Software Engineering Aspects
or Software Engineering, un ami qui vous veut du bien

Ian Welch
Ian.welch@ecs.vuw.ac.nz

2015 Wellington SDN Workshop

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

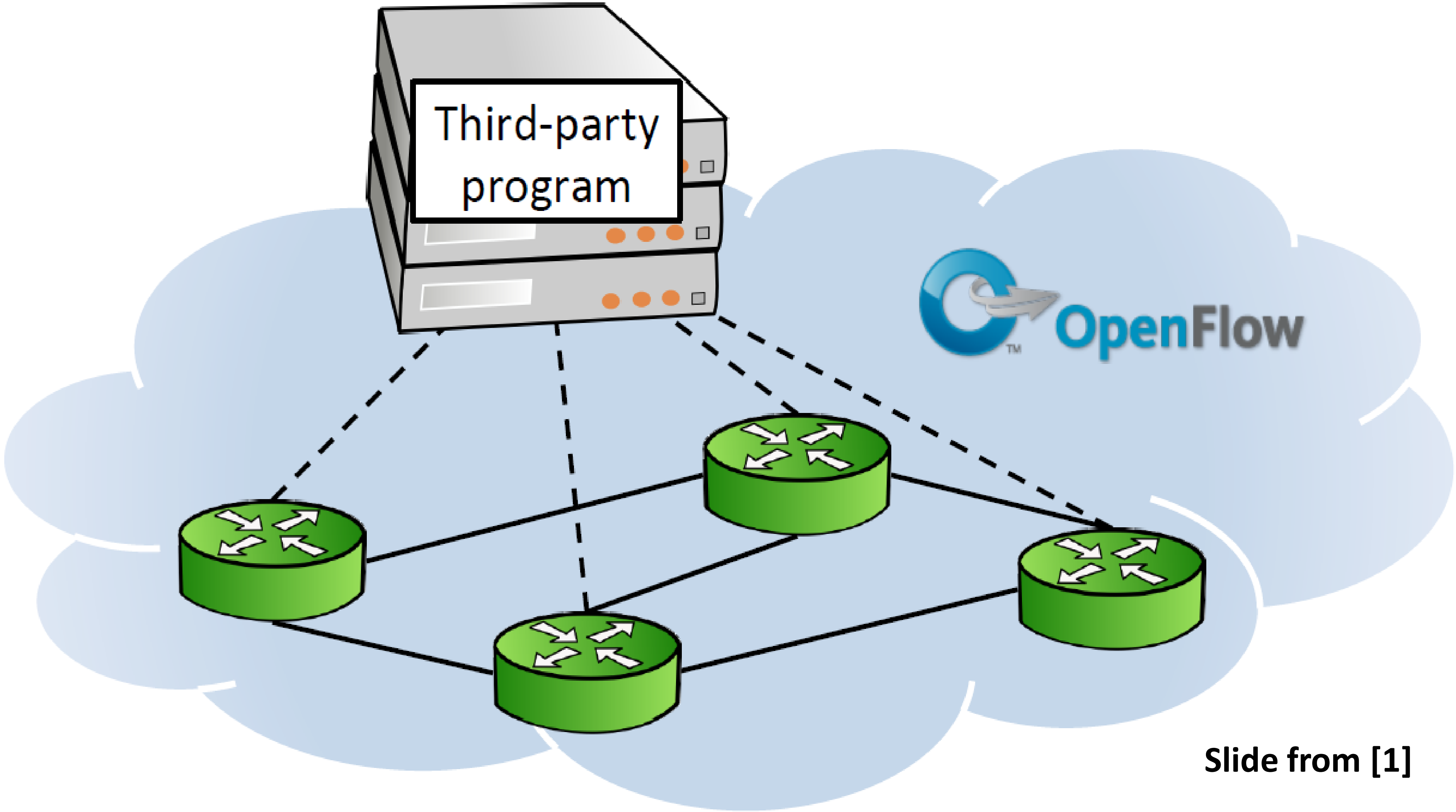
PICA8 OPEN NETWORKING

REANNZ

catalyst

IEEE

Software-Defined Networking (SDN)



Slide from [1]

Enables new functionality through programmability …

… at the risk of bugs

```
               Network Operating System

A fatal exception has occurred at 10.3.0.5/C0011E36 in OF(01) +
00010E36. The current OpenFlow application will be terminated.

* Press any key to terminate the current OpenFlow application
* Press CTRL+ALT+DEL again to restart your network. Your
  users will lose all network connectivity.

                    Press any key to continue
```

