Spicy: A Unified Deep Packet Inspection Framework Dissecting All Your Data

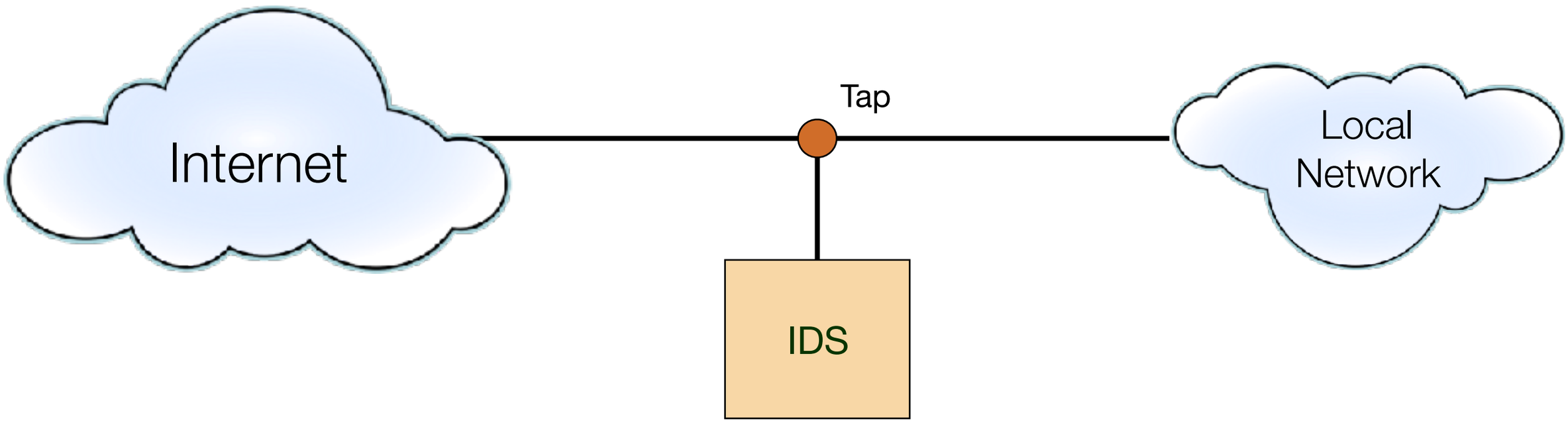
Robin Sommer

International Computer Science Institute, &
Corelight, Inc.

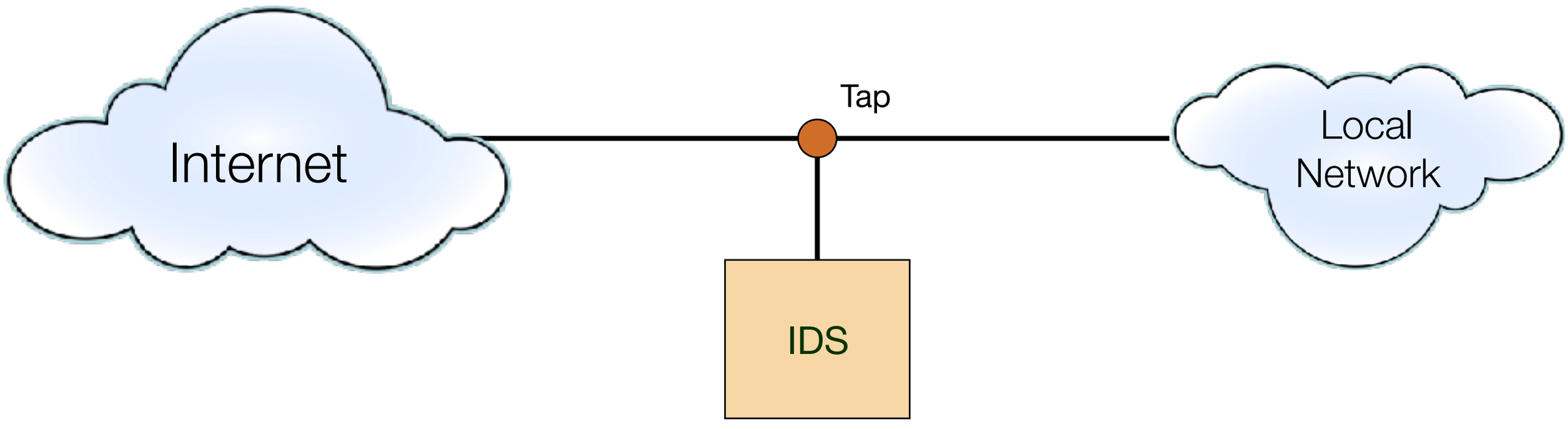
robin@icsi.berkeley.edu
robin@corelight.io

http://www.icir.org/robin

Deep Packet Inspection



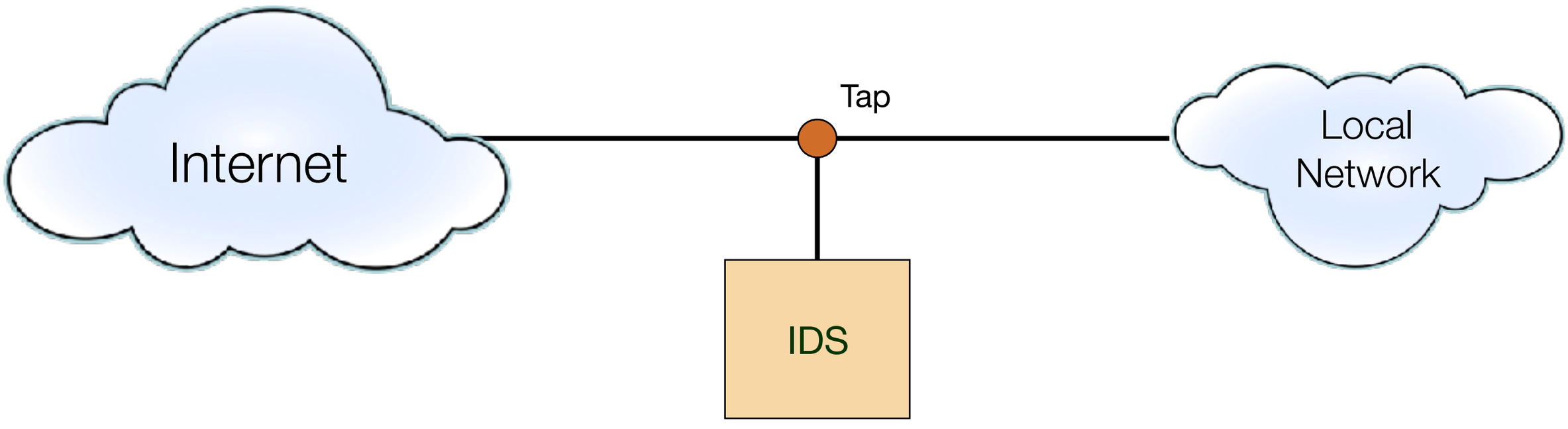
Deep Packet Inspection



Example: Finding downloads of known malware.

1. Find and parse all Web traffic.
2. Find and extract binaries.
3. Compute hash and compare with database.
4. Report, and potentially kill, if found.

