

Review of
**Model Checking Large Network
Protocol Implementations**

Madanlal Musuvathi
Dawson Engler

By
Vamsi Kambhampati
7th March 2006

What problem did the paper address?

Big Picture Problem

Proving correctness of communication protocols used in the Internet

Specific Problem

Verify large network protocol *implementations*

- Verify Transmission Control Protocol (TCP) in Linux Kernel

Why is it hard?

Protocol specification could be wrong!

Explosive state space

- Consider lost packets, re-transmitted packets, reordered packets, timeouts
- Many More ...

Why should we care?

- **Everyone uses the Internet**
 - Online business (e-commerce)
 - Home computing
 - Education
- Incorrect network protocols are harmful!
 - Loss of business (money)
 - Loss of human resources (time)
- User expectations:
 - “Is my network protocol correct?”
 - “Is my transport protocol inefficient?”
 - “Is my network protocol implementations vulnerable to attacks?”
- **Network protocols should be correct**

Approach: Model Checking

- **Prove protocol correctness using formal verification techniques**
 - A *model* of the protocol
 - Protocol implementation (ex: TCP in Linux)
 - A set of *correctness properties* (i.e., the *formal specification*)
 - Expressed as Linear Temporal Logic (LTL) formulas
- **“Do the correctness properties satisfy for *all* possible executions of the protocol?”**

Model Checking Example

```

Process main() {
  in = 0;
  a = 0;
  concurrent {
    inc(); dec();
  }
}
    
```

```

Process inc() {
  while ( in != 0 ) {
    skip;
  }
  if ( in == 0 ) {
    in = 1;
    a++;
    in = 0;
  }
}
    
```

```

Process dec() {
  while ( in != 0 ) {
    skip;
  }
  if ( in == 0 ) {
    in = 1;
    a--;
    in = 0;
  }
}
    
```

- What are the possible values taken by a?
- More specifically, what are the values taken by in?
- LTL formula: $[\] p$, where p is $(in == 0 \mid in == 1)$, and $[\]$ means “always” in LTL grammar

This reads, “*at all times in the execution of the program, in is either 0 or 1*”

Is this true for the above program?

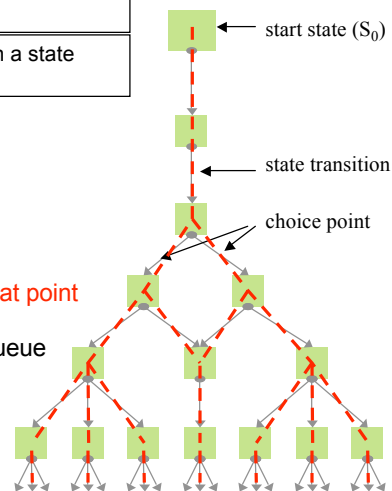
Model Checking Tool

State: The values of variables (globals, locals etc) at a program point

State transition: A possible execution that results in a state change

Algorithm:

1. Start from the initial state S_0
2. Execute a state transition
 - Generates new states
3. Add new states to queue
 - Ignore redundant states (using hash)
4. Check if correctness property holds at that point
 - If property does not hold \rightarrow report error
5. Enable (one of) unexplored state from queue
6. Repeat from step 2, until
 - No more resources left, or
 - All states are explored



The C Model Checker

- The C Model Checker (CMC)
 - Backtracking network simulator
 - Works directly on the implementation of the protocol written in C
- *Optimized* to handle large network protocols
 - Handles large states
 - Hash compaction algorithm
 - Incremental state processing
 - Handles state space explosion problem
 - Incremental heap canonicalization
 - Heuristic based exploration of “interesting” protocol behavior

Conclusions

- Model checking large network protocols possible with CMC
- Linux TCP implementation has 4 bugs!
- TCP specification is ambiguous
- CMC achieves large protocol coverage (92% combined coverage for Linux TCP implementation)

Critique

- Probably the first attempt to model check TCP
- Results are impressive,
 - 4 bugs in Linux TCP
 - 92% protocol coverage
- Assumes the user is aware of a lot of background in model checking
- Should have specified at least a few correctness properties for TCP
- Does not demonstrate the effort needed to model check a protocol

Relation to CS653

- I care 😊
 - Our project is all about model checking network protocols
- Model checking is a form of program analysis (for debugging programs)
- Demonstrates program analysis for *nondeterministic* systems
 - So far we looked at static program analysis for deterministic systems