Analyzing Protocol Implementations for Interoperability

Luis Pedrosa  Ari Fogel  Nupur Kothari
Ramesh Govindan  Ratul Mahajan  Todd Millstein

http://nsl.cs.usc.edu/Projects/PIC

May 6, 2015  NSDI'15  Oakland, CA
[Protocol] Interoperability

(noun)
The ability of two or more systems to successfully communicate in accordance with an agreed upon protocol.
What I Mean by Interoperability

[Protocol] Interoperability
(noun)

Interoperability is crucial for reliability and correctness in networked systems!

RFC 2616
HTTP/1.1
SIP Interoperability - Why Is It So Hard to Achieve?

The problem is that RFC 3261 that defines SIP has become “everything to everyone” and bloated in both size and in flexibility.
SIP Interoperability: Why Is It So Hard to Achieve? (Part II)

Let's move past the technical issues and talk about a far more difficult challenge—the politics of SIP Interoperability.

It appears to me that soon after the authors of RFC 3261 finished their work, the fun really started. As the development teams of the various product and application companies started to build their solutions based on RFC 3261, the looseness of the specification allowed them to make wildly different choices all "within specification." The result was that you had developers that had invested untold hours of hard work into developing a protocol stack that worked fine in their own lab and with their own products, but had serious interoperability issues with other vendors. To each of the developers, it appeared that "everybody else screwed up."
Interoperability is Important!

This isn't going away!
Interoperability Testing Today

Bake Off
Test case:

- Participant roles
- Topology
- Protocol interaction
- Inputs
Interoperability Testing Today

Test case:
- Participant roles
- Topology
- Protocol interaction
- Inputs
Interoperability Testing Today

Test case:
- Participant roles
- Topology
- Protocol interaction
- Inputs
Interoperability Testing Today

Test case:
- Participant roles ✔
- Topology ✔
- Protocol interaction ✔
- Inputs
Interoperability Testing Today

Test case:
- Participant roles
- Topology
- Protocol interaction
- Inputs
Interoperability Testing Today
Interoperability Testing Today

Manual testing is not enough!
- Human intensive
- Not comprehensive

INVITE sip:@usc.edu
We should automate the search for inputs that will lead to non-interoperabilities
Defining Interoperability

Sender

Request

Receiver

API
Defining Interoperability

Diagram showing the interaction between a Sender and a Receiver. The(Sender) sends a Request, which is successfully received. However, the Response is not successful. The Receiver responds to the Request.
Defining Interoperability

Every Possible Message
Defining Interoperability

Messages Sender Generates
Defining Interoperability

Messages Sender Generates

Messages Receiver Rejects
Defining Interoperability

Messages Sender Generates

Messages Receiver Rejects

Interoperability Issues
Defining Interoperability

[Diagram showing Venn diagram with one circle labeled "Messages Sender Generates" and another labeled "Messages Receiver Rejects". The overlapping area is labeled "Interoperability Issues".]
Contributions

- Automate finding non-interoperabilities
  - Use program analysis to map message space
Contributions

- Automate finding non-interoperabilities
  - Use program analysis to map message space

- Scale to real-world protocols
  - Guide and prune the exploration carefully
  - Leverage common constructs in protocol code
Contributions

- Automate finding non-interoperabilities
  - Use program analysis to map message space

- Scale to real-world protocols
  - Guide and prune the exploration carefully
  - Leverage common constructs in protocol code

- Find real interoperability issues
  - Evaluated on SPDY and SIP implementations
  - Found and reported ~20 issues
Protocol Interoperability Checker

Instrumented Code -> Sender Analysis -> Validation

Instrumented Code -> Receiver Analysis

Combined Paths -> SMT Solver

Test Cases

Program analysis to map message space

Interoperability Issues
Example: NetCalc

**NetCalc Client**

1: void compute(char op,  
2:     int arg1,  
3:     int arg2) {
4:     byte msg[9];
5:     
6:     if (op == '+')
7:         msg[0] = 0;
8:     else if (op == '-')
9:         msg[0] = 1;
10:    else throw exception;
11:   
12:   msg[1..4] = arg1;
13:   msg[5..8] = arg2;
14:   
15:   send(msg);
16: }