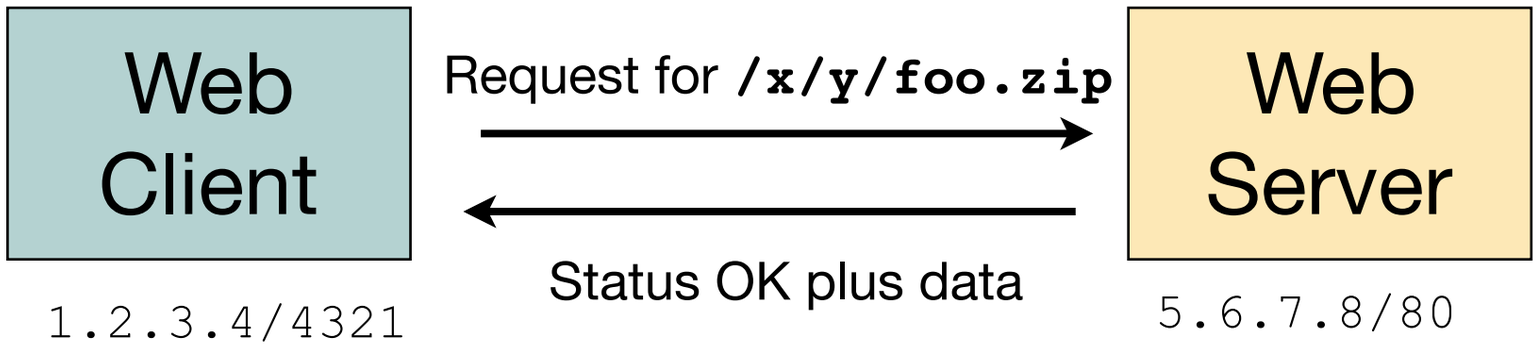
Deep Packet Inspection



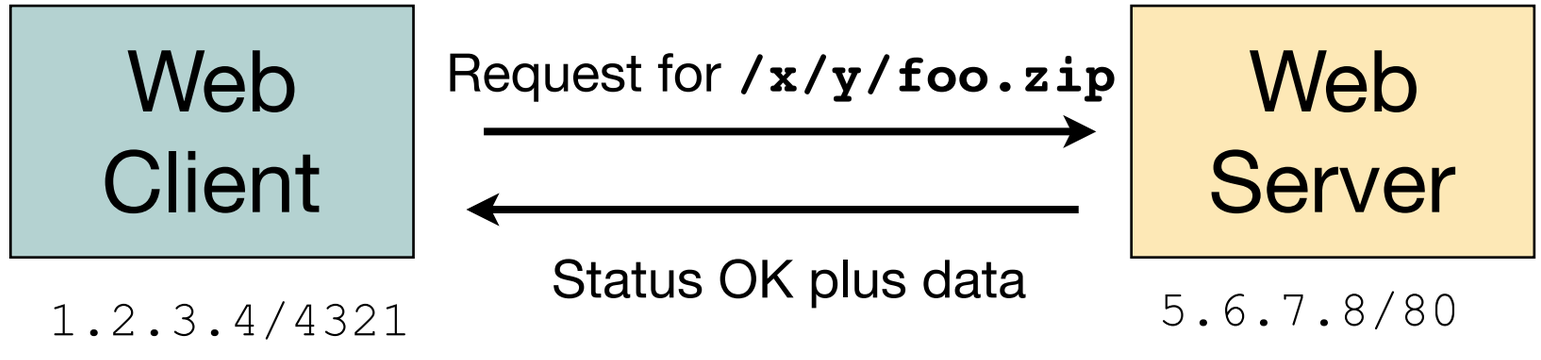
Example: Finding downloads of known malware.

1. Find and parse all Web traffic.
2. Find and extract binaries.
3. Compute hash and compare with database.
4. Report, and potentially kill, if found.

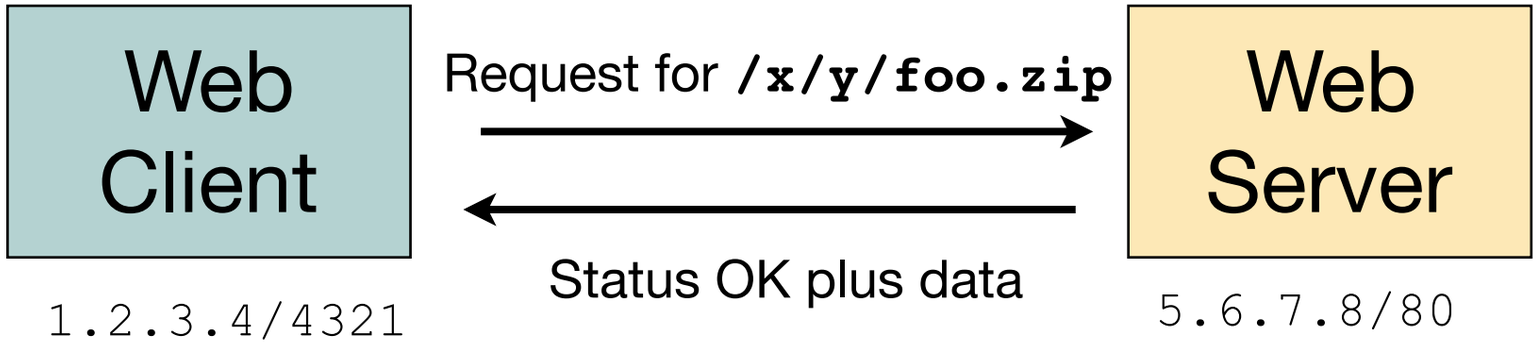
Protocol Parsing



Protocol Parsing



Protocol Parsing



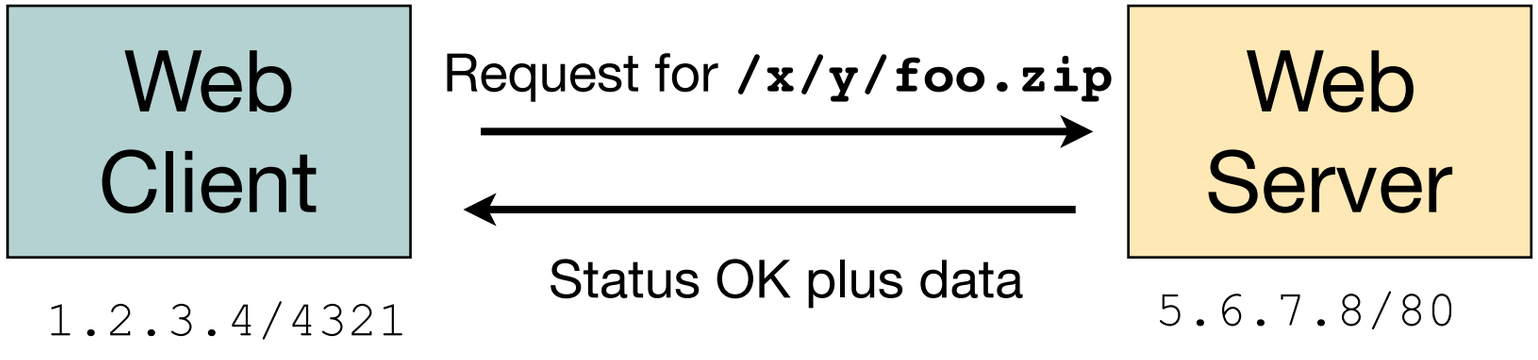
→ TCP connection established

```
GET /x/y/foo.zip HTTP/1.1 …
```

TCP stream reassembly for originator

→ Request for /x/y/foo.zip, protocol version 1.1, HTTP headers

Protocol Parsing



→ TCP connection established

```
GET /x/y/foo.zip HTTP/1.1 …
```

TCP stream reassembly for originator

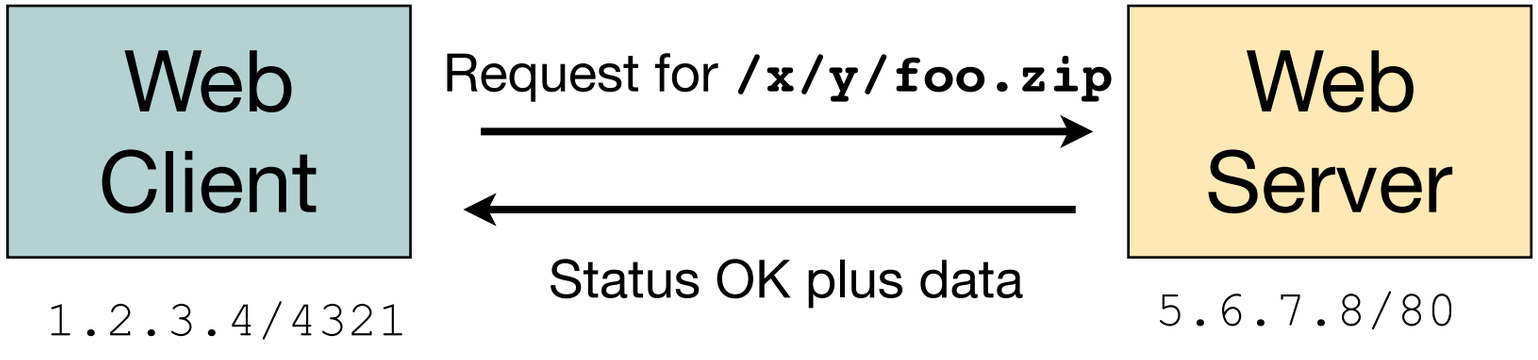
→ Request for /x/y/foo.zip, protocol version 1.1, HTTP headers

```
200 OK …
```

TCP stream reassembly for responder

→ Reply with page content for further analysis (e.g., hash; unpack & parse files)

Protocol Parsing



↳ TCP connection established

```
GET /x/y/foo.zip HTTP/1.1 …
```

TCP stream reassembly for originator

↳ Request for /x/y/foo.zip, protocol version 1.1, HTTP headers

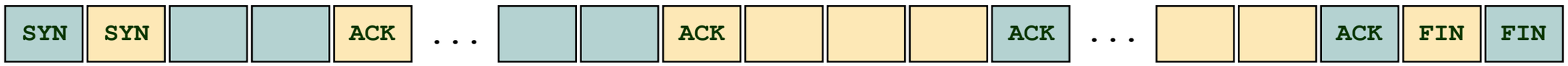
```
200 OK …
```

TCP stream reassembly for responder

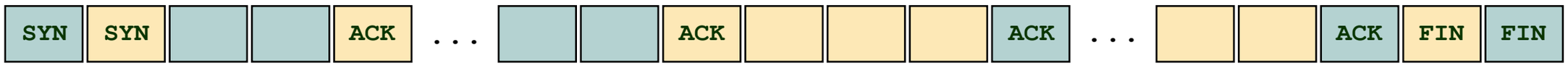
↳ Reply with page content for further analysis (e.g., hash; unpack & parse files)

