eventually adopted in Linux and Mac OS X

• Published in USENIX 93, SIGCOMM 99

• My apologies... escaped before it was done
  – never quite finished libpcap API, then I read about it in Rich Stevens’ TCP/IP Illustrated
  – Loris tells me I messed up the pcap file format 😞
Summary

• So, that’s my story of libpcap
• I’m very honored and excited to return to the packet capture community after all these years...
  – I would have thought all the problems were solved but as we dig deeper every day, it’s clear there is tons of opportunity for innovation...
  – I am looking forward to working with Loris, Gerald, and the community to continue to push the envelope
• It’s hard to make things easy, but it’s worth it in the end