**NetCalc Server**

1: int handleMessage(byte[] query) {
2:     int arg1 = query[1..4];
3:     int arg2 = query[5..8];
4:     int reply;
5:     
6:     switch (query[0]) {
7:         case 0:
8:             reply = arg1 + arg2;
9:             break;
10:        default:
11:             throw exception;
12:     }
13:   
14:   return reply;
15: }
**NetCalc Client**

```java
1: void compute(char op,
2:      int arg1,
3:      int arg2) {
4:      byte msg[9];
5:      
6:      if (op == '+'
7:         msg[0] = 0;
8:      else if (op == '-')
9:         msg[0] = 1;
10:      else throw exception;
11:      
12:      msg[1..4] = arg1;
13:      msg[5..8] = arg2;
14:      
15:      send(msg);
16: }
```

**NetCalc Server**

```java
1: int handleMsg(byte[] query) {
2:      int arg1 = query[1..4];
3:      int arg2 = query[5..8];
4:      int reply;
5:      
6:      switch (query[0]) {
7:         case 0:
8:            reply = arg1 + arg2;
9:            break;
10:        default:
11:            throw exception;
12:      }
13:      
14:      return reply;
15: }
```
NetCalc Client

```java
1: void compute(char op,
2:     int arg1,
3:     int arg2) {
4:     byte msg[9];
5:     
6:     if (op == '+')
7:         msg[0] = 0;
8:     else if (op == '-')
9:         msg[0] = 1;
10:    else throw exception;
11:    
12:    msg[1..4] = arg1;
13:    msg[5..8] = arg2;
14:    
15:    send(msg);
16: }
```
Symbolic Execution Primer

**NetCalc Client**

```java
1: void compute(char op,
2:         int arg1,
3:         int arg2) {
4:     byte msg[9];
5:     if (op == '+')
6:         msg[0] = 0;
7:     else if (op == '-')
8:         msg[0] = 1;
9:     else throw exception;
10:    msg[1..4] = arg1;
11:    msg[5..8] = arg2;
12:    send(msg);
13: }
```

**Path Condition:** Symbolic Store:

\[ \text{op} = \sigma_1 \]
NetCalc Client
1: void compute(char op,
2: int arg1,
3: int arg2) {
4: byte msg[9];
5: if (op == '+')
6: msg[0] = 0;
7: else if (op == '-')
8: msg[0] = 1;
9: else throw exception;
10: msg[1..4] = arg1;
11: msg[5..8] = arg2;
12: send(msg);
13: }

Path Condition: Symbolic Store:
\(\sigma_1 == '+'\)
\(\sigma_1\)
\(\sigma_2\)
\(\sigma_3\)
\(\emptyset\)

Path Condition: Symbolic Store:
\(\sigma_1 != '+'\)
\(\sigma_1\)
\(\sigma_2\)
\(\sigma_3\)
\(\emptyset\)
NetCalc Client

```java
void compute(char op, int arg1, int arg2) {
    byte msg[9];
    if (op == '+')
        msg[0] = 0;
    else if (op == '-')
        msg[0] = 1;
    else throw exception;
    msg[1..4] = arg1;
    msg[5..8] = arg2;
    send(msg);
}
```

Path Condition: Symbolic Store:
- $\sigma_1 = '+'$
- $\sigma_1 = \sigma_2$
- $\sigma_2 = \sigma_3$
- $\sigma_3$
- $msg = 0|\sigma_2|\sigma_3$
How PIC uses Symbolic Execution

Sender

+  -  !
How PIC uses Symbolic Execution

Sender

[Diagram showing a button with a plus and a minus sign]
How PIC uses Symbolic Execution

