Automatic Concolic Test Generation with Virtual Prototypes for Post-Silicon Validation

Kai Cong, Fei Xie, Li Lei

Presented by: Jianchao Gao, Dike Zhou

Outline

Background

Concolic Test Generation

Implementation

Experimental Results

Conclusion

Outline

Background

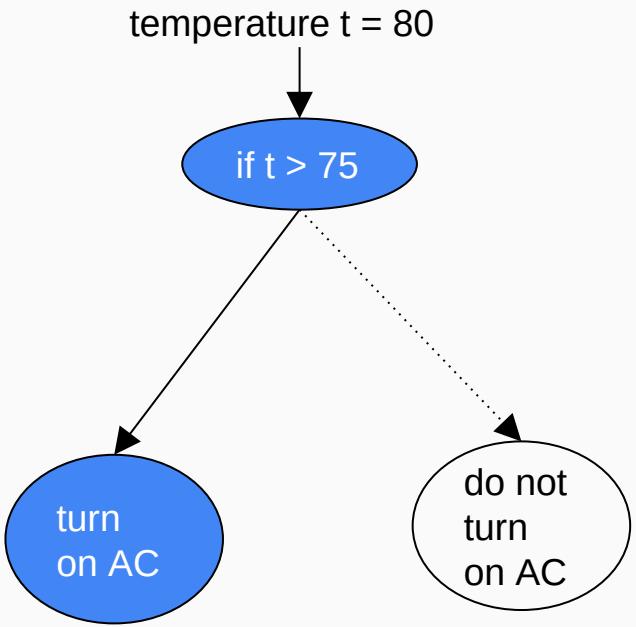
Concolic Test Generation

Implementation

Experimental Results

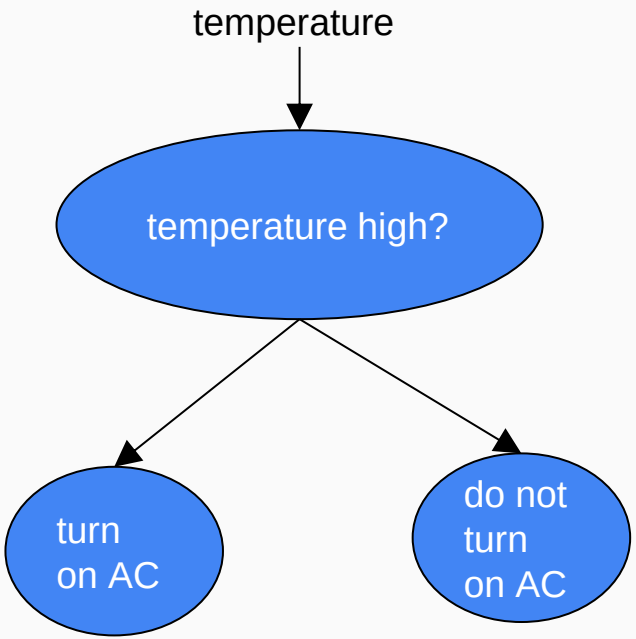
Conclusion

Concolic: CONCrete



Concrete Execution: Executed by concrete values

Concolic: symbOLIC



Symbolic Execution: Executed by symbolic values

KLEE: Symbolic execution engine

Quiz: Symbolic or Concrete?

```
memReq(Req r, Addr a, Val& v) {
        case(LOAD):
        loadFromMem(a, v)
        case(STORE):
        storeToMem(a, v)
}
```

```
foo() {
    Req r, Addr a, Val v
    memReq(r, a, v)
}
```

Ans: Symbolic.

