

Packetizer Framework

Hannes Mehnert

GPN5, 10. Juli 2006, Karlsruhe

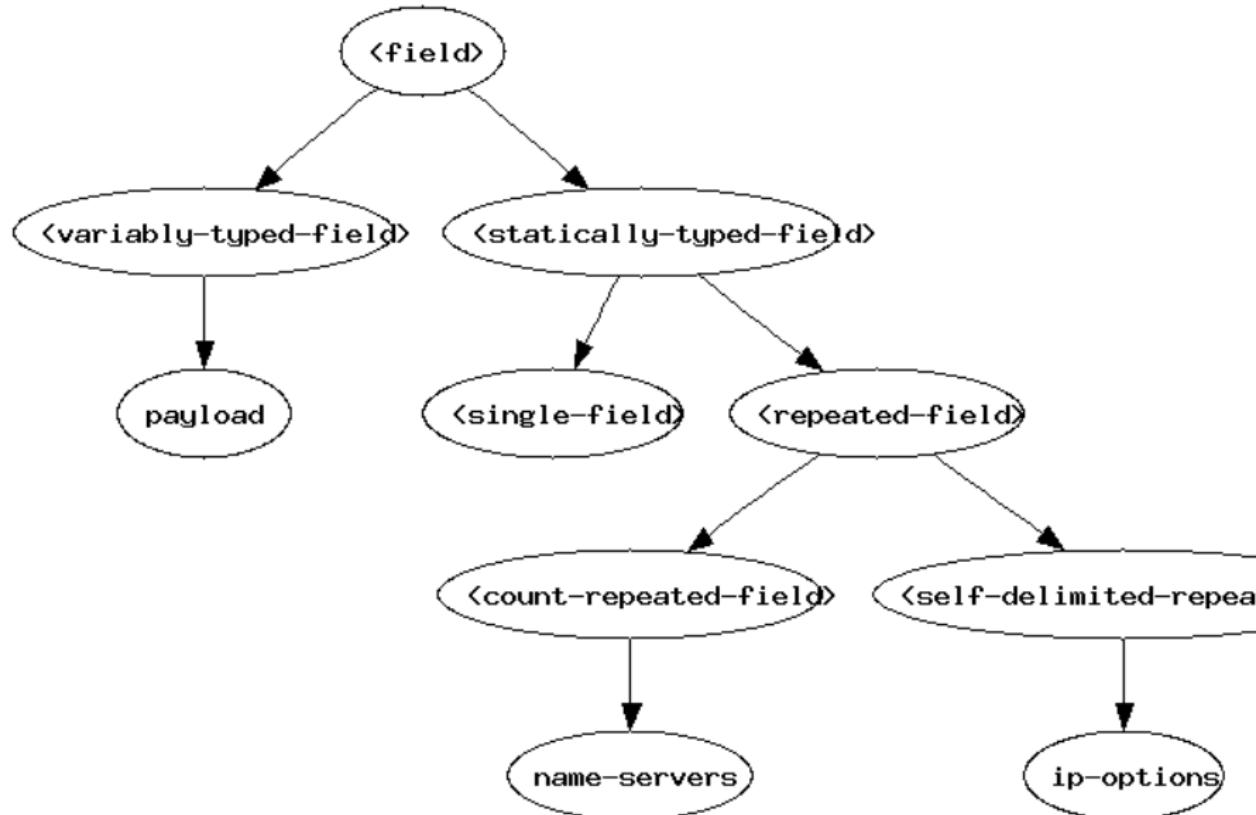
Packetizer

- ▶ inspired by scapy
- ▶ goal: domain specific language for describing byte-protocols
- ▶ examples: network protocols (IP, TCP, ethernet,...), filesystems, elf-header
- ▶ describe protocol once, get a high-performance, secure (bounds checked) parser and generator
- ▶ use as little C code as possible (too many bugs ;)

Example protocol definition

```
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;
```

