Distributed OpenFlow Testbed (DOT)
A Roadmap

Raouf Boutaba

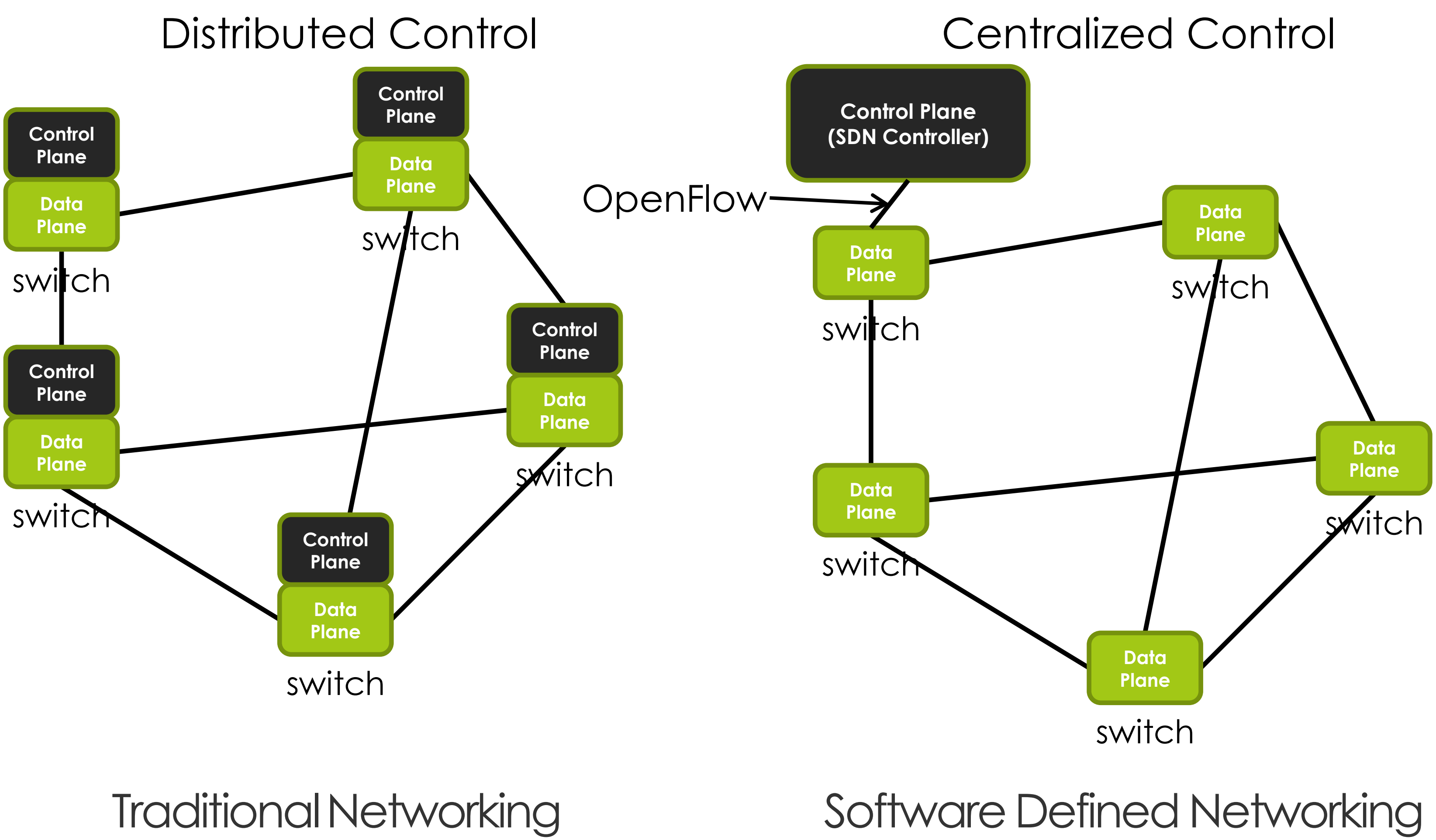
David R. Cheriton School of Computer Science
University of Waterloo

<date>

Outline

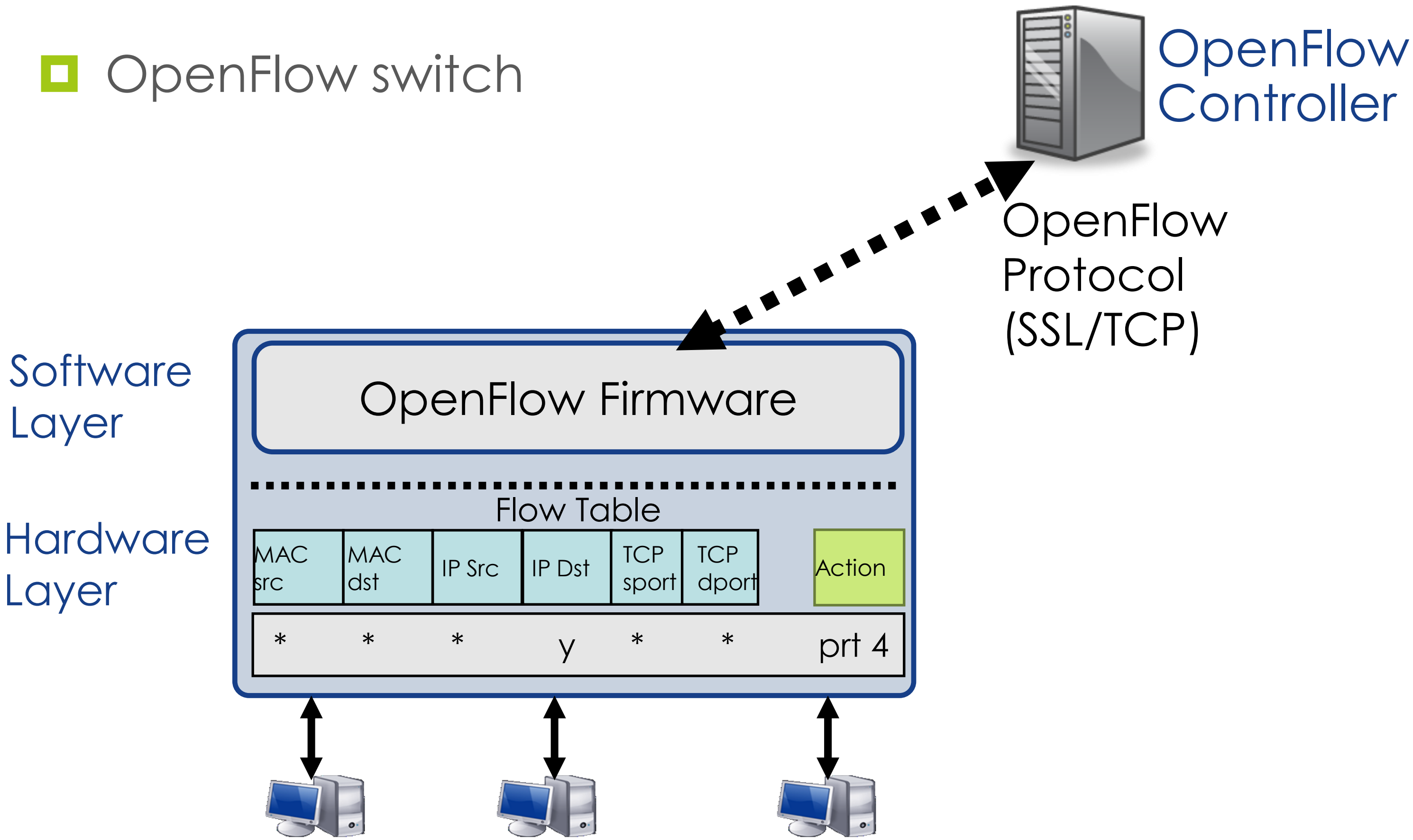
- Overview of SDN
- Our Work on Controller Provisioning
- Issues Faced During Simulation
- Design of DOT
- Back to Controller Provisioning
- Conclusion & Future Work

Traditional vs. Software Define Networking



OpenFlow in Action

- OpenFlow switch



OpenFlow in Action (cont.)

