Software-Defined Networking (SDN)

A New Network Paradigm

Susinthiran Sithamparanathan
Head engineer at the Institute of Theoretical Astrophysics, part time student, thesis “High Throughput Virtualization on OpenStack” @ Simula.

Ákos Pap
Master’s student from Hungary, Thesis: Mobile crowdsourcing @ Simula
Why do we need it?
Changes in traffic

- Video consumption → N-S traffic
- Data mining → E-W traffic
- More and more mobile devices
- Cloud computing → flexible, changing networks
- Solution: add more capacity! ...or?
- Other problems...
What is SDN?
**Definition**

*Software-Defined Networking (SDN) is an emerging network architecture where network control is decoupled from forwarding and is directly programmable.*

*Open Networking Foundation (ONF)*

**2 goals:**

1. Decoupling of data and control planes
2. Programmability of the control plane

**Results:**

simpler, cheaper network devices - intelligence is in the controller
Benefits

● Greater control of a network - through programming

● Enhanced configuration:
  ○ Currently no automatic and dynamic reconfiguration
  ○ Challenging to add, configure/re-configure new network devices due to heterogeneity
  ○ Involves some level of manual intervention → tedious and error-prone
  ○ SDN aims to unify it all (switches, routers, firewalls, NATs etc) in the control plane, programatically

● Improved performance:
  ○ Objective: maximize the utilization of network infrastructure - as a whole
  ○ Now: only in a subset of networks, without global knowledge → can lead to suboptimal performance, even conflicts
  ○ SDN aims to improve the performance globally for traffic scheduling, end-to-end congestion control etc
Benefits 2.

● Encouraged innovation in network architecture and operations:
  ○ Currently: proprietary HW → no modification, slow propagation
  ○ SDN: easily isolated → tested → easily moved to production

● Transition from testbed to production
  ○ No investment needed (no HW changes)
  ○ Easily reversible
Traditional vs Software Defined Networking

Network Applications
- MAC Learning
- Routing Algorithms
- Intrusion Detection System
- Load Balancer

SDN controller

Traditional vs software defined networking
Components of SDN
Reference model

SDN northbound interfaces (NBIs)
- A-CPI: Application-controller plane interface
- D-CPI: Data-controller plane interface

SDN controller

Application layer
- Application plane

Control layer
- Controller plane

Infrastructure layer
- Data plane

Network element

SDN application

Reference model
Infrastructure layer - Switching devices

Controller:
  for communicating with upper layers

Memory:
  storing routing rules
  storing statistics

→ no logic, no decision making!
Controller in an SDN switch:

- Communicates with the control layer (packet forwarding rules, link tunneling rules)
- Stores the rules (TCAM, SRAM)
- No need to run routing protocols
- Still need to store rules → same concepts as in routers last week
Infrastructure layer - Switching devices 3.

Data plane in an SDN switch:

- Packet forwarding based on IP and MAC, *plus*
- TCP/UDP port, VLAN tag, etc.

⇒ increased processing complexity
  - PC based → inefficient
  - HW classification on NICs
  - elephants → ASIC, mice → CPU
Categorization of SDN switches:

- **Implemented on general PC**
  - usually Linux, usually x86/x64
  - e.g. Pantou, OpenFlowClick (extension to the Linux kernel)
  - limited performance, but useful for VM servers (traffic is kept within the machine)

- **Implemented on open network hardware**
  - vendor independent programmable platform
  - e.g. NetFPGA, SwitchBlade, mostly lab prototypes (flexible yet performant)

- **Implementation on vendor’s switch**
  - released by the vendors
  - also FW updates that enable SDN
Reference model

-- SDN northbound interfaces (NBIs) --
A-CPI: Application-controller plane interface
D-CPI: Data-controller plane interface

-- SDN southbound interface --

Application layer
Application plane

Control layer
Controller plane

Infrastructure layer
Data plane

SDN application
SDN application

SDN controller

Network element
Network element
Network element

Reference model
Control layer - Programming

- CLI - e.g. Cisco IOS
  - designed for hardware configuration ↔ dynamic and stateful network status
  - limited by the capabilities of the hardware
  - Difficult and error prone to program

- SDKs built on top of high level languages (C++, Java, Python)
  - e.g. Frenetic (Pyretic)
    - `split = if_(match(dstip=IPAddr('10.0.0.1')),fwd(1),fwd(2))`
  - e.g. Maple
    - standard programming language to design algorithm run for each packet
    - Optimizer: records the run to a trace-tree → rules in the flow table of switches
    - Scheduler: assign a thread/memory to a switch
Control layer - Updating rules