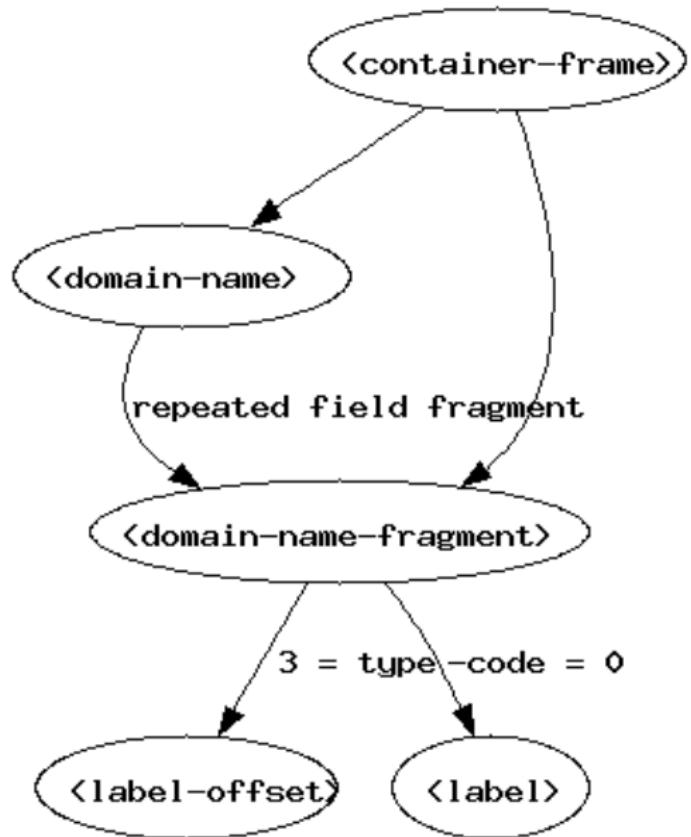
fields



frames

- ▶ leaf frames: <mac-address>, <ipv4-address>, <13bit-unsigned-integer>
- ▶ container frames: <dns-header>, <arp-frame>
- ▶ header frames: <ipv4-frame>, <ethernet-frame>

dns-frames



parse and assemble functions

- ▶ parse-frame (<frame-type>, <byte-vector>) => (<frame>)
- ▶ assemble-frame (<frame>) => (<byte-vector>)

<mac-address>

```
define n-byte-vector(mac-address, 6) end;

define method read-frame
  (type == <mac-address>, string :: <string>) => (res)
    let fields = split(string, ':');
    make(<mac-address>,
      data: map-as(<byte-vector>,
                    rcurry(string-to-integer, base: 16),
                    fields));
  end;

define method as
  (class == <string>, frame :: <mac-address>) => (string :: <string>);
  reduce1(method(a, b) concatenate(a, ":", b) end,
          map-as(<stretchy-vector>,
                 rcurry(integer-to-string, base: 16, size: 2),
                 frame.data))
  end;
```

<ethernet-frame> protocol definition

```
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;
```

protocol definition generates

- ▶ class definitions
 - ▶ <ethernet-frame>
 - ▶ <unparsed-ethernet-frame>
 - ▶ <ethernet-frame-cache>
 - ▶ <decoded-ethernet-frame>
- ▶ for each field, getters and setters are generated
 - ▶ destination-address
 - ▶ source-address
 - ▶ type-code
 - ▶ payload

dynamic start

```
define protocol dynamic-test (header-frame)
    field length :: <unsigned-byte>;
    field payload :: <raw-frame>,
        start: frame.length * 8;
end;
```

self-delimited repeated field

```
define protocol repeated-test (container-frame)
  repeated field bar :: <unsigned-byte>,
    reached-end?: method (frame) frame = 0 end;
end;
```

externally delimited repeated field

```
define protocol repeated-test (container-frame)
    field length :: <unsigned-byte>;
    repeated field bar :: <unsigned-byte>;
    field after :: <unsigned-byte>,
        start: frame.length * 8;
end;
```

count repeated field

```
define protocol count-repeated-test (container-frame)
  field count :: <unsigned-byte>;
  repeated field fragments :: <unsigned-byte>|,
    count: frame.count;
end;
```

default values

```
define protocol foo (container-frame)
    field foobar :: <unsigned-byte> = 23;
end;
```

fixup functions

```
define protocol foo (header-frame)
    field length :: <unsigned-byte>,
        fixup: byte-offset(frame-size(frame.payload));
    field payload :: <raw-frame>,
        length: frame.length * 8;
end;
```

