CSC 358 Principles of Computer Networks

Handout # 11: Software-Defined Networking

Joe Lim

Department of Mathematical and Computational Sciences

joe.lim@utoronto.ca

Announcements

- PA2: Due: Fri, Mar 30 @11:59PM
  - 6% : Working PA1
  - 12%: Working PA2
  - No penalty extension to Monday, Apr 2 @11:59PM
  - 10% Bonus Mark for Fri, Mar 30 @11:59PM submission
    - Only used to top off PA2 mark.
- Finals: Mon, Apr 9, 2018 from 9AM-12Noon in IB120
  - Please consult the official timetable website https://student.utm.utoronto.ca/examschedule/finalexams.php
  - 30 questions, 76/76, 14 pages

Extra Office Hours

- Friday, April 6th
- DH3095, 2:00PM – 4:00PM

- Please fill out the Course Evaluation

The Story So Far

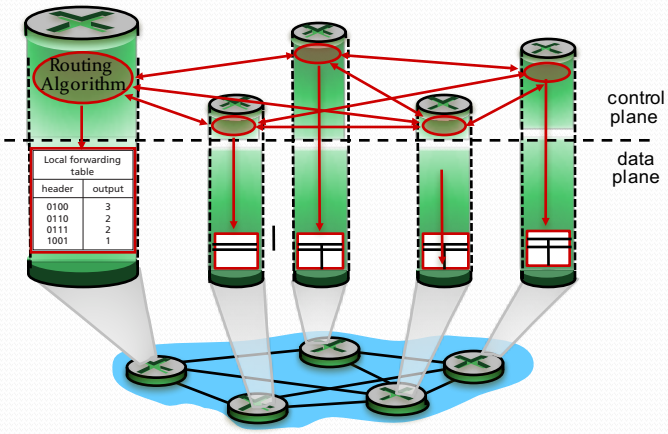
- Layering
  - Link layer
    - Media, framing, error detection/correction, switches, hubs, …
  - Network layer
    - Addressing (CIDR, subnet), routing and forwarding, DNS, BGP, …
  - Transport layer
    - TCP, UDP, flow control, congestion control, queue management, …
- **Misc**: Queueing Mechanisms, Middleboxes
- **Next**: Software-defined networks

Software Defined Networking (SDN)

- Internet network layer: historically has been implemented via distributed, per-router approach
  - *monolithic* router contains switching hardware, runs proprietary implementation of Internet standard protocols (IP, RIP, IS-IS, OSPF, BGP) in proprietary router OS (e.g., Cisco IOS)
  - different "middleboxes" for different network layer functions: firewalls, load balancers, NAT boxes, ..
- ~2005: renewed interest in rethinking network control plane

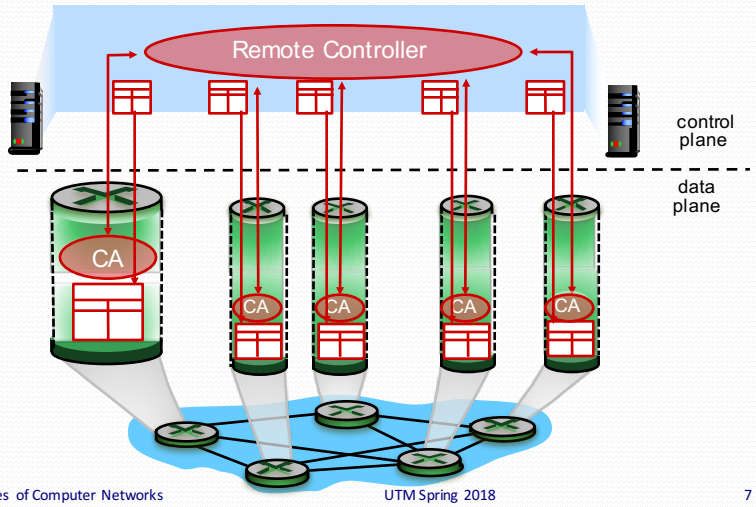
Recall ...

