Safe Protocol Language (SPL)

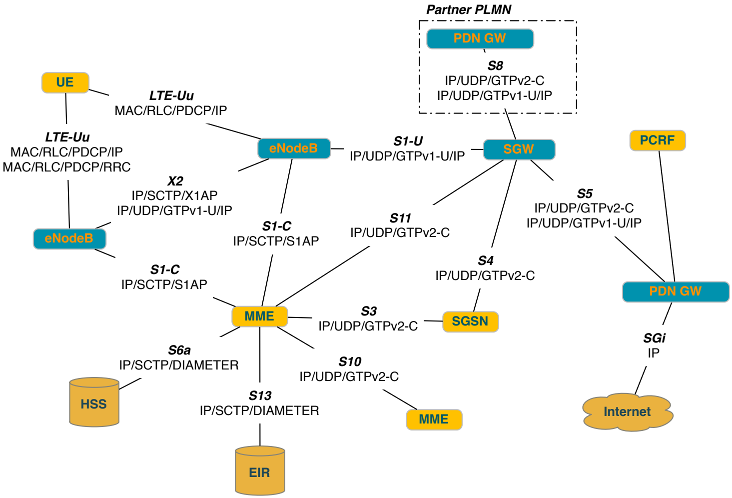
Jasson Casey

April 6th, 2012

Problem - Network Protocol Arms Race

- Network communication protocols
  - Constantly being design, updated, and implemented
- New areas of protocol growth
  - Long Termin Evolution (LTE)
  - IP Television (IPTV)
  - Virtual Networks in Cloud Enviornments
- Requires continuous new development

Problem - LTE Protocol Proliferation



Problem - Classic Software Engineering & Verification

- Protocol design may be faulty
  - lacks safety and/or liveliness
  - does not conform to specification
  - ineffecient
- Protocol implementation may be deficient
  - does not conform to design
  - overly permissive
  - ineffecient

Problem - High Cost & Low Reliability

- System Cost
  - Per protocol cost is measured in man years
  - Interesting problems involve systems of protocols
  - Exploration is prohibitive for all but institutional efforts
- System Reliability
  - Availability and security dependent on underlying protocols
  - Simple, mature protocols by large companies suffer
  - http://www.kb.cert.org/vuls : Apple, Cisco, etc.

Problem - Existing Solutions

- System Programming Languages
  - Levage libraries for common protocol idioms
  - Protocol concepts built on-top of system language (C/C++)
  - Unit and System testing required to catch faults
  - Ad-hoc and usually reactive approach
  - Increases cost of solution
- Formal Verification
  - Heavy duty model checking languages (Spin/Promela)
  - Required understanding: protocol and verification
  - High barrier to entry for non-verification communities
  - Increases cost of solution

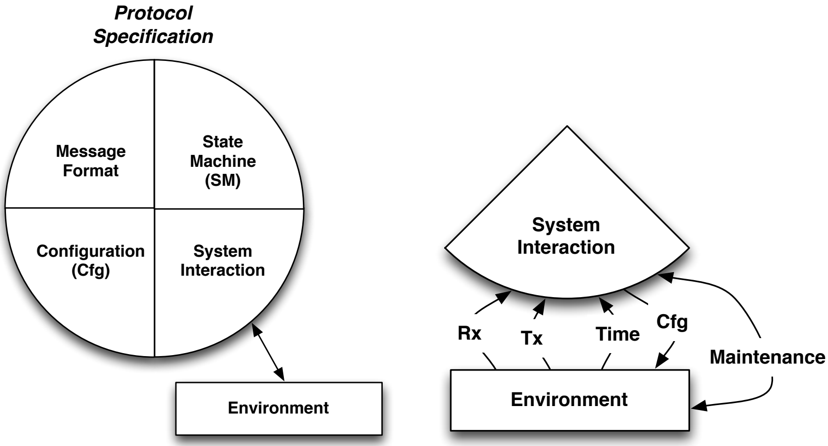
Prior Work

- Esterel - model checking langauge with code extraction [Bous91]
- HIPPCO - optimizing compiler for Esterel [Cast97]
- Prolac - functional language with sub-typing [Khol99]
- Click - configuration language for router components [Khol00]
- PacketTypes - declarative message format language [McCan00]
- Austin Protocol Compiler - certified compilation [McGui04]
- binpac - similar to PacketTypes but with ASCII formats [Pang06]
- protege - ad-hoc protocols for sensor nets [Wang11]