Virtual Prototypes

```
//device state
typedef struct E1000_st {
                …
} E1000State;
 //Interface register function
 static void  write_reg (...) {
                …
 }
 //device transaction function, invoked by reg_func
 static void start_xmit (E1000State *s) {
                …
 }
//environment function
static size_t receive (...) {
                …
}
```

Virtual prototype: fast and fully-functional software models of hardware system

4 fundamental portions

QEMU (Quick EMUlator): generic & open source machine emulator and virtualizer

Virtual devices have better observability and traceability

Outline

Background

Concolic Test Generation

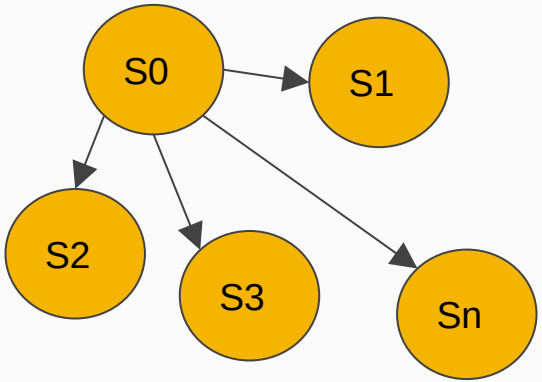
Implementation

Experimental Results

Conclusion

Motivation

Starting point: run symbolic execution from the reset state to explore all states



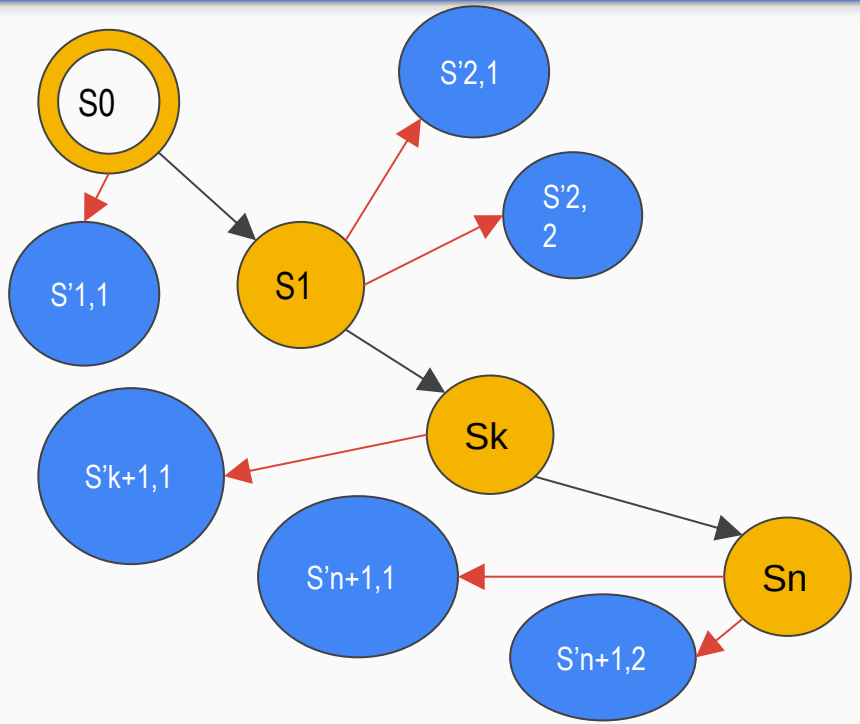
Issue?

How many states exist for a commercial used network adapter chip?

How many symbolic requests are needed to reach a specified state?

Combine concolic & symbolic tests together (concolic) to generate test cases in order to avoid these issues