- Individual routing algorithm components in each and every router interact with each other in control plane to compute forwarding tables



Logically Centralized Control Plane

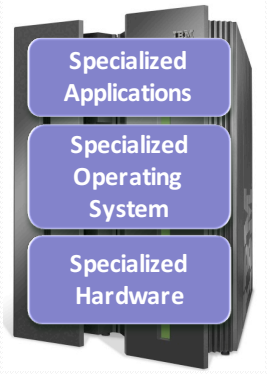
A distinct (typically remote) controller interacts with local control agents (CAs) in routers to compute forwarding tables

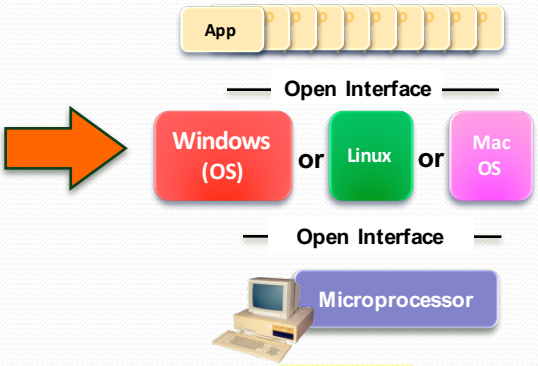


SDN

- Why a logically centralized control plane?
  - easier network management: avoid router misconfigurations, greater flexibility of traffic flows
  - table-based forwarding (recall OpenFlow API) allows "programming" routers
    - centralized "programming" easier: compute tables centrally and distribute
    - distributed "programming: more difficult: compute tables as result of distributed algorithm (protocol) implemented in each and every router
  - open (non-proprietary) implementation of control plane

Analogy: Mainframe to PC Evolution





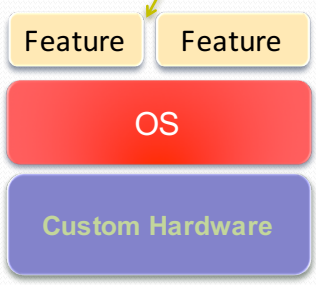
Vertically integrated
Closed, proprietary
Slow innovation
Small industry

Horizontal
Open interfaces
Rapid innovation
Huge industry

* Slide courtesy: N. McKeown

We Have Lost Our Way

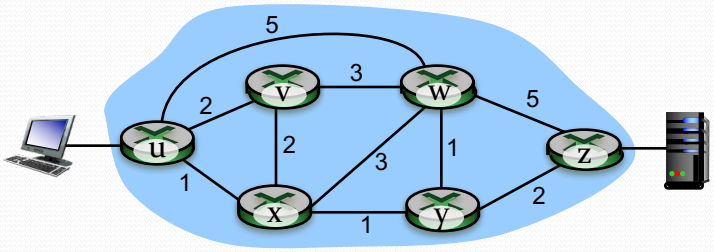
Routing, management, mobility management, access control, VPNs, …



Million of lines of source code
6,000 RFCs
Billions of gates
Bloated
Power Hungry

- Vertically integrated, complex, closed, proprietary
- Networking industry with "mainframe" mind-set

Traffic Engineering: Difficult Traditional Routing

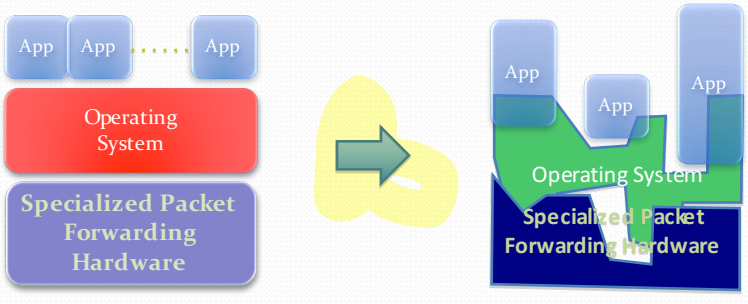


Q: what if network operator wants u-to-z traffic to flow along *uvwz*, x-to-z traffic to flow *xwyz*?