- OpenFlow Flow Table Entry

Rule	Action	Stats

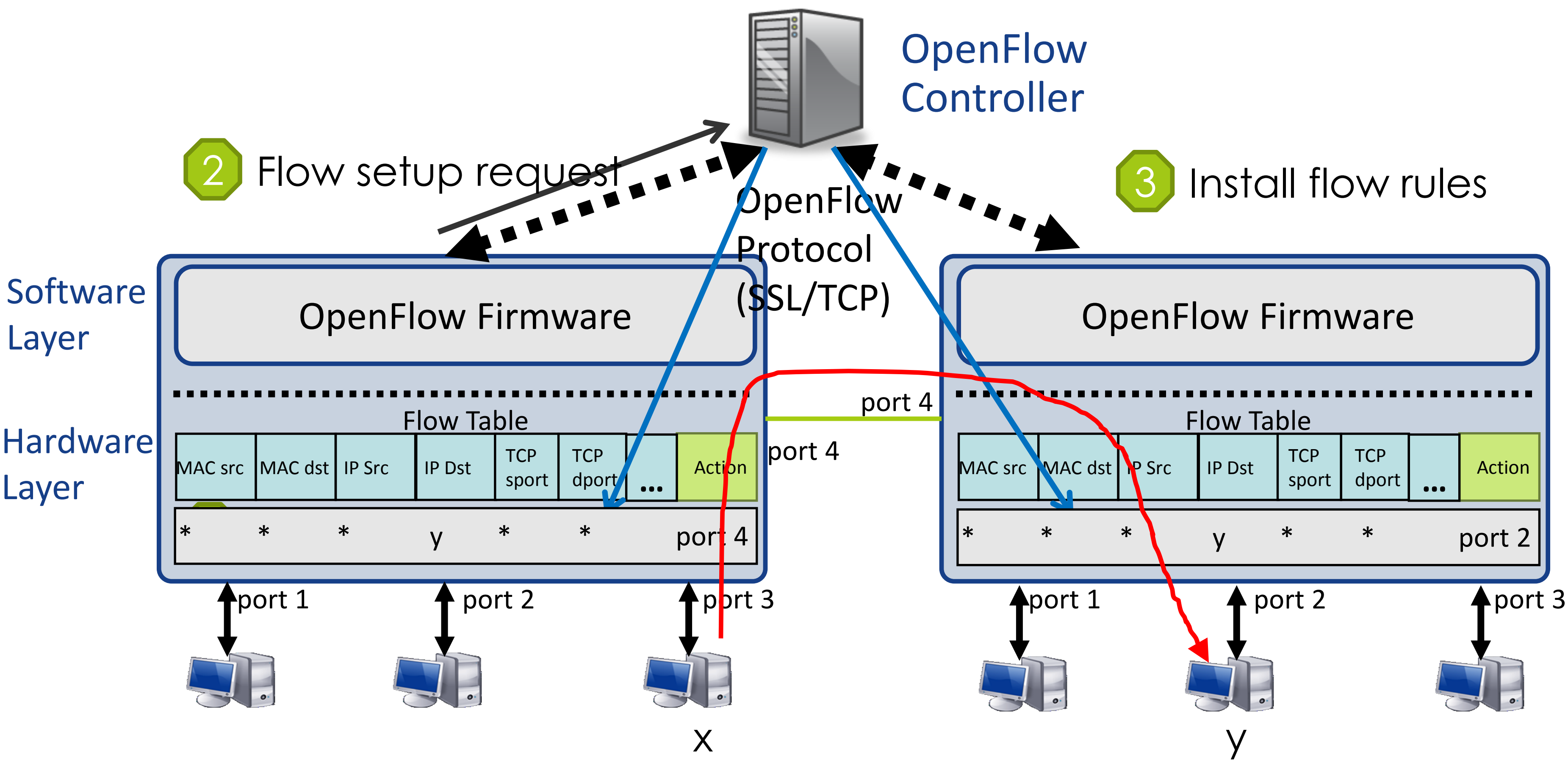
Packet + byte counters

1. Forward packet to port(s)
2. Encapsulate and forward to controller
3. Drop packet
4. Send to normal processing pipeline

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Proto	TCP sport	TCP dport

+ mask

OpenFlow in Action (cont.)

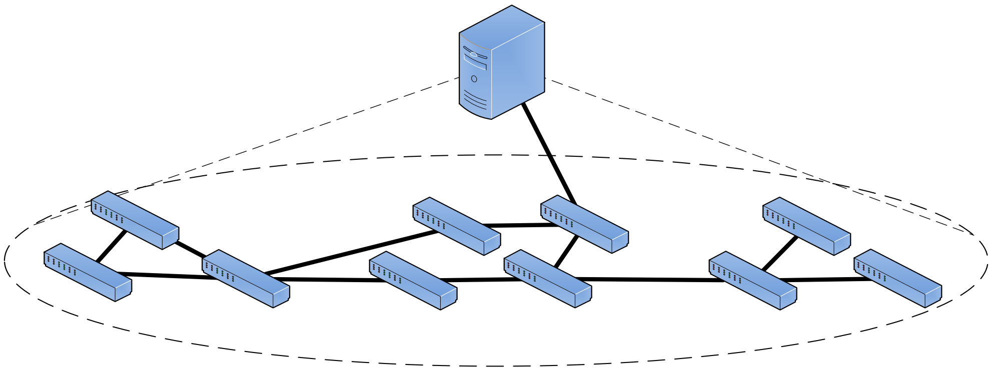


Outline

- Overview of SDN
- Our Work on Controller Provisioning
- Issues Faced During Simulation
- Design of DOT
- Back to Controller Provisioning
- Conclusion & Future Work

SDN with a Single Controller

- A single controller controls all switches in the network
- Advantages:
  - Centralized control
  - Ease of management
  - Network-wide view

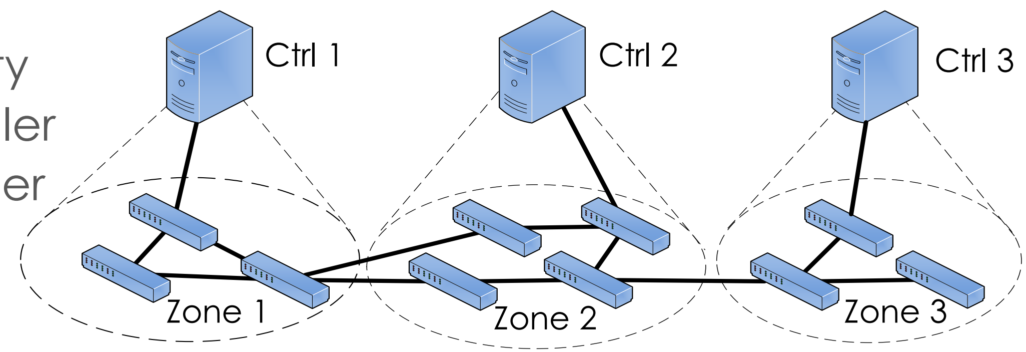


- Disadvantages:
  - High switch-to-controller latency
  - Limited processing capacity of controller

→Higher flow setup time

Multiple Controller

- Each controller controls a subset of the switches
- A switch communicates with just one controller
- Advantages:
  - Less processing capacity is required for each controller
  - Lower switch-to-controller latency



- Disadvantages:
  - Require state synchronization between controllers → Large control traffic overhead
  - Static switch-to-controller assignment → Overloaded controllers

What is Required?