Slide from [1]

Example: pyswitch

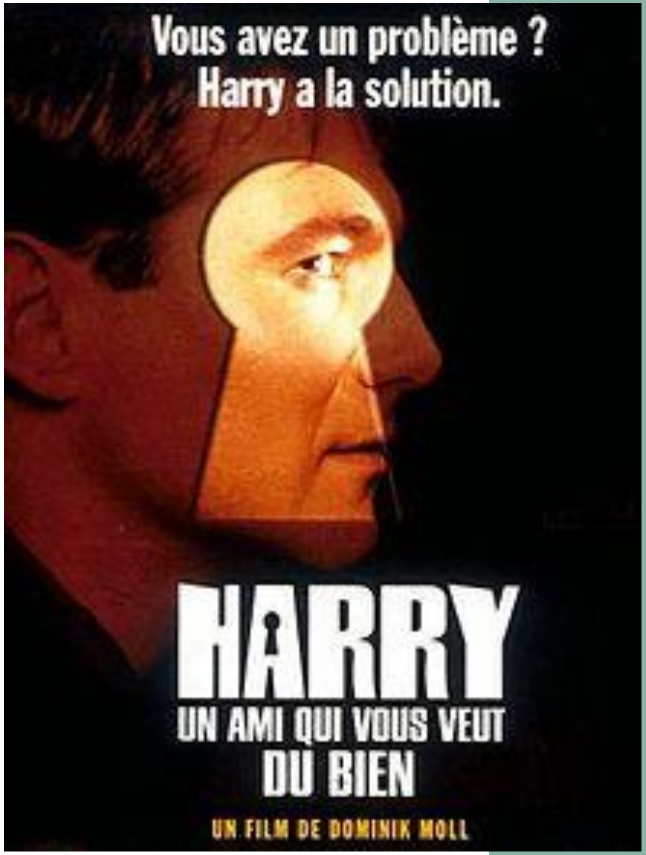
- Consider pyswitch (NOX distribution)
- Only 98 lines of code, what could go wrong?
- NICE automated bug finder found three software flaws missed despite testing by the developers.

Why so hard?

- Human factors, to err is human.
- Program does not execute in isolation.
- Data-plane driven i.e. packet content can change controller behaviour (think QoS applications)
- Complex network behaviour i.e. event ordering (packet arrivals, topology changes) affect program behaviour.
  - Race conditions can arise (inconsistencies between controller's view and actual switch state).
  - LOTS AND LOTS OF STATE THAT AFFECT PROGRAM EXECUTION!

Software Engineering

- Programmable networks may be buggy.
- However, programmable networks mean we can apply software engineering practices.