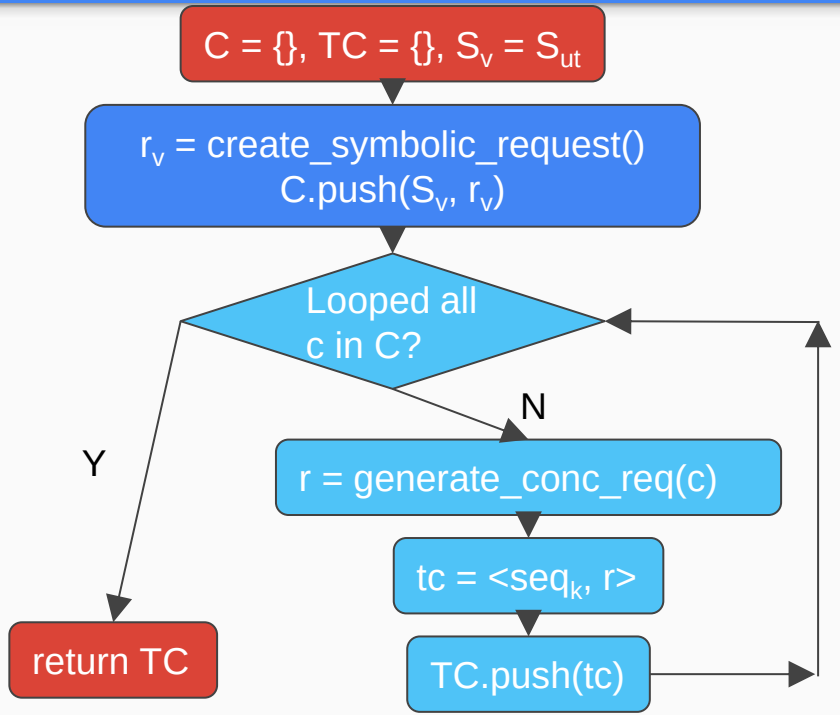
Concolic Test Generation



Apply concrete requests to generate a set of reproducible states ({S0, …, Sn})

- Apply symbolic requests on each states to reach more new states.
- Find the concrete requests that correspond to each transaction.
- Cover all the states along the path

Concolic Test Generation Algorithm



C: set of all computed symbolic executions

TC: set of all generated test cases

s_v: temporary state

r_v: temporary symbolic request

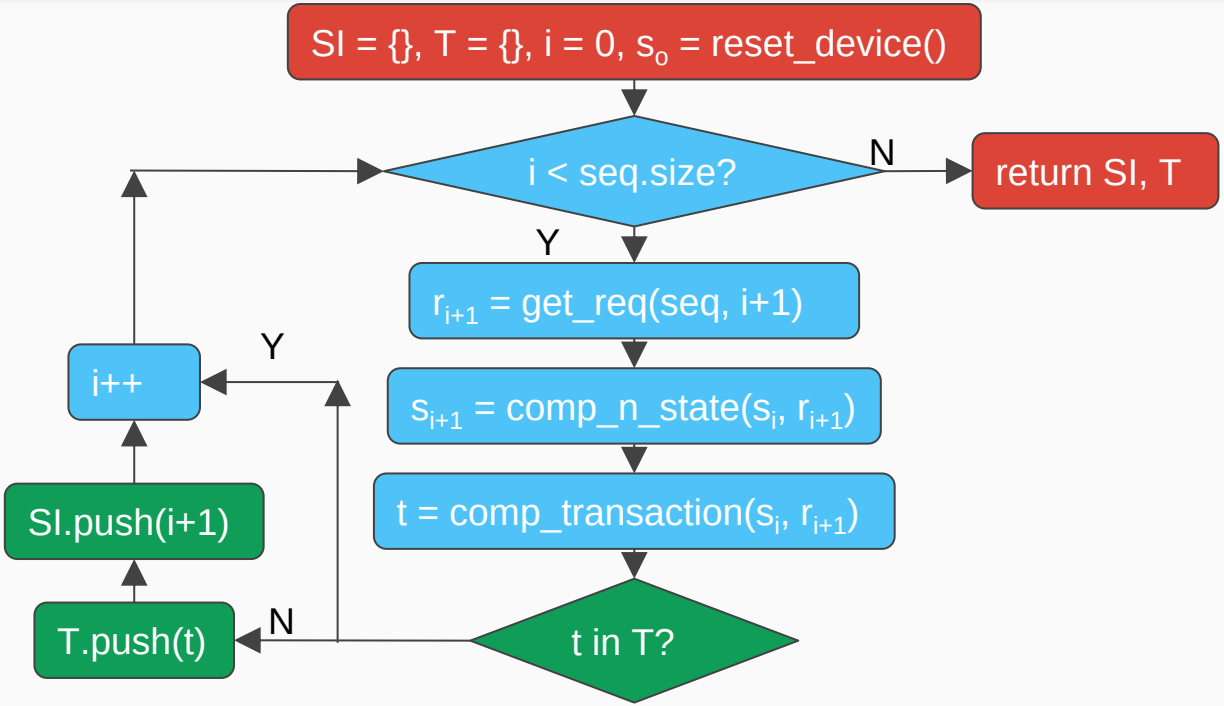
r: temporary concrete request

Transaction-Based Test Selection Strategy

Issues of the methodology:

- Large numbers of states generated from concolic execution
- Test cases that covers the same transactions

Transaction-Based Test Selection Strategy Algorithm



SI: set of indices of all selected states

T: all unique transactions

Outline

Background

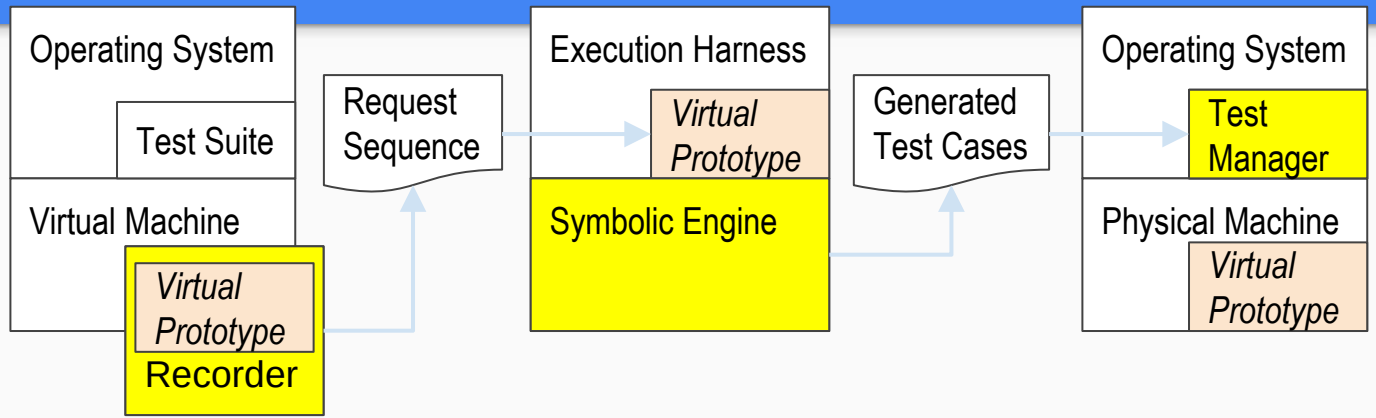
Concolic Test Generation

Implementation

Experimental Results

Conclusion

Framework



Recorder: Captures concrete requests

Symbolic Engine: Explore new states and create concrete test cases

Test Manager: Apply test cases to physical device and check for inconsistency

Harness

```
// Declarations of necessary variables
E1000State state; //Device state
target_phys_addr_t address; //Address
......

int main() {
```

```
// Stub functions
uint16_t net_checksum(uint32_t sum){
        .......
}
```

```
// Stub functions
uint16_t net_checksum_finish(uint32_t sum) {
    ......
}
```

Harness: a driver for the virtual devices

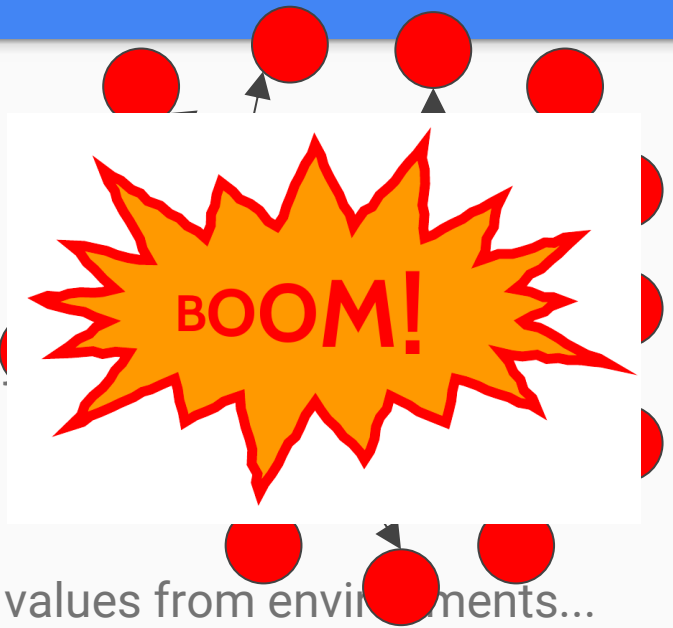
Four components:

1. Declarations of the state variable and parameters of entry functions
2. Code for making the state variable and parameters of entry functions symbolic
3. Non-deterministic calls to virtual device entry functions
4. Stub functions for virtual machine API functions invoked by virtual devices

Symbolic Execution Issues

Two problems

- Path explosion problem
  - Number of path grows exponentiall
- Environment interaction problem
  - API functions resulting in unknown values from envi ments...



Proposed Solutions

Path Explosion Problem

- Loop bound: applied on each loop whose loop condition is a symbolic expression
- Time bound: symbolic execution will terminate in some finite time

Environment Interaction Problem

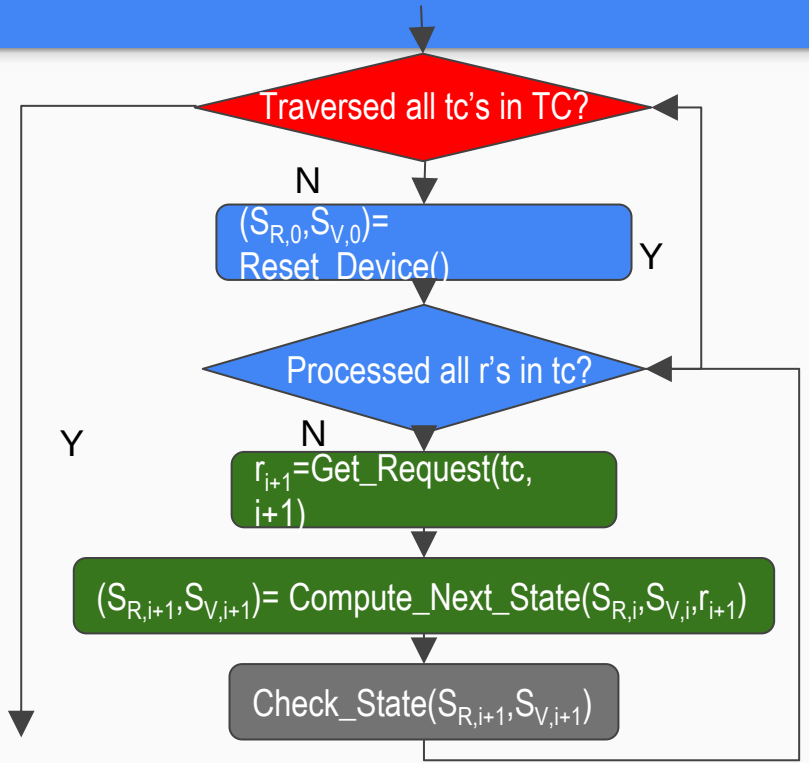
- If API function calls will affect states in virtual device, implement a stub function

Quiz: Coverage

By applying symbolic execution, can we achieve 100% state coverage?

By applying symbolic execution proposed in this paper, can we achieve 100% state coverage?