A: need to define link weights so traffic routing algorithm computes routes accordingly (or need a new routing algorithm)!

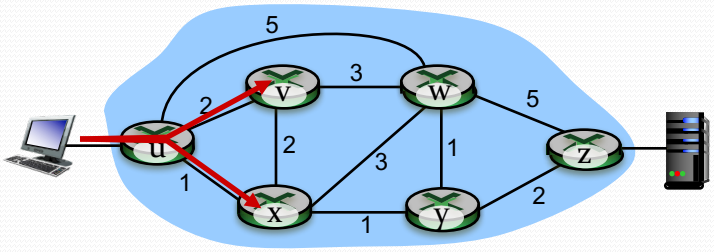
Link weights are only control "knobs": wrong!

Reality is Even Worse



- Lack of competition means glacial innovation
- Closed architecture means blurry, closed interfaces

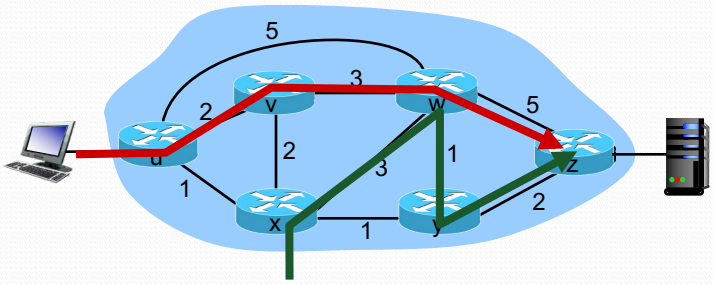
Traffic Engineering: Difficult



Q: what if network operator wants to split u-to-z traffic along uvwz *and* uxyz (load balancing)?

A: can't do it (or need a new routing algorithm)

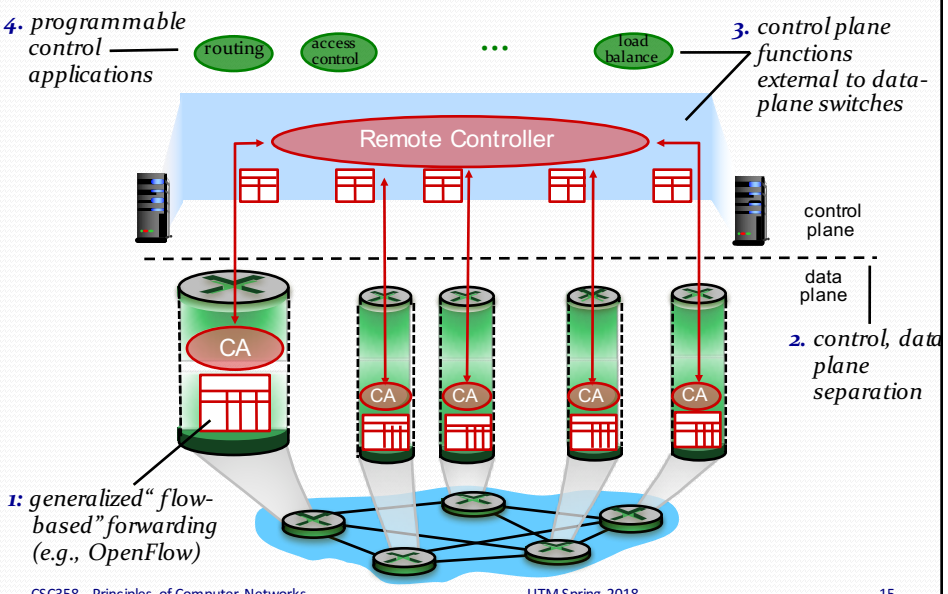
Traffic Engineering: Difficult



Q: what if w wants to route blue and red traffic differently?