↳ TCP connection tear down

Parsing Is Hard



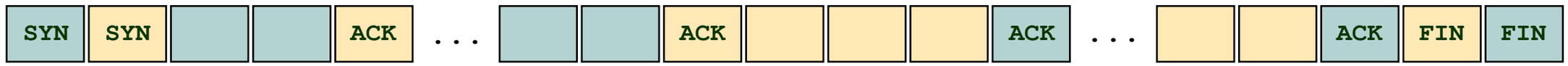
Parsing Is Hard



Must be robust

- Lots of “crud” in real-world networks
- Cannot trust input

Parsing Is Hard



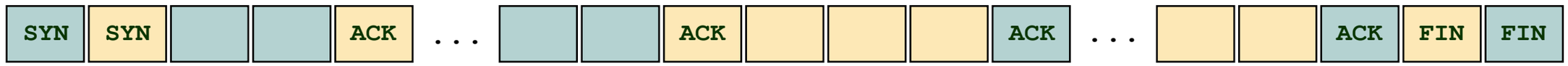
Must be robust

- Lots of “crud” in real-world networks
- Cannot trust input

Must be efficient

- 100,000s of concurrent connections
- Incremental processing for low latency & memory usage

Parsing Is Hard



Must be robust

- Lots of “crud” in real-world networks
- Cannot trust input

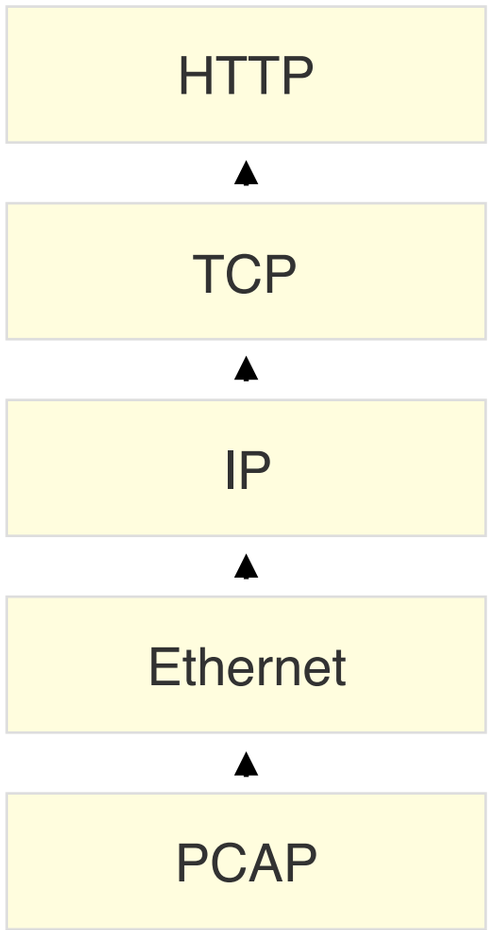
Must be efficient

- 100,000s of concurrent connections
- Incremental processing for low latency & memory usage

Must be complete

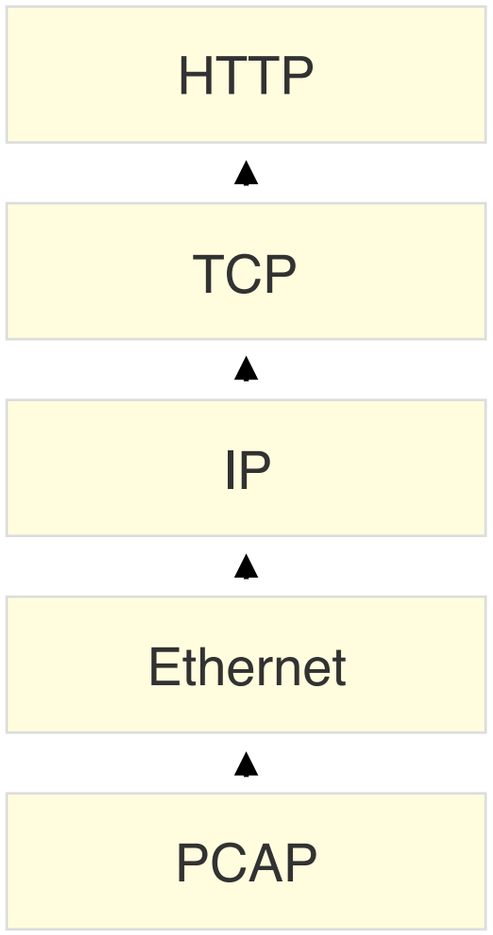
- Leaving out parts of the protocol opens evasion opportunities
- Protocols can be really complex (SMB …)

There are a lot of protocols out there …



Even a simple case involves 5 protocols

There are a lot of protocols out there …

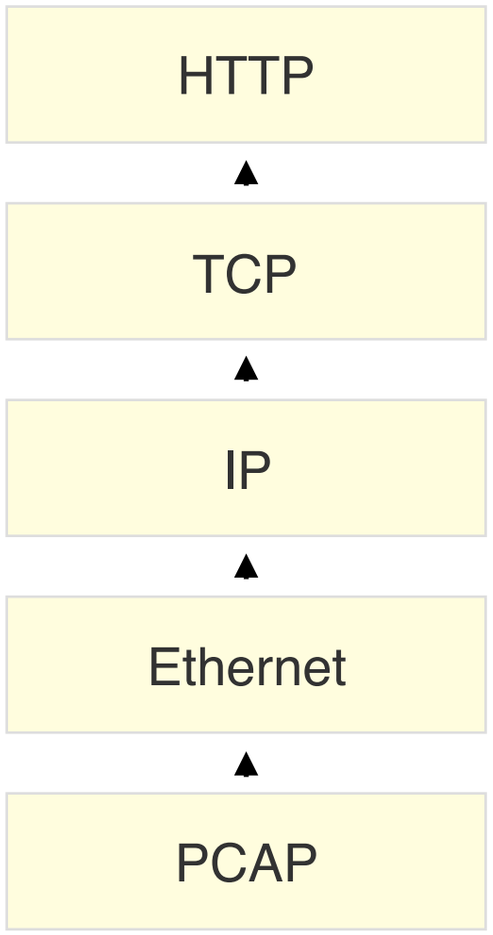


Even a simple case involves 5 protocols

A few popular protocols account for the bulk of traffic in most environments

(e.g., TCP/IP, HTTP, TLS, DNS, SMTP, IMAP)

There are a lot of protocols out there …

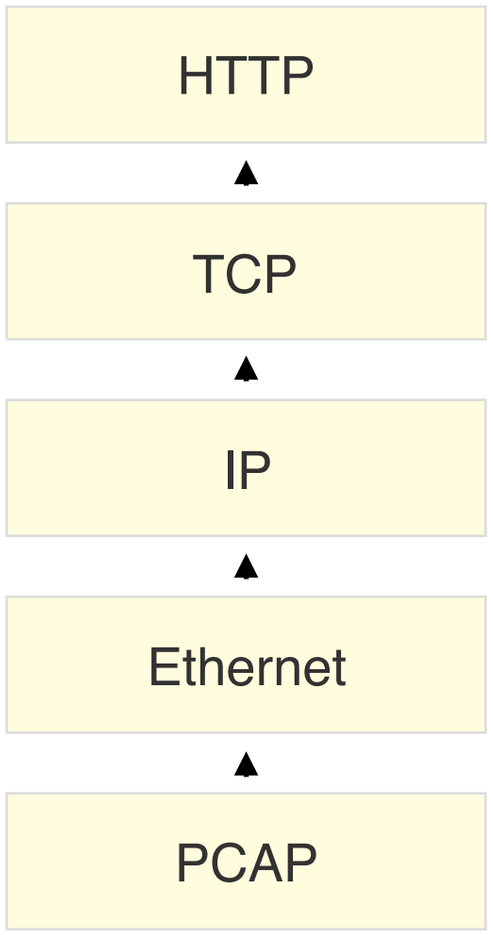


Even a simple case involves 5 protocols

A few popular protocols account for the bulk of traffic in most environments
(e.g., TCP/IP, HTTP, TLS, DNS, SMTP, IMAP)

Long tail of further protocols, often environment-specific
(e.g., SMB, Modbus, BACnet, more L2)

There are a lot of protocols out there …



Even a simple case involves 5 protocols

A few popular protocols account for the bulk of traffic in most environments

(e.g., TCP/IP, HTTP, TLS, DNS, SMTP, IMAP)

Long tail of further protocols, often environment-specific

(e.g., SMB, Modbus, BACnet, more L2)

File formats amplify the challenge

Example: Bro 2.5

AYIYA	Kerberos	
BitTorrent	Login	SIP
DCE_RPC	Modbus	SMTP
DHCP	MySQL	SNMP
DNP3	NCP	SOCKS
DNS	NFS	SSH
DTLS	NTP	SSL
FTP	NetBIOS	Syslog
Finger	PE	TCP
GTPv1	POP3	Telnet
Gnutella	Portmapper	Teredo
HTTP	Radius	UDP
ICMP	RDP	X509
IPv4/6	Rlogin	ZIP
IRC	Rsh	
Ident	SMB	

A Tale of Three Open-Source IDS

A Tale of Three Open-Source IDS

Shared parsers?