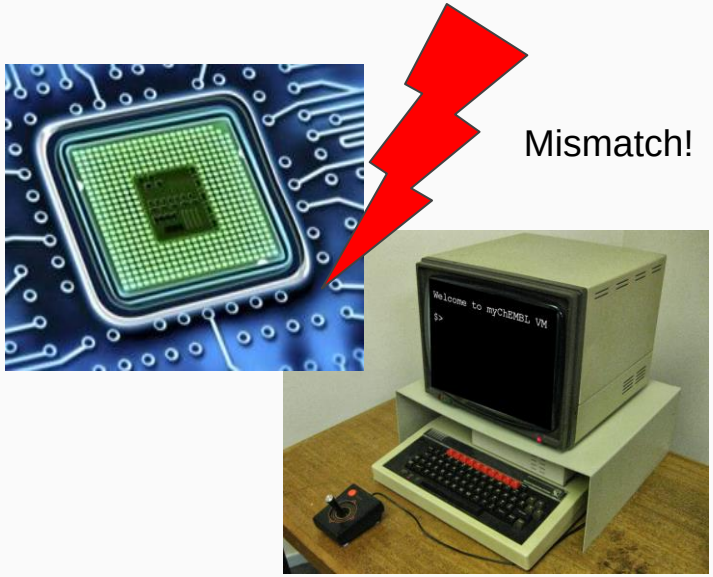
Test with Generated Test Cases



Application of test cases

1. Reset both real and virtual devices to desired state
2. Apply test cases to both devices and capture their concrete next-states
3. Check for any inconsistency

Test Case Replay



Upon the detection of an inconsistency, the triggering test case can be replayed on the virtual device

Better observability and controllability of transactions

Outline

Background

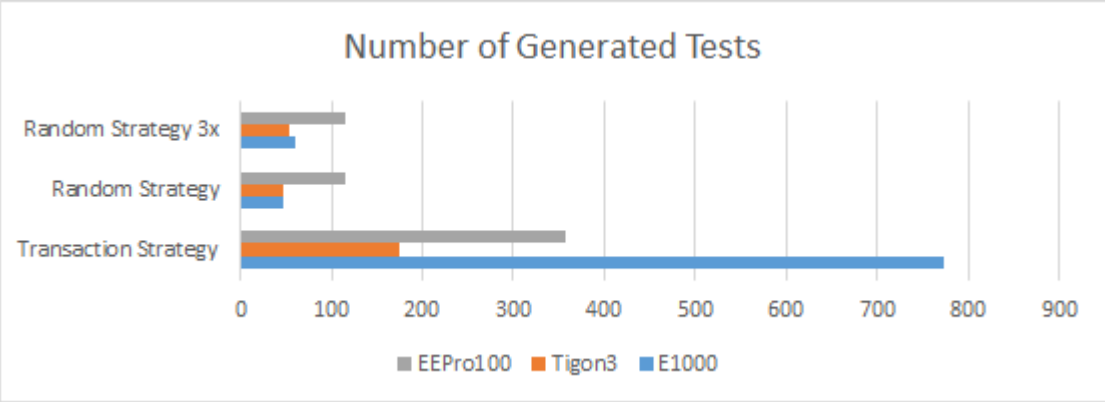
Concolic Test Generation

Implementation

Experimental Results

Conclusion

Compare Transaction-Based Strategy with Random Strategy



	Requests in Trace	Transaction Strategy		Random Strategy			
		States	Tests	States	Tests	States	Tests
E1000	64836	60	774	60	48	180	60
Tigon3	19157	52	175	52	46	156	54
EEPro100	41849	54	357	54	116	162	116

