HyperFlow
A Distributed Control Plane for OpenFlow

Amin Tootoonchian
Yashar Ganjali

System and Networking Group
Department of Computer Science
University of Toronto
Brief Overview of OpenFlow

- Root cause of network mgmt. & control complexity:
  - Tight coupling of control and data planes.

- OpenFlow separates the data and control planes:
  - Abstracts switches as programmable flow tables.
  - A *logically centralized* controller programs them.

- But current setups do not scale well.

OpenFlow extremely simplifies network control & mgmt.
A Network with a Single Centralized Controller Does Not Scale.

• Flow setup time for switches farther from controller is larger.

• Single controller can handle a limited number of datapath requests.

• End-to-end control bandwidth is limited.

Network operators need to deploy multiple controllers.
Distributed Control Plane Must Not Sacrifice Simplicity for Scalability!

• Key to OpenFlow’s simplicity:
  • Network control logic centralization.

• Trade-off:
  • Scalability (complete distribution)
  • Simplicity

• Distributed cp should be scalable, yet transparent to the control logic.

A distributed cp must keep network control logic centralized.
Our Approach: Push All State to All Controllers

• Make each controller think it is the only controller.
• Passively synchronize state among controllers.
  – With minor modifications to applications.
• How to synchronize state with minimal modification?
  – Capture controller events which affect controller state.
    • Controller events: e.g., OpenFlow messages (Packet_in_event, ...).
    • The number of such events is very small.
  – Replay these events on all other controllers.
HyperFlow Design

- HyperFlow has two components:
  - Controller component:
    - An event logger, player, and OpenFlow command proxy.
    - Implemented as a C++ NOX application.
  - Event propagation system:
    - A publish/subscribe system.

- Switches are connected to close controllers.

- Upon controller failure:
  - Switches are reconfigured to connect to another controller.
Overview of HyperFlow
HyperFlow Controller Component

• Event logger captures & serializes some ctrl events.
  – Only captures events which alter the controller state.
  – Serializes and publishes the events to the pub/sub.
• Event player deserializes & replays captured events.
  – As if they occurred locally.
• Command proxy sends cmds to appropriate switch.
  – Sends the replies back to the original sender.
Event Propagation System

• The pub/sub system has a network-wide scope.

• It has three channel types:
  – Control channel: controllers advertise themselves there.
  – Data channel: events of general interest published here.
  – Individual controllers’ channels: send commands and replies to a specific controller.

• Implemented using WheelFS, because:
  – WheelFS facilitates rapid prototyping.
  – WheelFS is resilient against network partitioning.
Are Controllers in Sync?

• How rapidly can network changes occur in HF?
  – Yet guarantee a bounded inconsistency window.
• The bottleneck could be either:
  – The control bandwidth.
  – The publish/subscribe system.
• The publish/subscribe system localizes the HyperFlow sync traffic.
  – The control bandwidth problem could be alleviated.

How many events can HF exchange with pub/sub per sec?
How Frequent Can a Network Change?

• Benchmarked WheelFS:
  - The number of 3KB-sized files HF can serialize & publish:
    • 233 such events/sec → not a concern (multiple publishers)
  - The number of 3KB-sized files HF can read & deserialize:
    • 987 such events/sec.

• However, HF can handle far larger number of events.
  - During spikes inconsistency window is not bounded.

No. of network changes on avg must be < 1000 events/sec.
Summary

• HyperFlow enables deploying multiple controllers.
  – Keeps network control logic centralized.
  – Yet, provides control plane scalability.

• It synchronizes network-wide view among controllers.
  – By capturing, propagating & playing a few ctrl events.

• It guarantees bounded window of inconsistency:
  – If network changes occur at a rate < 1000 event/sec.

• It is resilient to network partitioning.

• It enables interconnection of OpenFlow islands.
Current/Future Work

• We designed OpenBoot to bootstrap controller state very quickly.
  – Uses checkpoint/restart + event logging
  – Enables rapid recovery from controller failures.
  – Enables adaptive control plane scaling.
  – Enables continuous control plane operation.

• Improvements to the publish/subscribe system.
• Evaluation on a large testbed with realistic data.
Thanks for your attention.

Questions?
amin@cs.toronto.edu