

Secure Networking

Hannes Mehnert
14. October 2006

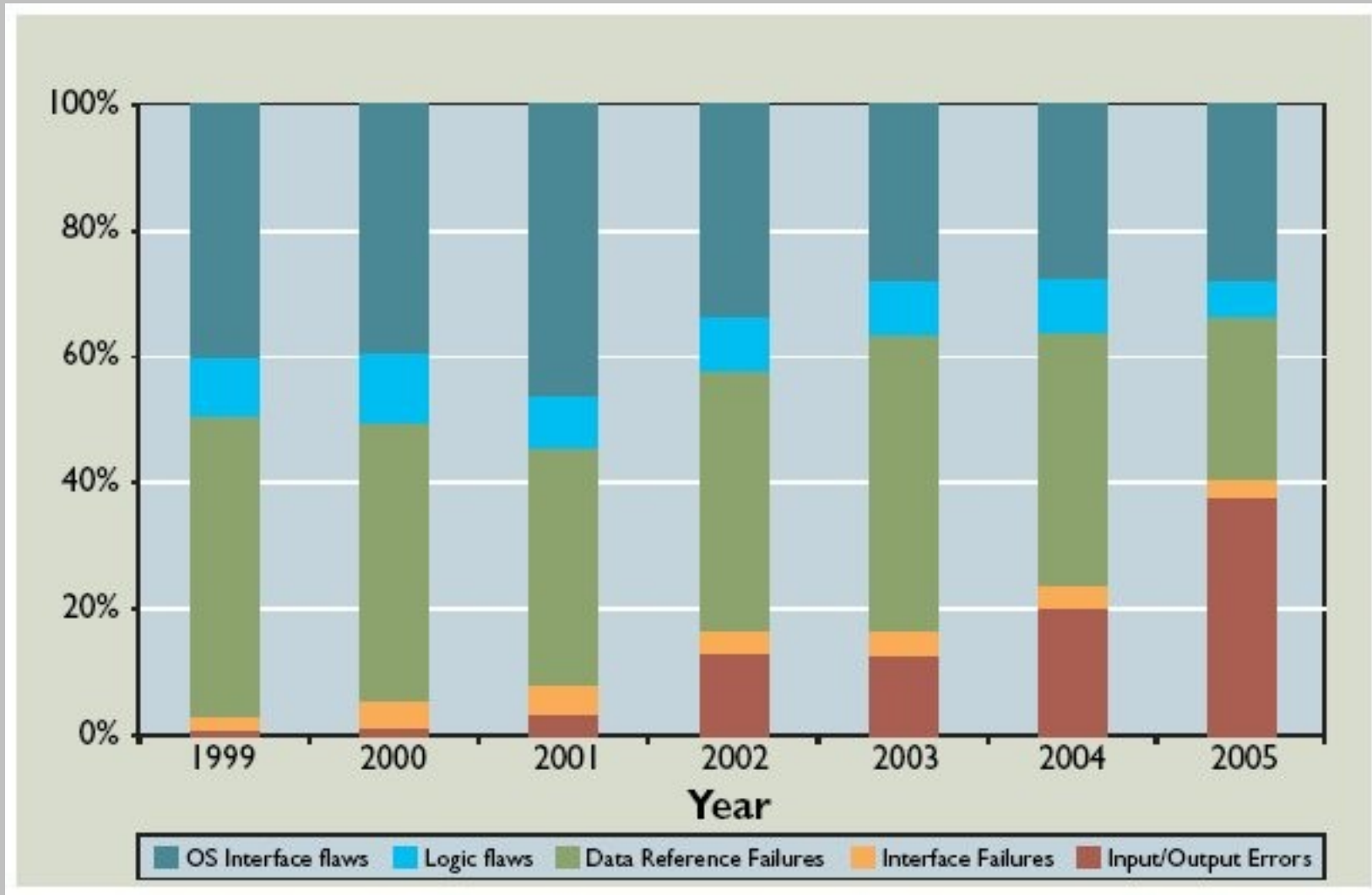


Overview

- Common software vulnerabilities
- Dylan
- Architecture of IP-Stack



CVE sorted by bug class



“Software Security is Software Reliability”, Felix Lindner, CACM 49/6



Data Reference Failures – Workarounds and solutions

- Buffer overflows:
 - Workarounds:
 - Stack canaries
 - Write xor execute
 - Randomized address spaces
 - Solution: Bounds checking
- Integer overflows:
 - Solution: bignums, exception on overflow
- Premature memory release
 - Solution: Automatic memory management



Input/Output Errors

- SQL injections
- Cross-site scripting
- Blue boxing
- 0-byte poisoning
- Perl OPEN



Interface Failures

- Format String Exploits
 - problem: varargs are not type safe
 - solution: language with type safe varargs
- Race conditions



OS Interface Flaws

- Directory Traversal
- Illegal File Access
- Remote Code Execution



Conclusion

- Prevention strategies exist for most bug classes
- Data Reference Failures can be avoided by choice of suitable programming language



Dylan

- Object oriented
- Functional aspects (higher order functions)
- Automatic memory management
- Dynamic and strong typing
- Bounds checks
- Optional type inference
- Supports encapsulation
- Features like scripting language (rapid prototyping), but compiled (performance)

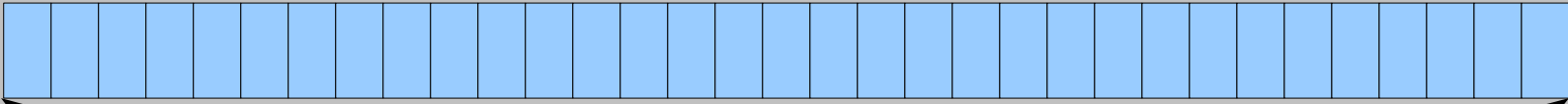


Architecture for secure networking

- Packetizer – to parse and assemble protocols
 - Inspired by scapy
- Flow-Graph library – to specify flow of packets
 - Inspired by click
- Layering-mechanism – to stack protocols
 - Inspired by conduit+



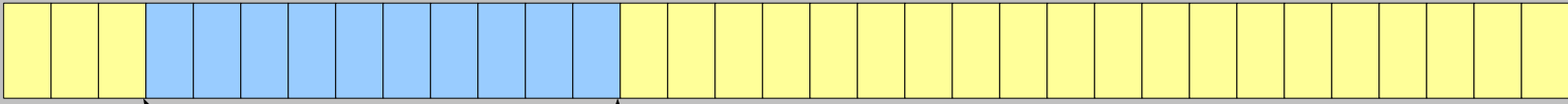
<stretchy-byte-vector-subsequence>



Start: 0

End: #f

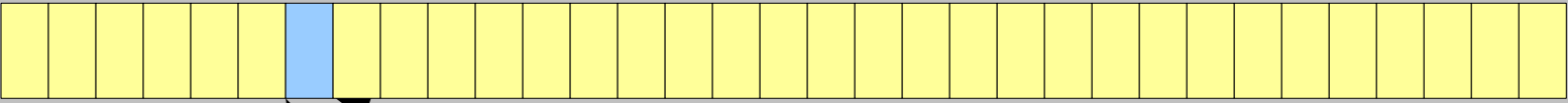
```
let subseq = subsequence(bytes, start: 3 * 8, length: 8 * 10)
```