- Dynamically provision controllers based on
  - Changing network conditions (traffic dynamics)
  - Switch-to-controller latency requirement
- Goals
  - Dynamically decide the number of controllers and their locations
  - Minimize flow setup time and control traffic
  - Minimize switch-to-controller reassignments

→ **Dynamic Controller Provisioning Problem (DCPP)**

Evaluation Process

- Preliminary thoughts
  - Deploy WAN topology on Mininet
  - Modify an SDN controller to support our solution
  - Evaluate the end-to-end Flow Setup time

What is Mininet?

- Mininet
  - De facto standard SDN emulator
  - Emulates an SDN network in a *single* machine
  - Uses Linux *container* to emulate hosts

Outline

- Overview of SDN
- Our Work on Controller Provisioning
- Issues Faced During Simulation
- Design of DOT
- Back to Controller Provisioning
- Conclusion & Future Work

Mininet – A good start! But….

Our experience

..we found that Mininet is inadequate for our purpose as it cannot handle the amount of traffic that we wanted to simulate….

"Dynamic controller provisioning in software defined networks" – Bari et al. (CNSM 2013)

Mininet – A good start! But….

In Mininet Support

….Scalability on a single system is something we can work on improving, but for now I'd recommend trying a smaller configuration on your hardware setup….

https://mailman.stanford.edu/pipermail/mininet-discuss/2012-June/000931.html

Mininet – A good start! But....

In Mininet Wiki

Mininet's original goal was "1000 nodes on your laptop" but such networks aren't really practical. ....

https://github.com/mininet/mininet/wiki/Ideas

DOT is Born

So…

Let's start with

DOT

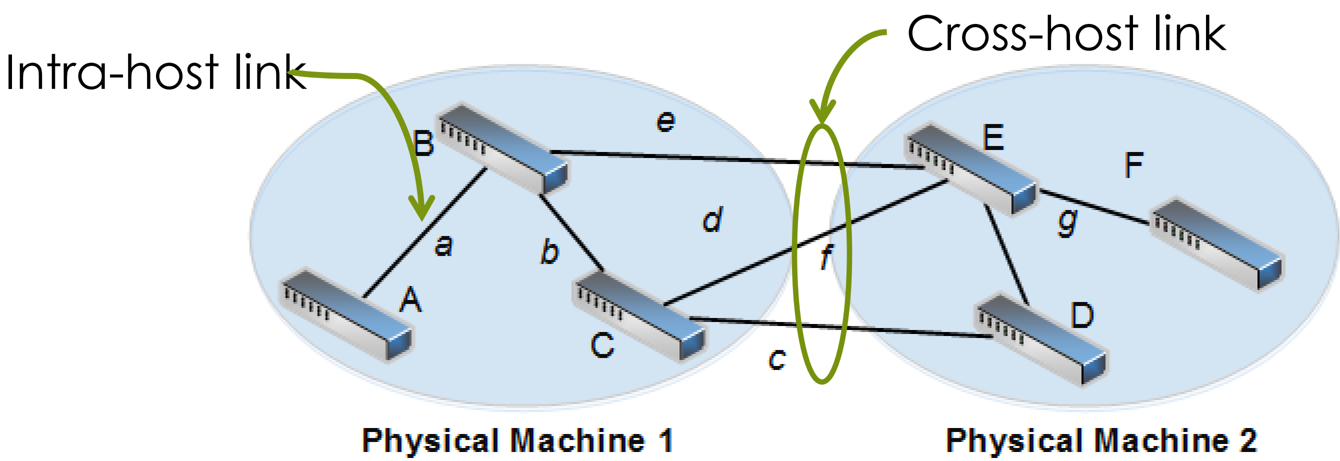
Distributed OpenFlow Testbed

Outline

- Overview of SDN
- Our Work on Controller Provisioning
- Issues Faced During Simulation
- Design of DOT
- Back to Controller Provisioning
- Conclusion & Future Work

Achieving Scalability

- Distributing the emulation across multiple physical machines
- Embedding algorithm partitions the *logical network* into multiple physical hosts
  - Formulated as an ILP
  - Proposed a greedy heuristic



Embedding: Formulation

- DOT embedding is formulated as an ILP
- Objective function

  Minimize $\boldsymbol{\alpha C^T + \beta C^E}$

- Where
  - $\boldsymbol{C^T}$ →Represents the number of cross-host links and their bandwidths
  - $\boldsymbol{C^E}$→Number of active physical hosts
- Constraints
  - Physical resource constraints
  - Cross-host link delay constraint

  ⟶ DOT embedding is NP-hard

Embedding: Heuristic

- **Switch selection**
  - Select a switch i using
    $$R_i = \gamma_D R_i^D + \gamma_B R_i^B + \gamma_N R_i^N$$
    - Where

      R_i^D →Degree ratio

      R_i^B →Resource ratio

      R_i^N →Neighbor ratio

- **Host selection**
  - Select an active physical host p for switch i
    $$F_{ip} = \lambda_R F_{ip}^R + \lambda_N F_{ip}^N$$
    - Where