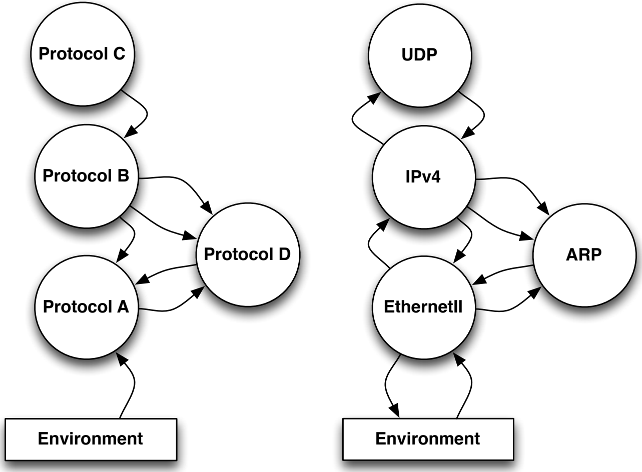
Language Research Objectives

- Simplify and unify design and implementation
  - Lower protocol knowledge/idioms into language
  - Shift verification reasoning from user to the compiler
  - Reduce the amount of code to define a protocol
- Leverage language theory and verification research
  - Validate message and logic handling with type systems
  - Validate state machine behavior with model checking

Intra-Protocol Anatomy



Inter-Protocol Anatomy



First Focus - Message Format

- Material portion of implementation effort
  - Messages are nested self-describing structures
  - Decode/encode functions are non-trivial
- Source of 2 classes of vulnerabilities
  - Value constraints
  - Structural constraints
  - Inhabited by numerous entries in Cert

First Focus - Message Format

- Support declarative format definitions
  - Type check definitions for structural inconsistency
  - Eliminate need to write decode and encode functions
  - Reduces development time and ability to inject errors
- Capture value and structural constraints
  - Use constraints during state machine type check
  - Find unsafe usages during compile
  - Automatically resolve unsafe usage during compile

Structure & Constraint Capture

- Capture constraints with macros and/or comments

```
struct iphdr {
#if defined(__LITTLE_ENDIAN_BITFIELD)
    __u8  ihl:4,
version:4;
#elif defined (__BIG_ENDIAN_BITFIELD)
    __u8  version:4,
ihl:4;
#else
#error   "Please fix <asm/byteorder.h>"
#endif
    __u8  tos;
    __be16   tot_len;
    __be16   id;
    __be16   frag_off;
    __u8  ttl;
    __u8  protocol;
    __sum16  check;
    __be32   saddr;
    __be32   daddr;
    /*The options start here. */
};
```

Explicit Constraint Checking

```
if (iph->ihl < 5 || iph->version != 4)
goto inhdr_error;

if (!pskb_may_pull(skb, iph->ihl*4))
goto inhdr_error;

iph = ip_hdr(skb);

if (unlikely(ip_fast_csum((u8 *)iph, iph->ihl)))
goto inhdr_error;

len = ntohs(iph->tot_len);
if (skb->len < len) {
    IP_INC_STATS_BH(dev_net(dev), IPSTATS_MIB_INTRUNCATEDPKTS);
    goto drop;
} else if (len < (iph->ihl*4))
goto inhdr_error;
```