Proposed selection strategy can generate many more tests than random strategy

Time Usage

Time usage of transaction-based selection strategy

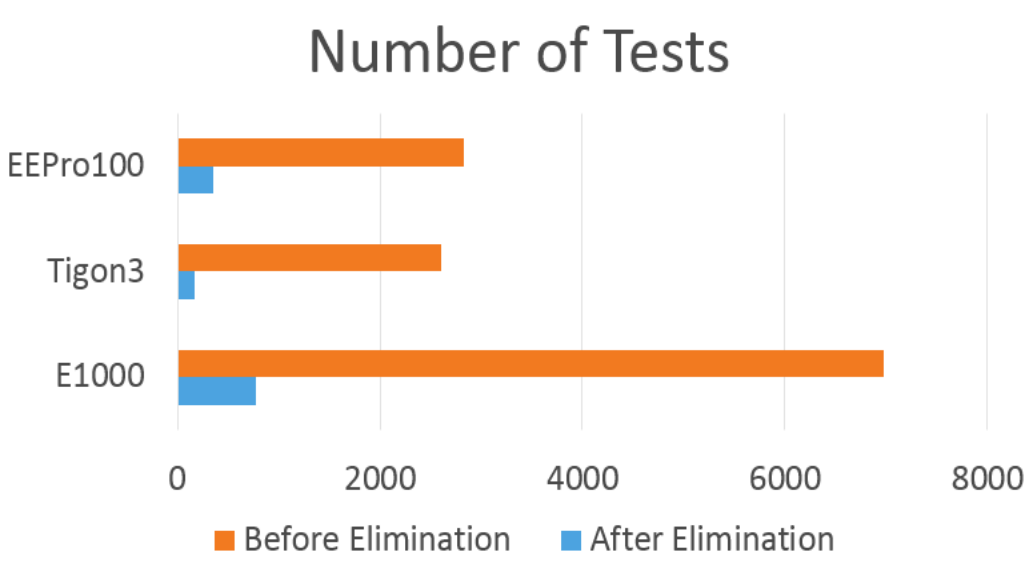
	States	Time(Min)		
		Selection	Generation	Overall
E1000	60	3.5	26.5	30
Tigon 3	52	2	17	19
EE Pro 100	54	2	91	93

Proposed selection strategies takes reasonable amount of time

Only two more new tests found on 6000 states while taking one day of work.

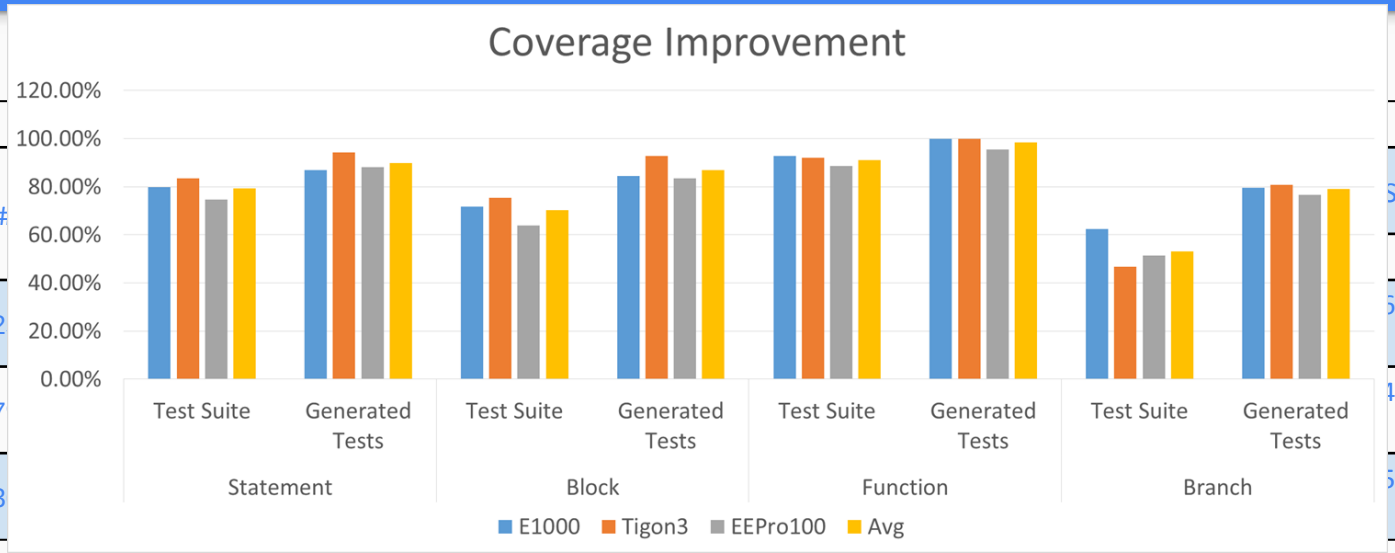
- Not cost-effective to capture all states
- Proposed selection strategy is efficient enough