None.

Every DPI application rewrites its parsers — usually in C/C++!

Opportunity: Provide Platform for Parsers

Opportunity: Provide Platform for Parsers

Protocols leverage a rather small set of patterns

- Readable line-based formats for text protocols
- Static “prototocol data units” (PDU) for binary protocols
- Request/response structure
- Common sub-formats (HTTP/MIME/ASN.1)
- Fragmentation (even at app layer!)

Opportunity: Provide Platform for Parsers

Protocols leverage a rather small set of patterns

- Readable line-based formats for text protocols
- Static “prototocol data units” (PDU) for binary protocols
- Request/response structure
- Common sub-formats (HTTP/MIME/ASN.1)
- Fragmentation (even at app layer!)

But: Potpourri of protocols remains diverse still

- Every protocol does *something* different

Opportunity: Provide Platform for Parsers

Protocols leverage a rather small set of patterns

- Readable line-based formats for text protocols
- Static “prototocol data units” (PDU) for binary protocols
- Request/response structure
- Common sub-formats (HTTP/MIME/ASN.1)
- Fragmentation (even at app layer!)

But: Potpourri of protocols remains diverse still

- Every protocol does *something* different

Can we leverage similarities, while remaining flexible?

Opportunity: Provide Platform for Parsers

Protocols leverage a rather small set of patterns

- Readable line-based formats for text protocols
- Static “prototocol data units” (PDU) for binary protocols
- Request/response structure
- Common sub-formats (HTTP/MIME/ASN.1)
- Fragmentation (even at app layer!)

But: Potpourri of protocols remains diverse still

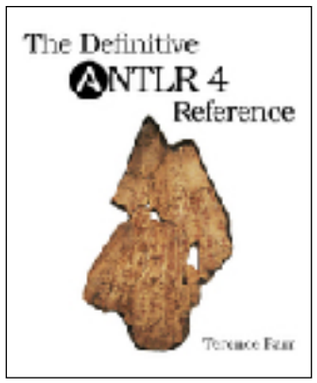
- Every protocol does *something* different

Can we leverage similarities, while remaining flexible?

Can we reuse code across applications?

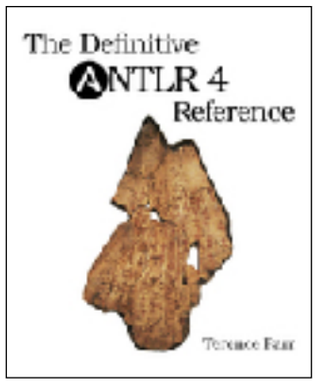
Meanwhile, in another domain …

There are powerful tools for implementing parsers for programming languages.



Meanwhile, in another domain …

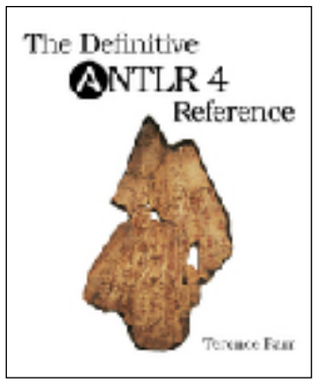
There are powerful tools for implementing parsers for programming languages.




```
exp: NUM               { $$ = $1;         }
   | exp '+' exp       { $$ = $1 + $2; }
   | exp '-' exp       { $$ = $1 - $2; }
   | exp '*' exp       { $$ = $1 * $2; }
   | exp '/' exp       { $$ = $1 / $2; }
```

Meanwhile, in another domain …

There are powerful tools for implementing parsers for programming languages.




```
exp: NUM                { $$ = $1;         }
     | exp '+' exp      { $$ = $1 + $2; }
     | exp '-' exp      { $$ = $1 - $2; }
     | exp '*' exp      { $$ = $1 * $2; }
     | exp '/' exp      { $$ = $1 / $2; }
```

Yacc → **yyparse()** → Host Application

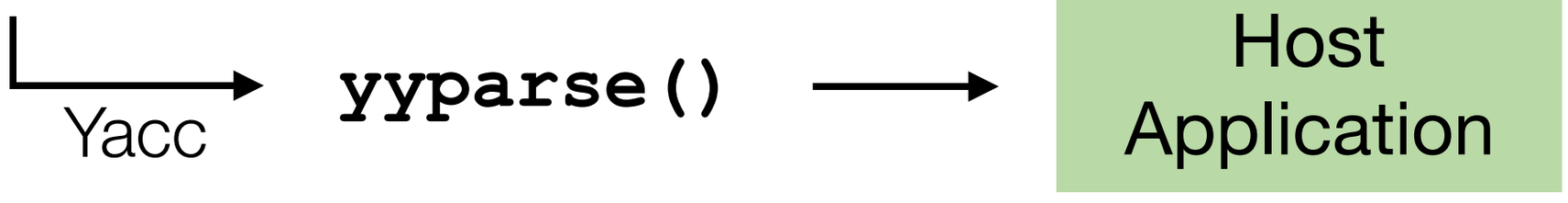
Meanwhile, in another domain …

There are powerful tools for implementing parsers for programming languages.

These parsers aren't suitable for DPI, unfortunately.

No support for concurrent, incremental processing

No support for domain-specific idioms



Domain-specific Parser Generation

binpac: A yacc for Writing Application Protocol Parsers

Ruoming Pang*
Google, Inc.

Vern Paxson
International Computer Science Institute

Robin Sommer
International Computer Science Institute

Larry Peterson
Princeton University

IMC 2006

Domain-specific Parser Generation

binpac: A yacc for Writing Application Protocol Parsers

Ruoming Pang*
Google, Inc.

Vern Paxson
International Computer Science Institute

Robin Sommer
International Computer Science Institute

Larry Peterson
Princeton University

IMC 2006

```
type ClientHello(rec: HandshakeRecord) = record {
    client_version: uint16;
    gmt_unix_time : uint32;
    random_bytes  : bytestring &length = 28;
    session_len   : uint8;
    session_id    : uint8[session_len];
    dtls_cookie   : case client_version of {
        DTLSv10, DTLSv12 -> cookie : ClientHelloCookie(rec);
        default          -> nothing: bytestring &length=0;
    };
    […]
}
```

TLS v3 Client Hello (Source: Bro's TLS analyzer)

Domain-specific Parser Generation

binpac: A yacc for Writing Application Protocol Parsers

Ruoming Pang*
Google, Inc.

Vern Paxson
International Computer Science Institute

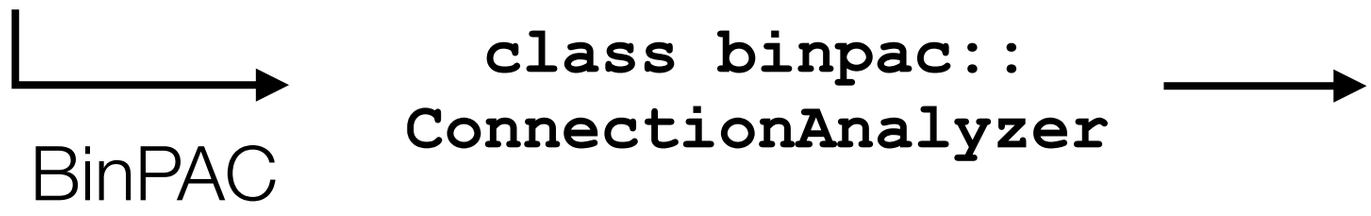
Robin Sommer
International Computer Science Institute

Larry Peterson
Princeton University

IMC 2006

```
type ClientHello(rec: HandshakeRecord) = record {
    client_version: uint16;
    gmt_unix_time : uint32;
    random_bytes  : bytestring &length = 28;
    session_len   : uint8;
    session_id    : uint8[session_len];
    dtls_cookie   : case client_version of {
        DTLSv10, DTLSv12 -> cookie : ClientHelloCookie(rec);
        default          -> nothing: bytestring &length=0;
    };
    […]
}
```

TLS v3 Client Hello (Source: Bro's TLS analyzer)



Domain-specific Parser Generation

binpac: A yacc for Writing Application Protocol Parsers

Ruoming Pang*
Google, Inc.