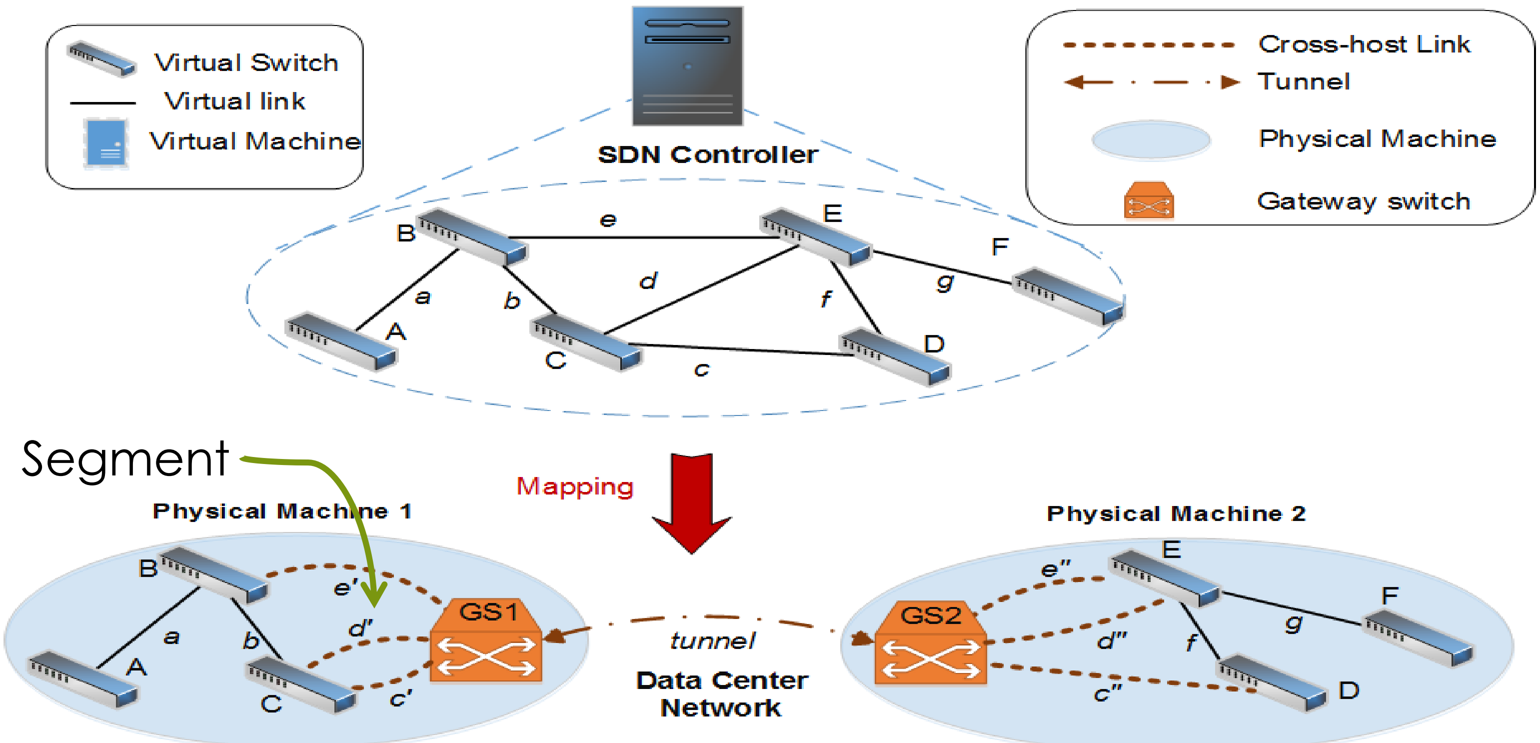
      F_{ip}^R →Residual capacity ratio

      F_{ip}^N →Locality ratio
  - Otherwise, activate another feasible host

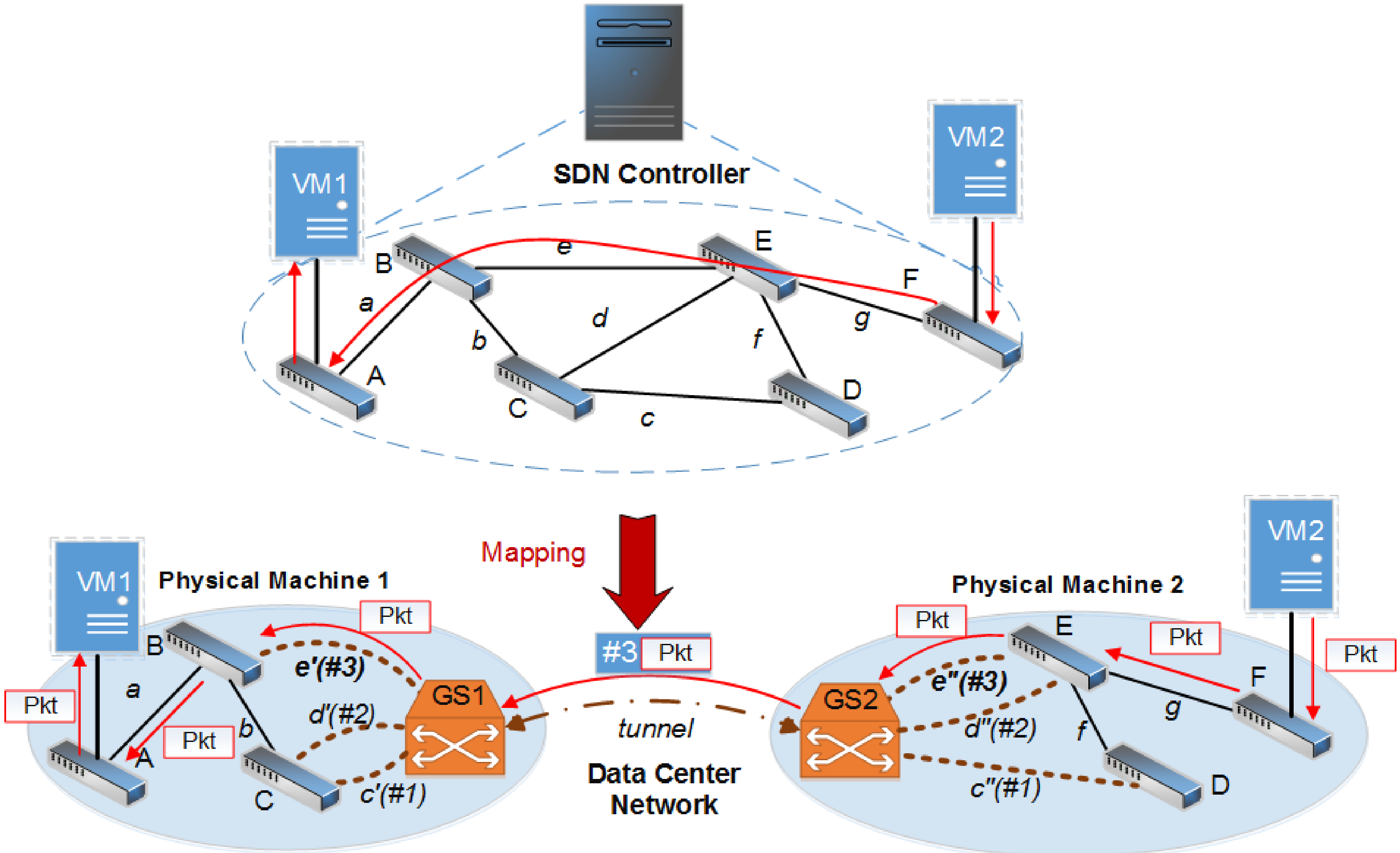
- Repeat until all switches are assigned or no embedding is possible with the policy

Achieving Transparency

- Gateway Switch (GS) is added to *each active physical* host
  - It unicasts packets passing through the cross-host links
  - It hides the partitioning from the SDN controller

