Software-Defined Networks:
How to deploy them.
How to update them.
How to distribute them.

I ❤ SDN! Where can I get it, how can I deploy it?

Prologue

Context: Network Virtualization

Context: Flexible Allocation and Migration of Resources
CloudNet Prototype: Connecting “Nano-Datacenters”

- Resources at POPs, street cabinets, ...
- E.g., network monitoring, compute/aggregator smart meter data, ...
- New economic roles

**Roles in CloudNet Arch.**

- **Service Provider (SP)** (offers services over the top)
- **Virtual Network Operator (VNO)** (operates CloudNet, Layer 3+, innovation)
- **Virtual Network Provider (VNP)** (resource broker, compiles resources)
- **Physical Infrastructure Provider (PIP)** (resource provider, knows infrastructure and demand)

Federated CloudNet Architecture (Prototype!)

**Robes in CloudNet Arch.**

- **Service Provider (SP)** (offers services over the top)
- **Virtual Network Operator (VNO)** (operates CloudNet, Layer 3+, innovation)
- **Virtual Network Provider (VNP)** (resource broker, compiles resources)
- **Physical Infrastructure Provider (PIP)** (resource provider, knows infrastructure and demand)

Federated CloudNet Architecture

**Robes in CloudNet Arch.**

- **Service Provider (SP)** (offers services over the top)
- **Virtual Network Operator (VNO)** (operates CloudNet, Layer 3+, innovation)
- **Virtual Network Provider (VNP)** (resource broker, compiles resources)
- **Physical Infrastructure Provider (PIP)** (resource provider, knows infrastructure and demand)

Federated CloudNet Architecture

**Robes in CloudNet Arch.**

- **Service Provider (SP)** (offers services over the top)
- **Virtual Network Operator (VNO)** (operates CloudNet, Layer 3+, innovation)
- **Virtual Network Provider (VNP)** (resource broker, compiles resources)
- **Physical Infrastructure Provider (PIP)** (resource provider, knows infrastructure and demand)

Prototype

- Open source, testbed in Berlin and Munich
- Entire pipeline: from specification over solving to resource signalling

**service and resource specification: FleRD**

respect privacy, keep spec flexibility, non-topological aspects

**two stage embedding: heuristic and optimization of heavy-hitters**

**signaling to allocation resources**

VLANs, SDN paths, ...
Why SDN and how to get it?

Where is SDN useful? And for what? Examples of Deployment?

- Datacenter
- Inter-Datacenter WANs
- IXP/ISP
- Enterprise Networks
- Wireless
- ... and more

SDN in Datacenter (1)

- Characteristics and Problems
  - Fat-tree networks
  - Quite homogenous (hardware, software), even clean-slate
  - Already quite virtualized (e.g., OpenVSwitch’s run on servers, realizes the fabric abstraction)

- What is SDN used for?
  - Virtualize: decouple applications from the physical infrastructure (e.g., to migrate VMs)
  - Isolate: e.g., different customers can use same virtual addresses
  - Performance: Higher utilization in Ethernet-based architectures

SDN in Datacenter (2)

- Examples of “Deployments”
  - PAST: more utilization for Ethernet-based architectures
  - SPAIN, VL2, NVP, ...
  - Often cleanslate architectures (possible in datacenters)

- E.g. PAST
  - Implements a Per-Address Spanning Tree routing algorithm
  - Network architecture for datacenter Ethernet networks
  - Preserves Ethernet’s self-configuration and mobility support while increasing its scalability and usable bandwidth
  - Performance comparable to or greater than Equal-Cost Multipath (ECMP) forwarding, which is currently limited to layer-3 IP networks, without any multipath hardware support
  - OpenFlow-based implementation
**SDN for Inter-Datacenter WAN (1)**

- **Characteristics and Problems**
  - Wide-area: bandwidth precious (WAN traffic grows at fastest rate)
  - Latency
  - Probably not so many sites
  - Many different applications and requirements

- **What is SDN used for?**
  - Improve link utilization
  - Prioritize traffic (bulk vs priority traffic: not possible in conventional control plane)

**SDN for Inter-Datacenter WAN (2)**

- **Examples of “Deployments”**
  - Google B4:
    - own hardware
    - implement own routing control plane protocol
    - application classes:
      - i) user data copies (e.g., email, documents, audio/video) to remote data centers for availability/-durability
      - ii) remote storage access for computation over inherently distributed data sources
      - iii) large-scale data push synchronizing state across multiple data centers
  - Ordered in increasing volume, decreasing latency sensitivity, and decreasing overall priority.