Vern Paxson
International Computer Science Institute

Robin Sommer
International Computer Science Institute

Larry Peterson
Princeton University

IMC 2006

BinPAC works, but solves the problem only partially.

Remains limited to syntax, cannot express logic.

Still needs custom C++ for logic & integration.

Remains limited to app protocols & connection structure.

Lacks support for higher-level idioms.

BinPAC → `class binpac::ConnectionAnalyzer` → Host Application

New Framework: Spicy

Integrates experience from many years of writing parsers manually and with BinPAC.

corelight

New Framework: Spicy

Integrates experience from many years of writing parsers manually and with BinPAC.

Expresses both syntax and logic

Supports protocols and file formats

Facilitates composition and reuse

Supports error handling and recovery

Just-in-time compilation via LLVM

Spicy Example: Parsing SMTP Banners

```
220 mx.foo.com ESMTP Postfix
```

Spicy Example: Parsing SMTP Banners

```
220 mx.foo.com ESMTP Postfix
```

```
module SMTP;

export type Greeting = unit {
                : /220 +/;
    domain      : /[^ ]+/;
                : / */;
    protocol: /(E?SMTP)?/;
                : / */;
    software: /[^ ]*/;

    on %done { print self; }
}
```

smtp.spicy

Spicy Example: Parsing SMTP Banners

```
220 mx.foo.com ESMTP Postfix
```

```
module SMTP;

export type Greeting = unit {
                : /220 +/;
    domain      : /[^ ]+/;
                : / */;
    protocol: /(E?SMTP)?/;
                : / */;
    software: /[^ ]*/;

    on %done { print self; }
}
```

smtp.spicy

```
# echo "220 mx.foo.com ESMTP Postfix" | spicy-driver smtp.spicy
<domain=mx.foo.com, protocol=ESMTP, software=Postfix>
```

Host Application API

```
# Compile Spicy code just-in-time (C++)

auto ctx = new spicy::CompilerContext();

atuo llvm_module = ctx->compile(“smtp.spicy”);

auto linked_module = ctx->linkModules("SMTP", llvm_module);

auto jit = ctx->jit(linked_module);

auto parse_func = jit->nativeFunction(“smtp_greeting_parse”)
auto resume_func = jit->nativeFunction(“smtp_greeting_resume”)
```

Host Application API

```
# Compile Spicy code just-in-time (C++)

auto ctx = new spicy::CompilerContext();

atuo llvm_module = ctx->compile(“smtp.spicy”);

auto linked_module = ctx->linkModules("SMTP", llvm_module);

auto jit = ctx->jit(linked_module);

auto parse_func = jit->nativeFunction(“smtp_greeting_parse”)
auto resume_func = jit->nativeFunction(“smtp_greeting_resume”)
```

```
# Feed data into parser (C).

hlt_bytes* data = hlt_bytes_new_from_data(“220 mx.foo.”);
void* cookie = (*parse_func)(data);

hlt_bytes* next = hlt_bytes_new_from_data(“.com ESMTP Postfix”);
hlt_bytes_append(data, next);
cookie = (*resume_func)(cookie);
```

A File Format: Tar

A File Format: Tar

```
module tar;

export type Archive = unit {
   files: list<File>;
        : uint<8>(0x0);
        : bytes &length=511;
};

type File = unit {
   header: Header;
   data  : bytes &length=self.header.size;
         : bytes &length=512-(self.header.size mod 512)
};

type Type = enum {
   REG=0, LNK=1, SYM=2, CHR=3, BLK=4, DIR=5, FIFO=6
};
```

```
type Header = unit {
 name  : bytes &length=100;
 mode  : bytes &length=8;
 uid   : bytes &length=8;
 gid   : bytes &length=8;
 size  : bytes &length=12 &convert=$$.to_uint(8);
 mtime : bytes &length=12 &convert=$$.to_time(8);
 chksum: bytes &length=8  &convert=$$.to_uint(8);
 tflag : bytes &length=1  &convert=$$.to_uint(8);
 lname : bytes &length=100;
       : bytes &length=88; # Skip further fields
 prefix: bytes &length=155;
       : bytes &length=12;.

 var full_path: bytes;

 on %done {
  if ( ! self.tflag ) self.tflag = Type::REG;
  self.full_path = self.prefix + b"/"+ self.name;
 }
```

A File Format: Tar

```
module tar;

export type Archive = unit {
   files: list<File>;
        : uint<8>(0x0);
        : bytes &length=511;
};

type File = unit {
   header: Header;
   data  : bytes &length=self.header.size;
         : bytes &length=512-(self.header.size mod 512)
};

type Type = enum {
   REG=0, LNK=1, SYM=2, CHR=3, BLK=4, DIR=5, FIFO=6
};


type Header = unit {
 name  : bytes &length=100;
 mode  : bytes &length=8;
 uid   : bytes &length=8;
 gid   : bytes &length=8;
 size  : bytes &length=12 &convert=$$.to_uint(8);
 mtime : bytes &length=12 &convert=$$.to_time(8);
 chksum: bytes &length=8  &convert=$$.to_uint(8);
 tflag : bytes &length=1  &convert=$$.to_uint(8);
 lname : bytes &length=100;
       : bytes &length=88; # Skip further fields
 prefix: bytes &length=155;
       : bytes &length=12;.

 var full_path: bytes;

 on %done {
  if ( ! self.tflag ) self.tflag = Type::REG;
  self.full_path = self.prefix + b"/"+ self.name;
 }
```

```
# tar tvf mp.tar
foobar/staff       0 2016-05-15 18:58 mp/
foobar/staff   39548 2016-05-15 18:58 mp/part01.txt
foobar/staff   39503 2016-05-15 18:58 mp/part02.txt*/

# cat print-tar.spicy
module PrintTar;

import tar;

on tar::Archive::%done {
 print self.files;
}

# cat mp.tar | spicy-driver tar.spicy print-tar.spicy
[<header=<name=b"mp/", mode=b"000755", uid=b"000771",
  gid=b"000024", size=0, mtime=2016-05-16T02:58:19Z,
  chksum=5100, tflag=DIR>, data=b"",
   […], full_path=b"mp/">]

[<header=<name=b"mp/part01.txt", mode=b"000644",
 uid=b"000771", gid=b"000024", size=39548,
 mtime=2016-05-16T02:58:19Z, chksum=6351, tflag=REG>,
 data=b"A seashore. Some way out to sea […]",
 […], full_path=b"mp/part01.txt">]


[<header=<name=b"mp/part02.txt", mode=b"000644",
 uid=b"000771", gid=b"000024", size=39503,
 mtime=2016-05-16T02:58:11Z, chksum=6348, tflag=REG>,
 data=b"A man appears on the top of a sand […]",
 […] full_path=b"mp/part02.txt">]
```

Composition: Pipelining Layers

Composition: Pipelining Layers

```
type HTTP::Body = unit(msg: Message, delivery_mode: DeliveryMode) {

  var data: sink;

  on %init {
      # Add parser for body content (e.g., application/x-tar)
      self.data.connect_mime_type(msg.content_type);

      if ( msg.content_encoding == b"gzip" ) {
              self.data.add_filter(Spicy::Filter::GZIP);
 }

  switch ( delivery_mode ) {
    DeliveryMode::EndOfData -> : bytes &eod -> self.data;
    DeliveryMode::Length    -> : bytes &length=msg.content_length -> self.data;
    DeliveryMode::Multipart -> : list<[^\r\n]*\r?\n/> &until($$ == msg.boundary)
                                 foreach { self.data.write($$); }
};
```

Error Recovery

Error Recovery

```
type HTTP::Requests = unit {
    requests: list<Request> &synchronize;
};

type HTTP::Request = unit {
    request: RequestLine;
    message: Message;
};

type HTTP::RequestLine = unit {
    %synchronize-at = /^(GET|POST|HEAD) /;

     method: Token;
           : WhiteSpace;
        uri: Token;
           : WhiteSpace;
           : /HTTP\//;
    version: /[0-9]+\.[0-9]*/;
           : NewLine;
};

type HTTP::Message = unit { ... }
```

