Finding Bugs in Programmable Data Plane Generators

Jiamin Cao

Yu Zhou, Chen Sun, Lin He, Zhaowei Xi, Ying Liu
Programmable Data Plane Gains Significant Traction

**Intent**
- Routing
- Telemetry
- Security

Network environment
- DSL language

Hardware feature
- Hardware capacity

DSL Program

DSL Compiler

Binary

Programmable Device
Data Plane Generators Make Programming Easier

Intent
- Routing
- Telemetry
- Security

DP Generator
- Network environment
- Hardware feature
- DSL language
- Hardware capacity

DSL Program

DSL Compiler
- Binary
- Binary
- Binary

Programmable Device
Understanding the Correctness of DP Generators

How to guarantee the correctness of DP generators?

Intent Violation

Security Vulnerability

Generator Crash

Intent → DP Generator → DP Program

Inconsistent

Intent → DP Generator → No Output
Existing Work is Not Designed to Debug DP Generators

- **Intent**
- **DP Generator**
- **DP Program**
- **DP Program Verifier**

- **Cannot enumerate all intents**
- **Involve massive manual efforts**

SIGCOMM18 p4v
SIGCOMM18 vera
SIGCOMM20 bf4
SIGCOMM21 Aquila...
Firebolt: Finding Bugs in Programmable DP Generators
Firebolt Workflow

1. Generating Reasonable Intents

2. Generating Representative Intents

3. Deriving Specifications

4. Formalizing DP Programs

5. Correctness Verification

Intent Generation

Program Verification

Output

Input

Semantic constraints

Per-symbol specification

Intent grammar

Intent

Speciﬁcation

Verifier

DP Generator

Z3 Formula

Security vulnerability

Intent violation

Crash bug

Grammar specification
1. Generating Reasonable Intents
Generating Reasonable Intents

Syntax-Guided Intent Generation $\Rightarrow$ Syntactically-Correct Intents

Intent generation graph

1. **Start from start symbol**
2. **Grow graph using expansion rules**
3. **Collect leaf nodes as output intents**

Start Symbol | Expansion Rule
---|---
\langle prog \rangle ::= \langle stmt \rangle
\langle stmt \rangle ::= \langle name \rangle = \langle query \rangle
\langle query \rangle ::= \langle map \rangle | \langle filter \rangle

**BNF Grammar Example**

\[
\langle prog \rangle ::= \langle stmt \rangle
\langle stmt \rangle ::= \langle name \rangle = \langle query \rangle
\langle query \rangle ::= \langle map \rangle | \langle filter \rangle
\]

\[
\langle prog \rangle \rightarrow \langle stmt \rangle \rightarrow \langle name \rangle = \langle query \rangle \rightarrow \langle name \rangle = \langle map \rangle \rightarrow \langle name \rangle = \langle filter \rangle
\]
### Generating Reasonable Intents

**Semantic Constraint Enforcement**

**Semantically-Valid Intents**

#### Invalid Semantics

- **Undefined reference**
  
  ```
  def R2 = func1 (R1, ...);
  // R1 is not defined
  ```

- **Repeated definition**
  
  ```
  def R2 = func1 (...);
  def R2 = func2 (...);
  ```

#### Context-Aware Semantic Constraint

- **Dependency-type:**
  
  - if \( \exists \langle r1 \rangle \text{ on } \langle n1 \rangle, \exists \langle r2 \rangle \text{ on } \langle n2 \rangle \)
  
  - variable declaration
  
  - variable definition

- **Exclusion-type:**
  
  - if \( \exists \langle r1 \rangle \text{ on } \langle n1 \rangle, \nexists \langle r2 \rangle \text{ on } \langle n2 \rangle \)
  
  - variable definition
  
  - (same) variable definition
Firebolt Workflow

Intent Generation

1. Input

Intent grammar

Semantic constraints

Per-symbol specification

2. Generating Representative Intents

Intent

Intent

Intent

DP Generator

Specification

Verifier

Output

DP Program

Z3 Formula

Crash

Intent violation

Security vulnerability
Generating Representative Intents

1. Boundary Rule (0)

2. Random Rule (327)

3. Previously selected rules

Wide parameter range (many)

Keep representative rules (few)

Cyclic symbol reference (infinite)

Break symbol recurrence (finite)

\[ \langle N \rangle ::= 0 \sim 65535 \]

\[ \langle S \rangle ::= \langle N \rangle + \langle N \rangle \]

\[ \langle N \rangle + \langle N \rangle \]

\[ \langle S \rangle \]

65536² intents

\[ \langle S \rangle ::= x \mid \langle S \rangle + \langle S \rangle \]

\[ \langle S \rangle + \langle S \rangle \]

\[ \langle S \rangle + \langle S \rangle + \langle S \rangle \]