Start: 3

End: 13

```
subsequence(subseq, start: 3 * 8, length: 8)
```

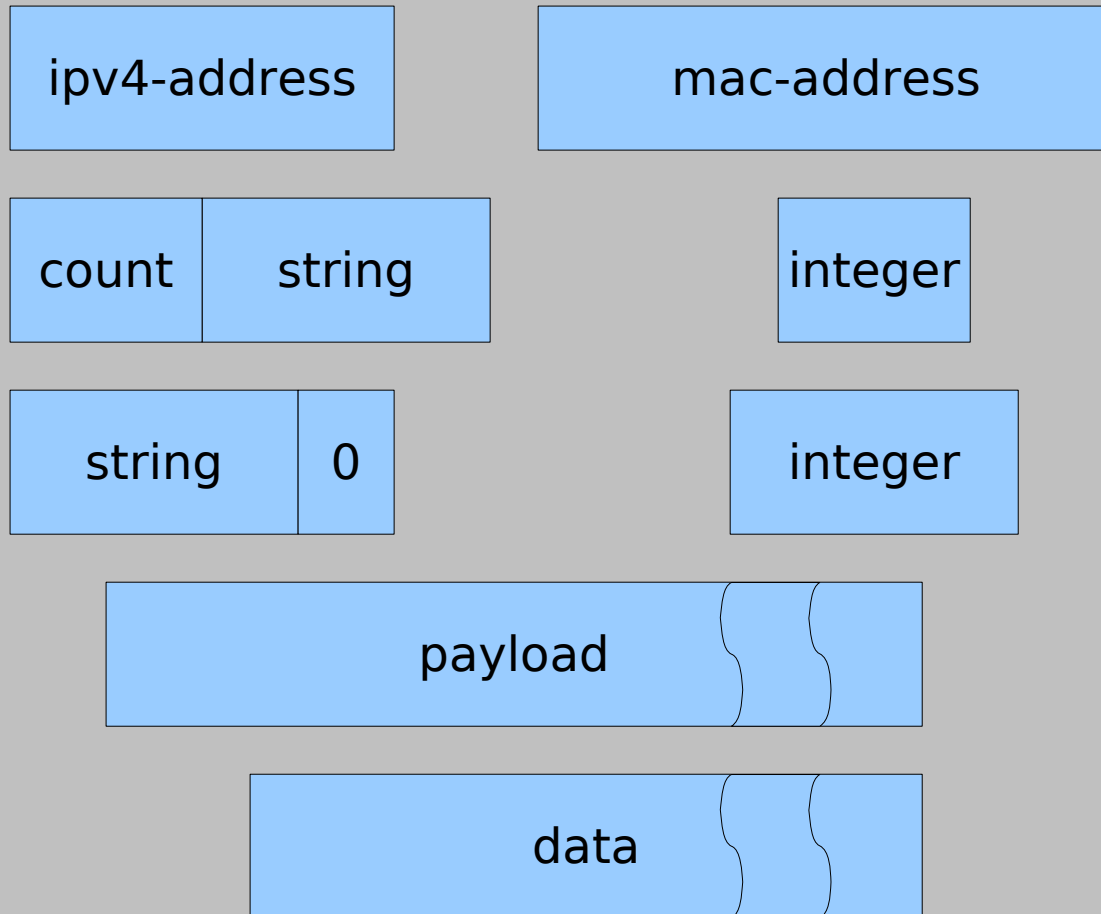


Start: 6

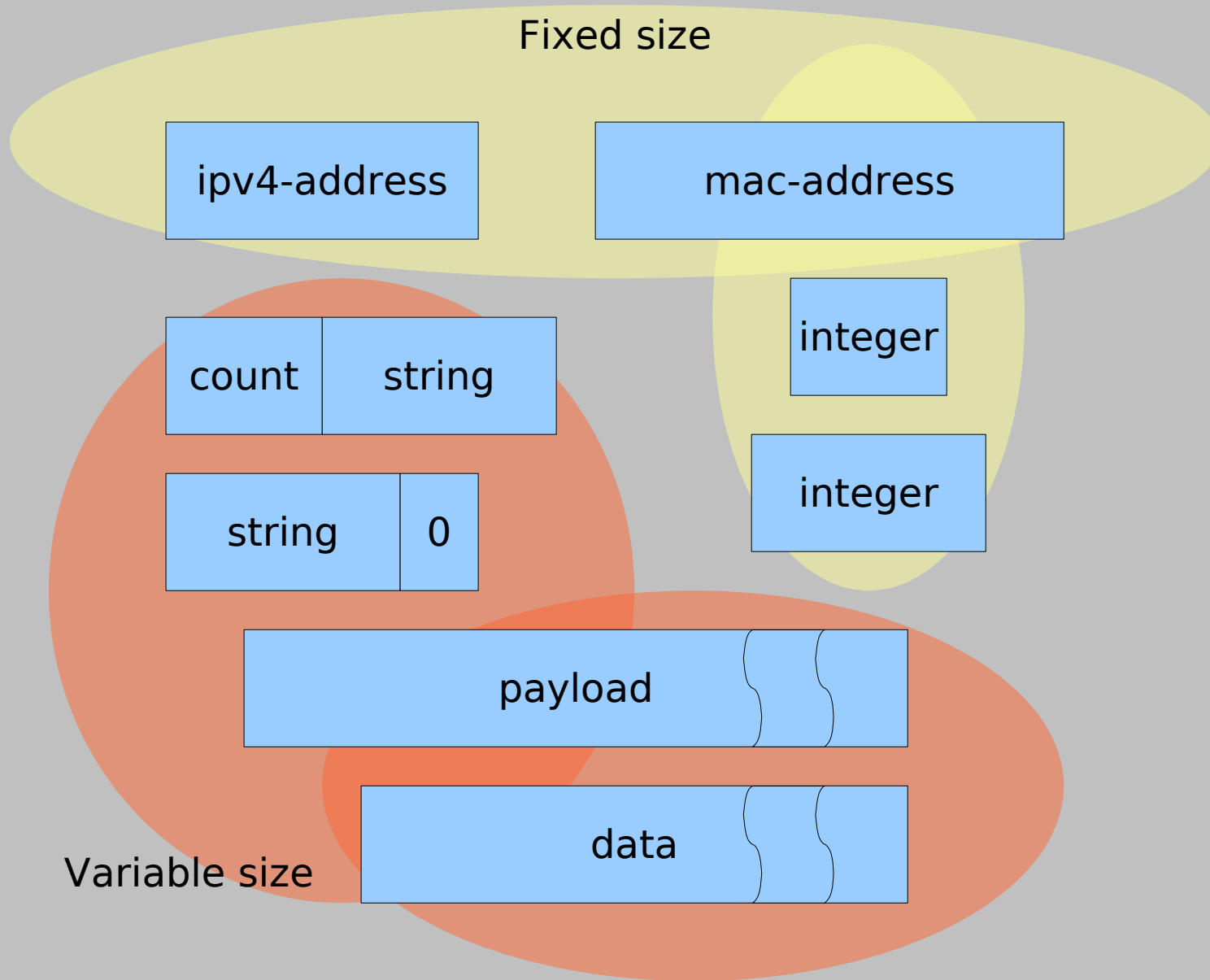
End: 7



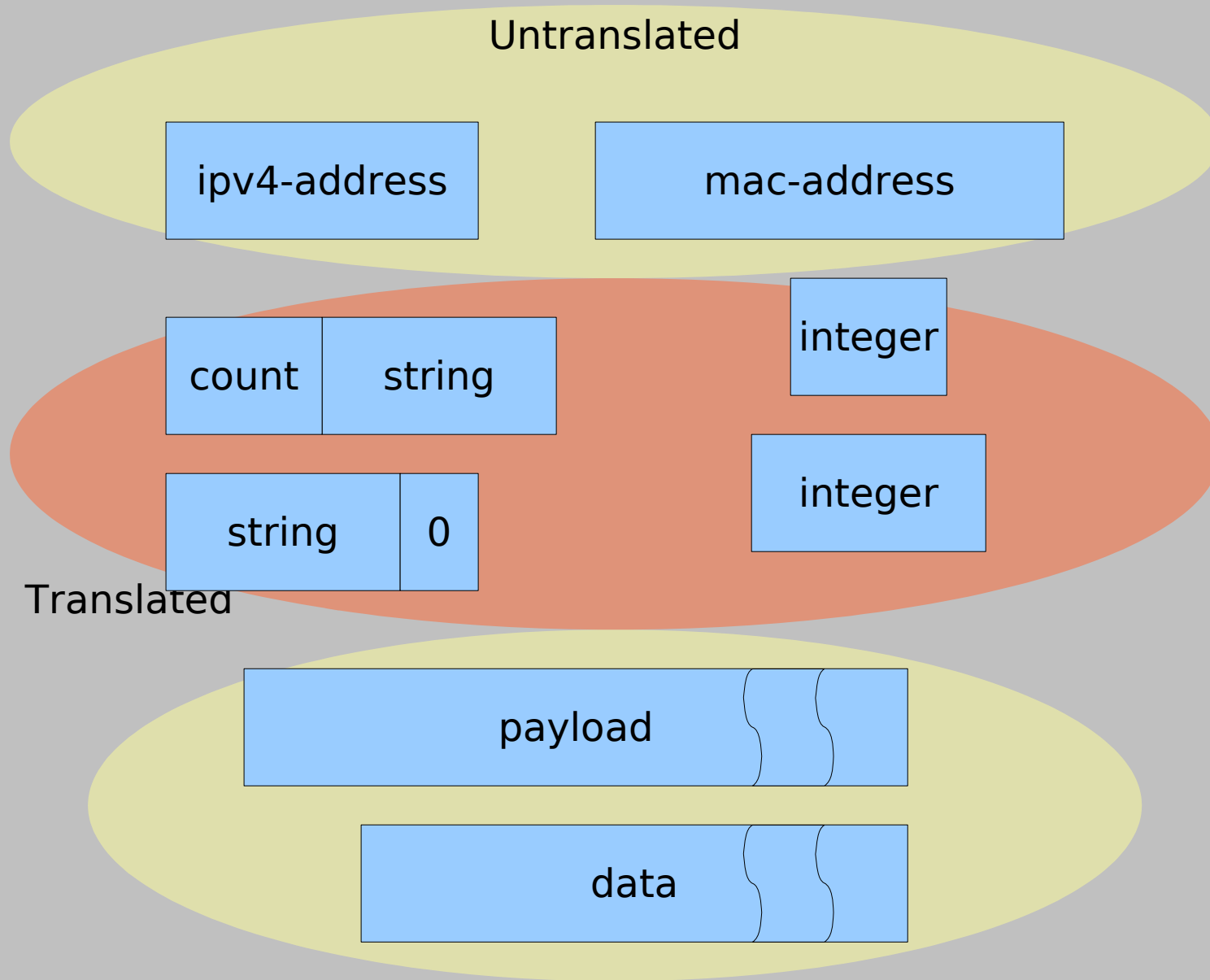
Frames



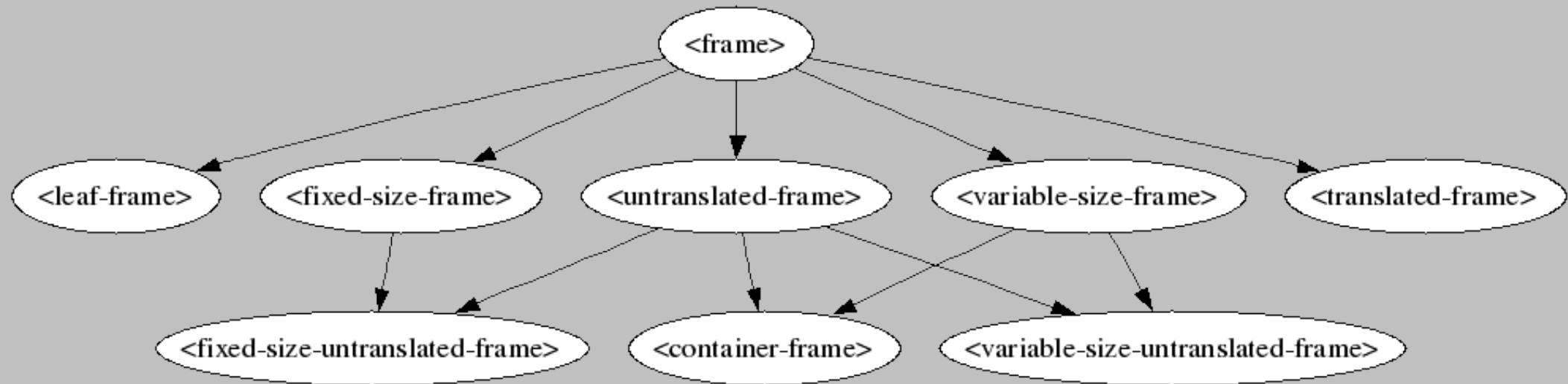
Frames – Size property



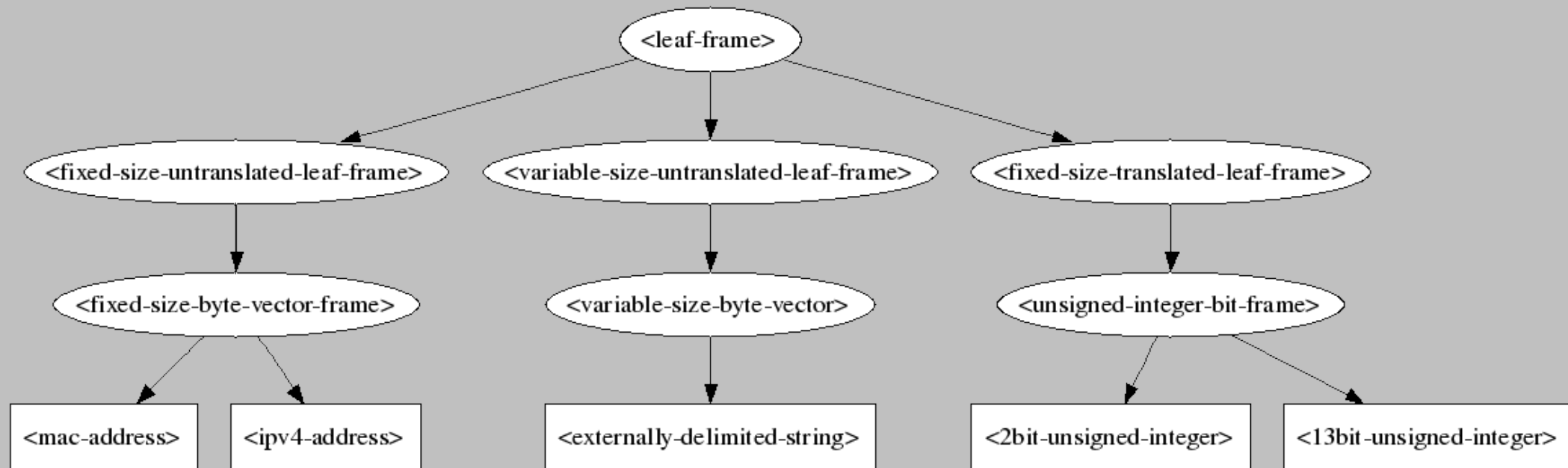
Frames – Translation property



<frame>

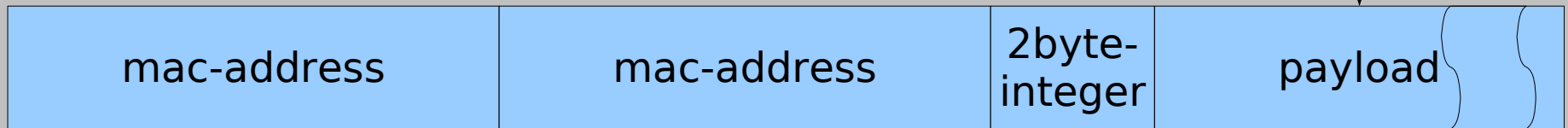


<leaf-frame>



Container Frames

```
Name: payload
type: select (type-code)
    #x800 => <ipv4-frame>
    #x806 => <arp-frame>
end,
static-start: 14 * 8;
```



Name :destination-address
