CSC 2229 – Software-Defined Networking

Handout # 4:
Scaling Controllers in SDN - HyperFlow

Professor Yashar Ganjali
Department of Computer Science
University of Toronto

yganjali@cs.toronto.edu
http://www.cs.toronto.edu/~yganjali

Joint work with Amin Tootoonchian.

Announcements

- Final project guidelines and schedule posted on class website.
  - Groups of 1 to 3 students

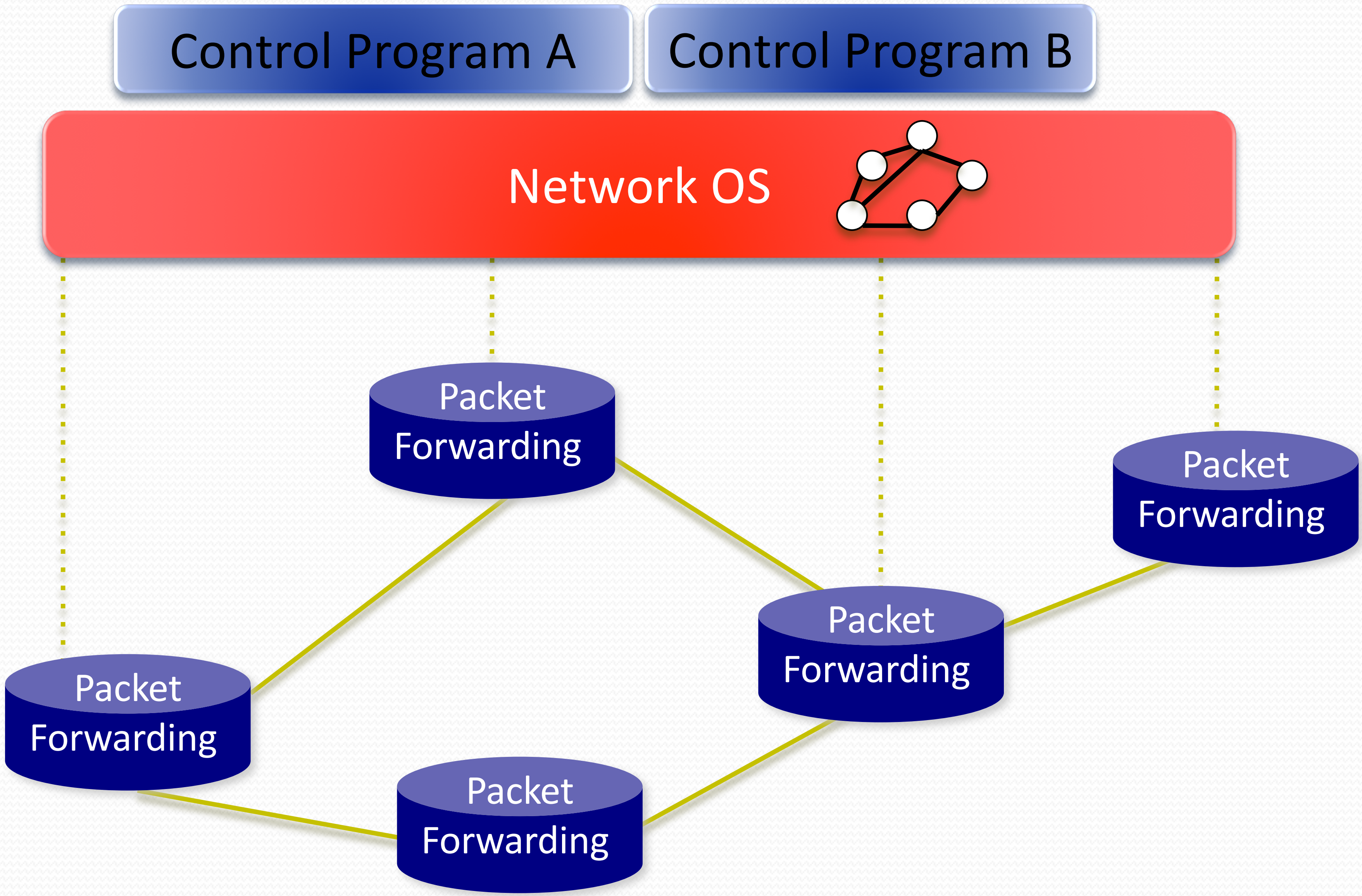
- Project proposal (2 pages + references)
  - Due: Fri. January 31 (5PM)
  - Skim 15-16 papers, quickly read 4-5, deep read 2-3
  - Select a problem, question the assumptions, evaluations…
  - Come and talk to me during the office hour