The controller creates a new rule → installs it on the switches

Consistency?

a) Strict consistency
   - per packet / per flow, using versioning
   - either the old rule applies or the new rule applies → consistently throughout the lifetime of the packet/flow
   - drawback: memory consumption (2 set of rules in the switch during transition)

b) Eventual consistency
   - Later packets use the new rules
   - Step 1: redirect to controller (buffers)
     - Step 2: change rules
     - Step 3: release packets, process them with the new rules
Control layer - Network status

- Build NW topology → help making decisions
- Traffic statistics: duration time, packet number, data size, and bandwidth share of a flow
- Push / pull mode
- Traffic Matrix (TM): “TM reflects volume of traffic that flows between all possible pairs of sources and destinations in a network”
- OpenSketch: three stage pipeline - hashing, filtering, and counting (of packet fields)
- 1 centralized controller → overloading, security, etc. problems ⇒ multiple, replica controllers!
- Problem: syncing NW status → HyperFlow, SDNi, Onix
Control layer - policy/rule validation

- Detect and resolve conflicting rules
- Model-checking - verify reachability, loop-freeness
- Static and dynamic validation
Control layer performance

- The control layer is of high importance with regard to the performance of SDN networks.
- Serving packet forwarding rules to switching devices, sending rules updates and network status collection generate traffic between controllers and switching devices.
- There is a need to address scalability of centralized controllers.
Control layer performance

Increase the processing ability in software:

- Maestro, NOX/NOX-MT, McNettle and Beacon
- parallelism and batching. Used in Maestro, NOX-MT and McNettle
- Maestro: parallelism, I/O batching, core and thread binding. Demoed near linear performance scalability on multi-core CPUs.
- Benchmark of NOX vs. NOX-MT vs Maestro vs Beacon: NOX-MT has better min and max response time, and max throughput, but...
Control layer performance

Increase the processing ability in software:

- McNettle: an SDN controller written in Haskell
- McNettle: schedules event handlers, allocates memory, optimizes message parsing and serialization, optimizes cache usage, OS processing and runtime system overhead by reducing system calls
- McNettle can service up to 5000 switches using 1 single controller with 46 cores, and with a throughput of 14 million flows per second
Throughput scaling of various controllers
Control layer performance

Reduce request frequency:

- Heavy request load → longer delay in SDN controllers
  ⇒ strategies needed to decrease the request frequency
- 2 strategies:
  - Modify switching devices to handle requests in the data plane or near the data plane
  - Refine the structure of switching device organization
Control layer performance

Distributing rules across “authority switches”:

- Divert the packets through “authorized switches” to access needed rules, without requests to controllers → some paths might be long!
- DevoFlow handles “mice” flows in switching devices.
- DevoFlow installs a small set of packet forwarding rules in switching devices, supporting ECMP routing and rapid re-routing when designated port goes down without the need to contact controllers.
Control layer performance

Proper organization and labour division of switching devices:

- Kandoo is a framework for preserving scalability without changing switching devices and uses 2-layer architecture to handle (most of) the recent events locally, bottom layer being group of controllers w/o a network-wide view
- Top layer logically centralized controller with network-wide view and handling rare events, e.g. request for routing decisions.
- Heavy communication load is offloaded to highly replicable local controllers at the bottom layer
Control layer performance

Performance benchmarking:

- Why?
- Cbench and OFCBenchmark are tools designed for controller benchmarking
- Cbench: generates requests for packet forwarding rules and watches the responses from the controller
- Provides aggregated statistics of controller throughput and response time for all the switching devices
- Detailed controller behaviour hard to explore with Cbench!
Control layer performance

Performance benchmarking:

- OFCBenchmark provides fine-grained statistics for single switching device.
- Provides statistics of response rate, response time and amount of unanswered packages for every switching device
### SDN northbound interfaces (NBIs)
- **A-CPI**: Application-controller plane interface
- **D-CPI**: Data-controller plane interface

### SDN southbound interface
- Network element
- Network element
- Network element

### Reference model

1. **Application layer**
   - Application plane
2. **Control layer**
   - Controller plane
3. **Infrastructure layer**
   - Data plane
Application layer

- Global network view for SDN apps through northbound interface to controllers using Application Layer Traffic Optimization (ALTO) and eXtensible Session Protocol (XSP)
- High level language used by SDN apps to manipulate underlying networks → PaaS model for networking
- Popular SDN apps: load-balancing and cross-layer design
Application layer