type: mac-address
static-start: 0
static-length: 6 * 8

Name: source-address
type: mac-address
static-start: 6 * 8
static-length: 6 * 8

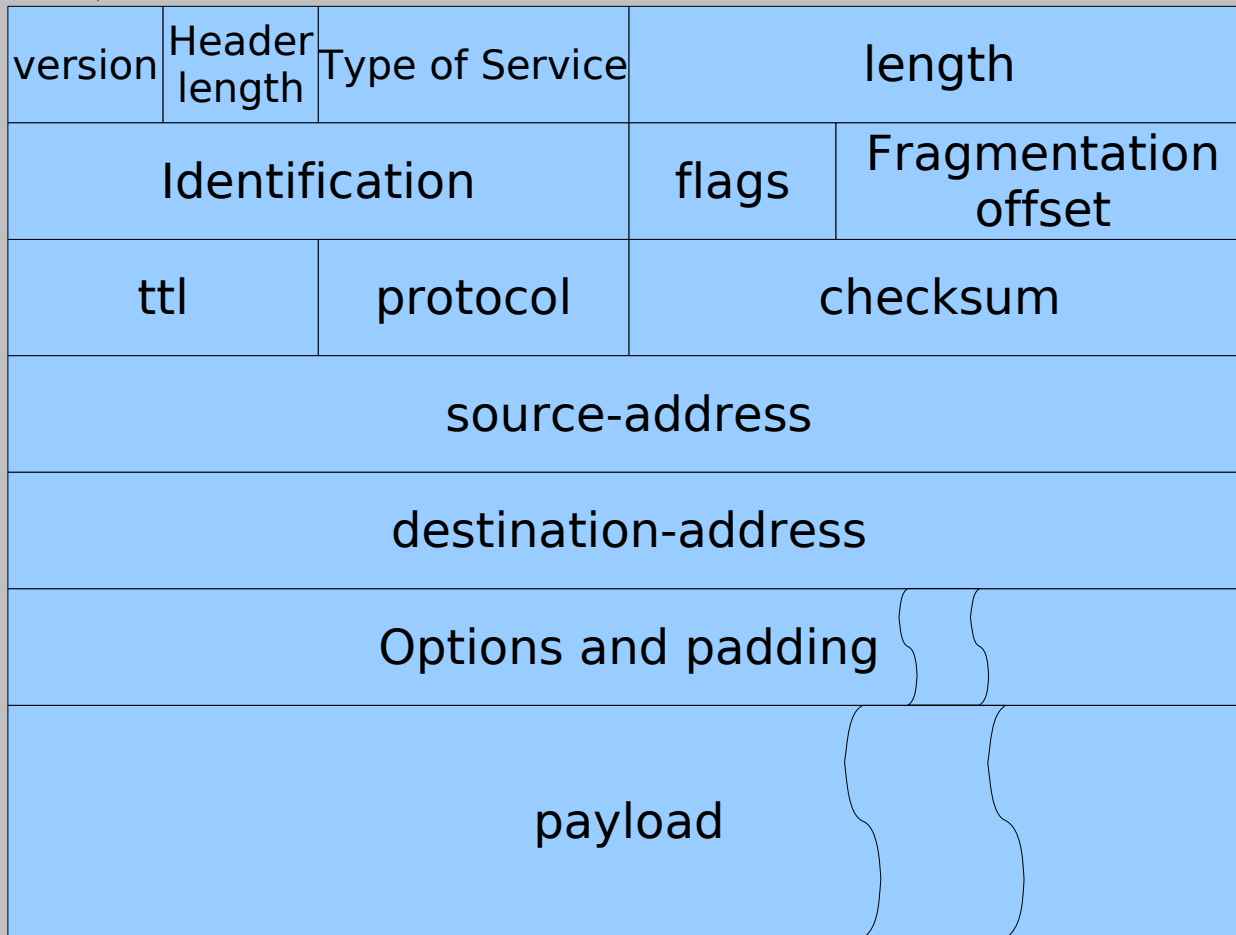
Name: type-code
type: 2byte-big-endian-unsigned-integer
static-start: 12 * 8
static-length: 2 * 8



IPv4

default-value: 4

fixup: ceiling/(size(options) + 20, 4)



fixup: frame.header-length + size(frame.payload)

type-function: select (frame.type-code)
6 => <tcp-frame>
17 => <udp-frame>
end,
start: frame.header-length * 4 * 8,
end: frame.length * 8;

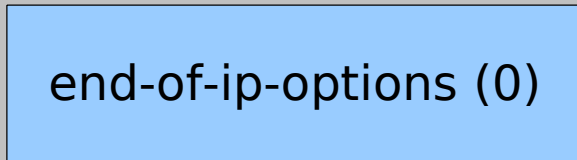
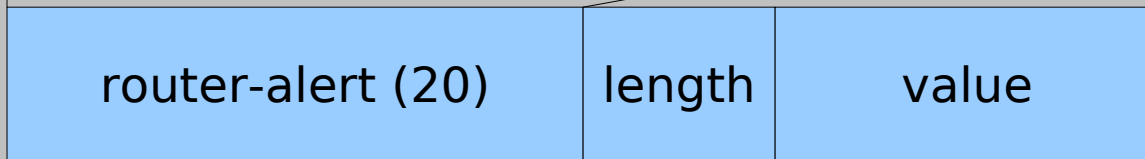


