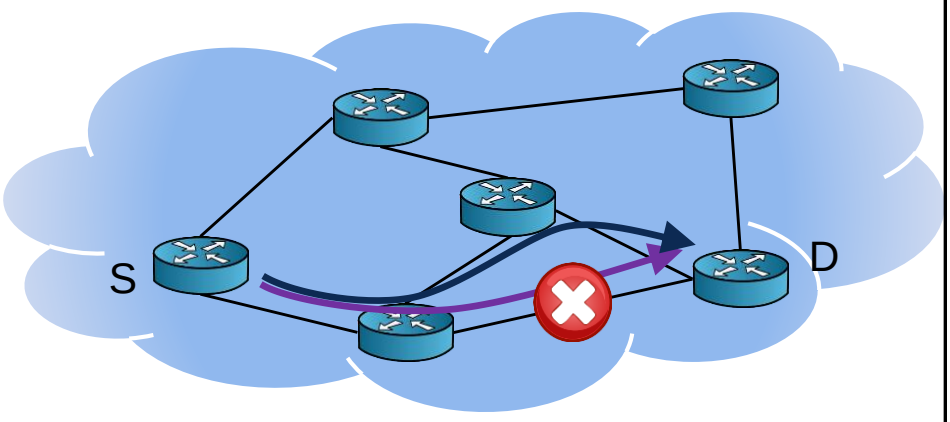
Automatic Failure Recovery for OpenFlow (AFRO)

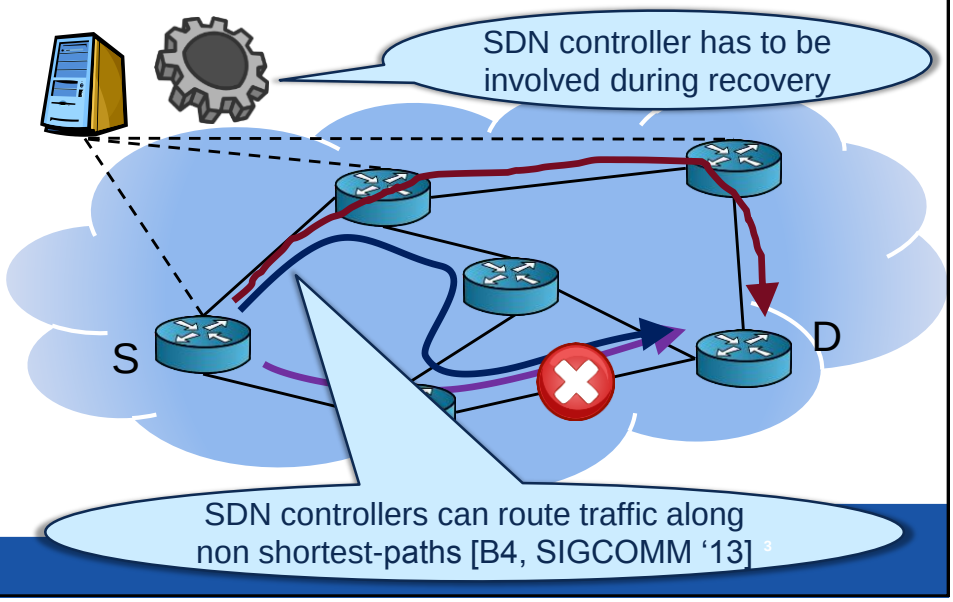
Project 3

Maciej Kuźniar (maciej.kuzniar@epfl.ch)