Based on combinatorial theory, see more in our paper!
Firebolt Workflow

Input

Intent Generation

Intent

Semantic constraints

1. Generating Reasonable Intents

2. Generating Representative Intents

Per-symbol specification

Program Verification

DP Generator

DP Program

Specification

Verifier

Output

Crash bug

Intent violation

Security vulnerability

1. Generating Reasonable Intents

2. Generating Representative Intents

Grammar

Semantic constraints

Per-symbol specification

Output

Crash bug

Intent violation

Security vulnerability
Firebolt Workflow

Input

Intent Generation

Semantic constraints

Program Verification

Per-symbol specification

Intent grammar

Intent

3. Deriving Specifications

DP Generator

Specification

Verifier

DP Program

Z3 Formula

Output

Crash

bug

Intent violation

Security vulnerability

Grammar

Semantic constraints

Spec

Per-symbol specification

Compilation

Verifying

Specifying

Generating

Input

Intent

Program Verification

Output

Crash

bug

Intent violation

Security vulnerability
**Deriving Specifications Automatically**

**Observation**

Intents are generated by expanding grammar symbols

**Key idea**

Write specifications of each grammar symbol

Compose symbol specifications

**General and flexible format:**

```
expression :: field == value
```

```
def DEC_FUNC:
    counter = 0

def EXEC_FUNC:
    return (field).exec() == (value).exec()
```

```
def DEC_FUNC:
    counter = 0

def EXEC_FUNC:
    if (expression).exec() == true:
        counter = counter + 1
    return counter
```

```
def DEC_FUNC:
    counter = 0

def EXEC_FUNC:
    if (pkt.tcp.sport == 80) == true:
        counter = counter + 1
    return counter
```

**A simple example (counter)**

```
def DEC_FUNC:
    counter = 0

def EXEC_FUNC:
    if (expression).exec() == true:
        counter = counter + 1
    return counter
```

```
def DEC_FUNC:
    counter = 0

def EXEC_FUNC:
    return (field).exec() == (value).exec()
```
Firebolt Workflow

Input

Intent Generation

Semantic constraints

Per-symbol specification

Program Verification

Intent grammar

Intent

DP Generator

4. Formalizing DP Programs

Specification

Verifier

Output

DP Program

Z3 Formula

Crash bug

Intent violation

Security vulnerability
**Formalizing DP Programs**

Each Programmable Block
*(parser, ingress, egress, deparser, etc.)*

Z3 Formulas

**Example: Match action tables with entries**

```
action a (p) {x = p;}

table t {
  key = k : exact;
  actions = {
    a; no_op;
  }
  default_action = no_op;
}

Table entries: 1 => (1)
```

**Input parameter**
- Match key of table t

**Free Z3 variables**
- \( (_\text{BitVec}\ 32)\) k

**Output parameter**
- Variable y

**Output Z3 expression**
- \( (_\text{BitVec}\ 32)\) y = (if (k=1) 1 else y))
**Firebolt Workflow**

1. **Intent Generation**
   - Input grammar
   - Intent

2. **Program Verification**
   - DP Generator
   - DP Program
   - Z3 Formula

3. **Output**
   - Crash bug
   - Intent violation
   - Security vulnerability

4. **Semantic constraints**
   - Intent

5. **Per-symbol specification**
   - Verification
Evaluation

Implementation
• Ubuntu 16.04 virtual machine with 4GB RAM and two 2.3GHz CPU cores
• ~2000 lines of Python/C++ code (built atop OSDI20-Gauntlet)
• 3 open-source DP generators under test
  • SIGCOMM16-Marple, SIGCOMM17-Sonata, USENIX20-Poise
• 2 DP program verification tools for comparison
  • SIGCOMM21-Aquila, SIGCOMM18-p4v

Evaluation Goals
• Bug Coverage
  • How many bugs can Firebolt find?
• Scalability
  • How long does it take to find bugs?
  • How many human efforts does Firebolt take?
Bug Coverage

How many bugs can Firebolt find?

<table>
<thead>
<tr>
<th>DP Generator Under Test</th>
<th># Generated Intents</th>
<th># Detected Bugs / # Intents Causing Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Crash Bug</td>
</tr>
<tr>
<td>Marple</td>
<td>7341</td>
<td>1 / 12</td>
</tr>
<tr>
<td>Sonata</td>
<td>7912</td>
<td>0 / 0</td>
</tr>
<tr>
<td>Poise</td>
<td>2362</td>
<td>0 / 0</td>
</tr>
</tbody>
</table>

- Detect bugs in all three generators
- Detect altogether 5 security vulnerabilities, 13 intent violations, 1 crash bug
# Scalability

How many human efforts does Firebolt require? **O(100) LoC**

How long does it take? To debug a DP generator? **< 25 minutes**

<table>
<thead>
<tr>
<th>DP Generator Under Test</th>
<th>Human-written LoC</th>
<th>Running Time (Total / Average)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Intent Grammar</td>
<td>Semantic Constraints</td>
</tr>
<tr>
<td>Marple</td>
<td>93</td>
<td>70</td>
</tr>
<tr>
<td>Sonata</td>
<td>34</td>
<td>10</td>
</tr>
<tr>
<td>Poise</td>
<td>25</td>
<td>25</td>
</tr>
</tbody>
</table>

Can Firebolt save human efforts? **0.1% to 0.01%**

<table>
<thead>
<tr>
<th>DP Generator Under Test</th>
<th>Verifying One Program</th>
<th>Verifying All Programs</th>
<th>Finding All Bugs (1 Bug / 1 Program)</th>
</tr>
</thead>
<tbody>
<tr>
<td>p4v</td>
<td>O(1K)</td>
<td>O(1M)</td>
<td>O(10K)</td>
</tr>
<tr>
<td>Aquila</td>
<td>O(100)</td>
<td>O(100K)</td>
<td>O(1K)</td>
</tr>
<tr>
<td>Firebolt</td>
<td>O(100)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

• Firebolt is the first tool designed to debug DP generators
  o Thoroughly explore the intent space to generate syntactically-correct, semantically-valid, and representative intents
  o Automatically verify DP programs by formalizing programs and producing specifications
  o Achieve high bug coverage and high scalability on three DP generators
Thanks for your interest in Firebolt

cjm21@mails.tsinghua.edu.cn