Frame inheritance, repeated fields

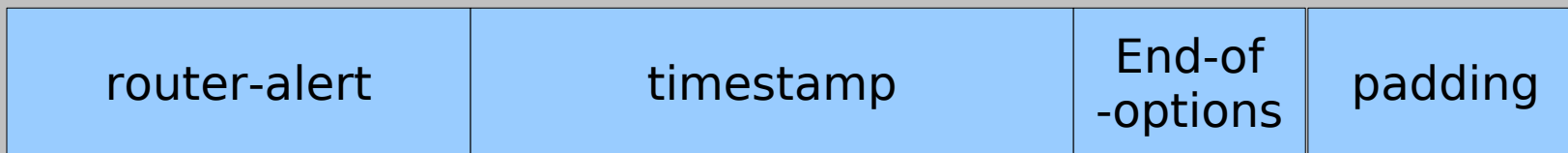
ip-option-header



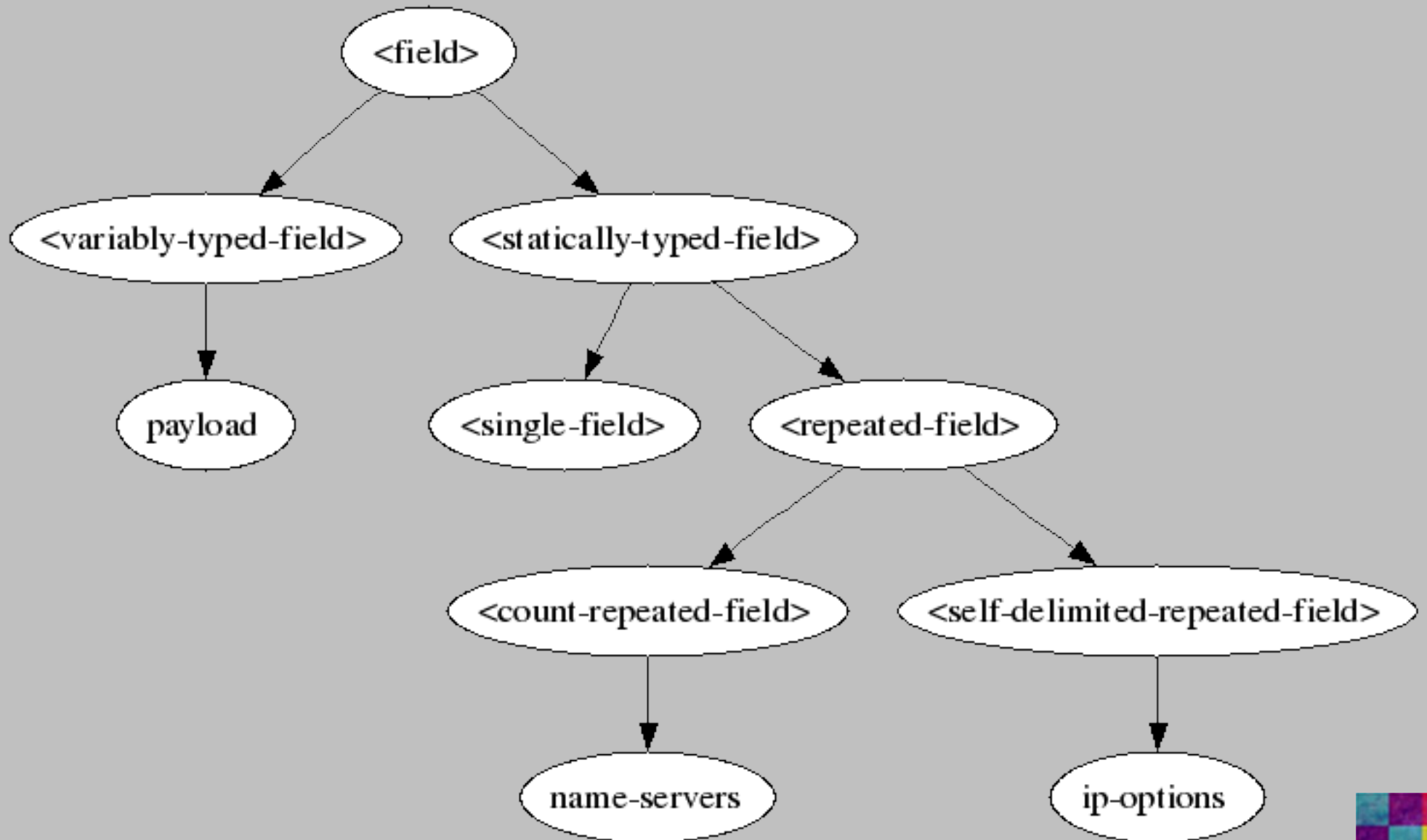
ip-options



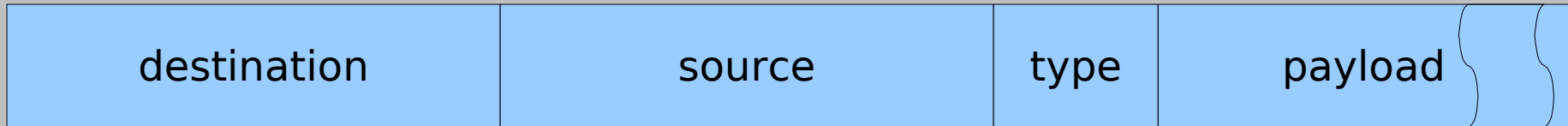
Options field in ipv4-frame



<field>



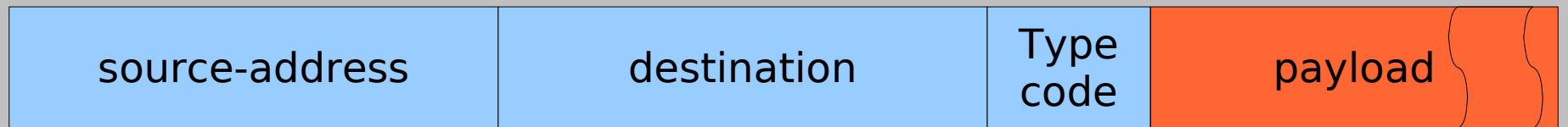
Packetizer Code example



```
define protocol ethernet-frame (header-frame)
  field destination-address :: <mac-address>;
  field source-address :: <mac-address>;
  field type-code
    :: <2byte-big-endian-unsigned-integer>;
  variably-typed-field payload,
    type-function: select (frame.type-code)
      #x800 => <ipv4-frame>;
      #x806 => <arp-frame>;
      otherwise <raw-frame>;
  end;
end;
```

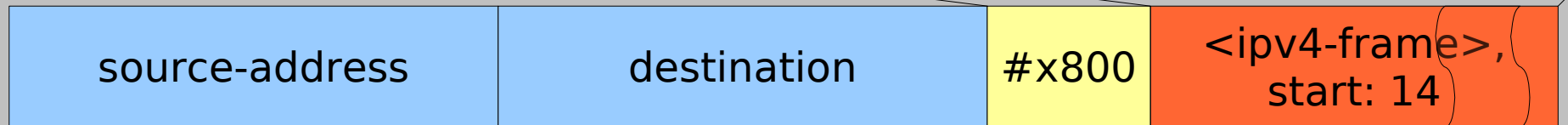


Parsing ethernet-payload

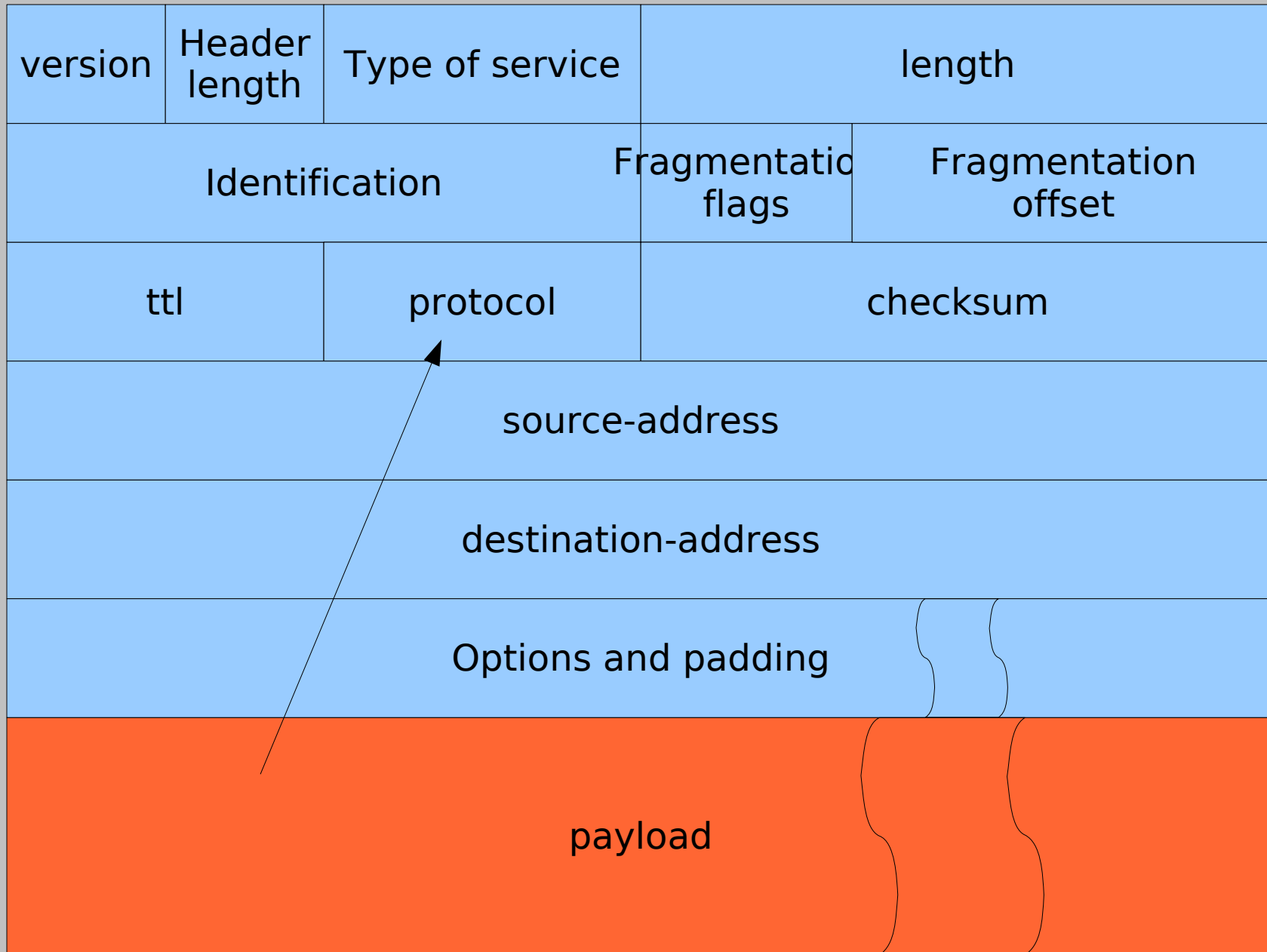


parse type code
static start, static size

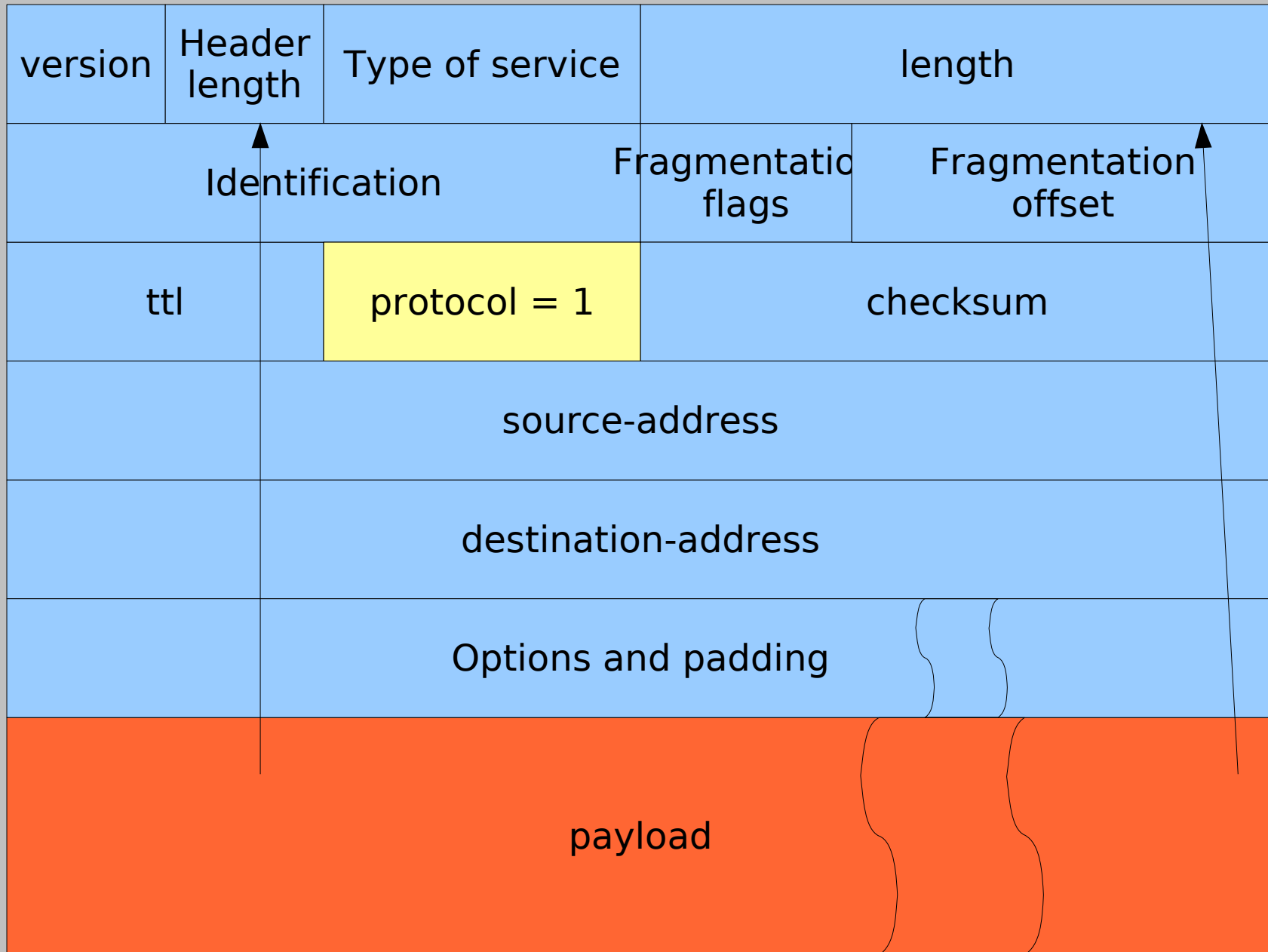
00,de,ad,be,ef,00,00,00,00,12,23,34,08,00,11,12,13,14,15,16,17,18,19,1a,1b, ...