Format Declaration Types

- uint t_1 t_2 - new type
  - $t_1 =$ expression defining the number size of this uint in bits
  - $t_2 =$ expression defining the initial/default value of this type
- pad t - new type
  - $t =$ expression defining the number of bits to skip
- array t_1 t_2 - new type
  - $t_1 =$ expression defining the type contained in the array
  - $t_2 =$ expression defining the initial/default value of the array

Format Declaration Types

- pdu *id* $\{\ l_i = t_i\}$ - new type
  - id = idenifier for the pdu record
  - l_i = label for this particular attirbute
  - t_i = term to assign to label of this attribute
- enum *id* $\{\ p_i : t_i\ \}$
  - id = idenifier for the enum variant
  - p_i = predicate used to identify the variant type
  - t_i = term used under this predicate

Format Example

```
pdu LLC ...
pdu IPv4 ...
pdu ARP ...

enum L2_type
   <= 0x0600 : LLC
   == 0x0800 : IPv4
   == 0x0806 : ARP

pdu EtherII
   uint 48 dst
   uint 48 src
   uint 16 type_len
   [uint 8] * payload
   uint 32 crc
```

Format - Structural Dependencies

```
pdu Test
   uint 1 flag
   pad 7
   if flag == 1 then
      uint 16 extra
   [uint 8] * payload
```

- Potential unsafe usage of extra

```
rcv :: Test -> Unit
rcv pkt =
   examine pkt.extra
```

Aside - Lambda Calculus

- Formalized by Alonzo Chruch
- Developed to study the foundations of mathematics
- Foundation of programming langauge theory

Aside - Lambda Calculus

- Inductively defined language of terms
  - t ::= x | λx.t | t t
- variable - x
- abstraction - λx.t
- application - t t

Aside - Lambda Calculus

- Evaluation Rules / Dynamics

$$\frac{t_1 \to t_1'}{t_1 \ t_2 \to t_1' \ t_2} \ App$$

$$\frac{t_2 \to t_2'}{v_1 \ t_2 \to v_1 \ t_2'} \ App$$

$$\frac{}{(\lambda x.t) \ v \to [x \mapsto v] \ t} \ App$$

Aside - Lambda Calculus

- The λ calculus is turing complete
- tru $= \lambda$t.λf.t;
- fls $= \lambda$t.λf.f;
- if_then_else $= \lambda$l.λm.λn. l m n;
- if_then_else tru x w

Aside - Lambda Calculus

- Enrich system with types, an environment, and more terms
- t ::= x | λx.t | t t | if t then t else t
- t ::= pred t | succ t | iszero t
- t ::= true | false | 0
- T ::= Bool | Nat | T $\rightarrow$ T
- $\Gamma ::= \emptyset \mid \Gamma, x : T$

Aside - Lambda Calculus

- Statics / Type Relations : $\Gamma \vdash t : T$

$$\frac{}{\Gamma \ \vdash \ \ true : Bool} \ T_{true}$$

$$\frac{}{\Gamma \ \vdash \ \ false : Bool} \ T_{false}$$

$$\frac{}{\Gamma \ \vdash \ \ 0 : Nat} \ T_{zero}$$

$$\frac{\mathrm{x{:}T} \in \Gamma}{\Gamma \ \vdash \ \ x : T} \ T_{store}$$

$$\frac{\Gamma \ \vdash \ \ t : \mathrm{Nat}}{\Gamma \ \vdash \ \ succ\ \mathrm{t} : \mathrm{Nat}} \ T_S$$

$$\frac{\Gamma \ \vdash \ \ t : \mathrm{Nat}}{\Gamma \ \vdash \ \ pred\ \mathrm{t} : \mathrm{Nat}} \ T_P$$

$$\frac{\Gamma \ \vdash \ \ t : \mathrm{Nat}}{\Gamma \ \vdash \ \ iszero\ \mathrm{t} : \mathrm{Bool}} \ T_Z$$