Announcements

- In class presentations
  - In two weeks?
  - In three weeks from now?
  - Email me with your preferences

- Papers [3] and [4] posted
  - Please read before the next class

Software Defined Network (SDN)



Network OS

Network OS: distributed system that creates a consistent, up-to-date network view

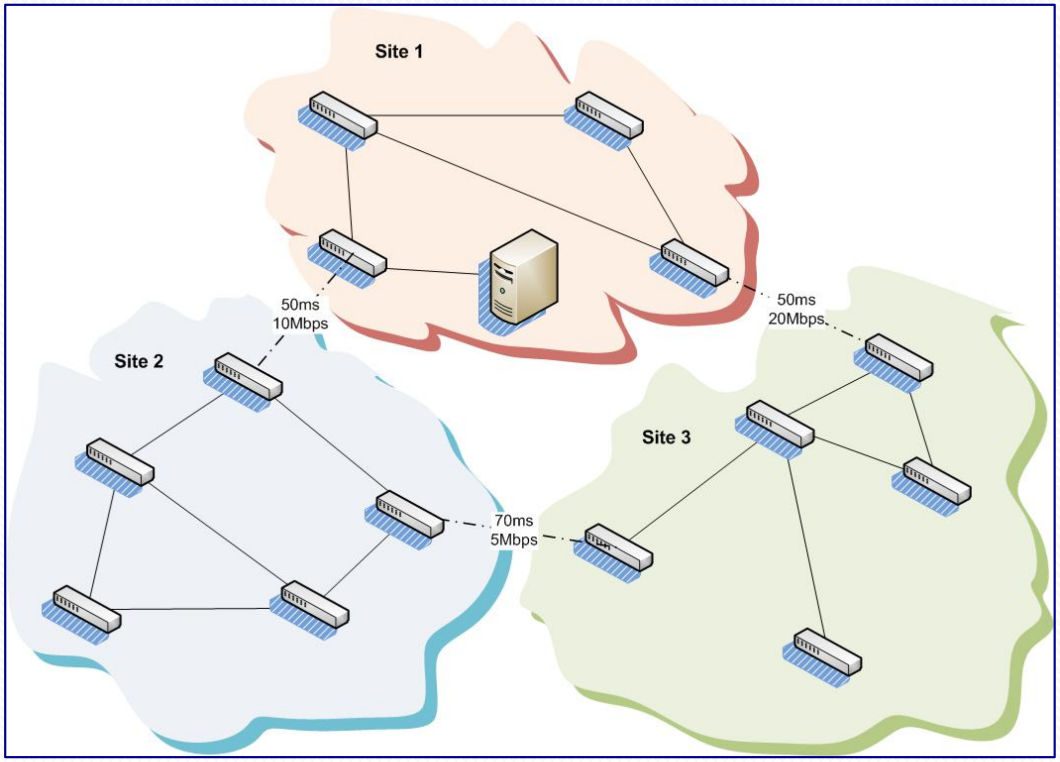
- Runs on servers (controllers) in the network

Control program: operates on view of network

- **Input**: global network view (graph/database)
- **Output**: configuration of each network device