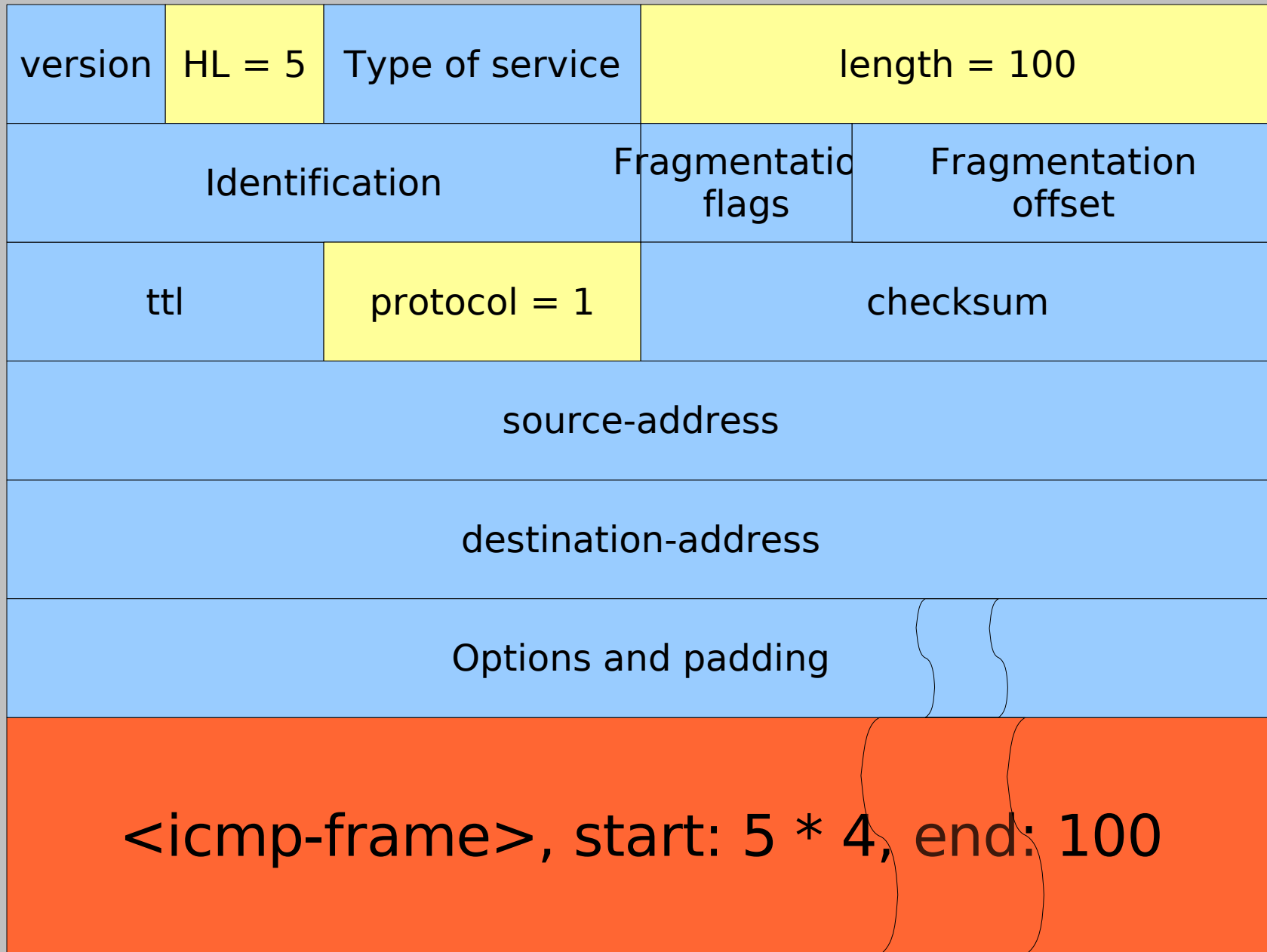
Parsing IPv4 payload - type



Parsing IPv4 payload - boundaries



Parsing payload of IPv4



```

define protocol ipv4-frame (header-frame)
  field version :: <4bit-unsigned-integer> = 4;
  field header-length :: <4bit-unsigned-integer>,
    fixup: ceiling/(reduce(\+, 20, map(frame-size, frame.options)), 4);
  field type-of-service :: <unsigned-byte> = 0;
  field total-length :: <2byte-big-endian-unsigned-integer>,
    fixup: frame.header-length * 4 + frame-size(frame.payload);
  field identification :: <2byte-big-endian-unsigned-integer> = 23;
  field evil :: <1bit-unsigned-integer> = 0;
  field dont-fragment :: <1bit-unsigned-integer> = 0;
  field more-fragments :: <1bit-unsigned-integer> = 0;
  field fragment-offset :: <13bit-unsigned-integer> = 0;
  field time-to-live :: <unsigned-byte> = 64;
  field protocol :: <unsigned-byte>;
  field header-checksum :: <2byte-big-endian-unsigned-integer> = 0;
  field source-address :: <ipv4-address>;
  field destination-address :: <ipv4-address>;
  repeated field options :: <ip-option-frame> = make(<stretchy-vector>),
    reached-end?: method(value :: <ip-option-frame>)
      instance?(value, <end-of-option-ip-option>)
    end;
  variably-typed-field payload,
    start: frame.header-length * 4 * 8,
    end: frame.total-length * 8,
    type-function: payload-type(frame);
end;

```

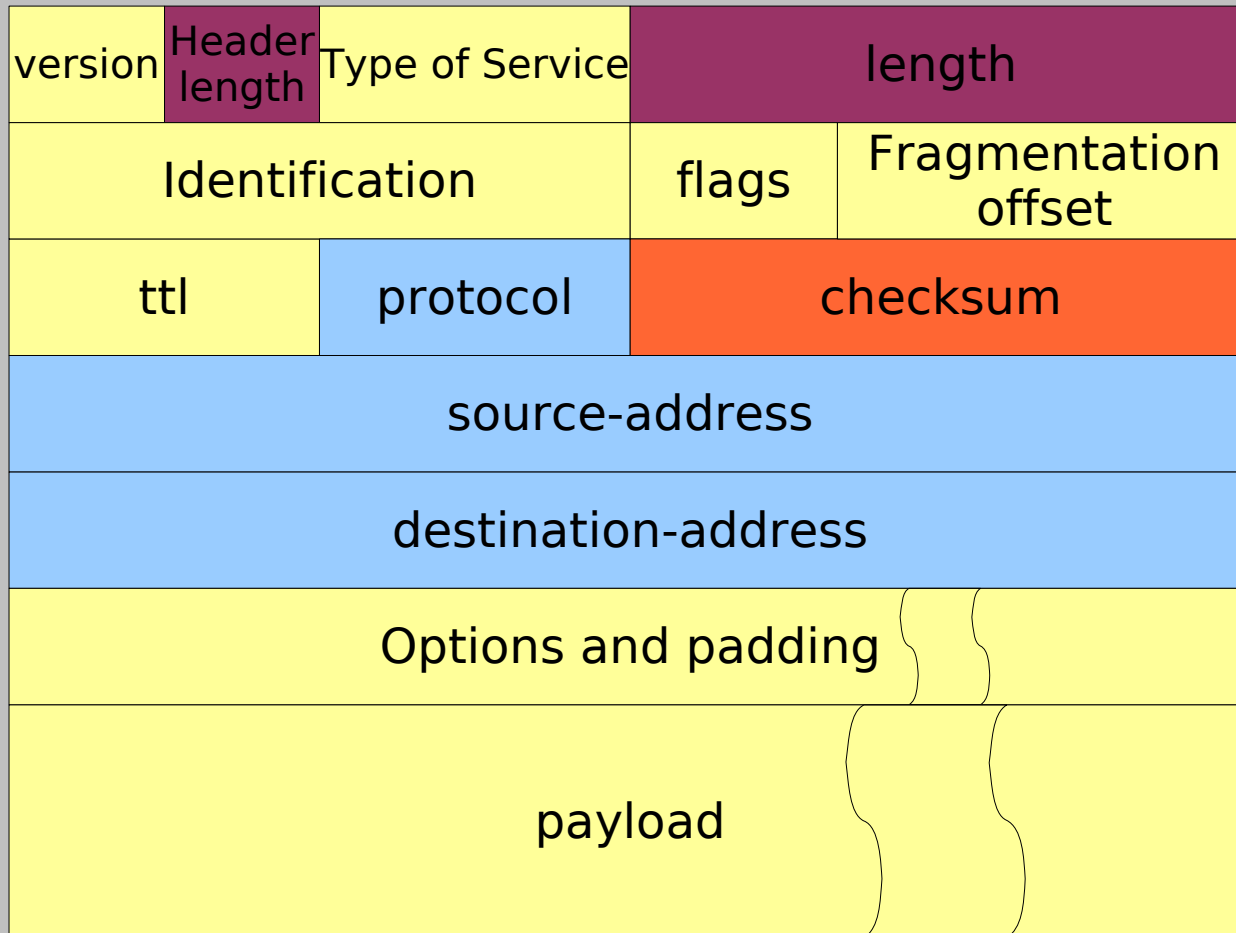


Parse code example

```
let frame = parse-frame(<ipv4-frame>,
                        packet: some-data);
format-out("Source address %=\n",
          frame.source-address);
```



Assembly IPv4



User provided

Default value

fixup

fixup!