Load balancing:

- widely deployed as front-end to direct client requests to particular back-end
- Dedicated load balancers are expensive → do it the SDN way!
- Algorithm based on SDN packet forwarding rules → arrange IP prefixes based on client’s unique IP and binary tree → divide the traffic using wild card rules
- Differentiated algorithm: web or email traffic?
Application layer

Cross-layer design & boundless roaming:

- Breaking the layers of OSI → allow inter-layer information exchange → easily achieved with SDN → QoS guarantee and better apps performance, e.g. video conferencing
- QoS-aware Network Operating System (QNOX) offers the abovementioned
- Again, SDN’s convenient reconfigurability → configure network on-the-fly using apps dynamic → improved apps performance with low config overhead
- Heterogeneous systems from multiple carriers → unified with SDN → seamless handover of mobile devices!
Application layer - Network maintenance

- 60%+ of network downtime is caused by humans
- NW tools (ping, tracert, etc.): difficult to automate
- SDN: inherently centralized and automated + has a global view and control
- Has all the data necessary to automate maintenance tasks

- *ndb* (like gdb) provides traces of network events (via “postcards”)
  OFRewind: same, but with playback possibility

- Immediate rule replacement in case of failure
Can we save energy with clever routing?

- energy-aware data-link adaptation, traffic proxying, infrastructure, application
- determine minimum data-links → dynamically power down redundant links in case of low traffic
Network Virtualization

Aim: allow multi heterogeneous network architectures to cohabit on a shared infrastructure, e.g IaaS

- Slice the physical network → virtual instances → assign different users, controllers or SDN apps
- Conventional virtualization methods (tunneling, VLAN, MPLS tags) → tedious configurations on network devices.
- FlowVisor:
  - can be used to create virtual networks from a physical network
  - share physical network with clear isolation between users
- Another approaches?
SDN & Cloud

- Extends the IaaS model beyond the compute and storage resources -> network services
- Virtual switching: used for intra communication of VMs on the same host - > but no sufficient network visibility and control. Solution?
  - Open vSwitch: provides virtual edge switching for VMs, reports network status to and receive rules from SDN controllers.
- VM migration: limited to single broadcast domain -> no inter datacenter VM migration. Solution?
  - Locator/Identifier Separation Protocol (LISP): OpenDaylight SDN controller uses LISP to preserve VM connectivity during the migrations.
OpenFlow

- protocol / standard for SDN
- initially: to enable easy network experiments in a campus network
- builds on flow tables in modern devices
- operates on a separate, secure TCP channel

- OpenFlow inspired SDN concepts, and as SDN matures, it influences the OpenFlow development
Flow table entry in OpenFlow

1. Forward Packet to Port(s)
2. Encapsulate and forward to controller
3. Drop Packet
4. Sent to normal processing pipeline
OpenFlow 2.

- first reference implementation (0.1.0): 2007 Nov; 1.0: 2009 Dec.
- New features and evolving specification with each release
- Now: managed by ONF, and in active development
- Both open network HW and vendor switches + many commercial controllers (and some open-source)

- E.g. Stanford OpenRoads project (5 48-port switches, 30 WiFi APs, 1 WiMax base station)
  E.g. Google B4 (connecting data centers, hybrid, open source Quagga as controller, near 100% link utilization)
OpenFlow 3.

- NOX (first OpenFlow controller)
  - “allows applications to be implemented based on a centralized network view using high level names as opposed to distributed algorithms over low-level addresses”
  - Python or C++ (core is implemented in C++)

- Mininet
  - network simulator for rapid prototyping of an OpenFlow network
  - offers a CLI to manage and interact with the network
Design Guidelines

- Collaboration between vendors, academia, communities and HW/SW knowledge needed for success
- SDN switching devices requires more memory and better processing power while keeping the cost down
- Must be able to play nicely together with different technologies like wireless, optical transmission → more control over the infrastructure → efficient infrastructure resource utilization
Design Guidelines

● Policies → low level rules translation without conflicts
● Large networks → multiple controller → keep in sync
● Improve controller’s performance: SW optimizations and algorithm analysis
● Leverage innovative ideas of SW devs → solutions beneficial for economic, social and environment.
Conclusion

● SDN considered a great solution to meet: convenient Internet access, more bandwidth requirements and dynamic management from SPs
● SDN offers enhanced configuration, improved performance and encouraged innovation
● SDN de-facto standard today: OpenFlow
References