**SDN for IXP/ISP (1)**

- **Characteristics and Problems**
  - IXP: layer-2 internet exchange points (multiple providers)
  - ISP: wide-area carrier network, dumb bit-pipe providers?
  - Today’s inter-domain routing protocol BGP: inflexible, difficult to manage, troubleshoot, and secure

**SDN for IXP/ISP (2)**

- **What is SDN used for?**
  - ISP
    - introduce new services
    - e.g., based on multiple header fields
    - e.g. video over better paths, traffic differentiation, QoS, …: CloudNets
    - manage traffic directly (not via weights and logging into devices)
  - IXP
    - More expressive policies
    - from multiple ISPs, depending on application, etc.
    - implementing business contracts (more than hop-by-hop forwarding)
    - distant networks can exercise “remote control” over packet handling
    - And and and: inbound traffic engineering, redirection of traffic to middleboxes, wide-area server load balancing, blocking of unwanted traffic, etc.

- **Examples of “Deployments”**
  - SDX

**SDN for Enterprise Networks (1)**

- **Characteristics and Problems**
  - Organically grown, “unstructured” (not clean-slate!)
  - Many legacy devices, management complex
  - Utilization often low
  - Outages costly (and not business expertise!)

**SDN for Enterprise Networks (2)**

- **What is SDN used for?**
  - Simplify network management: automated troubleshooting and network management
  - Logically centralized control (instead of 1000s of config files distributed over devices)

- **Examples of “Deployments”**
  - Stanford CS building OpenFlow deployment

**SDX**
How to Introduce SDN (and Operate as a Hybrid Network)?

- Datacenter
- Inter-Datacenter WANs
- IXP/ISP
- Enterprise Networks

Deploy SDN in Datacenter

- Where to deploy?
  - Usually deployed on (software) edge only: there translate logical to physical addresses, use access control mechanism, etc.
  - OpenVSwitch, run on servers (can terminate links at VM hypervisor)
  - Inside the network: e.g., simple "fabric" (forwarding only), classic multipath-equal cost control platform, etc.

- How to deploy?
  - Software update in the hypervisor (e.g. OpenVSwitch)

Deploy SDN in Inter-Datacenter WAN

- Where to deploy?
  - Replace IP "core" routers (running BGP) at border of datacenter (end of long-haul fiber)

- How to deploy?
  - Gradually replace routers
  - However, benefits arise only after complete hardware overhaul of network (after years)

Deploy SDN in Inter-Datacenter WAN

Deploy SDN in IXP and ISP

Deployment options?

- Single-site deployment (SDN controller = "smart route server" on behalf of the participating ASes at the exchange)

- Or multi-site deployment: SDN controllers across multiple exchange points coordinate to enable more sophisticated wide-area policies

Deploy SDN in Enterprise Network

Let’s shift gears and focus on enterprise network in more detail!

How to deploy SDN in enterprise?
Deploy SDN in Enterprise Network

Let's shift gears and focus on enterprise network in more detail!

How to deploy SDN it in enterprise?
- First idea: Full deployment (replace all legacy devices)
- Migros-budget idea: Edge deployment, like in datacenter

Pro and Cons?

First Option: Full Upgrade (Of All Devices)

Too expensive! Must upgrade to SDN incrementally…

Second Option: Edge-Based Approach

Police traffic at all (SDN) ingress ports, then “tunnel” through legacy network

- Full control over access policy
- New network functionality at edge

- Bad for enterprise networks:
  - Unlike datacenters, edge does not terminate at VM hypervisor but at access switch
  - Hundreds of switches at edge!

The Enterprise Network…: Where the heck is the edge?!

A real large-scale campus network

TheEnterpriseNetwork….: Where the heck is the edge?!

Still too expensive! Many legacy devices, large edge, …

Reasons Incremental Deployment in Enterprise

We want more flexible and even more incremental deployment!

Because?
- Budget constraints
- Confidence building: gradually open scope rather than flag-day event
- Want to benefit from SDN already after buying the first switch: unrealistic?
Dual-Stack Approach:
- Partition flow space into several disjoint slices
- Assign each slice to either SDN or legacy processing
- Does not address how the legacy and SDN control planes should interact
- Nor how to operate the resulting network as an SDN
- Requires a contiguous deployment of hybrid programmable switches (process both according to legacy and SDN mechanisms)

Edge-based + Tunnel
- High cost of deployment
- Impairs the ability to control forwarding decisions within the core of the network (e.g., load balancing, waypoint routing)
- Management benefits do not extend to legacy devices

Panopticon realizes full SDN from partial SDN deployment!
Transition to SDN control plane before hardware is fully installed.
I.e., (1) integrates legacy and SDN switches, (2) exposes an SDN control plane on an abstract network view, (3) supports arbitrary deployment, (4) supports arbitrary deployment, (5) supports arbitrary deployment, (6) supports arbitrary deployment.