A: can't do it (with destination based forwarding, and LS, DV routing)

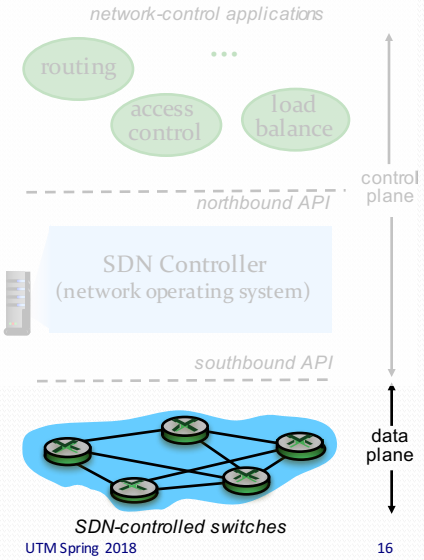
Software Defined Networking (SDN)



SDN Perspective: Data Plane Switches

Data plane switches

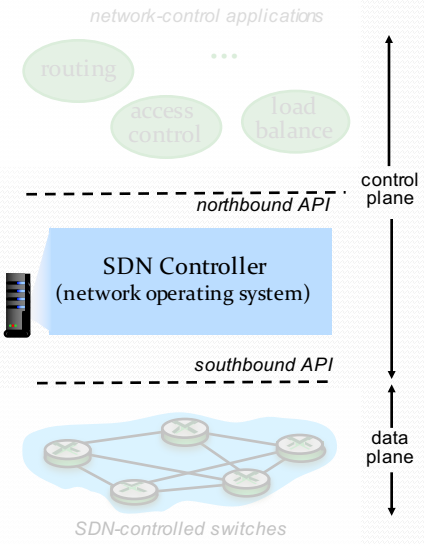
- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
- switch flow table computed, installed by controller
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable and what is not
- protocol for communicating with controller (e.g., OpenFlow)



SDN Perspective: SDN Controller

SDN controller (network OS):

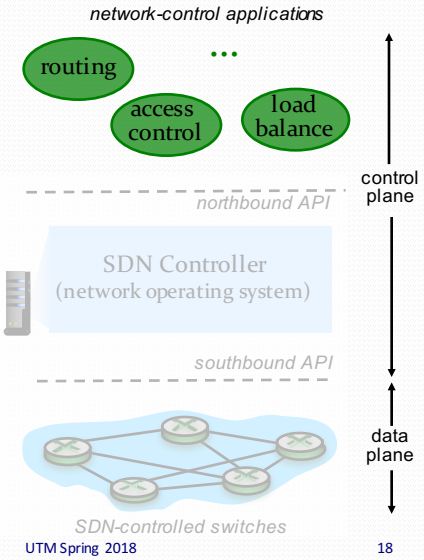
- maintain network state information
- interacts with network control applications "above" via northbound API
- interacts with network switches "below" via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



SDN Perspective: Control Applications

network-control apps:

- "brains" of control: implement control functions using lower-level services, API provided by SND controller
- *unbundled:* can be provided by 3rd party: distinct from routing vendor, or SDN controller