Sender

Receiver

Background  Motivation  Design  Evaluation  Conclusion
How PIC uses Symbolic Execution

Sender

Receiver
How PIC uses Symbolic Execution

Sender

Receiver

!
Combining Sender & Receiver

+ 

- 

!
Combining Sender & Receiver

+ -> ! -> +
-

- !
Combining Sender & Receiver

SMT Solver
Combining Sender & Receiver

![Diagram showing SMT Solver process]

SMT Solver
Combining Sender & Receiver

SMT Solver
Combining Sender & Receiver

\[ \text{op} = \div \]
\[ \text{arg1} = 2 \]
\[ \text{arg2} = 3 \]

SMT Solver
Combining Sender & Receiver

Potentially HUGE!

op = '
arg1 = 2
arg2 = 3

SMT Solver
Potentially HUGE!

Independently analyzing sender and receiver **does not scale!**

op = '-'
arg1 = 2
arg2 = 3
Scaling: Joint Analysis

Messages Sender Generates

Interoperability Issues

Messages Receiver Rejects
Scaling: Joint Analysis

- Messages Sender Generates
- Messages Receiver Rejects

Interoperability Issues
Scaling: Joint Analysis

- Messages: Sender Generates
- Messages: Receiver Rejects
- Interoperability Issues

Background  Motivation  Design  Evaluation  Conclusion
Scaling: Joint Analysis

Interoperability Issues

Messages Sender Generates

Messages Receiver Rejects
We should scope the receiver-side analysis within the sender context.
Joint Symbolic Execution

NetCalc Server

```
1: int handleMessage(byte[] query) {
2:     int arg1 = query[1..4];
3:     int arg2 = query[5..8];
4:     int reply;
5:     switch (query[0]) {
6:         case 0:
7:             reply = arg1 + arg2;
8:             break;
9:         default:
10:             throw exception;
11:     }
12: }
13: 
14: return reply;
15: }
```
NetCalc Server

```java
1: int handleMsg(byte[] query) {
2:     int arg1 = query[1..4];
3:     int arg2 = query[5..8];
4:     int reply;
5:     
6:     switch (query[0]) {
7:         case 0:
8:             reply = arg1 + arg2;
9:             break;
10:        default:
11:            throw exception;
12:     }
13: 
14:     return reply;
15: }
```
NetCalc Server

```java
1: int handleMsg(byte[] query) {
2:     int arg1 = query[1..4];
3:     int arg2 = query[5..8];
4:     int reply;
5:     switch (query[0]) {
6:         case 0:
7:             reply = arg1 + arg2;
8:             break;
9:         default:
10:             throw exception;
11:         }
12:     } return reply;
13: }
```
NetCalc Server

1: int handleMsg(byte[] query) {
2:     int arg1 = query[1..4];
3:     int arg2 = query[5..8];
4:     int reply;
5:     
6:     switch (query[0]) {
7:         case 0:
8:             reply = arg1 + arg2;
9:             break;
10:        default:
11:            throw exception;
12:    }
13: }
14: return reply;
15: }
Joint Symbolic Execution

NetCalculate Server

```java
1: int handleMsg(byte[] query) {
2:   int arg1 = query[1..4];
3:   int arg2 = query[5..8];
4:   int reply;
5:   
6:   switch (query[0]) {
7:     case 0:
8:       reply = arg1 + arg2;
9:       break;
10:      default:
11:         throw exception;
12:     }
13: }
14: return reply;
15: }
```
NetCalc Server

1: int handleMessage(byte[] query) {
2:     int arg1 = query[1..4];
3:     int arg2 = query[5..8];
4:     int reply;
5: 
6:     switch (query[0]) {
7:         case 0:
8:             reply = arg1 + arg2;
9:             break;
10:        default:
11:            throw exception;
12:     }
13: 
14:     return reply;
15: }