Network Failures



SDN Failures



Failure Recovery in Controllers

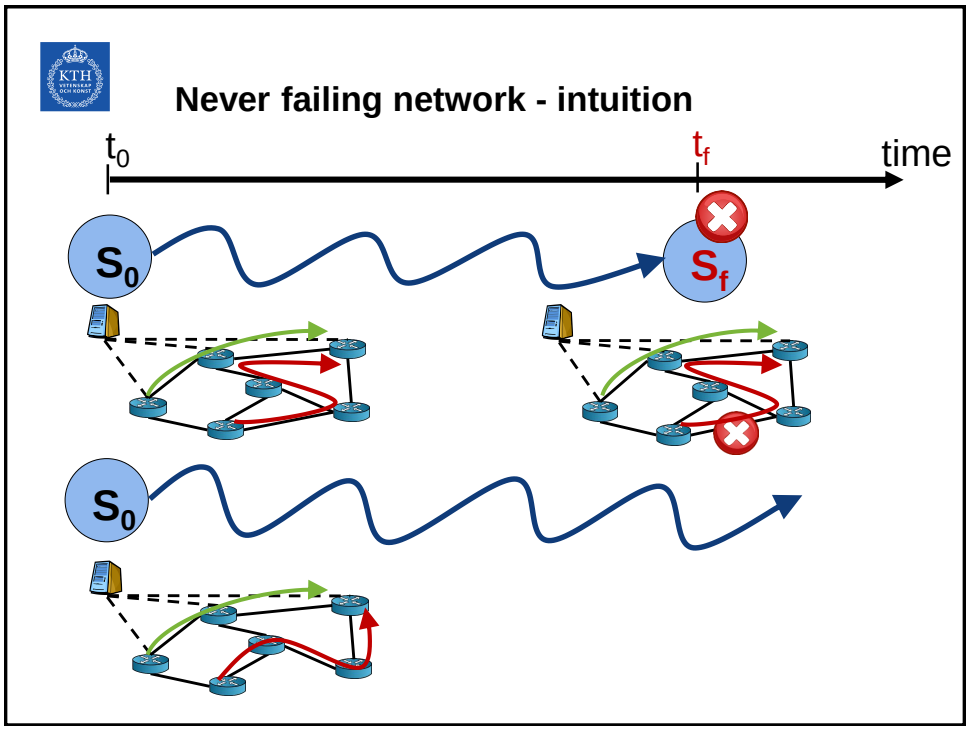
Difficult

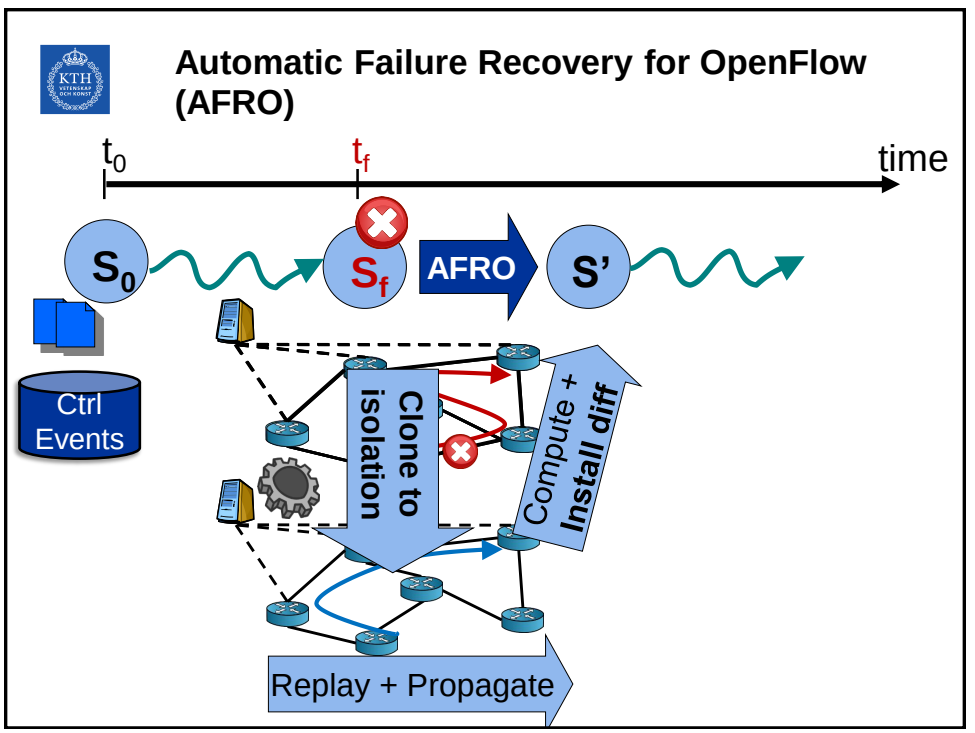
Time consuming

Error prone

Solution:
Decouple failure recovery and application logic

Abstraction of a network that never fails


Never failing network - intuition
t_0
t_f
time
S_0
S_f
S_0


Automatic Failure Recovery for OpenFlow (AFRO)
t_0
t_f
time
S_0
S_f
AFRO
S'
Ctrl Events
Clone to isolation
Compute + Install diff
Replay + Propagate

Project overview

- Implement AFRO as a layer below the controller
- Test the implementation in a network emulator
- Evaluate performance

AFRO as a proxy

- AFRO is a layer between switches and controller
- Intercepts communication
  - Can remove or add messages
- Implement as a transparent proxy in C++
  - Use existing libraries to (de)serialize messages
    - oflib by CPqD (OF 1.3)
    - Flowgrammable (OF 1.0 – 1.4)
    - Controllers like NOX

AFRO - modules

- State tracking – event recording
- Failure detection
- State computation
- Network reconfiguration

- Parallel network emulation

State tracking

- **Intercept**
  - (PacketIn) events coming from switches
  - Messages going to switches
- **Filter** and store relevant events
  - events coming from switches at network edge
- **Expire** events that are no longer relevant
  - Many flows are short
  - Remove event if a flow installed in response to this event is no longer in the network

Failure detection

- Use OpenFlow messages
  - Port Status message
  - Switch disconnecting

State computation

- Replay events in an emulated environment
  - Start a **copy of the controller** in a fresh state
  - Create an **emulated network copy**
    - Build switch models or reuse existing software switches (references at the end)
  - **Inject** events to the emulator and let them **propagate**

Replay

Injecting events in an arbitrary order is wrong

- Order of events should reflect the order in original network
- When an event was processed?
  - Requires full control over switches, links, controller
- Sometimes may be relaxed
  - Parallel replay – when is it possible?

Network reconfiguration

1. **Compute** minimal rule difference
2. **Reconfigure** switches

And

3a. **Copy state** from emulated controller to the real one
Or
3b. Transparently **switch controllers**

Parallel network emulation

- **Multiple emulators** with different network topologies
- Replicate events and inject to all emulators **instantly**
- After failure use state from a correct emulator

- Reuse previous code
- More emulators than one machine can handle
- Requires communication between **many machines**

Testing

- Use Mininet
- Implement a few controllers:
  - With custom failure recovery
  - Without failure recovery
- Compare AFRO and custom recovery

Expected outcomes

- Well documented and working code
  - Easy to deploy
  - No crashes
- Functionality at least equal to Python prototype
- High performance
  - Recording of over 40k events/sec
  - Replay of over 20k events/sec

Not ambitious enough?

Problems that require conceptual work:

- Event filtering strategies
- Parallel replay strategies
- Other OpenFlow events

References

AFRO description:

- http://web.ict.kth.se/~dejanko/documents/publications/afro-tr-jan14.pdf

OpenFlow serialization libraries:

- oflib: http://cpqd.github.io/ofsoftswitch13
- Flowgrammable: http://flowgrammable.org/

OpenFlow software switches:

- reference: http://archive.openflow.org/downloads/openflow-1.0.0.tar.gz
- OVS: http://openvswitch.org
- CPqD: http://cpqd.github.io/ofsoftswitch13