

A LANGUAGE-BASED APPROACH FOR ROBUST AND EFFICIENT NETWORK APPLICATION PROTOCOL IMPLEMENTATIONS

Laurent Burgy, Laurent Réveillère
 Université de Bordeaux, INRIA / LaBRI
 {burgy,reveillere}@labri.fr

Julia Lawall
 University of Copenhagen, DIKU
 julia@diku.dk

Gilles Muller
 École des Mines de Nantes, INRIA / LINA
 Gilles.Muller@emn.fr

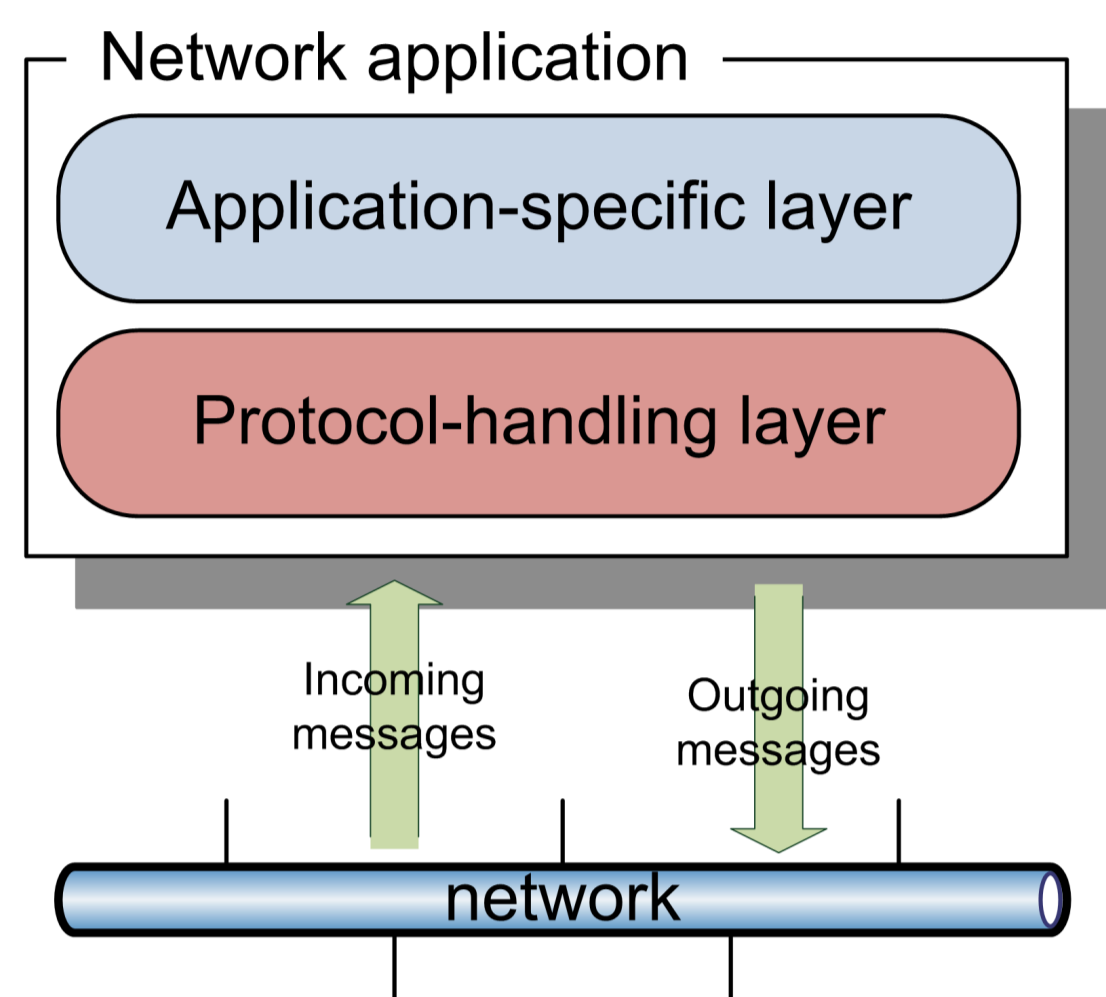
Problem

HTTP-like application-layer protocols

- HTTP, SIP, RTSP
- Defined in RFCs using ABNF notation

Network applications built on application-layer protocols

- Instant Messaging clients, Multimedia players, HTTP/SIP Servers and Proxies...