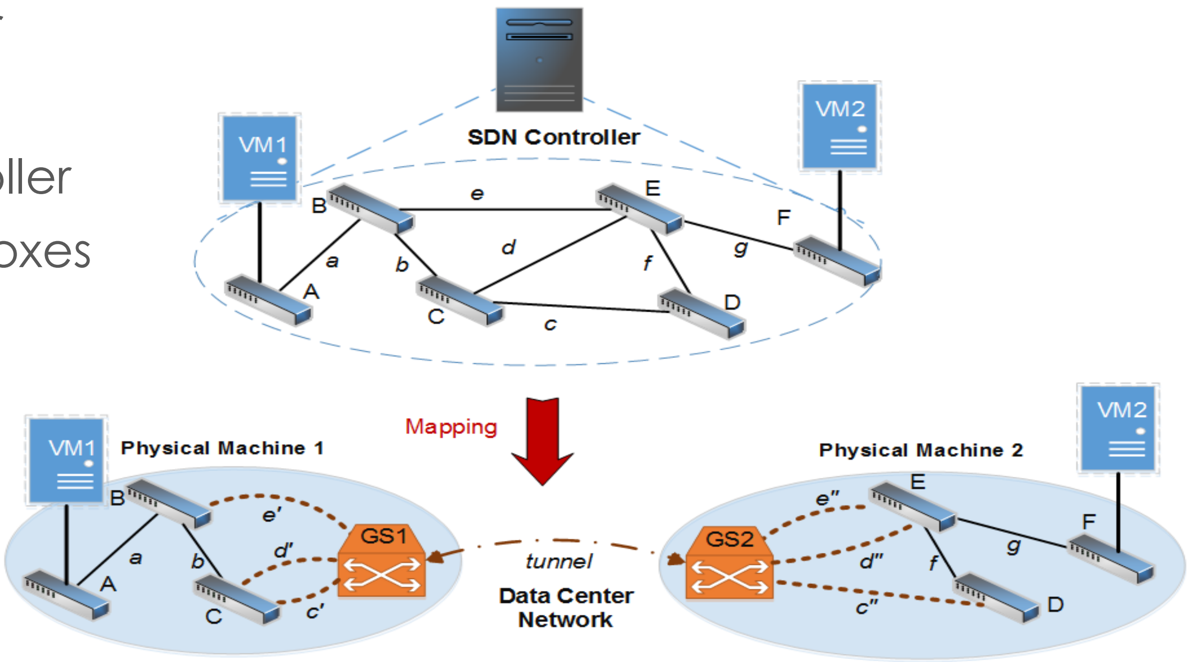


Inter-host Traffic Forwarding



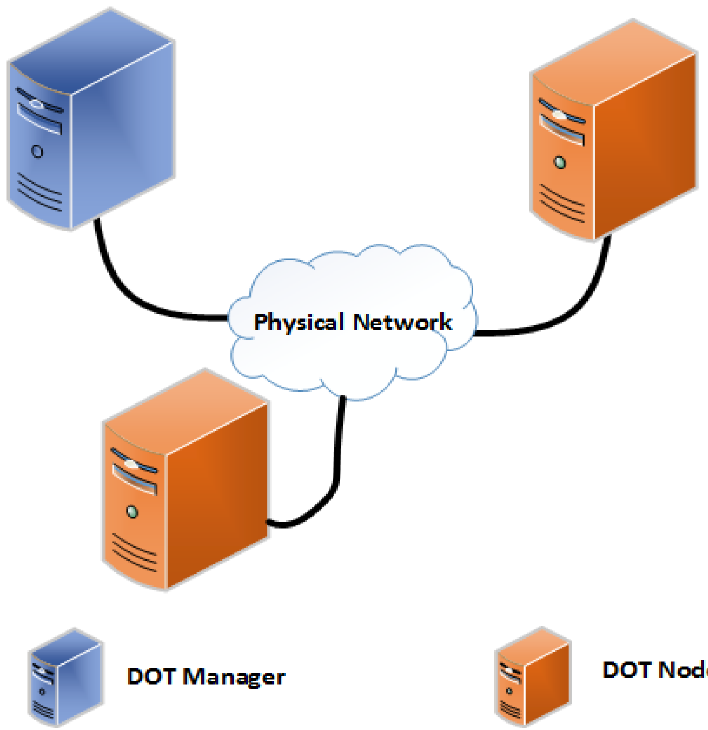
Achieving Flexibility

- DOT supports
  - Container based virtualization
  - Full virtualization (End-hosts → full fledged VM)
- VMs can be used for
  - Generating traffic
  - Running SDN controller
  - Deploying middleboxes



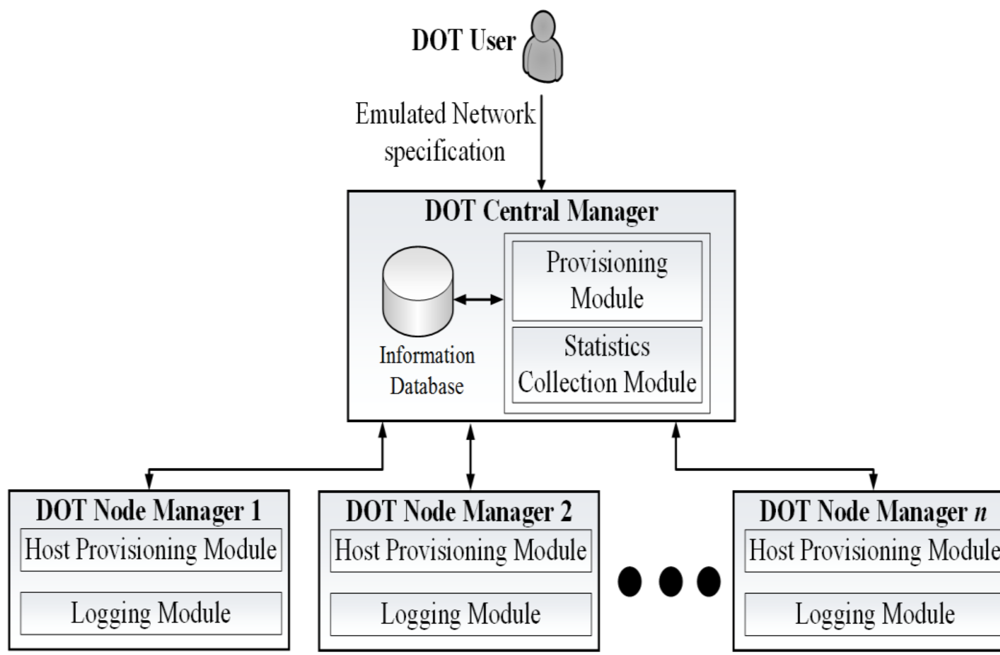
A Typical DOT Deployment

- DOT uses one *DOT Manger* and one or more *DOT Nodes*
- DOT Manager
  - Allocates and provisions the virtual infrastructure
  - Provides centralized access and monitoring facility
- DOT Node
  - Hosts the virtual switches and VMs



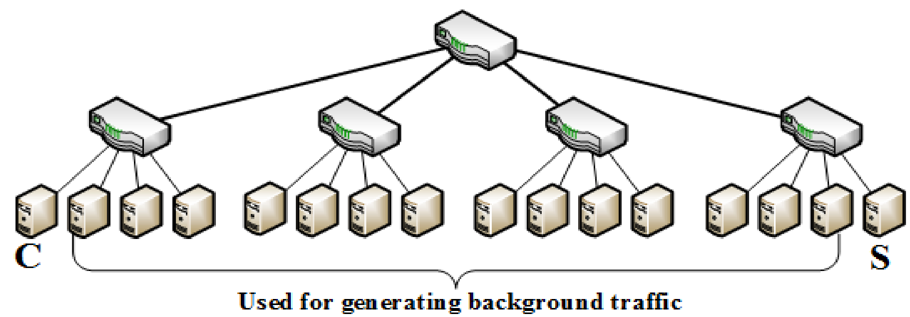
Management Architecture

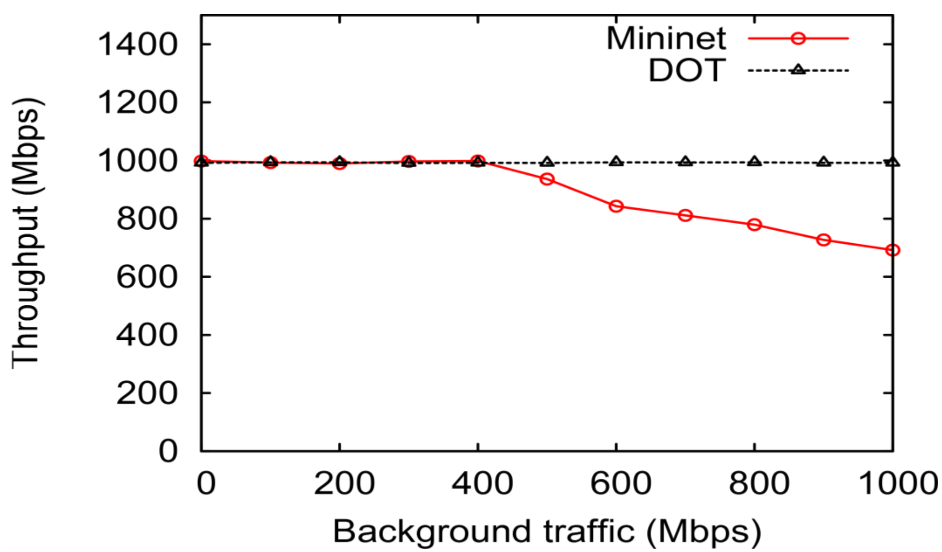
- DOT Central Manager
  - Provisioning module runs an embedding algorithm to determine the placement and instructs the host provisioning module about it.
  - Statistics collection module gathers information from logging modules of each DOT nodes
- DOT Node Manager
  - Host Provisioning module is responsible for allocating and configuring the virtual instances
  - Logging module collects local statistics



