Challenges in Softwarized Communication Systems

- Software plays an increasingly important role in networking
  - Protocols, billions of apps, etc.
  - Network elements become flexible (SDN, NFV, In-network processing)
- Important: Analysis of real code – not models

Goal: devise a new methodology for
Software Analysis of Interacting Systems?
Rigorous, automated and effective!
State of the Art in Distributed Systems Testing

Testbeds, Prototypes
Emulation

Random testing

Simulation

Model-based proofs

Automatism
Coverage
Effectiveness

Automatism
Coverage
Effectiveness

Automatism
Coverage
Effectiveness

Automatism
Coverage
Effectiveness

Traditional
Symbolic Execution

http://comsys.rwth-aachen.de
Symbolic Execution: A Simple Example

```c
int get_range(int x) {
    if (x == 0)
        return blue();
    if (x < 50) {
        if (x > 10)
            return red();
        return green();
    }
    return orange();
}
```

But, is Symbolic Execution able to analyze *networked* systems?

---

Symbolic Execution and Networked Systems

- Symbolic analysis of networked systems?

Additional influence factor:
- Consider packets as additional input
- Symbolic analysis of networked systems?

Additional influence factor:
- Consider packets as additional input
- It may arrive at various/any times

Need for a rigorous analysis of any input at any time

Symbolic Analysis of Network Input
Symbolic Execution of Networked Systems

- Symbolic analysis of network input

Symbolic Distributed Execution (SDE)

Branching within a node causes branches in all other nodes
SDE: State Explosion

- **Test scenarios**
  - Grid with $n^2$ nodes (example: 49)
  - Transmissions via a static path
  - Symbolic network failures
  - 10s simulated time

- **Results using the conservative approach (49 nodes)**
  - The basic implementation of the formal model of SDE is not scalable
  - Results using conservative and lazy forking algorithms

SDE: Elimination of Redundant States

- **Test scenarios**
  - Grid with $n^2$ nodes (example: 49)
  - Transmissions via static path
  - Symbolic network failures
  - 10s simulated time

- **Results using conservative and lazy forking algorithms**
  - Significant elimination of duplicates enables much more scalable analyses
Liveness of a Protocol – Infinite Loop Detection

- **Why are infinite loops an issue with protocols?**
  - The outmost (protocol) loop should run infinitely (intended loop)
  - The input handler should always finalize (non-intended loop)
  - Infinite inner loop is a bug

- **When is a loop infinite?**
  - If it comes to the same state, again and again!
    - maybe with (different) intermediate steps

- **When is a loop erroneous**
  - If it does not consume any input any more?

- **How can we detect re-occurring same states?**

Source: baynote.com
Efficient Implementation of Same State Detection

- Two states are the same if all their memory is the same
  - Including call stack and instruction pointer
- Compare each new state $S_x$ to all its predecessor states
  - How can this be achieved efficiently?

1. bool x = false;
2. x = true;
3. while (x) {
   4. x = true;
5. }

Efficient same state detection?
- Compare hashes instead of states
- Hash after blocks, not instructions

memcmp(NULL1, NULL2, 0xFF...FF);

For every predecessor state
Naïve implementation is prohibitively expensive!
Efficient Implementation of Same State Detection

- Two states are the same if all their memory is the same
  - Including call stack and instruction pointer
- Compare each new state $S_X$ to all its predecessor states
  - How can this be achieved efficiently?

```
1    bool x = false;
2    x = true;
3    while (x) {
4      x = true;
5    }
```

Efficient same state detection?
- Compare hashes instead of states
- Hash blocks, not instructions
- Compute hashes iteratively

Results
- So far, a total of seven previously undetected bugs were detected
  - Five bugs in the GNU Coreutils
    - e.g. in “tail”: 130 line while(1) loop calling 2 functions
  - Two bugs in busybox
    - e.g. In a 490 line while(1) loop calling 2 functions
  - All bugs have been reported, confirmed and fixed
  - The coreutils bugs have existed for over 12 years!
The Next Challenge

Symbolic Analysis
of Temporal Uncertainty

Analyzing uncertain event times – Why is time so important?

- State of system at arrival time of input determines the behavior
- Rigorous analysis requires analysis of all points in time!
- Moreover, time is continuous – not discrete!
Symbolic Time: Symbolic analysis of uncertain event times

Problems
- Time is continuous – not discrete
- Temporal dependencies in code
- Deriving all combinations and dependencies

Challenges
- How to derive temporal equivalence classes?
- How to detect them?
- How to make sure to consider all cases?

Challenges in Softwarized Communication Systems

- Trend: Software plays an increasingly important role in networking
  - Protocols, billions of apps, etc.
  - Network elements become flexible (SDN, NFV, In-network processing)
- Important: Analysis of real code – not models

Reliability!
(bugs, loops)

Predictable?
(performance, resources)
Symbolic Analysis of Protocol / NF Performance

Performance Prediction of Softwarized Network Functions

- **Challenge: Prediction of Processing Effort/Time of a NF**
  - Necessary processing resources?
  - Expected/worst latency?
  - Achievable data rate?
  - Influence among NFs?
  - Are we under attack?
  - …

- **Influence Factors**
  - Code of the NF
  - Input Traffic (Pattern, Volume)
  - CPU Execution
    - Superscalar execution
    - Branch prediction
    - Caching

Achieved throughput per processing effort of a NF
Challenges in Software-Defined Communication Systems

- **Trend:** Software plays an increasingly important role in networking
  - Protocols, billions of apps, etc.
  - Network elements become flexible (SDN, NFV, In-network processing)
- **Important:** Analysis of real code – not models

**Networked Systems (protocols, apps)**

**Latency-critical networked control**

**Reliability!**
- (bugs, loops)

**Predictable!**
- (performance, resources)
Symbolic Analysis of Networked Systems

Klaus Wehrle
Joint work by the COMSYS team

SYMBIOSYS project homepage
→ https://comsys.rwth-aachen.de/research/projects/symbiosys/

http://comsys.rwth-aachen.de    klaus@comsys.rwth-aachen.de