Protocol-handling layer

- Message Decoding
 - Message Validation
 - Message Forging/Modifications
- ⇒ Invariant across all applications based on the same protocol

Application-specific layer

- Dedicated to each application
- Implements the application logic
- Relies on the protocol-handling layer to manipulate the message

Protocol-Handling Layer must be correct/defect-free and efficient

Existing Solutions

Hand-writing code

```
for (tmp=p;tmp<end;tmp++){
  switch(*tmp){
    case ' ':
      switch(state){
        case FIN_HIDDEN:
        case FIN_ALIAS:
          param->type=state;
          param->name.len=tmp-param->name.s;
          state=L_PARAM;
          goto endofparam;
        case FIN_BRANCH:
        case FIN_TTL:
        case FIN_MADDR:
        case FIN_RECEIVED:
        case FIN_RPORT:
        case FIN_I:
          param->type=state;
          param->name.len=tmp-param->name.s;
          state=L_VALUE;
          goto find_value;
        ...
      }
  }
}
```

- SIP Express Router parse_via.c
 - 2035 LOC, 64 switch, 564 case, 232 break
 - Far from ABNF specification
 - Difficult to write and maintain

- Correctness
 - How to guarantee robustness?
 - What about completeness?
 - Buffer-Overflow attacks

- Efficiency
 - Both in terms of **memory** and **time**

⇒ **Difficult to reconcile efficiency and robustness**

Using parser-generators

YACC

⇒ Too low-level

DataScript, PacketTypes

⇒ Only for binary protocols

PADS

⇒ Generic for processing ad hoc data, not protocol-oriented

Binpac, GAPA

⇒ Language tightly-coupled to the underlying framework

⇒ **Formalism often far from ABNF notation**

⇒ **Target only parsing phase**

Our Approach: Zebu

```
message sip3261 {
  request {
    ; Request only
    requestLine = Method.method SP Request-URIuri SP SIP-Version

    ; Constraints that apply only for the CSeq of a request
    header CSeq { CSeq.method == requestLine.method }

    ; Constraints that apply only for the Max-Forwards of a request
    header Max-Forwards { mandatory }
  }

  response {
    ; Response only
    statusLine = SIP-Version SP Status-Code:code SP Reason-Phrase:rphrase
    [...]
  }

  enum Method = INVITEm / ACKm / OPTIONSm / BYEm / CANCELm / REGISTERm / extension-method
  extension-method = token
  INVITEm = %x49.4E.56.49.54.45 ; INVITE in caps
  [...]

  struct Request-URI = SIP-URI / SIPS-URI / absoluteURI { lazy }
  [...]

  uint16 Status-Code = Informational / Redirection / Success / Client-Error / Server-Error
  / Global-Failure / extension-code
  uint16 Global-Failure = "600" ; Busy Everywhere
  / "603" ; Decline
  / "604" ; Does not exist anywhere
  / "606" ; Not Acceptable
  uint16 extension-code = 3DIGIT { extension-code >= 100 && extension-code <= 699 }
  [...]

  ; Header CSeq
  header CSeq = 1*DIGIT:number as uint32 LWS Method:method

  ; Header Max-Forwards
  header Max-Forwards = 1*DIGIT:value as uint32 { mandatory }

  ; Header To
  header To { "to" / "t" } = ( name-addr / addr-spec:uri ) *( SEMI to-param ) { mandatory, ReadOnly }
  name-addr = [ display-name ] LAQUOT addr-spec:uri RAQUOT
  struct addr-spec = SIP-URI / SIPS-URI / absoluteURI { lazy }
  [...]
}
```

Zebu in a nutshell

- Declarative Domain-Specific Language
- Generates **correct** and **efficient** protocol messages handling layers for application layer network protocols.
- Extends ABNF with domain-specific annotations
- Focuses on HTTP-like protocols

Zebu language annotations

- Define the memory layout required by the application
- Define the semantic interpretation of field values (types)
- Describe which fields can be processed (read/write/etc.)
- Express constraints on field values (bounds)
- Specify when a field must be processed (on-demand parsing)

The Zebu Compiler

- Generates C code
 - Memory layout tailored to application needs
 - Stubs (getters/setters) to be used by the application
- Enables compile-time/run-time verifications

Current Results

Mutation Analysis

- Injection of incorrect mutated messages
 - SIP
 - * Zebu generated code **detects 100%** of injected mutants
 - * Typical implementations **detect between 17% and 25%**
 - RTSP
 - * Zebu generated code **detects 100%** of injected mutants
 - * Typical implementations **detect between 0.1% and 27%**
- Injection of messages derived from an RFC to torture SIP implementations
 - Zebu generated code **does not consider any correct message as invalid**
 - Typical implementations **consider up to 4% of correct messages as invalid**

Performance (preliminary results)

- From **1.66 times faster to 6 times slower** than SIP Express Router
- From **3 times to 17 times faster** than the OSIP library

Future Work

- Performance improvement
 - Reducing memory footprint
 - Reducing execution time
- Memory predictability
- Implementation of other protocols and applications
- Support of protocol extension

<http://phoenix.labri.fr/software/zebu/>