How to realize a full SDN from a partial SDN deployment?
Match-Action Semantics

Given: partially upgraded network (hybrid SDN+legacy switches)
How to realize a full SDN from a partial SDN deployment?

Given: partially upgraded network (hybrid SDN+legacy switches)
SDN Switches: e.g., for policy enforcement, middlebox traversals, access control, ...

Solution: Waypoint Enforcement

Insight #1:
≥ 1 SDN switch →
Policy enforcement

Insight #2:
≥ 2 SDN switch →
Fine-grained control

Ensure that all traffic to/from an SDN-controlled port always traverses at least one SDN switch

SDN Waypoint Enforcement

Get flexibility with more SDN switches

Insight #1:
≥ 1 SDN switch →
Policy enforcement

Insight #2:
≥ 2 SDN switch →
Fine-grained control

Traffic load-balancing

Also: capacity beyond trees!
How to achieve Waypoint Enforcement?

How do we do this?

Use per-port VLANs to isolate traffic!

Per-Port VLANs

VLAN1

VLAN2

What about port A?

Can reach two SDN switches!

VLAN3

VLAN4

Give Port A two more VLANs!

Really good idea?
Per-Port VLANs

Idea:
1. Can reuse VLANs: different domain/"island"!
2. Can use single VLAN for port A: still full flexibility.

Putting it together: Panopticon Cell Blocks

Cell Blocks = conceptually group SDN ports (connected "legacy islands")

SDN switches separate different islands: e.g., can reuse VLAN tags.

Putting it together: Panopticon Solitary Confinement Trees

Per-port spanning tree (VLAN) to entire SDN frontier:
ensures waypoint enforcement and traffic isolation.
Larger frontier = more choice (MAC learning inside)

Talk Overview: Incremental SDN Deployment in Enterprise

Hybrid Operation
Operate the network as a (nearly) full SDN

Deployment Planning
Determine efficient partial SDN deployment

Where to deploy SDN Switches=

What if you could choose where to place SDN switches?

1. What is the "price of Panopticon"?
2. Optimization criteria?
A strength of SDN: flexible traffic management!
So how to update network configuration?

But what about multi-author policies?
E.g., Alice in charge of setting up tunnels, Bob in charge of ACLs, …

But what about distributed control planes?

First: A Simplistic Model for SDN

SDN
- Control of (forwarding) rules in network from simple, logically centralized vantage point
- Flow concept: install rules ("matches") to define flow (match L2 to L4)
- Match-Action concept: apply actions to packet (forward to port A, add tag, …)
- Allows to express global network policies, e.g., load-balancing, adaptive monitoring / heavy hitter detection, …

How to install a new policy “consistently”?

High-Level Results

- Evaluated architecture on a large campus network (1713 L2 and L3 switches)
- Traffic matrix derived from LBNL traces
- Upgrading 6% of distribution switches →
  - 100% SDN-controlled ports
  - avg. path stretch < 50%
  - max. link util. < 70%

A strength of SDN: flexible traffic management!
Use Case: Network Updates

How to do network updates in this model?
Network update (here): change path of packet class (e.g., header).

Possible criteria?

Abstractions for Consistent Network Update

Definition
- Policy: Here, path to be traversed by packet (of certain header)

Goals
- Per-packet consistency: per packet only one policy applied (during journey through network)
- Per-flow consistency: all packets of a given flow see only one policy

Why per-flow?
- E.g., packets of same TCP should reach same server.

Installation: 2-Phase Update Protocol

Initially
- SDN Match-Action
  - Match header (define flow)
  - Execute action (e.g., add tag or forward to port)
- Consistent Update: 2-phase
  - At internal ports: add new rules for new policy with new tag
  - Then at ingress ports: start tagging packets with new tag

Installation: 2-Phase Update Protocol

SDN Match-Action
- Match header (define flow)
- Execute action (e.g., add tag or forward to port)
- Consistent Update: 2-phase
- At internal ports: add new rules for new policy with new tag
- Then at ingress ports: start tagging packets with new tag

How to guarantee per-packet consistency but still update network paths eventually?