corelight

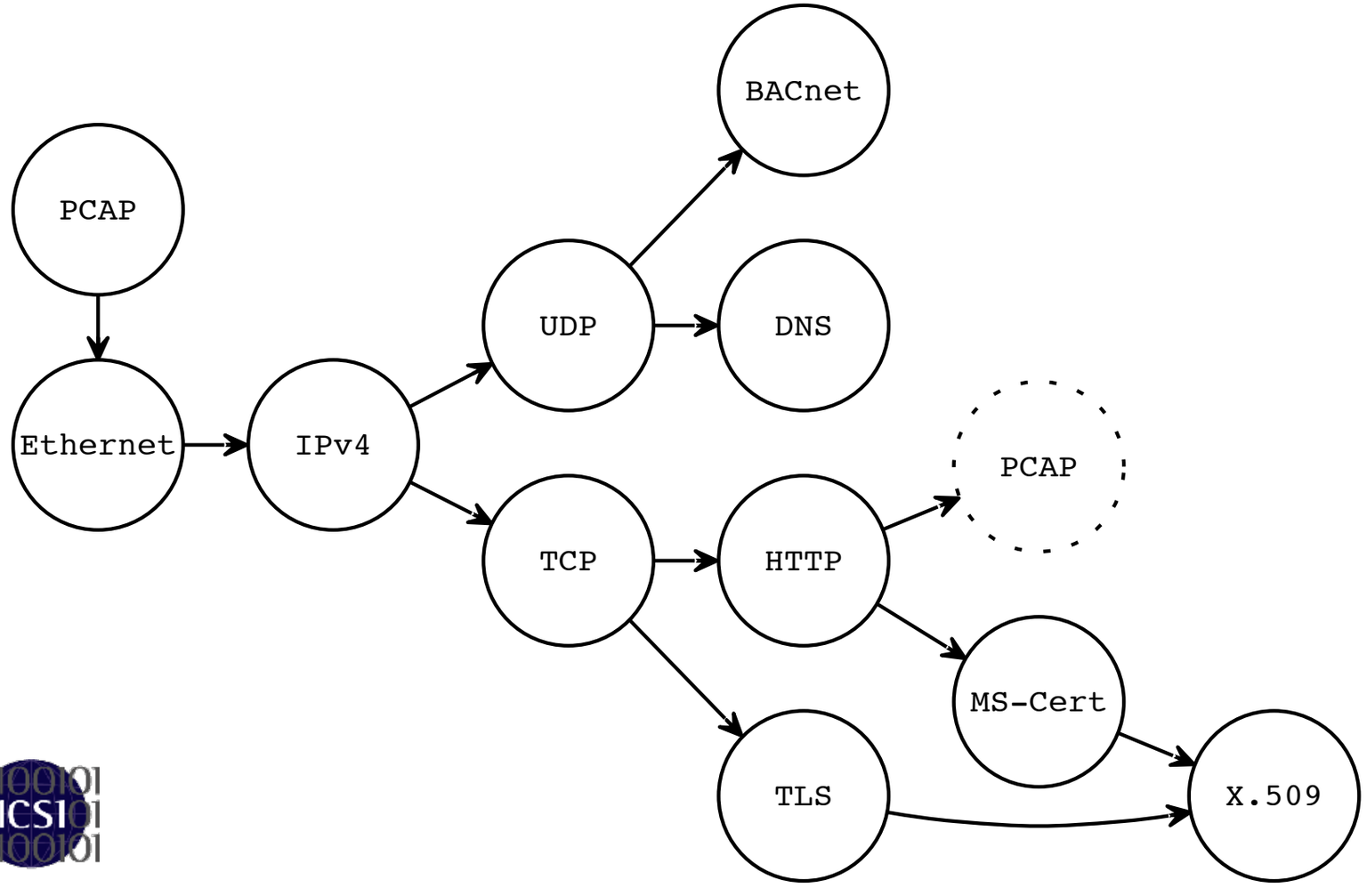
Evaluation: Writing Spicy Parsers

Evaluation: Writing Spicy Parsers

Protocols	BACnet (ASHRAE/ANSI 135), DNS, Ethernet (IEEE 802.3), PCAP, HTTP, IPv4, RTMP (handshake), SSH (banner), SMTP (greeting), SMB2, TCP, TLS, TFTP, UDP
File Formats	ASF (headers & metadata), ASN.1, Gzip (header), ZIP (header), MS Certificate Store, Tar, X.509 certificates

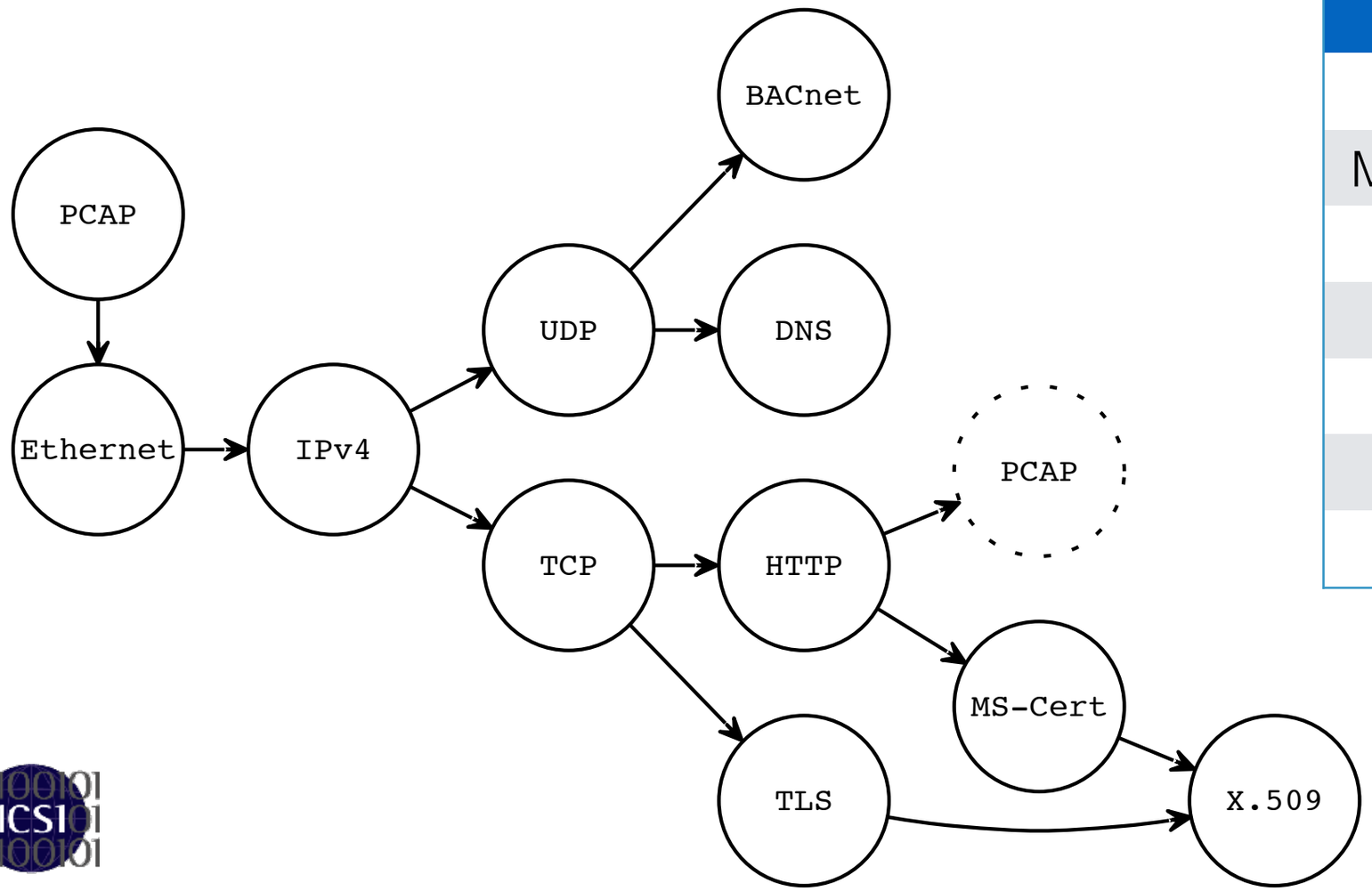
Evaluation: Writing Spicy Parsers

Protocols	BACnet (ASHRAE/ANSI 135), DNS, Ethernet (IEEE 802.3), PCAP, HTTP, IPv4, RTMP (handshake), SSH (banner), SMTP (greeting), SMB2, TCP, TLS, TFTP, UDP
File Formats	ASF (headers & metadata), ASN.1, Gzip (header), ZIP (header), MS Certificate Store, Tar, X.509 certificates



Evaluation: Writing Spicy Parsers

Protocols	BACnet (ASHRAE/ANSI 135), DNS, Ethernet (IEEE 802.3), PCAP, HTTP, IPv4, RTMP (handshake), SSH (banner), SMTP (greeting), SMB2, TCP, TLS, TFTP, UDP
File Formats	ASF (headers & metadata), ASN.1, Gzip (header), ZIP (header), MS Certificate Store, Tar, X.509 certificates



Trace 1
X.509
MS Cert Store
HTTP
TCP
IP
Ethernet
PCAP

Trace 2
X.509
TLS
TCP
IP
Ethernet
PCAP
HTTP
TCP
IP
Ethernet
PCAP

Evaluation: Real-world Performance

Add Spicy plugin for Bro to compare parsing with a native Bro.

Traces: HTTP: 1/25 of Berkeley port 80 traffic.
30GB trace, 52min, 340k messages.

DNS: Full Berkeley port 53 traffic.
1GB trace, 10min, 65M messages.