Assembly code example

```
let v4-frame = make(<ipv4-frame>,  
  source-address: ipv4-address("23.23.23.23"),  
  destination-address: ipv4-address("42.42.42.42"),  
  protocol: 23);  
let byte-vector = assemble-frame(v4-frame).packet;
```



Assembly IPv4 – fixup

```
define method fixup! (frame :: <ipv4-frame>)  
  frame.header-checksum  
    := calculate-checksum(frame.packet);  
  fixup!(frame.payload);  
end;
```



Filter language

- Operators

- And &
- Or |
- Not ~

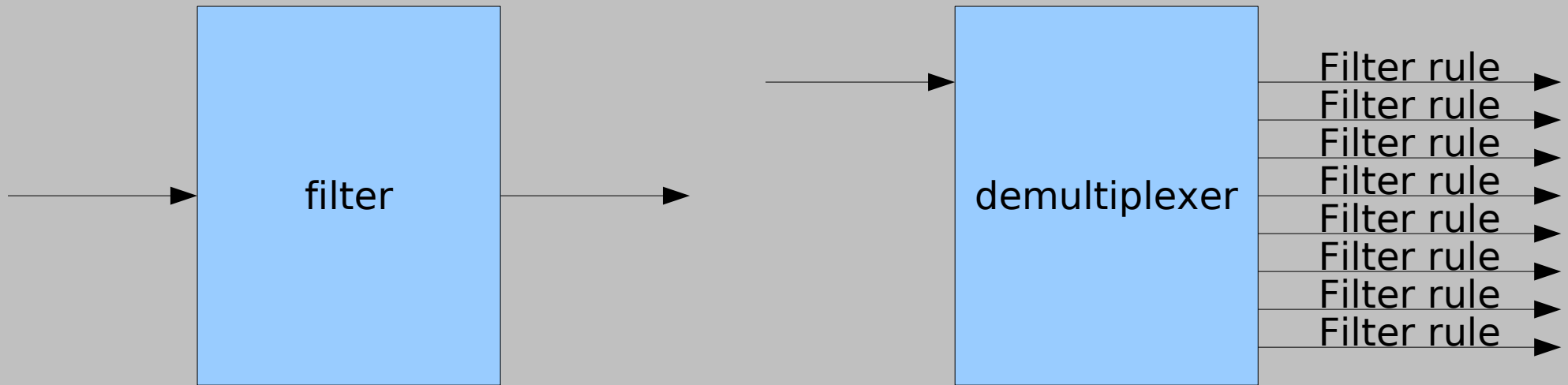
- Rules

- Presence of a frame-type (“ipv4”, “~ (dns)”)
- Value of a field (“ipv4.destination-address = 23.23.23.23”)

- “(udp.source-port = 53) | (udp.destination-port = 53)”

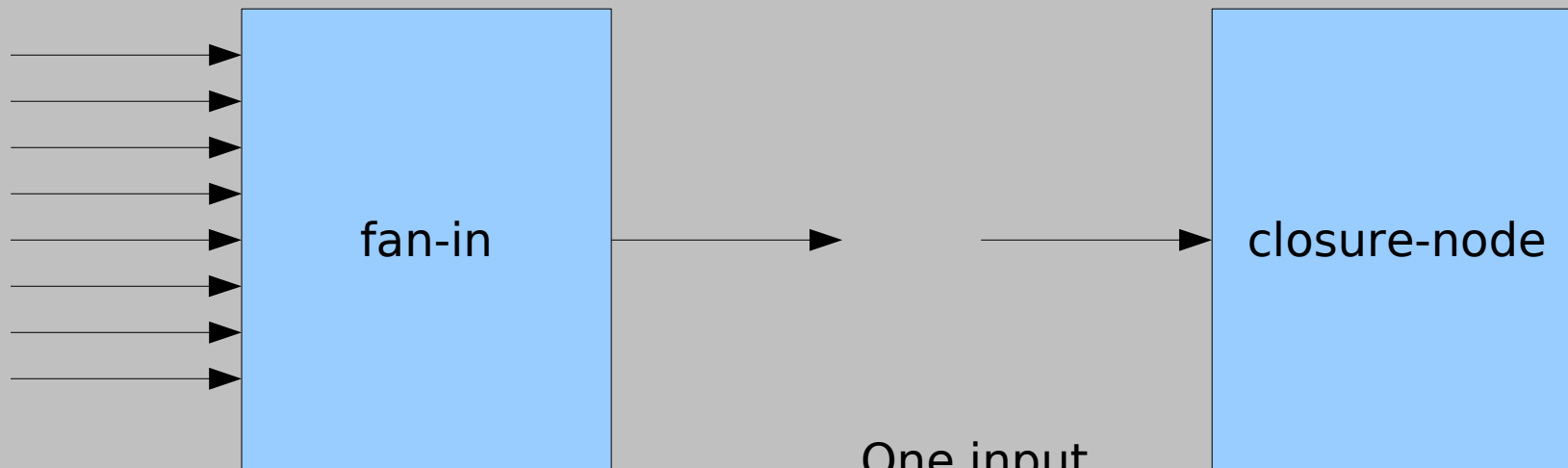


Flow-Graph



One input
one output

One input
multiple outputs,
each is associated with a filter rule



Multiple inputs
one output

One input
executes closure
with each packet received

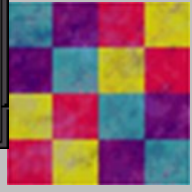


Example: simple-sniffer

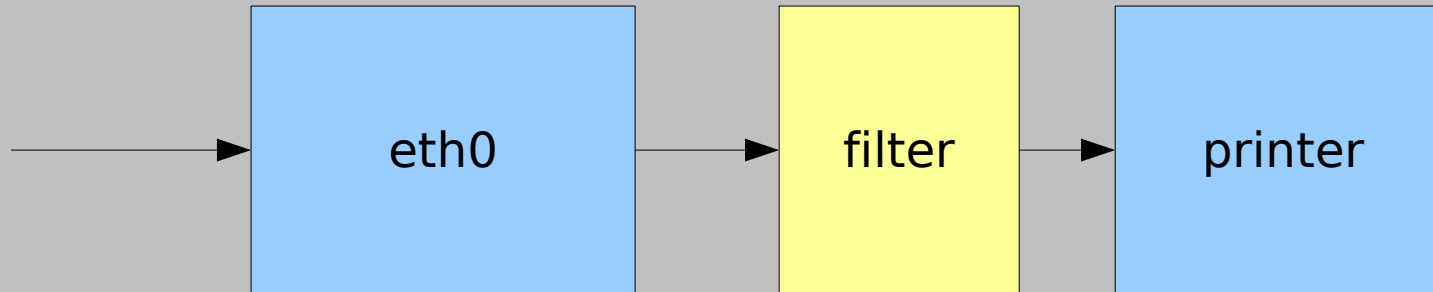


```
let eth0 = make(<ethernet-interface>, name: "eth0");
connect(eth0, make(<summary-printer>));
toplevel(eth0);
```

```
hannes@localhost:~
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.29 DST 192.168.1.35/TCP AR port 945 -> 57014
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.29 DST 192.168.1.35/TCP AR port 244 -> 57015
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.29 DST 192.168.1.35/TCP AR port 1413 -> 57016
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.29 DST 192.168.1.35/TCP AR port 877 -> 57017
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.29 DST 192.168.1.35/TCP AR port 532 -> 57018
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.29 DST 192.168.1.35/TCP AR port 404 -> 57019
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.29 DST 192.168.1.35/TCP AR port 16959 -> 57020
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.29 DST 192.168.1.35/TCP AR port 1030 -> 57021
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.29 DST 192.168.1.35/TCP AR port 4333 -> 57022
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.29 DST 192.168.1.35/TCP AR port 165 -> 57023
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.29 DST 192.168.1.35/TCP AR port 582 -> 57024
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.29 DST 192.168.1.35/TCP AR port 739 -> 57025
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.29 DST 192.168.1.35/TCP AR port 1214 -> 57026
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.29 DST 192.168.1.35/TCP AR port 957 -> 57027
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.29 DST 192.168.1.35/TCP AR port 290 -> 57028
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.29 DST 192.168.1.35/TCP AR port 1987 -> 57029
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.29 DST 192.168.1.35/TCP AR port 557 -> 57030
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.29 DST 192.168.1.35/TCP AR port 305 -> 57031
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.29 DST 192.168.1.35/TCP AR port 27000 -> 57032
ETH 00:0D:60:B1:E5:C8 -> 00:13:49:06:41:4E/IP SRC 192.168.1.35 DST 213.73.91.42/TCP AP port 64529 -> 1234
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.42 DST 192.168.1.35/TCP A port 1234 -> 64529
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/IP SRC 213.73.91.42 DST 192.168.1.35/TCP AP port 1234 -> 64529
ETH 00:0D:60:B1:E5:C8 -> 00:13:49:06:41:4E/IP SRC 192.168.1.35 DST 213.73.91.42/TCP A port 64529 -> 1234
```