Test Case Redundancy Elimination



Number of test cases is reduced by one order of magnitude

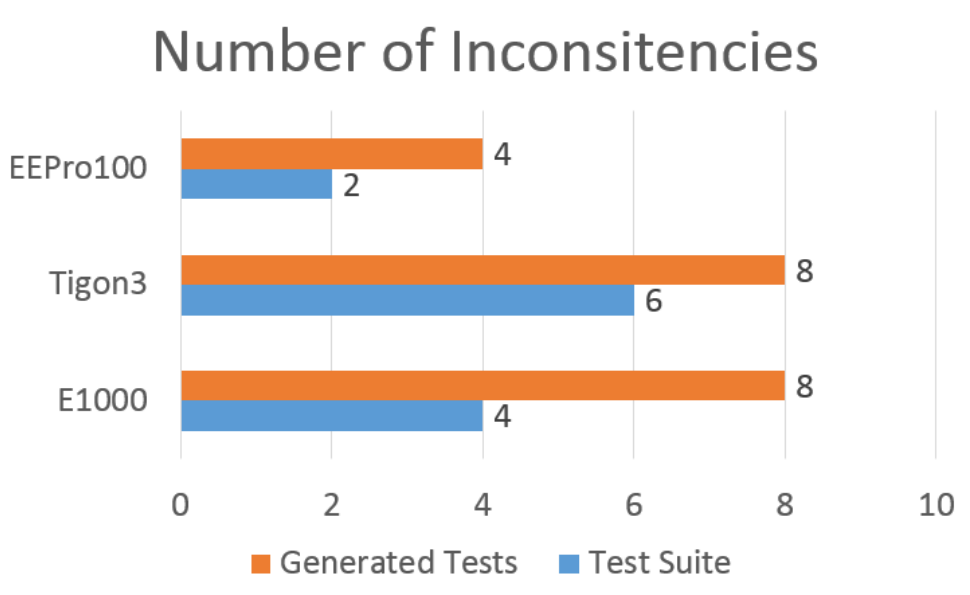
Coverage Improvement



Virtual Prototype	#	Branch			
		Suite	Generated Tests		
		%	#	%	
E1000	32	62.50 %	210	79.55%	
Tigon3	17	46.67 %	97	80.83%	
EEPro100	23	61.33 %	115	76.67%	

Coverage improved in all cases
For E1000 and Tigon3, function coverage can achieve 100%
For Tigon3, branch coverage can be improved by more than 30%

Inconsistencies



Two types of inconsistencies:

- Devices are not initialized correctly
- Devices update reserved registers

Outline

Background

Concolic Test Generation

Implementation

Experimental Results

Conclusion

Conclusion

ACTG with virtual prototypes can leverage observability and traceability of virtual prototypes

This approach can generate effective test cases in a modest amount of time using transaction-based test selection strategy, and improve coverage as well

Several inconsistencies found by tests generated under proposed approach

Questions?

Debate:

1. The proposed method requires a virtual device which should be exactly the same in transactions in order to generate a valuable feedback in replay. This can be very effort consuming. Worth it?

1. Can the proposed method replace ordinary test case selection methods, even though the proposed method cannot cover all possible transactions?

Experiment Setup

Summary of three virtual prototypes

	Virtual Prototype		Harness	
	Lines	Functions	Lines	Entry Functions
E1000	2099	53	74	4
Tigon3	4648	34	80	4
EEPro100	2178	70	85	7

Lightweight harnesses: <100 lines of code

Summary of test suite

Category	***Commands***	***Descriptions***
Basic Programs	ifup	Bring a network interface up
	ifdown	Take a network interface down
	ifconfig	Configure a network interface
	ping	Send ICMP ECHO_REQUEST
	scp	Copy files between network hosts

Capture this test suite from concrete executions of virtual devices in QEMU