Evaluation: Real-world Performance

Add Spicy plugin for Bro to compare parsing with a native Bro.

Traces: HTTP: 1/25 of Berkeley port 80 traffic.
30GB trace, 52min, 340k messages.

DNS: Full Berkeley port 53 traffic.
1GB trace, 10min, 65M messages.

Correctness

Spicy captures protocols correctly.

Evaluation: Real-world Performance

Add Spicy plugin for Bro to compare parsing with a native Bro.

Traces: HTTP: 1/25 of Berkeley port 80 traffic.
30GB trace, 52min, 340k messages.

DNS: Full Berkeley port 53 traffic.
1GB trace, 10min, 65M messages.

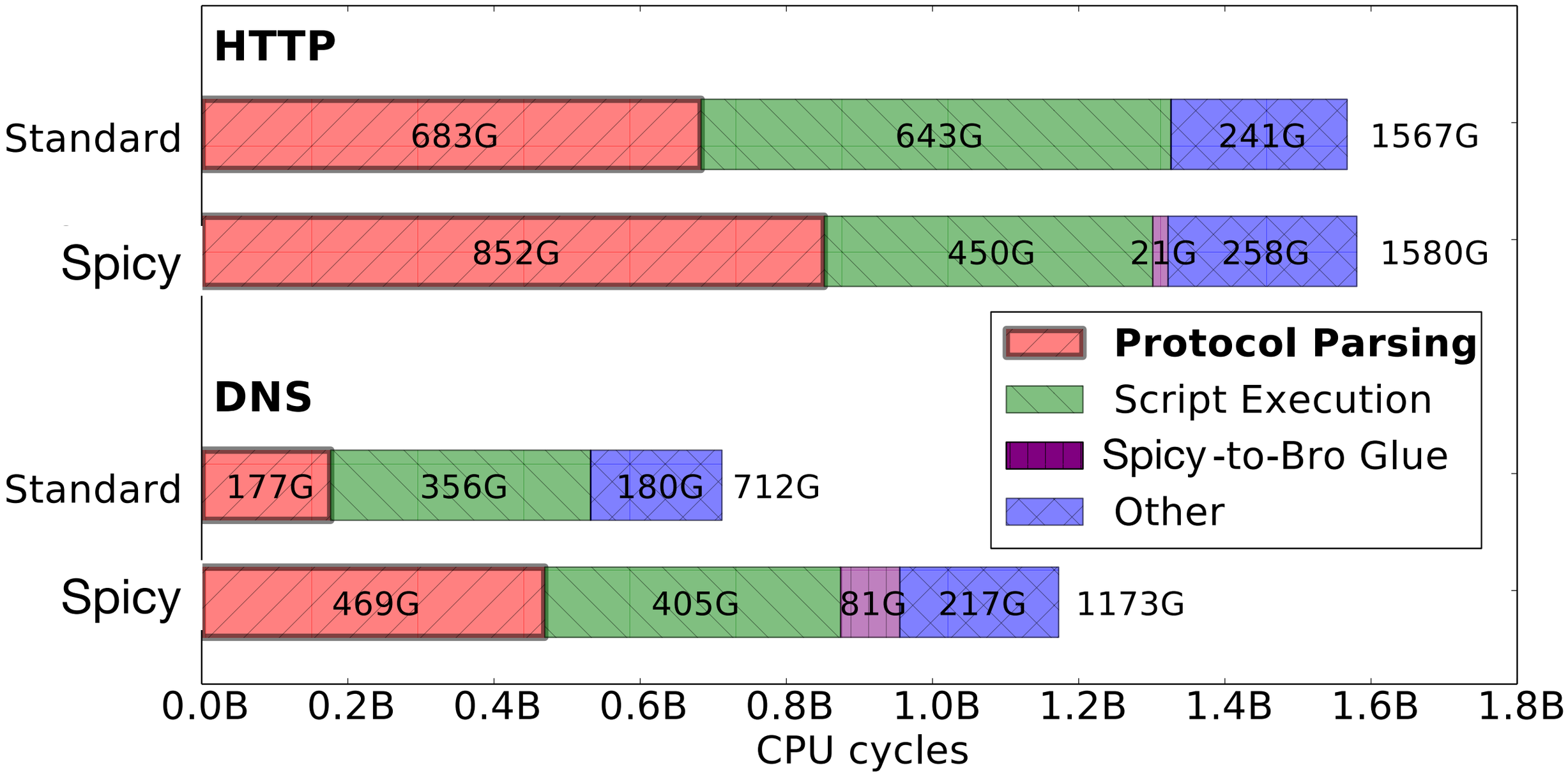
Correctness

Spicy captures protocols correctly.

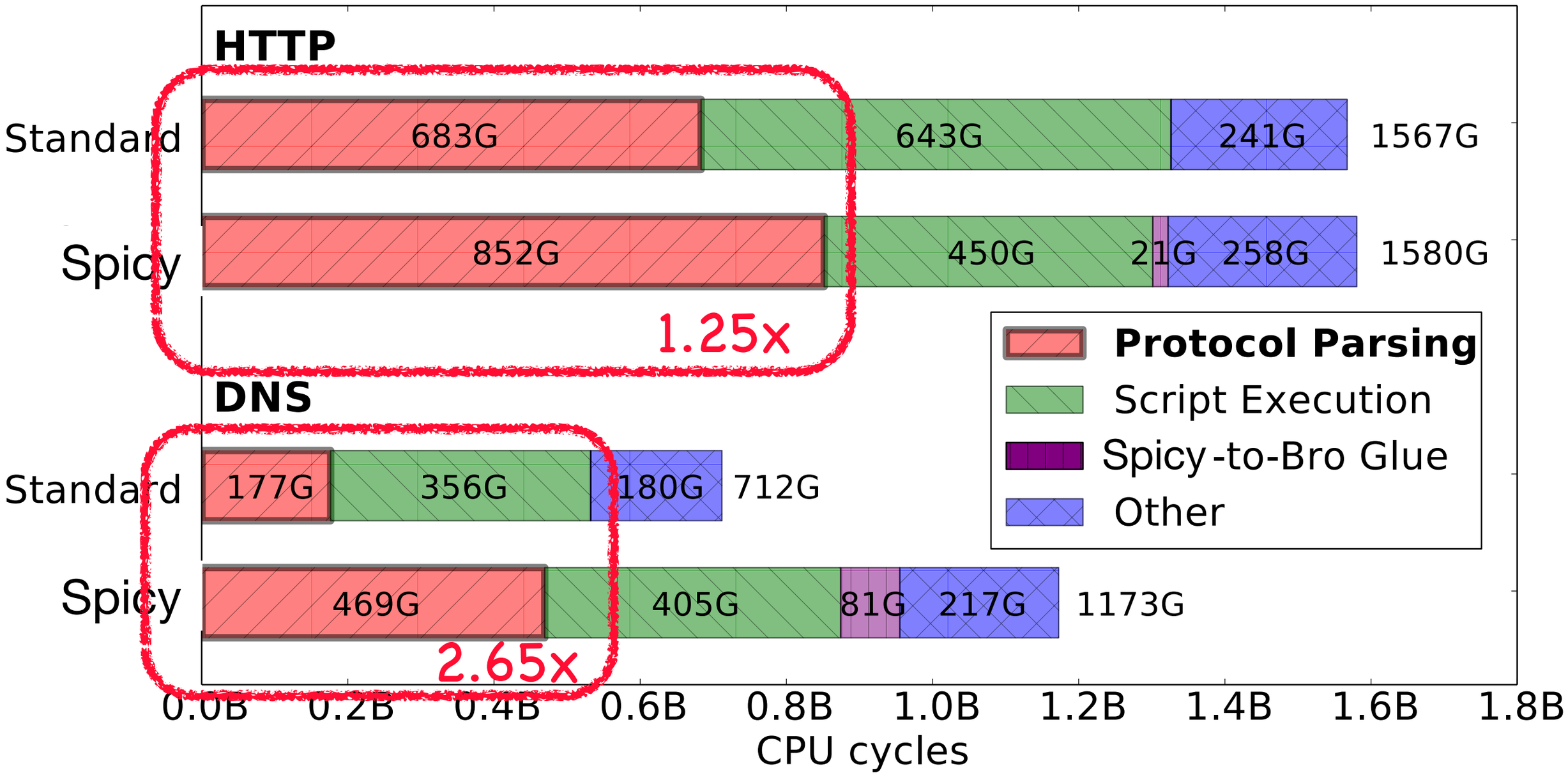
Performance

Let's see.

Performance: Spicy vs. C++ in Bro



Performance: Spicy vs. C++ in Bro



Bro Integration: “3rd Generation Parsers”

Bro Integration: “3rd Generation Parsers”

Generation 1: Manually written C++ code.

Bro Integration: “3rd Generation Parsers”

Generation 1: Manually written C++ code.

Generation 2: BinPAC - “yacc for protocols”.