Aside - Lambda Calculus

- Statics / Type Relations : $\Gamma \vdash t : T$

$$\frac{\Gamma \ \vdash \ \ t_1 \ : \ \mathrm{Bool} \qquad \Gamma \ \vdash \ \ t_2 \ : \ \mathrm{T} \qquad \Gamma \ \vdash \ \ t_3 \ : \ \mathrm{T}}{\Gamma \ \vdash \ \mathrm{if} \ t_1 \ \mathrm{then} \ t_2 \ \mathrm{else} \ t_3 \ : \ \mathrm{T}} \ T_{ifelse}$$

$$\frac{\Gamma, \mathrm{x} : \ T_1 \ \vdash \ \ t_2 \ : \ T_2}{\Gamma \ \vdash \ \ \lambda x : T_1 . t_2 : T_1 \to T_2} \ T_{abs}$$

$$\frac{\Gamma \ \vdash \ \ t_1 : T_1 \to T_2 \qquad \Gamma \ \vdash \ \ t_2 : T_1}{\Gamma \ \vdash \ \ t_1 \ t_2 : T_2} \ T_{app}$$

Aside - Lambda Calculus

- Dynamics / Evaluation : $t \to t'$

$$\frac{}{\text{P } 0 \to 0}\; E_{P0}$$

$$\frac{}{\text{Z } 0 \to \text{true}}\; E_{Z0}$$

$$\frac{}{\text{Z (S nv)} \to \text{false}}\; E_{ZS}$$

$$\frac{}{\text{P (S nv)} \to \text{nv}}\; E_{PS}$$

$$\frac{}{\text{if true then } t_2 \text{ else } t_3 \to t_2}\; E_{ifelse-true}$$

$$\frac{}{\text{if false then } t_2 \text{ else } t_3 \to t_3}\; E_{ifelse-false}$$

Aside - Lambda Calculus

- Dynamics / Evaluation : $t \to t'$

$$\frac{}{(\lambda x : T_1 . t_2)\ v_3 \to [x \mapsto v_3]\ t_2}\ E_{app}$$

$$\frac{\text{t} \to t'}{\text{pred t} \to \text{pred } t'}\ E_P$$

$$\frac{\text{t} \to t'}{\text{succ t} \to \text{succ } t'}\ E_S$$

$$\frac{\text{t} \to t'}{\text{iszero t} \to \text{iszero } t'}\ E_Z$$

Aside - Lambda Calculus

- Dynamics / Evaluation : $t \rightarrow t'$

$$\frac{t_1 \rightarrow t_1'}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \rightarrow \text{if } t_1' \text{ then } t_2 \text{ else } t_3} E_{ifelse}$$

$$\frac{t_1 \rightarrow t_1'}{t_1 \; t_2 \rightarrow t_1' \; t_2} E_{app}$$

$$\frac{t_2 \rightarrow t_2'}{v_1 \; t_2 \rightarrow v_1 \; t_2'} E_{app}$$

Aside - Lambda Calculus

- Safety of the simply typed λ calculus
- Progress - if $\Gamma \vdash t : T$ then $t \rightarrow t'$ or $t \approx v$
- Preservation - if $\Gamma \vdash t : T$ and $t \rightarrow t'$ then $\Gamma \vdash t' : T$
- Well typed programs do not 'go wrong'

Terms

Basic term structure found in $\lambda_{\rightarrow}$

$t_{base} ::= t\ t \mid \lambda \mathrm{x}.t \mid \text{if } t \text{ then } t \text{ else } t \mid t \text{ as T} \mid \text{let x} = t \text{ in } t \mid \{x_i = t_i^{i \in 1..n}\} \mid t.\mathrm{x} \mid \text{nil} \mid \text{cons t t}$

Augmenting terms allowing for structural types

$t_{ext} ::= \text{array } t\ t \mid \text{uint } t\ t \mid \text{pad } t \mid \text{pdu } t \mid \text{enum } t$