Joint Symbolic Execution enables scaling to large implementations.
Other Scaling Challenges

- Challenges with symbolic execution
  - Inherently incomplete; huge search-space
  - Typically tries to explore all paths
  - PIC only cares about a few
Other Scaling Challenges

- Challenges with symbolic execution
  - Inherently incomplete; huge search-space
  - Typically tries to explore all paths
  - PIC only cares about a few

- Solution: Guide and prune search-space
  - Prior art: Directed Symbolic Execution
  - Dispersive Exploration: used in receiver analysis
  - Return Normalization: mitigate local minima

---

1 K.-K. Ma, K. Y. Phang, J. S. Foster, M. Hicks. Directed symbolic execution. SAS'11
## Evaluation

<table>
<thead>
<tr>
<th>Implementation</th>
<th>SPDY Library</th>
<th>Annotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>spdylay-0.3.7</td>
<td>22739</td>
<td>23</td>
</tr>
<tr>
<td>spdylay-1.3.1</td>
<td>25495</td>
<td>23</td>
</tr>
<tr>
<td>nginx-1.5.5</td>
<td>99446</td>
<td>4</td>
</tr>
<tr>
<td>nginx-1.7.4</td>
<td>104364</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Implementation</th>
<th>SIP – Session Initiation Protocol Library</th>
<th>Annotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>eXoSIP-3.6.0</td>
<td>43990</td>
<td>38</td>
</tr>
<tr>
<td>PJ SIP-1.12</td>
<td>83916</td>
<td>21</td>
</tr>
</tbody>
</table>
## Evaluation

<table>
<thead>
<tr>
<th>SPDY Implementation</th>
<th>Library</th>
<th>Annotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC can analyze reasonably large code-bases of ~100 kSLOC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The amount of developer input is small with ~10s SLOC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Library</th>
<th>Annotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>eXoSIP-3.6.0</td>
<td>43990</td>
<td>38</td>
</tr>
<tr>
<td>PJSIP-1.12</td>
<td>83916</td>
<td>21</td>
</tr>
</tbody>
</table>
## Results: Summary

<table>
<thead>
<tr>
<th>SPDY</th>
<th>Issues Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>Liberal Sender</td>
<td>4</td>
</tr>
<tr>
<td>Conservative Receiver</td>
<td>4</td>
</tr>
<tr>
<td>Bug</td>
<td>1</td>
</tr>
<tr>
<td>Optional Feature</td>
<td>2</td>
</tr>
<tr>
<td>Unsupported Version</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SIP – Session Initiation Protocol</th>
<th>Issues Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>Liberal Sender</td>
<td>8</td>
</tr>
<tr>
<td>Optional Feature</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: Liberal Sender

INVITE sip:@usc.edu
Example: Liberal Sender

URI standard is ambiguous about empty usernames

INVITE sip:@usc.edu
Example: Liberal Sender

SUBSCRIBE

event="\r\n"

EXOSIP

PJSIP

Background  Motivation  Design  Evaluation  Conclusion
Example: Liberal Sender

eXoSIP doesn't check inputs; mangles message

SUBSCRIBE
event="\r\n"

PJSIP
Example: Optional Feature

SUBSCRIBE

event="aaa"

EXOSIP

SIP signaling SDK

PJSIP

501 Not Implemented
Example: Optional Feature

Fixing and reanalyzing reveals deeper issues!

SUBSCRIBE
event="aaa"

501 Not Implemented
Example: Optional Feature

spdylay

path="/\%00"

400 Bad Request

NGiNX
Example: Optional Feature

Client

**spdylay**

path="/\%00"

400 Bad Request

Application

RESTful API

**NGINX**
Conclusions

- Automated non-interoperability finding with PIC
  - Joint symbolic execution
  - Dispersive exploration
  - Return normalization
  - Bugs found and fixed!

- Future Work
  - Generalized analysis framework: poster + demo!