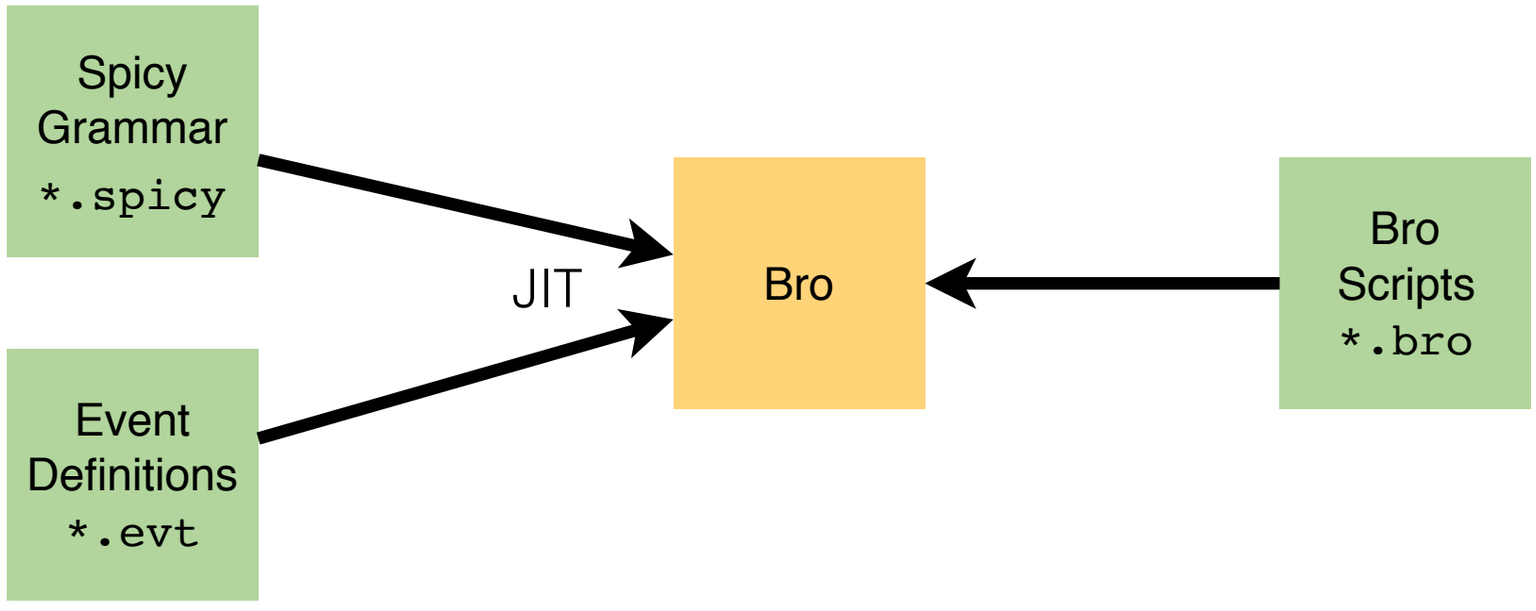
corelight

Bro Integration: “3rd Generation Parsers”

Generation 1: Manually written C++ code.

Generation 2: BinPAC - “yacc for protocols”.

Generation 3: Spicy - A “closed” system.



Advanced Spicy Features

Composibility

Error detection & recovery

Protocol detection

Reassembly/defragmentation

Generating wire format

Implementation: HILTI Toolchain

HILTI: An Abstract Execution Environment for Deep, Stateful Network Traffic Analysis

Robin Sommer
ICSI / LBNL
robin@icir.org

Matthias Vallentin
UC Berkeley
vallentin@icir.org

Lorenzo De Carli
University of Wisconsin-Madison
lorenzo@cs.wisc.edu

Vern Paxson
ICSI / UC Berkeley
vern@icir.org

IMC 2014

Implementation: HILTI Toolchain

HILTI: An Abstract Execution Environment for Deep, Stateful Network Traffic Analysis

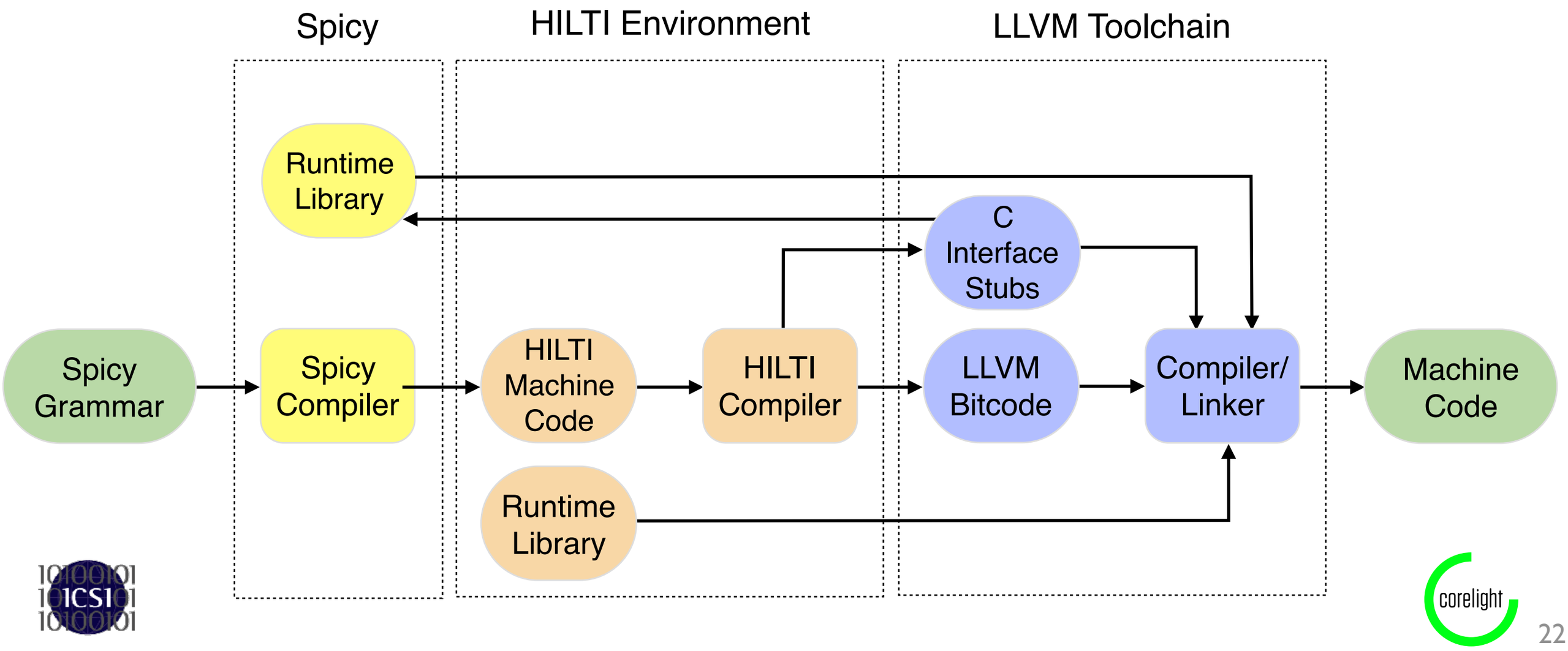
Robin Sommer
ICSI / LBNL
robin@icir.org

Matthias Vallentin
UC Berkeley
vallentin@icir.org

Lorenzo De Carli
University of Wisconsin-Madison
lorenzo@cs.wisc.edu

Vern Paxson
ICSI / UC Berkeley
vern@icir.org

IMC 2014



The HILTI Model

Secure Execution Environment

Sandboxed execution

Automatic memory management

Performance via Abstraction

Transparent improvement under the hood

Integration of non-standard hardware

High-level, global compiler optimizations

Automatic parallelization

Facilitating Reuse

Means and glue to share functionality

HILTI library of common high-level components

Summary

Spicy is a next-generation parser generator
for deep packet inspection systems.

Summary

Spicy is a next-generation parser generator for deep packet inspection systems.

Expresses both syntax and semantics

Supports protocols and file formats

Facilitates composition and reuse

Supports error handling and recovery

Just-in-time compilation via LLVM

Summary

Spicy is a next-generation parser generator for deep packet inspection systems.

Expresses both syntax and semantics

Supports protocols and file formats

Facilitates composition and reuse

Supports error handling and recovery

Just-in-time compilation via LLVM

Open-source, BSD-licensed prototype.

`http://www.icir.org/hilti`

Questions?

Robin Sommer

International Computer Science Institute, &
Corelight, Inc.

robin@icsi.berkeley.edu
robin@corelight.io

http://www.icir.org/robin

Corelight is hiring!

The Bro Project
www.bro.org
info@bro.org
@Bro_IDS

Professional Bro Solutions
www.corelight.io
info@corelight.io
@corelight_inc