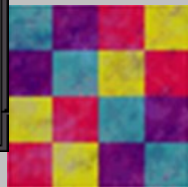


Example: simple-sniffer with filter

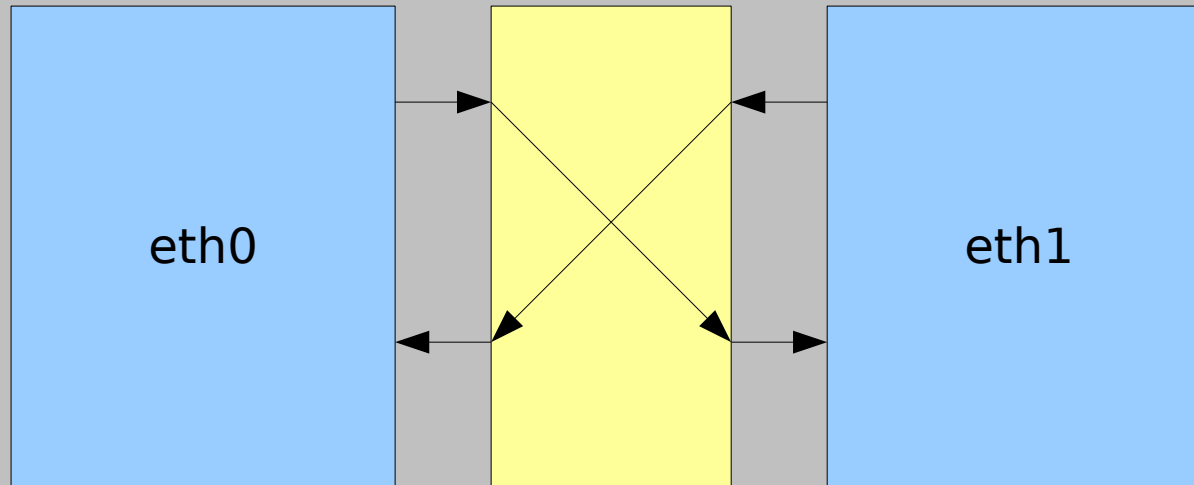


```
let eth0 = make(<ethernet-interface>, name: "eth0");
let filter = make(<frame-filter>, frame-filter: "arp");
connect(eth0, filter);
connect(filter, make(<summary-printer>));
toplevel(eth0);
```

```
hannes@localhost:~
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.97 tell 192.168.1.35
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.98 tell 192.168.1.35
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.99 tell 192.168.1.35
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.100 tell 192.168.1.35
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.101 tell 192.168.1.35
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.102 tell 192.168.1.35
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.103 tell 192.168.1.35
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.104 tell 192.168.1.35
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.105 tell 192.168.1.35
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.106 tell 192.168.1.35
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.107 tell 192.168.1.35
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.108 tell 192.168.1.35
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.109 tell 192.168.1.35
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.110 tell 192.168.1.35
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.111 tell 192.168.1.35
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.112 tell 192.168.1.35
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.113 tell 192.168.1.35
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.1 tell 192.168.1.35
ETH 00:13:49:06:41:4E -> 00:0D:60:B1:E5:C8/ARP 192.168.1.1 IS-AT 00:13:49:06:41:4E
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.2 tell 192.168.1.35
ETH 00:50:8B:4C:85:3C -> 00:0D:60:B1:E5:C8/ARP 192.168.1.2 IS-AT 00:50:8B:4C:85:3C
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.3 tell 192.168.1.35
ETH 00:0D:60:B1:E5:C8 -> FF:FF:FF:FF:FF:FF/ARP WHO-HAS 192.168.1.5 tell 192.168.1.35
```



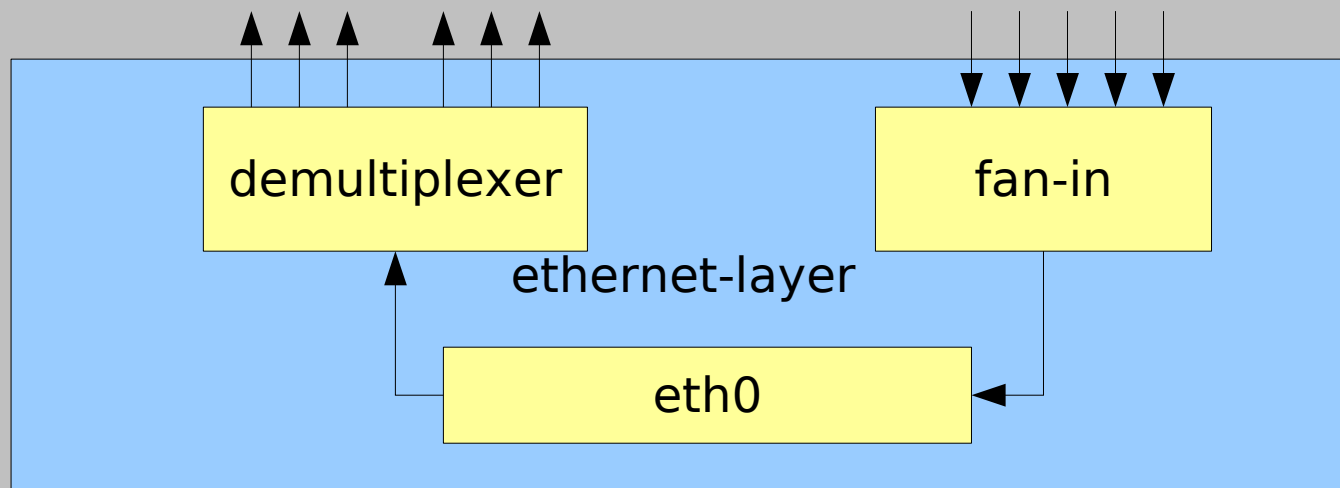
Example: bridge



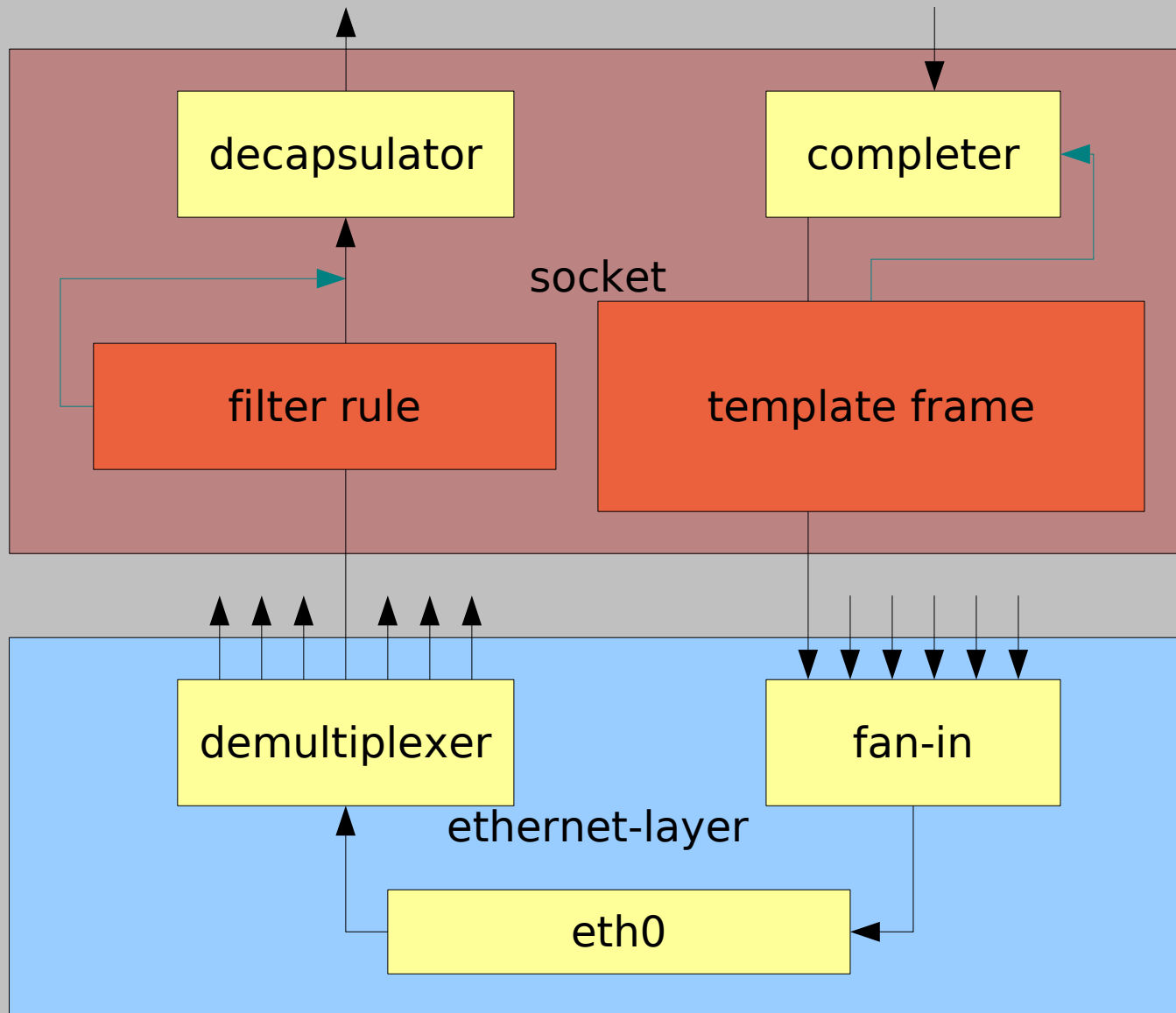
```
let eth0 = make(<ethernet-interface>, name: "eth0");  
let eth1 = make(<ethernet-interface>, name: "eth1");  
connect(eth0, eth1);  
connect(eth1, eth0);  
make(<thread>, function: curry(toplevel, eth0));  
make(<thread>, function: curry(toplevel, eth1));
```