Scalability in Control Plane

- Original SDN design suffered from scalability problems

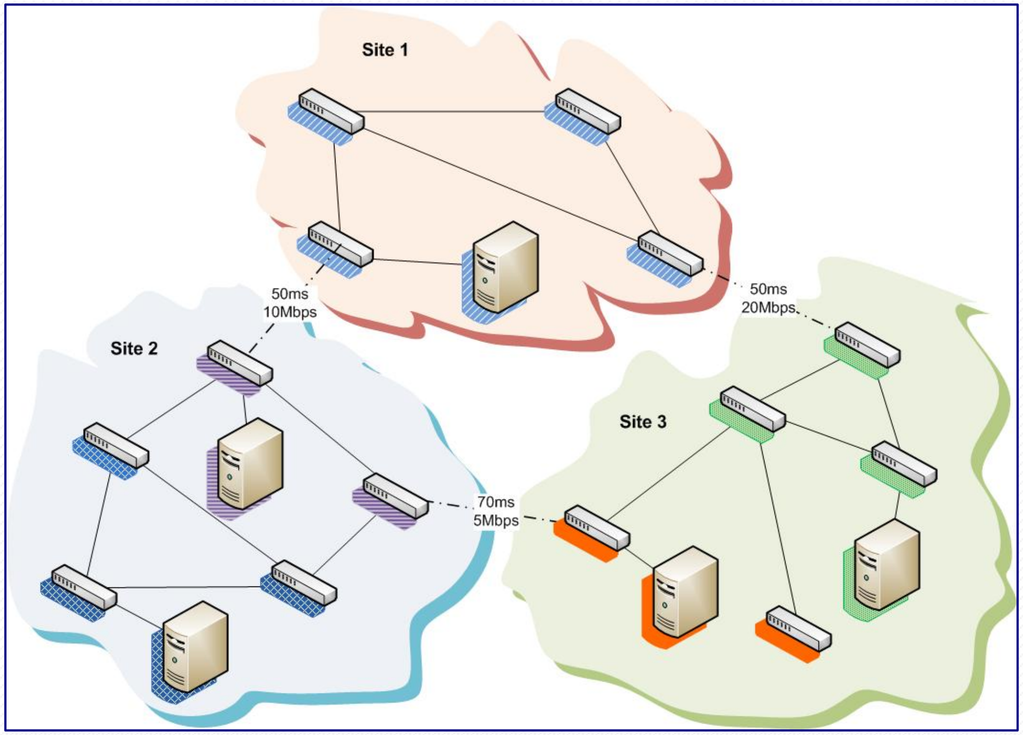


- Network growth →
  - More traffic; and
  - Higher CPU load for controller
- Geographic distribution →

Trivial solution: deploy multiple controllers!

Simplicity vs. Scalability

- Trade-off between
  - Simplicity
  - Scalability
- OpenFlow had chosen simplicity
  - Network control logic is centralized
- **Question**: Can we have the best of both worlds?



Idea: distributed the control plane but keep the network control logic centralized.

Why Not Direct Synchronization?

- Requires significant modifications to controller apps.
- The synchronization procedure must handle state conflicts.
- Synchronization traffic grows with the number of apps.

HyperFlow

- Our approach:
  - Push all state to all controllers
  - Make each controller think it is the only controller
- Synchronize state among controllers
  - With minor modifications to applications
- **Key idea**: capture controller events which affect controller state
  - Controller events: e.g., OpenFlow messages (Packet_in_event, …)
- **Observation**: the number of such events is very small
- Replay these events on all other controllers

Improved scalability, preserved simplicity.