Comparison to Mininet

- We consider a fat-tree topology
- We run *iperf* to generate traffic between two hosts
- Foreground traffic
  - UDP traffic at a constant rate of 1000Mbps between C and S
- Background traffic
  - 7 UDP client-server pairs are chosen randomly



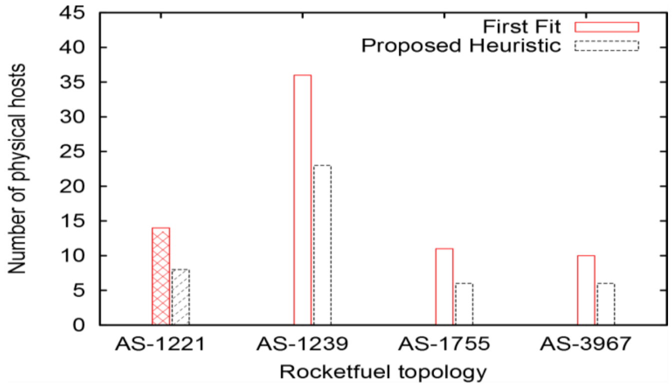


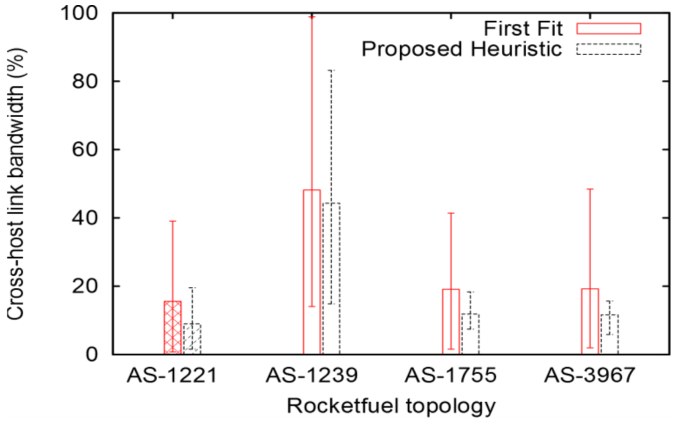
Embedding Algorithm

- We compare four different topologies (from RocketFuel [1])

Topology	#of Switch	#of Link
AS-1221	108	306
AS-1239	315	1944
AS-1755	87	322
AS-3967	79	294

- We compare the proposed heuristic with *First Fit* approach for these topologies.



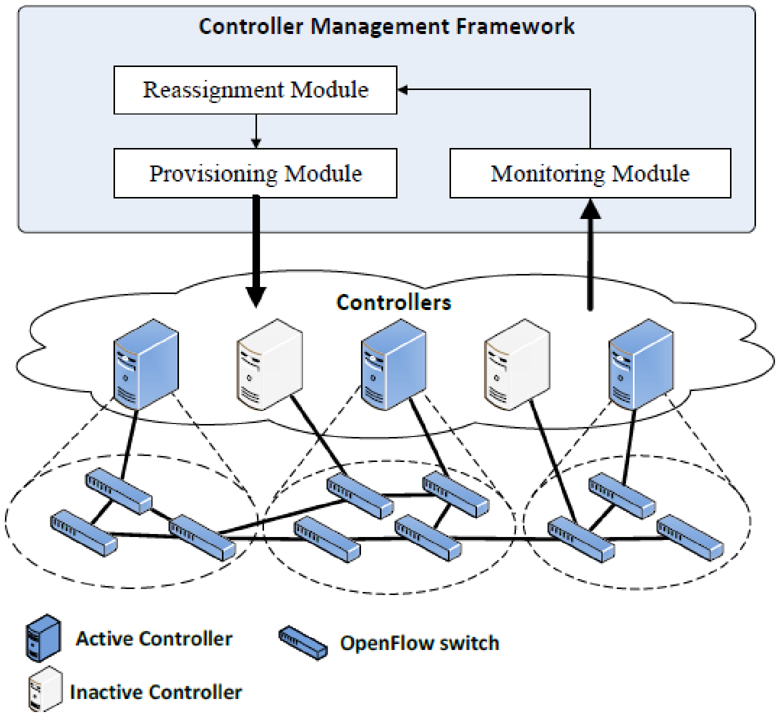


[1] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson.Measuring ISP topologies with rocketfuel. *IEEE/ACM Trans. Netw.* 12, 1 (February 2004), 2-16.

Outline

- Overview of SDN
- Our Work on Controller Provisioning
- Issues Faced During Simulation
- Design of DOT
- Back to Controller Provisioning
- Conclusion & Future Work

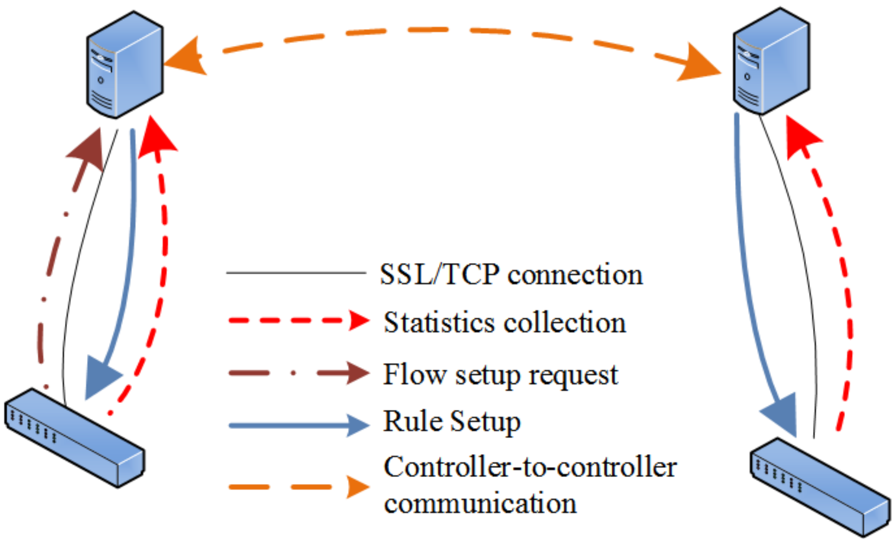
Management Framework



- Monitoring Module
  - Monitors controllers and collects statistics about the traffic
- Reassignment Module
  - Decides the number of controllers, their locations and the switch-to-controller assignment based on network conditions
- Provisioning Module
  - Provisions controllers and assigns switches to them

Problem Formulation

- DCPP can be formulated as an ILP
- Objective function **Minimize** $\alpha C_l + \beta C_p + \gamma C_s + \lambda C_r$
- Where
  - C_l = Statistics collection cost
  - C_p = Flow setup cost
  - C_s = Synchronization cost
  - C_r = Switch reassignment cost



- Constraints
  - Controller capacity constraint
  - Switch-to-controller delay constraint

$\longrightarrow$ DCPP is NP-hard.