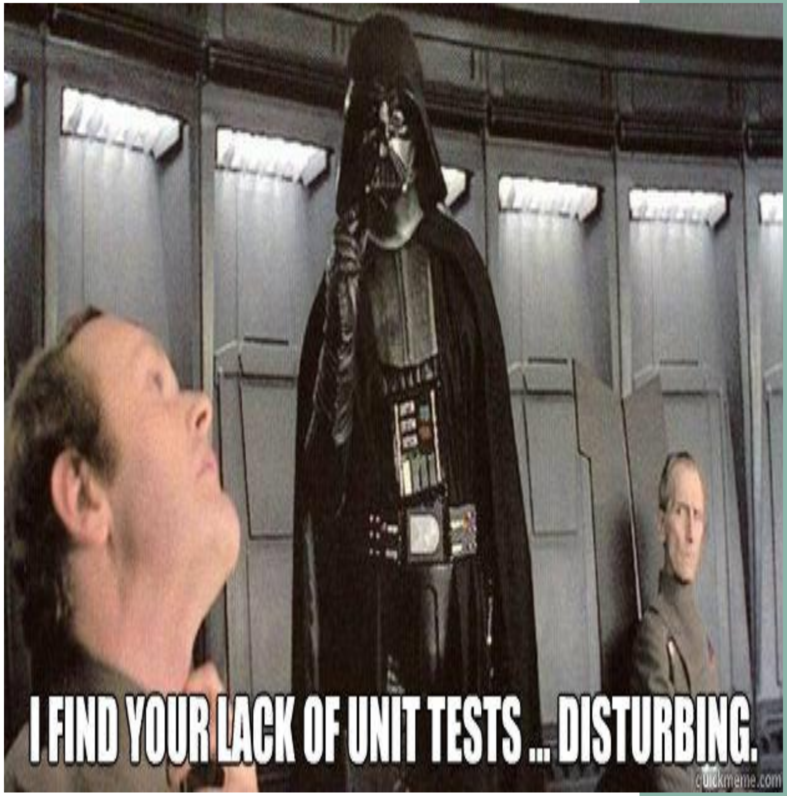


Non-exhaustive list of approaches

- Write tests for the SDN applications:
  - Hand-crafted tests (unit tests)
  - Automated test generation (black-box specification-based, white-box source code based, example-based fuzzing)
- Interactive debuggers (gdb but for networks)
- Domain-specific languages (first-class abstractions, declarative, use of compilers).

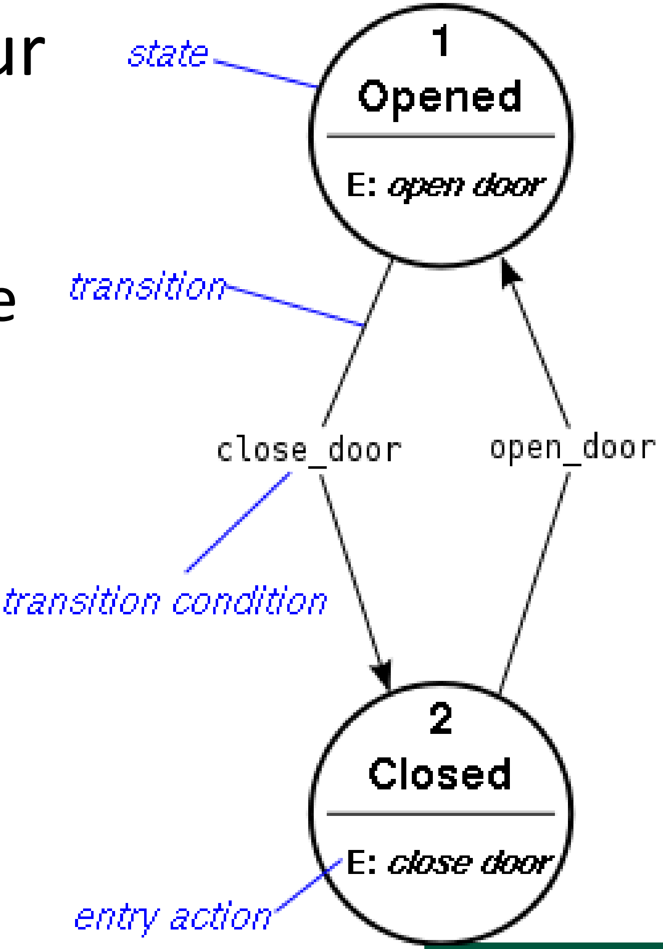
Unit tests

- Focus in the smallest testable parts of the application.
- Ideally write the tests (expected behaviour) as you go along.
- Run the test as part of the compilation process.
- Frameworks exist for python, has been applied to controllers (Ryu).
- Good for implementation errors, less so for finding design flaws.



Automated test generation

1. Specify the expected behaviour (finite state machine) (see [2])
   - Tests = executable paths
   - Generate unit tests that exercise these paths
   - Problem: building the specification
   - Problem: not just the controller but the switches and hosts
2. Extract a specification from code (see [1]):
   - Problem: very detailed models can lead to too many tests



NICE tool (2012)