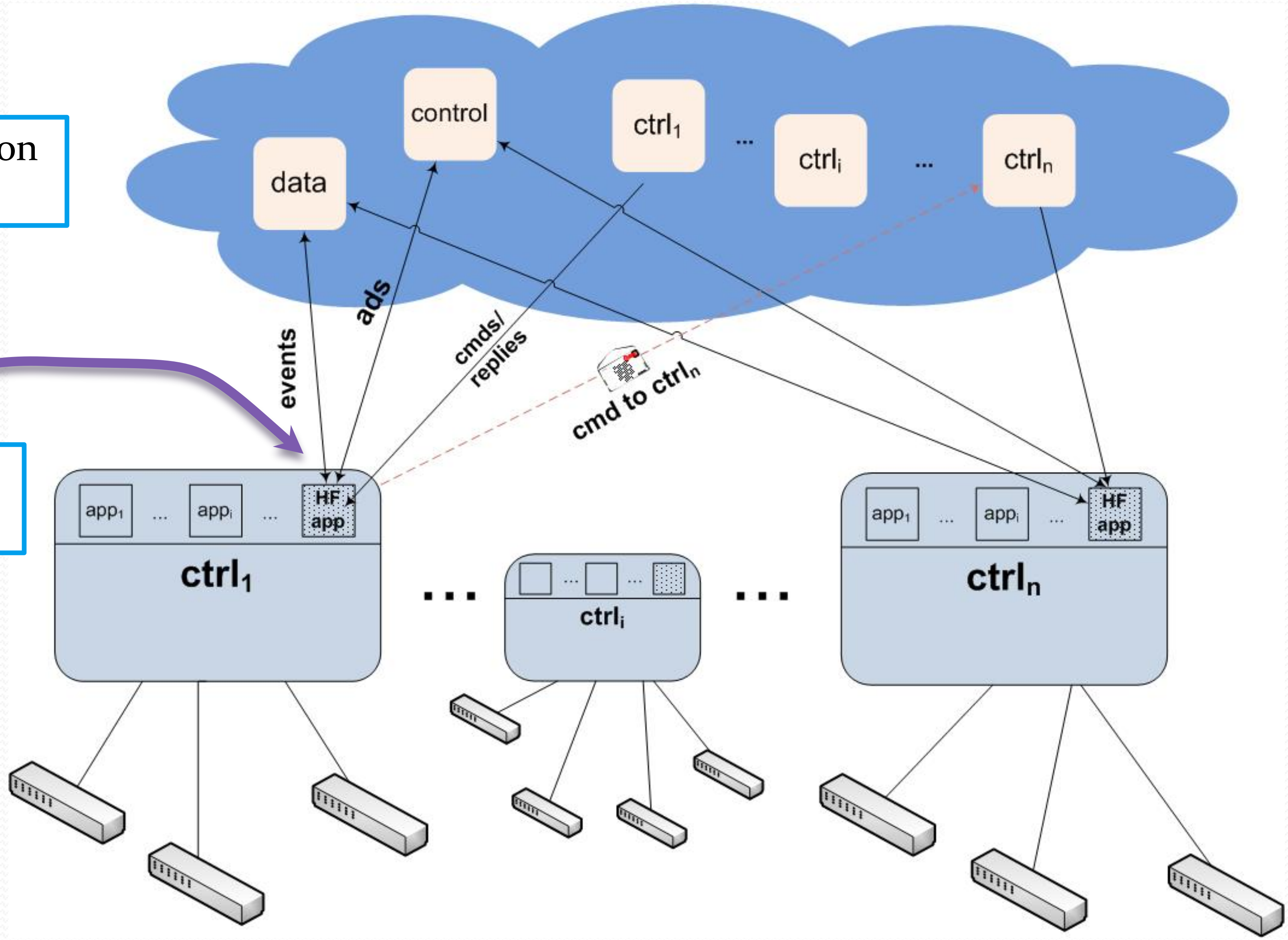
HyperFlow Design

- HyperFlow has two major 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 nearby controllers
- Upon controller failure:
  - Switches are reconfigured to connect to another controller

HyperFlow Architecture

Event Propagation System

Logger, Player, and Command Proxy



HyperFlow Controller Component

- Event logger captures & serializes some control events.
  - Only captures events which alter the controller state.
  - Serializes and publishes the events to the pub/sub.
- Event player de-serializes & replays captured events.
  - As if they occurred locally.
- If we need to modify a switch remotely
  - Command proxy sends commands to appropriate switches.
  - 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.
  - Direct messaging 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?

- **Question**: How rapidly can network changes occur in HyperFlow?
  - 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 HyperFlow exchange with the publish/subscribe system per sec?

How Often Can the Network Change?

- Benchmarked WheelFS:
  - The number of 3KB-sized files HyperFlow can serialize & publish:
    - 233 such events/sec → not a concern
  - The number of 3KB-sized files HyperFlow can read & deserialize:
    - 987 (multiple publis
- Howe n handle of events
  - During spikes in onsistency window is ot bounded.

Switch, host, link changes

Typically 10s of events/sec for thousands of hosts

Number of network changes on average must be < 1000 events/sec.

Detour: OpenBoot

- Controllers fail or are disrupted
  - Updating or patching controller, installing new applications, and bugs
- Controller failure can cause disruptions in network operation
- Reconstruction of state, and convergence takes time
- ***OpenBoot***: zero-downtime control plane service
- Based on the same idea of HyperFlow
  - Log controller events
  - Plus, check-pointing
- In case of failure
  - Replay all events to recover state

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.