Proposed Heuristics

- We propose two heuristics to solve DCPP
  - Greedy Knapsack based (DCP-GK)
  - Simulated Annealing based (DCP-SA)

- DCP-GK provides quick but inferior solutions

- DCP-SA provides good solutions, but requires longer time to find solutions

Greedy Knapsack Based (DCP-GK)

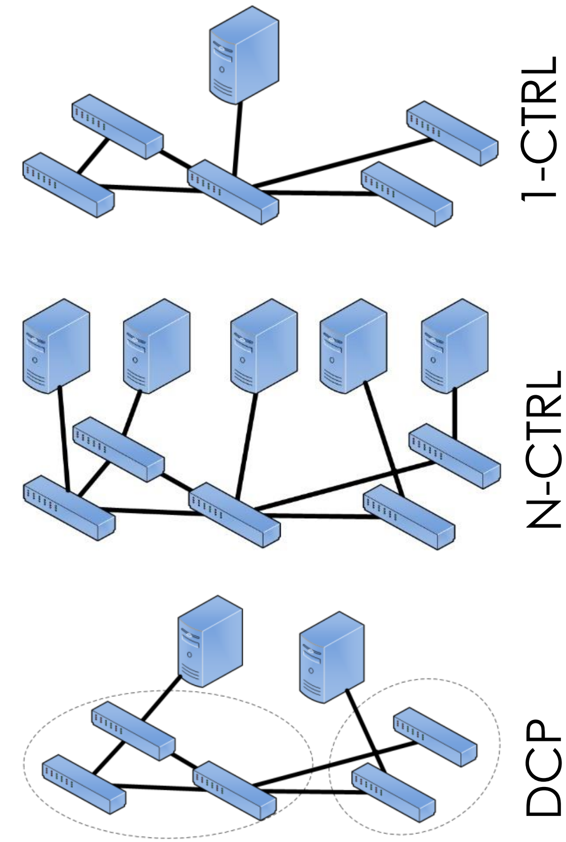
- Each controller is modeled as a knapsack
  - Capacity of the knapsack = number of flow-setups/sec
- Each switch is an object to be included in a knapsack
  - Weight = number of flow setup requests/sec
  - Profit = Inverse of switch to current controller's distance
- Procedure
  1. Repeat the following steps until either all switches are assigned to a controller or the controller set is exhausts
     - Step 1: Pick the controller with minimum total distance from all switches
     - Step 2: Use Greedy Knapsack approach to assign unassigned switches to the controller (subject to delay constraint)
  2. Randomly assign the remaining switches

Simulated Annealing Based (DCP-SA)

- DCPP is solved in two phases:
  - Phase 1: find a feasible assignment from the current one
    - For each overloaded controller
      - Select the switch sending maximum requests to it
      - Assign the switch to the most underused controller if the delay and capacity constraint are satisfied
      - Otherwise provision a new controller
    - Repeat until capacity and delay constraints are satisfied for all controllers
  - Phase 2: optimize the assignment by local search moves
    - Relocate switch
    - Swap switches
    - Activate a new controller
    - Merge controllers

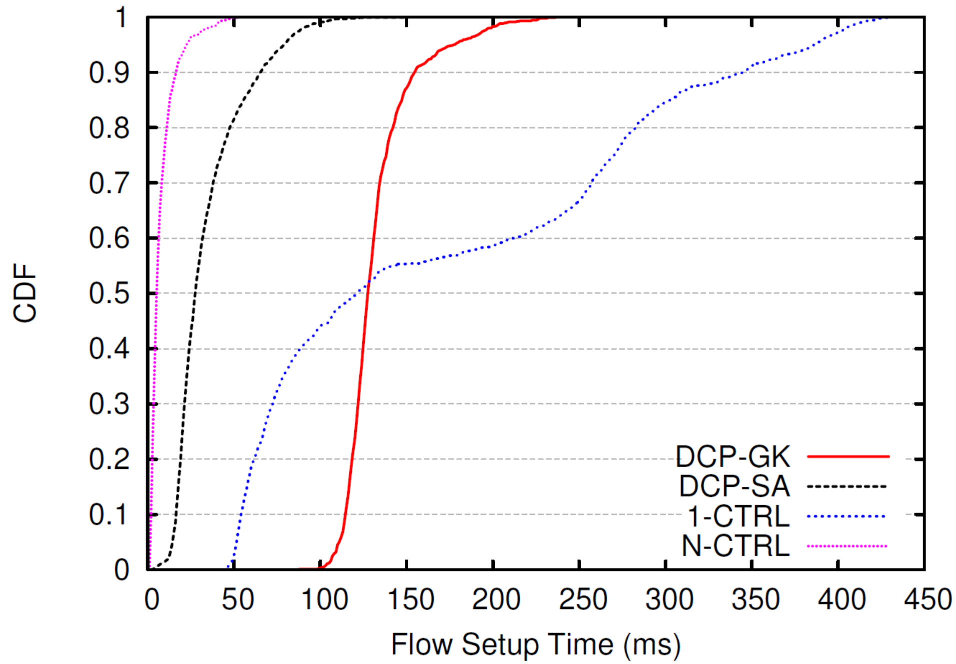
Simulation Setup

- We consider 3 scenarios
  - 1-CTRL: A single controller for all switches
  - N-CTRL: One controller for each switch
  - DCP: Dynamic controller provisioning
- Topology
  - 108 nodes, 306 links (from RocketFuel [1])
- Traffic
  - Based on a realistic traffic trace [2]
  - End-to-end TCP flows



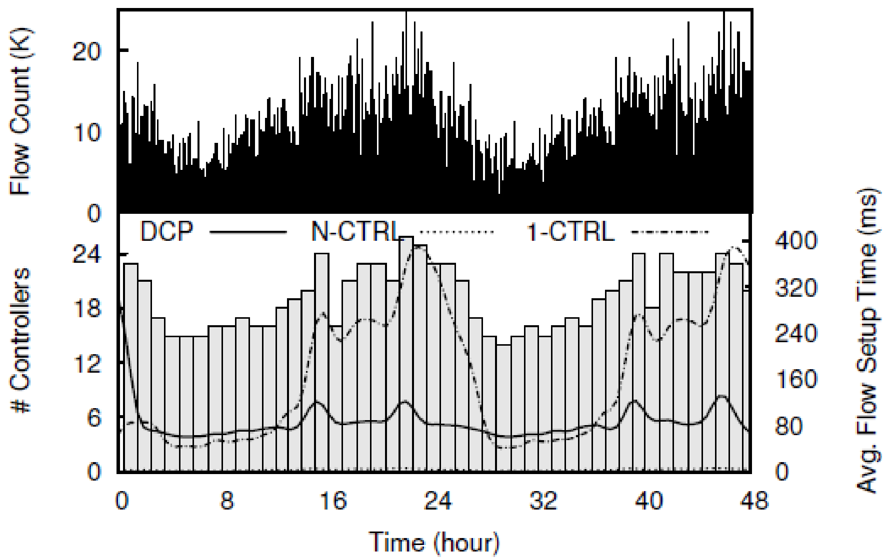
[1] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with rocketfuel. In SIGCOMM 2002, pages 133–145.
[2] S. Gebert, R. Pries, D. Schlosser, and K. Heck. Internet access traffic measurement and analysis. In Traffic Monitoring and Analysis, volume 7189 of LNCS, pages 29–42. 2012.