<domain-name>

```
define protocol domain-name (container-frame)
    repeated field fragment :: <domain-name-fragment>,
        reached-end?: method(frame :: <domain-name-fragment>)
            frame.type-code = 3 | frame.length = 0
        end;
    end;

define protocol domain-name-fragment (container-frame)
    field type-code :: <2bit-unsigned-integer>;
end;

define protocol label-offset (domain-name-fragment)
    field offset :: <14bit-unsigned-integer>;
end;

define protocol label (domain-name-fragment)
    field length :: <6bit-unsigned-integer>;
    field raw-data :: <externally-delimited-string>,
        length: frame.length * 8;
end;
```

custom parse-frame for <domain-name>

```
define method parse-frame
  (frame-type == <domain-name-fragment>,
   packet :: <byte-sequence>,
   #key start :: <integer> = 0,
   parent :: false-or(<container-frame>) = #f)
=> (value :: <domain-name-fragment>,
     next-unparsed :: false-or(<integer>))
let domain-name
  = make(unparsed-class(<domain-name-fragment>),
         packet: subsequence(packet, start: byte-offset(start)));
let label-frame-type
  = select (domain-name.type-code)
    0 => <label>;
    3 => <label-offset>;
    otherwise => signal(make(<malformed-packet-error>))
  end;
  parse-frame(label-frame-type, packet, start: start, parent: parent);
end;
```

filter language

- ▶ inspired by ethereal and tcpdump
- ▶ operators: & | ~
- ▶ rules:
 - ▶ presence of frame-type ("ipv4", "tcp", "dns")
 - ▶ value of field ("ipv4.destination-address = 23.23.23.1")
- ▶ "(udp.source-port = 53) | (udp.destination-port = 53)"

flow & network-flow

- ▶ inspired by click and netgraph
- ▶ nodes have push or pull in- and outputs
- ▶ decapsulator, demultiplexer, ethernet-interface,
pcap-file-reader/writer, summary-printer, verbose-printer,
fan-in, fan-out, ...

pcap file reader

```
define class <pcap-file-reader> (<single-push-output-node>)
  slot file-stream :: <stream>, init-keyword: stream:;
end;

define method toplevel (reader :: <pcap-file-reader>)
  let file = as(<byte-vector>,
                stream-contents(reader.file-stream));
  let pcap-file = parse-frame(<pcap-file>, file);
  for(frame in pcap-file.packets)
    push-data(reader.the-output, frame.payload)
  end
end;
```

simple sniffer

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

sniffer & gui-sniffer

- ▶ inspired by tcpdump and ethereal
- ▶ support for reading and writing pcap files
- ▶ listen on raw network interfaces

proof of concept implementations

- ▶ traffic accounting
- ▶ icmp responder
- ▶ connection tracking
- ▶ filtering bridge

roadmap

- ▶ protocol layer implementation
- ▶ support for more protocols
- ▶ configuration management
- ▶ forwarding engine
- ▶ routing engine
- ▶ shell (eval)
- ▶ full operating system

links

- ▶ Source <http://www.opendylan.org/cgi-bin/viewcvs.cgi/trunk/libraries/>
- ▶ or via subversion `svn co svn://svn.gwydiondylan.org/scm/svn/dylan/trunk/libraries`
- ▶ Genera (lisp machine) defstorage macro
- ▶ Click <http://www.read.cs.ucla.edu/click/>
- ▶ Netgraph <http://www.freebsd.org/cgi/man.cgi?query=netgraph&sektion=4>
- ▶ Scapy <http://www.secdev.org/projects/scapy/>
- ▶ Ethereal <http://www.ethereal.com>
- ▶ tcpdump <http://www.tcpdump.org>