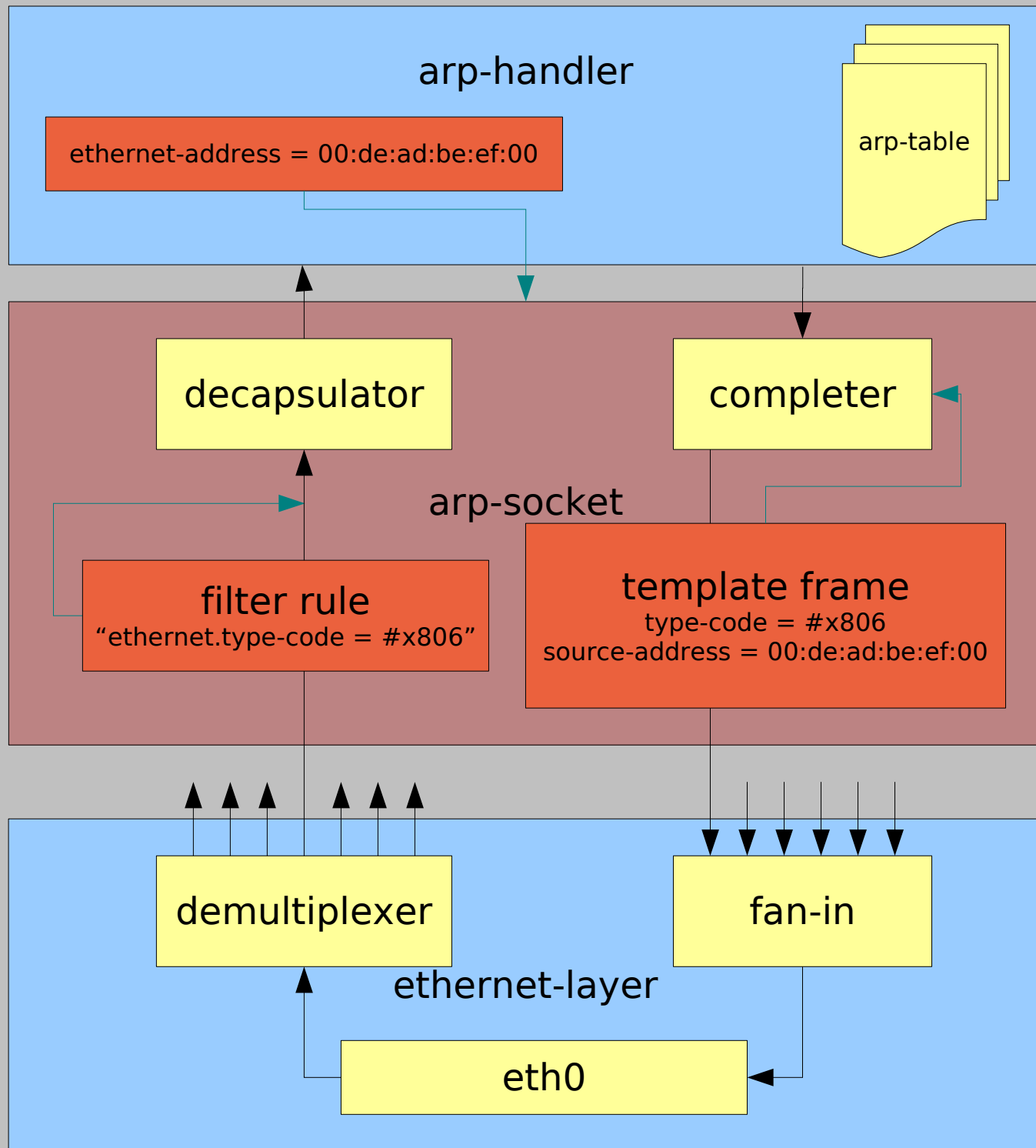
Layering – ethernet layer



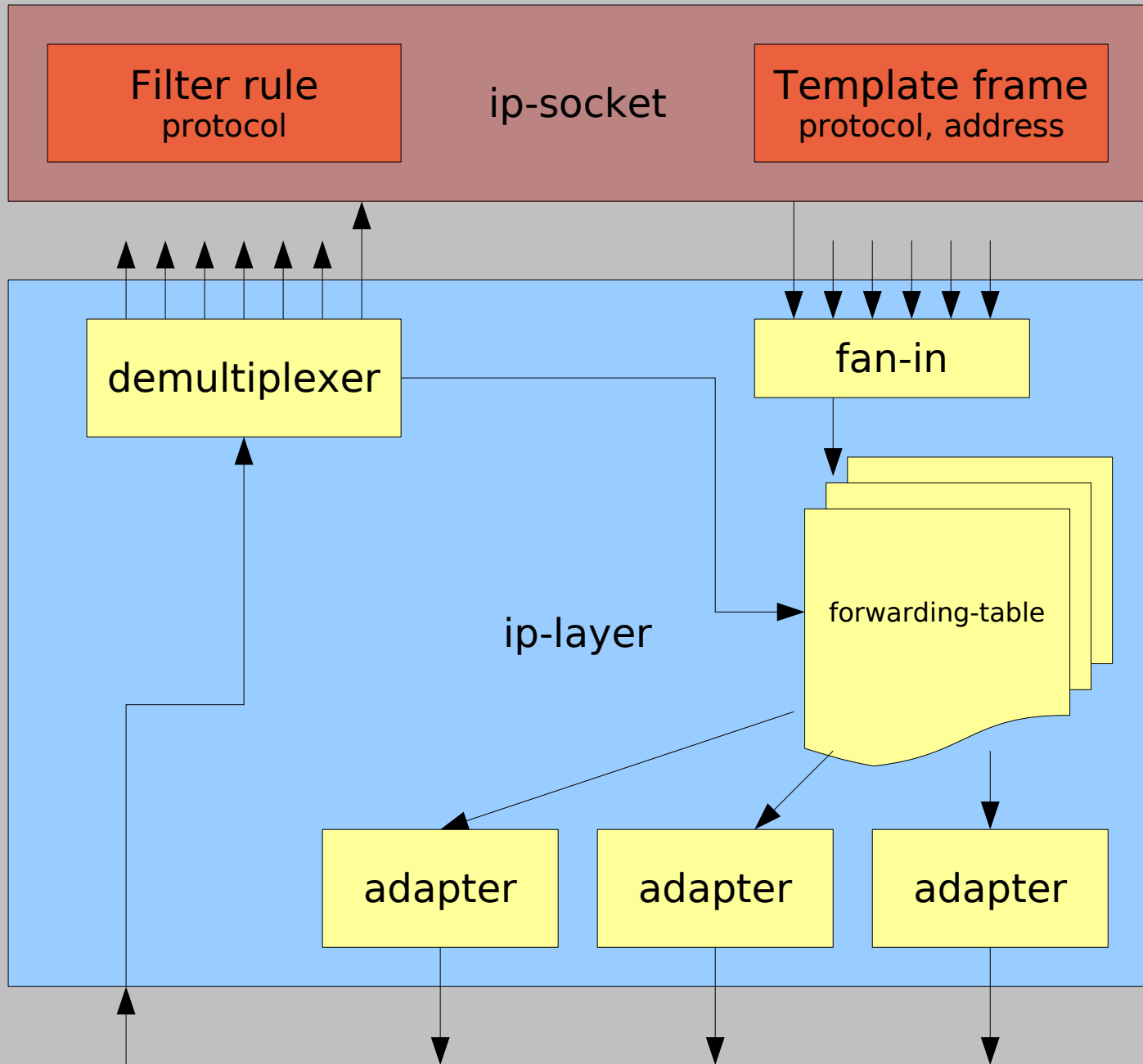
Layering - Socket



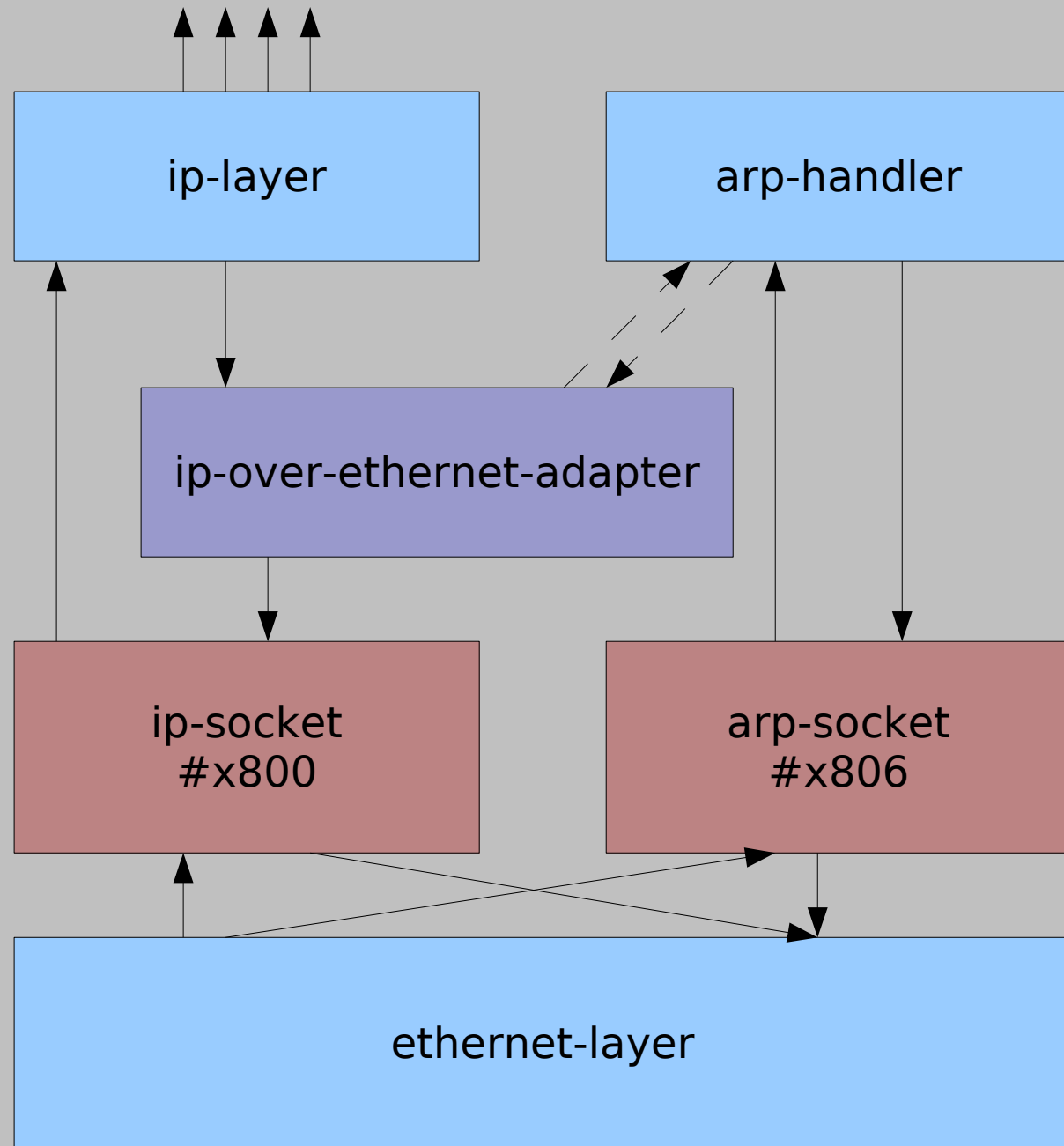
ARP



IP



IP-over-ethernet Adapter



Conclusion

- Software Architecture is Software Security
- IP Stack without remote exploits



Links

- Dylan Website <http://www.opendylan.org>
- Software Security is Software Reliability
<http://doi.acm.org/10.1145/1132469.1132502>
- Click <http://www.read.cs.ucla.edu/click/>
- Scapy <http://www.secdev.org/projects/scapy/>
- Conduit+: A Framework for Network Protocol Software
 - Hueni, Johnson, Engel, OOPSLA '95