Flow-setup Time CDF

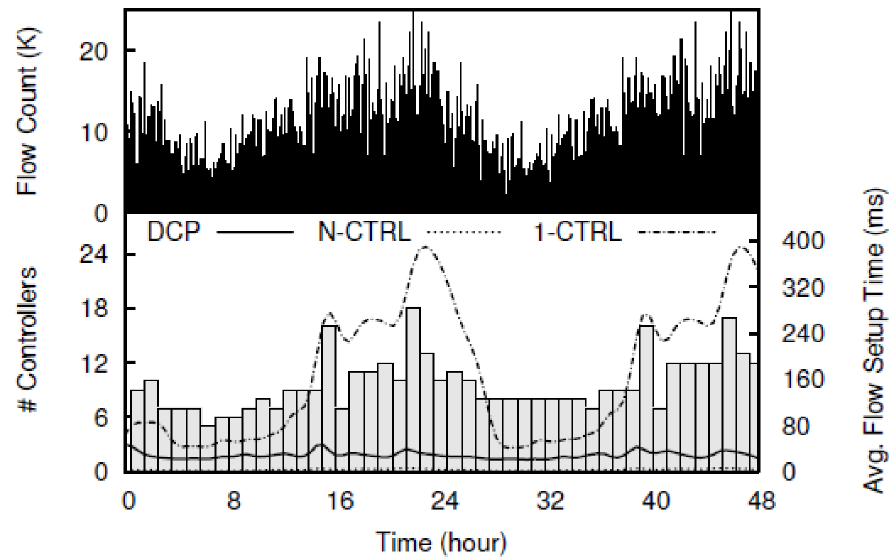


- N-CTRL provides minimal flow-setup time
- DCP-GK and DCP-SA both are better than 1-CTRL
- DCP-SA performs better than DCP-GK

Number of Controllers and Flow-setup Time



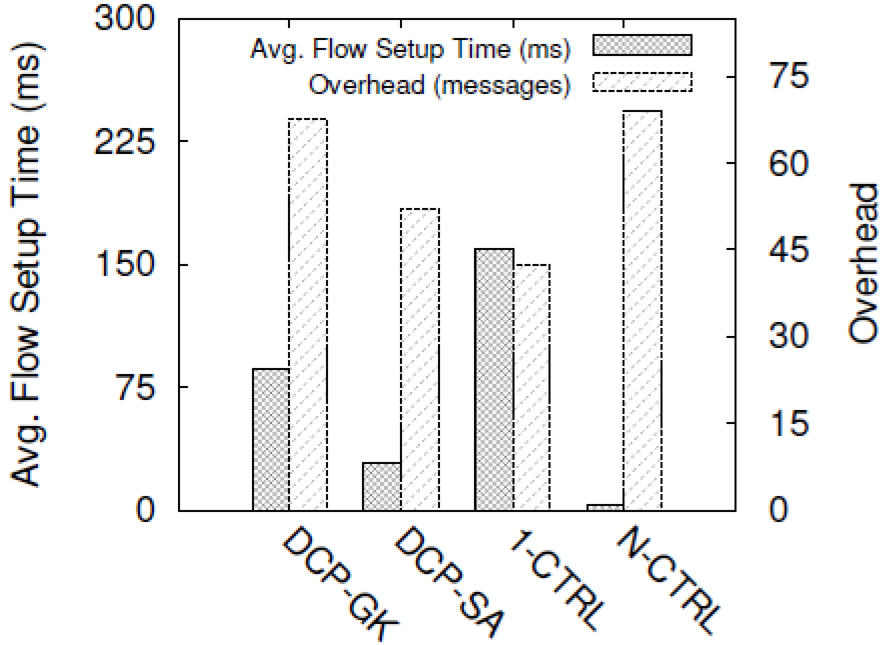
(a) DCP-GK



(b) DCP-SA

- DCP-SA required less controllers than DCP-GK
- In case of 1-CTRL flow-setup time varies with traffic
- DCP-GK and DCP-SA adapt to traffic changes

Summary of Overhead and Flow Setup Time



- N-CTRL has lowest flow-setup time, but largest overhead
- 1-CTRL has lowest overhead, but highest flow-setup time
- DCP-SA performs much better than DCP-GK

Outline

- Overview of SDN
- Our Work on Controller Provisioning
- Issues Faced During Simulation
- Design of DOT
- Back to Controller Placement
- Conclusion & Future Work

Conclusion & Future Work

- Until today, DOT
  - Solves scalability problem of Mininet
  - Hides distributed deployment of virtual infrastructure from SDN controller
  - Provides opportunities to emulate a wider range of network services
  - Support configurable logging facility
  - Provide APIs for monitoring and management
- Future DOT will
  - Provide *multi-user* support
  - Have auto scaling feature
  - Have provision for physical switch integration

DOT Portal

dothub.org

Distributed OpenFlow Testbed

An emulator for large scale OpenFlow networks

Home | Installation | Tutorial | Publications | Archive | FAQ

What is DOT?

Distributed OpenFlow Testbed (DOT) is a tool for emulating large scale OpenFlow based Software Defined Networks. DOT distributes the emulated network across several physical machines to provide guaranteed CPU time, bandwidth and network latency for the emulated components (i.e., switches, hosts and links). It can scale with the network size and traffic volume. It also has built-in support for configuring and monitoring the emulated components from a central point. DOT is an outcome of an ongoing research project of the WatSDN research group at the University of Waterloo.

Recent News

- October 22, 2014: DOT*v1.0* is released
- June 2, 2014: DOT is accepted at SIGCOMM 2014 (Demo)
- November 5, 2013 :DOT is accepted at NOMS 2014
- September 2, 2013: DOT *v0.1* is released

DOT: Installation

dothub.org/installation

Distributed OpenFlow Testbed

An emulator for large scale OpenFlow networks

Home | **Installation** | Tutorial | Publications | Archive | FAQ

Overview

DOT runs on a set of physical machines. One of these physical machines orchestrates the whole emulation process. We call this physical machine the **DOT Manager**. The other physical machines are called **DOT Nodes**. These machines contain virtual switches and virtual machines used to build the emulated topology.

i) The DOT Manager must be setup before any DOT Node. ii) The username for all DOT Nodes must be same. We recommend using a separate physical machine for DOT

Recent News

- October 22, 2014: DOT*v1.0* is released
- June 2, 2014: DOT is accepted at SIGCOMM 2014 (Demo)
- November 5, 2013 :DOT is accepted at NOMS 2014
- September 2, 2013: DOT *v0.1* is released

DOT: Tutorial

dothub.org/tutorial

Distributed OpenFlow Testbed

An emulator for large scale OpenFlow networks

Home Installation **Tutorial** Publications Archive FAQ

Emulating a topology in DOT

Requirements

Physical Environment

The physical environment should contain a dedicated machine for deploying the DOT Manager and one or more machines for deploying the DOT Node(s). The DOT configuration file should contain the hostname, IP address and resource (CPU,

Recent News

- October 22, 2014: DOT*v1.0* is released
- June 2, 2014: DOT is accepted at SIGCOMM 2014 (Demo)
- November 5, 2013 :DOT is accepted at NOMS 2014
- September 2, 2013: DOT *v0.1* is released