Built-in Types

$\tau_{constant}$ ::= Nat | Nat^{∞} | Char | Bool | Unit
$\tau_{composite}$ ::= $\tau^{\top}$ | $\tau \rightarrow \tau$ | $[\tau]$ | $\tau ? t$
$\sigma ::= \tau \mid \alpha \mid \forall \alpha . \sigma$
$\tau^{\top} = \tau \cup \{\top\}$, where $\top$ indicates the undefined value
$\text{Nat}^{\infty} = \text{Nat} \cup \{\infty\}$
$\tau ? t$ = type τ's presence depends on a predicate t being true

Type Constructors

$$\pi ::= \mathit{array} \mid \mathit{uint} \mid \mathit{pad} \mid \mathit{pdu} \mid \mathit{enum}$$
$$\mathit{array} :: \forall \alpha . \alpha \rightarrow \mathit{Nat}^{\infty} \rightarrow *$$
$$\mathit{uint} :: \mathit{Nat} \Rightarrow \mathit{Nat}^{\top} \Rightarrow *$$
$$\mathit{pdu} :: \{\pi\}^{+} \Rightarrow *$$
$$\mathit{enum} :: \{\forall \alpha . \alpha \rightarrow \mathit{Bool}, \tau\}^{+} \Rightarrow *$$
$$\mathit{pad} :: \mathit{Nat} \Rightarrow *$$

PDU Example

Surface Syntax

```
pdu EtherII
   uint 48 dst
   uint 48 src
   uint 16 type_len
   [uint 8] * payload
   uint 32 crc
```

Resulting Term

let EtherII = pdu { dst=uint 48 $\perp$, src=uint 48 $\perp$, type_len=uint 16 $\perp$, payload=[uint 8] ∞ $\perp$, crc = uint 32 $\perp$} in ...

PDU Type Checking

The sequence of inner types incrementally contribute to the environment.

$$\frac{\Gamma \;\vdash\; t_1 : T_1 \qquad \forall i_{\in 2..N}\; \Gamma, \{t_1 : T_1, .., t_{i-1} : T_{i-1}\} \;\vdash\; t_i : T_i}{\Gamma \;\vdash\; \{t_1, .., t_N\} : \{T_1, .., T_N\}}$$

Dependency Environment

Maybe we need an environment to carry dependencies arround

```
pdu Test
   uint 1 flag
   pad 7
   IF flag == 1
     uint 16 extra
   END
```

$\delta =$ special dependency term

$$\frac{\Gamma \vdash t : \mathit{Bool} \qquad \Gamma \vdash \delta : T}{\Gamma \vdash \mathit{IF}\ t : \mathit{Unit}, \delta : T?t}$$

$$\frac{\Gamma \vdash \delta : T}{\Gamma \vdash \mathit{End} : \mathit{Unit}, \delta : \mathit{Unit}}$$

Reserved Attributes

$$attr_{pdu} ::= \text{x.bits} \mid \text{x.bytes}$$
$$attr_{enum} ::= \text{x.type} \mid \text{x.enum}$$

```
enum EnumTest
   == 74 : uint 32
   > 100 : [char] 128

pdu PduTest
   uint 8 kind
   EnumTest type payload

rcv :: PduTest -> Unit
rcv pkt =
   f pkt.bits pkt.bytes pkt.payload.type pkt.payload.enum
```

Enum Example

```
pdu LLC ...
pdu IPv4 ...
pdu ARP ...

enum L2_type
   <= 0x0600 : LLC
   == 0x0800 : IPv4
   == 0x0806 : ARP

pdu EtherII
   uint 48 dst
   uint 48 src
   uint 16 type_len
   L2_type type_len payload
   uint 32 crc
```

Simple Exmaple

```
rcv :: ARP -> Unit
rcv pkt as ARP =
   if pkt.type == Request
      then send pkt.mac ( arp_who_has pkt.ip )
      else arp_update pkt.mac pkt.ip

rcv :: IPv4 -> Unit
rcv pkt as IPv4 =
   if pkt.ttl == 0
      then send pkt.src ( make_dst_unreach pkt )
      else fwd ( dec_ttl pkt )

rcv :: EtherII -> Unit
rcv pkt as EtherII =
   rcv pkt.payload
```