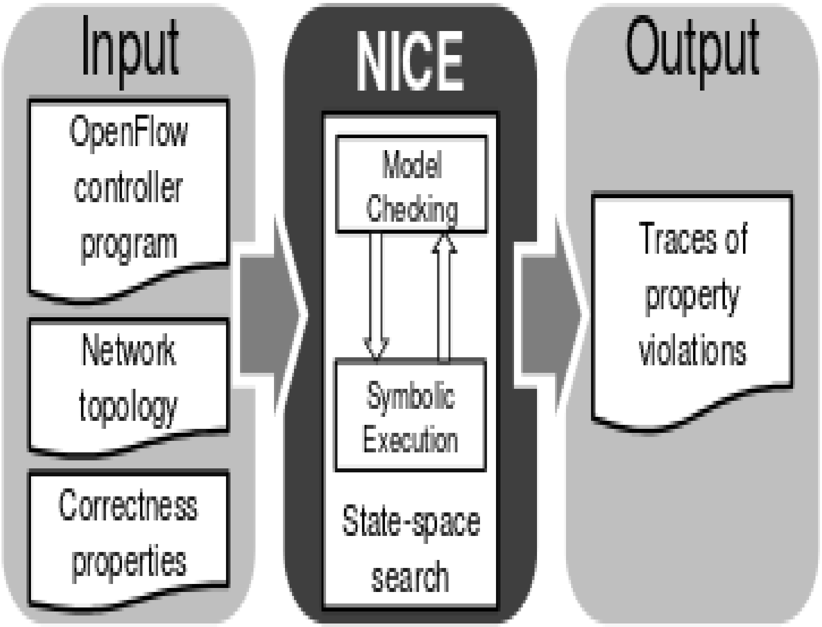


Figure 2: Given an OpenFlow program, a network topology, and correctness properties, NICE performs a state-space search and outputs traces of property violations.

- Correctness conditions -- safety (something bad never happens) and liveness (eventually something good happens).
- Trace allows you to identify what led to correctness being violated
- Found 11 mostly design bugs in 3 real applications.
- http://code.google.com/p/nice-of/
- Cannot guarantee found all bugs but this is same issue as with other testing approaches.
- Scalability is still an issue with this approach, requires access to the source code.

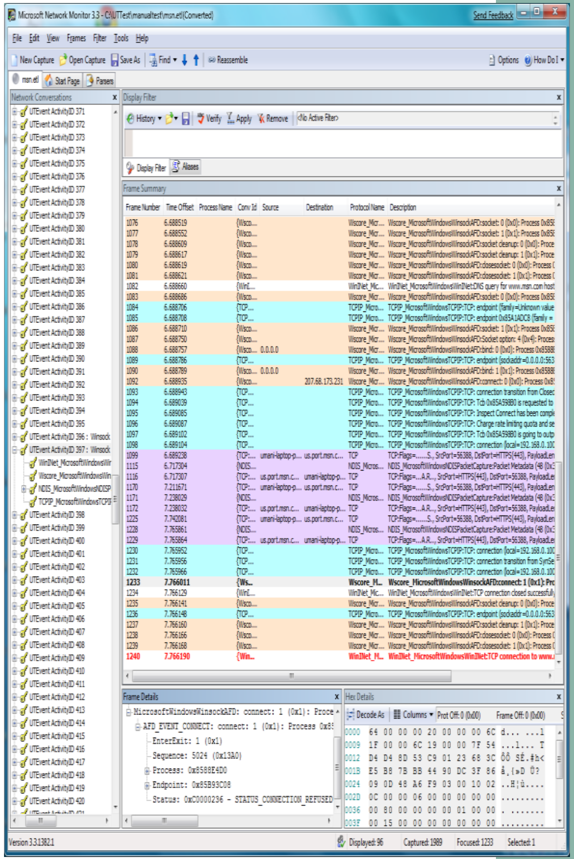
Automated test generation

3. Fuzz testing (see [3]):

- Generate invalid input (from specifications or based upon examples of real inputs).
- Can be used in a black-box manner.
- Specify invariants (“loss of connectivity”, “access control violations”).
- Violation of an invariant indicates a bug, output is a trace of the inputs leading to failure .
- Used by commercial developers of production controllers.
- Problem: Tedious process to analyse the trace because includes irrelevant inputs.