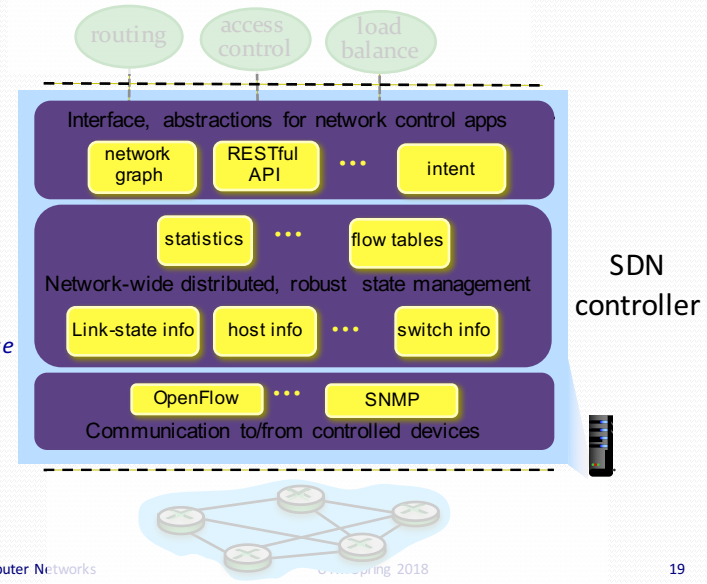


Components of SDN Controller

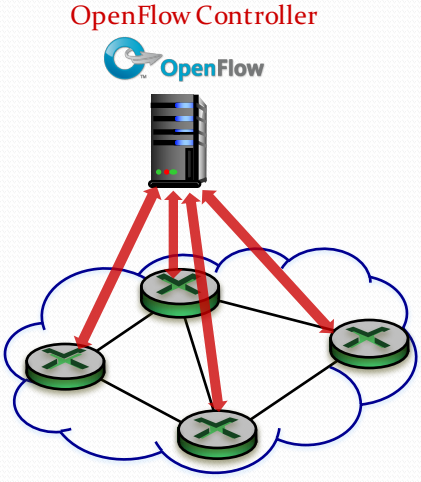
Interface layer to network control apps: abstractions API

Network-wide state management layer: state of networks links, switches, services: a *distributed database*

communication layer: communicate between SDN controller and controlled switches



OpenFlow Protocol

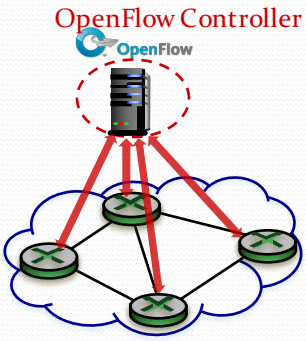


- operates between controller, switch
- TCP used to exchange messages
  - optional encryption
- three classes of OpenFlow messages:
  - controller-to-switch
  - asynchronous (switch to controller)
  - symmetric (misc)

OpenFlow: Controller-to-Switch Messages

Key controller-to-switch messages

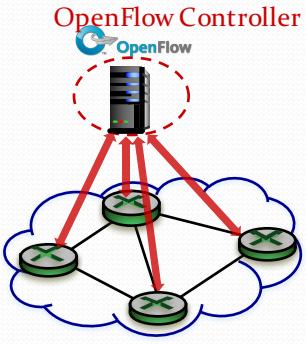
- *features:* controller queries switch features, switch replies
- *configure:* controller queries/sets switch configuration parameters
- *modify-state:* add, delete, modify flow entries in the OpenFlow tables
- *packet-out:* controller can send this packet out of specific switch port



OpenFlow: Switch-to-Controller Messages

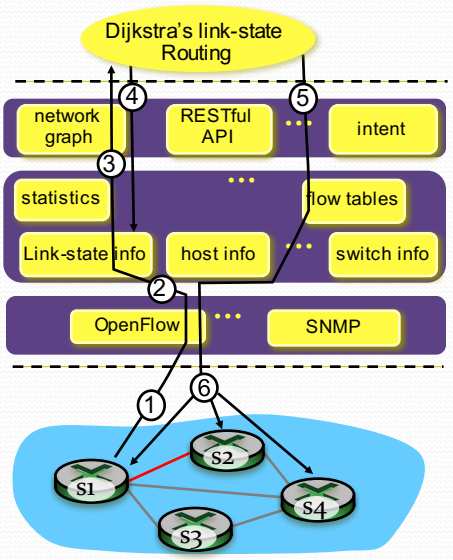
Key switch-to-controller messages

- *packet-in:* transfer packet (and its control) to controller. See packet-out message from controller
- *flow-removed:* flow table entry deleted at switch
- *port status:* inform controller of a change on a port.