Checking Example

```
pdu Foo
   uint 1 flag
   pad 7
   if flag == 1
      then uint 32 seq

rcv :: Foo -> Unit
rcv pkt =
  do_somthing pkt.seq
```

Potential error accessing parameter that is conditionally runtime dependent

Conditional Types

Let the type of seq depend on its conditional structural

$$\frac{\ldots}{\Gamma \vdash \text{seq} : \text{uint } 32 \ ? \ \text{flag} == 1}$$

Only type check dependent terms if they exist in a branch of control flow that covers the dependency.

```
rcv :: Foo -> Unit
rcv pkt =
   if pkt.flag == 1 or pkt.flag != 0 or ...
      then do_something pkt.seq
```

Predicate Introduction

- Δ - set of known facts
- conditional terms introduce predicates
- evaluate sub-terms using new fact established by predicate

$$\frac{\Delta|\Gamma \vdash t_1 : Bool \qquad \Delta,\{t_1\}|\Gamma \vdash t_2 : \tau \qquad \Delta,\{\neg t_1\}|\Gamma \vdash t_3 : \tau}{\Delta|\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : \tau}\ \mathbf{T}_{if}$$

Dependency Elimination

- deduce dependency from known facts
- eliminate dependency in resulting type

$$\frac{\Delta|\Gamma \;\vdash\; t : \tau?\delta \qquad \Delta \models \delta}{\Delta|\Gamma \;\vdash\; t : \tau}\ \mathbf{T}_{DepElim_1}$$

Dependency Deferral

- use of type without proof of dependency
- rewrite term using conditional
- eliminate dependency of type

$$\frac{\Delta|\Gamma \;\vdash\; t : \tau?\delta \qquad \Delta \not\models \delta}{\Delta|\Gamma \;\vdash\; t : \tau?\delta \rightsquigarrow \text{if } \delta \text{ then } t \text{ else } \mathit{error} : \tau}\ \mathbf{T}_{DepElim_2}$$

References

- P. McCann, S. Chandra. Packet Types: Abstract Specification of Network Protocol Messages. ACM SIGCOMM Proc. on Applications, Technologies, Architectures, and Protocols for Computer Communication 2000.
- R. Pang, V. Paxson, R. Sommer, L. Peterson. Binpacc: A yacc for Writing Application Protocol Parsers. ACM SIGCOMM Internet Measurement Conference 2006.
- Y. Wang, V. Gaspes. An Embedded Language for Programming Protocol Stacks in Embedded Systems. ACM PEPM 2011.
- K. Fisher, Y. Mandelbaum, D. Walker. The Next 700 Data Description Languages. ACM POPL 2006.
- E. Kholer, M. Kaashoek, D. Montgomery. A Readable TCP in the Prolac Protocol Language. ACM SigComm 99.
- E. Kohler, R. Morris, B. Chen, J. Jannotti, M. Kaashoek. The click modular router. ACM Transactions on Computer Systems 2000.
- T. McGuire, M. Gouda, The Austin Protocol Compiler, Springer 2004.
- F. Boussinot, R. Simone. The Esterel Language. Proceedings of the IEEE 1991.
- C. Castelluccia, P. Hoschka. A Compiler-Based Approach to Protocol Optimization. Second Workshop on High Performance Communications Subsystems 1995.

References

- C. Castelluccia, W. Dabbous, S. OMalley. Generating Efficient Protocol Code From an Abstract Specification. IEEE Transactions on Networking 1997.
- F. Boussinot, R. Simone. The Esterel Language. Proceedings of the IEEE 1991.
- C. Castelluccia, P. Hoschka. A Compiler-Based Approach to Protocol Optimization. Second Workshop on High Performance Communications Subsystems 1995.

- C. Castelluccia, W. Dabbous, S. OMalley. Generating Efficient Protocol Code From an Abstract Specification. IEEE Transactions on Networking 1997.