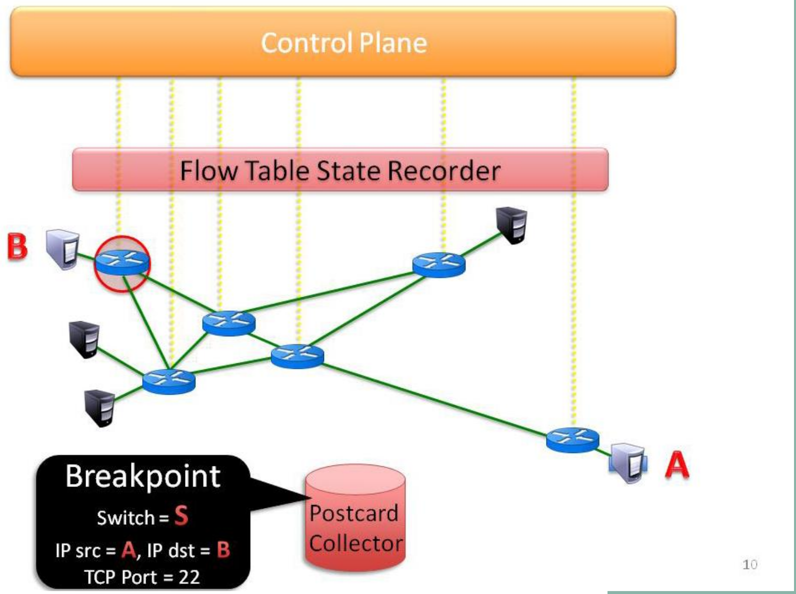
SDN Troubleshooting System (2014)

- Finding the root cause requires ignoring irrelevant events.
- SDN Troubleshooting System (STS) automates finding a minimum set of events.
- Applied to five production controllers.
- Significant reduction in size of traces
  - 1500 events to 2 events.
- Note focus isn't bug finding (fuzzing does that), more diagnosis.
- No need to have access to source code.



Interactive Debugger (2012)

- Most IDEs support interactive debugging.
- ndb – debugger for SDN applications
- Breakpoint:
  - Filter (header, switch)
- Backtrace:
  - Path of flow taken by packet
  - State of flowtable at each switch
- Implementation doesn't require changes to switches but does introduce extra traffic.
- Prototype – several limitations
  - OpenFlow 1.0
  - some table state not captured
  - some problems dealing with duplicate packets generated due to retransmissions



Breakpoint

Switch = S

IP src = A, IP dst = B

TCP Port = 22

Domain Specific Languages

- Language for expressing solutions to problems in a specific domain.
  - Example: make, lex, bison, yacc, regexp …
- Choose abstractions that are more expressive than base language.
  - Embed in existing language (Pyretic – extends python, composition of modules together, enhanced reuse of code that has already been tested) [6]
  - Create a new language (Fat tire 2013 – regexp) [5]
- Declarative DSLs compile down into implementations (less error prone).
  - Example SQL-like language used by Frenetic project, programmer concentrates on what rather than how [6].
  - Compiler technology used here to achieve the how.
- Advantages: reuse, automation, portability.

Summary

- Programmable networks vulnerable to bugs due to programming or design errors that can bring down your network.
- But because programmable we can apply software engineering techniques to avoid, find or prevent these bugs from leading to failure.
- Possibly this might even lead to more reliable networks than non-SDN ones because lack of tools for existing systems to catch configuration errors

Sources

[1] M. Canini, D. Venzano, P. Pereˇś́ ını, D. Kostic, and J. Rexford. A nice way to test openflow applications. In Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), pages 127–140, San Jose, CA, 2012. USENIX. Slides: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/canini

[2] Model Based Black-Box Testing of SDN Applications J Yao, Z Wang, X Yin, X Shi, J Wu, Y Li CoNEXT, 2014 - dl.acm.org

[3] C. Scott, A. Wundsam, B. Raghavan, A. Panda, A. Or, J. Lai, E. Huang, Z. Liu, A. El-Hassany, S. Whitlock, H. Acharya, K. Zarifis, and S. Shenker. Troubleshooting blackbox sdn control software with minimal causal sequences. In Proceedings of the 2014 ACM Conference on SIGCOMM, pages 395–406. ACM, 2014. http://ucb-sts.github.com/experiments

[4] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Maziéres, and Nick McKeown. 2012. Where is the debugger for my software-defined network?. In Proceedings of the first workshop on Hot topics in software defined networks (HotSDN '12). ACM, New York, NY, USA, 55-60.

[5] Mark Reitblatt, Marco Canini, Arjun Guha, and Nate Foster. 2013. FatTire: declarative fault tolerance for software-defined networks. In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking (HotSDN '13).

[6] Nate Foster, Michael J. Freedman, Arjun Guha, Rob Harrison, Naga Praveen Katta, Christopher Monsanto, Joshua Reich, Mark Reitblatt, Jennifer Rexford, Cole Schlesinger, Alec Story, and David Walker. Languages for software-defined networks. IEEE Communications Magazine, 51(2):128-134, 2013