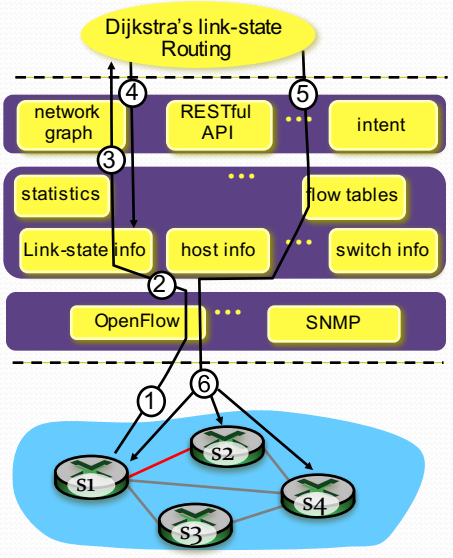
Fortunately, network operators don't "program" switches by creating/sending OpenFlow messages directly. Instead use higher-level abstraction at controller

SDN: Control/Data Plane Interaction Example



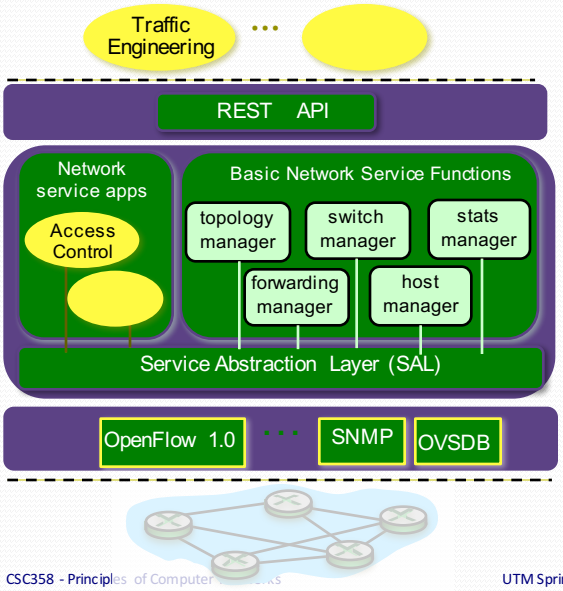
1. S1, experiencing link failure using OpenFlow port status message to notify controller
2. SDN controller receives OpenFlow message, updates link status info
3. Dijkstra's routing algorithm application has previously registered to be called when ever link status changes. It is called.
4. Dijkstra's routing algorithm access network graph info, link state info in controller, computes new routes

SDN: Control/Data Plane Interaction Example



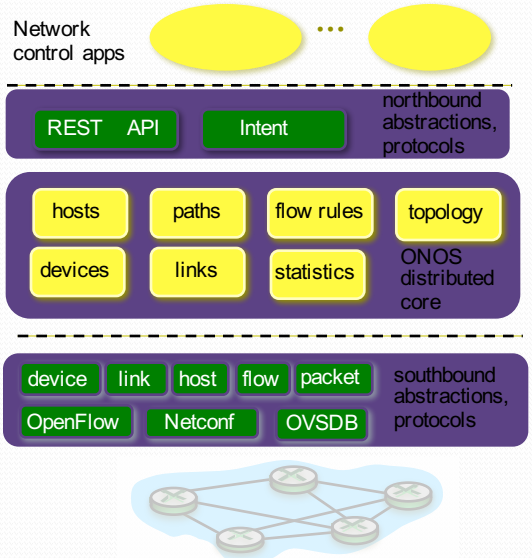
5. link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed
6. Controller uses OpenFlow to install new tables in switches that need updating

OpenDaylight (ODL) Controller



- ODL Lithium controller
- network apps may be contained within, or be external to SDN controller
- Service Abstraction Layer: interconnects internal, external applications and services

ONOS Controller



- control apps separate from controller
- intent framework: high-level specification of service: what rather than how
- considerable emphasis on distributed core: service reliability, replication performance scaling

SDN: Challenges

- hardening the control plane: dependable, reliable, performance-scalable, secure distributed system
  - robustness to failures: leverage strong theory of reliable distributed system for control plane
  - dependability, security: "baked in" from day one?
- networks, protocols meeting mission-specific requirements
  - e.g., real-time, ultra-reliable, ultra-secure
- Internet-scaling

Food for Thought

- What are the challenges in switching from traditional networks to OpenFlow networks?
- What are the opportunities?