Transmissions asynchronous...
2-Phase Update Protocol

**Phase 1**
- SDN Match-Action
  - Match header (define flow)
  - Execute action (e.g., add tag or forward to port)
- Consistent Update: 2-phase
  - At internal ports: add new rules for new policy with new tag
  - Then at ingress ports: start tagging packets with new tag

**Phase 2**
- SDN Match-Action
  - Match header (define flow)
  - Execute action (e.g., add tag or forward to port)
- Consistent Update: 2-phase
  - At internal ports: add new rules for new policy with new tag
  - Then at ingress ports: start tagging packets with new tag

**Abstractions for Consistent Network Update**

**Definition**
- Policy: Here, path to be traversed by packet (of certain header)

**Goals**
- Per-packet consistency: per packet only one tag applied (during journey through network)
- Per-flow consistency: all packets of a given flow see only one policy

**Number of tags?**
One per policy update.

**Do I need FIFO?**
Homework.

**Policy Update in Distributed Setting?!**

**Middleware**

**Desirable criteria?**

**Number to do it per-flow?**
Preinstall new policy at lower priority, but keep tagging packets of old flows. Idea: let old microflows expire (via timeout), but need to identify active flows, and do not want too many rules for each individual flow. Alternative: end-host feedback...
What about failures?

**Desirable criteria?**

 middleware

**Policy Update: Goals**

**Definition**
- **Policy**: Here, path to be traversed by packet (of certain header)

**Goals**
- All-or-nothing: policy fully installed or not at all
- Conflict-free: never two conflicting policies
- Progress: non-conflicting policy eventually installed; and: at least one conflicting policy
- Per-packet consistency: per packet only one policy applied (during journey through network)

... despite failures!

**How to realize?**

Need to (1) compose, (2) install, (3) make fault-tolerant! How?

Install and Compose!

**Definition**
- **Policy**: Here, path to be traversed by packet (of certain header)

**Goals**
- All-or-nothing: policy fully installed or not at all
- Conflict-free: never two conflicting policies
- Progress: non-conflicting policy eventually installed; and: at least one conflicting policy
- Per-packet consistency: per packet only one policy applied (during journey through network)

... despite failures!

**How to realize?**

Need to (1) compose, (2) install, (3) make fault-tolerant! How?

Install and Compose!

compose and install concurrent policies, with redundancy/"helping"

2-Phase RSM

Pyrethic
Building Blocks: Installation and Composition

- **Centralized installation:**
  - 2-phase consistent policy installation protocol
  - Designed with centralized controller in mind

- **Policy Composition Language:**
  - Frenetic/Pyrethic: policy composition
  - Parallel composition "|" enough

Remark: Composition Semantics External

- Policy: here defined over (header) domain ("flow space")
- Policy priority
- Implies rules on switch ports
- Conflict = overlapping domains, different treatment

Distributed Systems Issues: What does it require?

**Goals**

- All-or-nothing: policy fully installed or not at all
- Conflict-free: never two conflicting policies
- Progress: non-conflicting policy eventually installed; and: at least one conflicting policy
- Per-packet consistency: per packet only one policy applied (during journey through network)

What else do you always want in distributed systems?

Linearizability...
Goal: Serializable!

**Example**

Three switches, three policies, policy 1 and 2 with independent flow space, policy 3 conflicting.

Goal: Packets should not see conflicting Policy 3.
Goal: There should exist a linear history! = Looks as though the application of policy updates is atomic and packets cross the network instantaneously.

Left: Concurrent history: 3rd policy aborted due to conflict.
Right: In the sequential history, no two requests applied concurrently. No packet is in flight while an update is being installed.

No packet can distinguish the two histories!

Remark: Impossible Without Atomic Read-Modify-Write Ports

**Thm:** Without atomic rmw-ports, per-packet consistent network update is impossible if a controller may crash-fail.

**Proof:** Consensus not always possible!

QED

Solutions?

The TAG Solution

- Atomic RMW ports: "see before write!"
- Naïve solution: pre-install tags for each possible path internally, only need to tag packets at ingress port with path-tag!
  - Can synchronize "globally" at ingress port: not a distributed problem anymore…
  - If policy involves multiple ingress ports, go through ingress ports in pre-defined order. Copy rules where needed (if process died)

add tag: Ingress port
Relation Transactional Memory

- Related to shared memory problems:

![Diagram showing Read/Write Processes: Update-Transactions](image)

Read Processes: Traffic Transaction

Shared Memory

Note:
- Read process may have side-effects under monitoring rules!
- Read transaction must succeed and cannot wait

Efficient Solution

- Less than n tags? Processes must share: Consensus?!
- With n tags: Replicated State Machine, distributed counter to get next tag…

Conclusion

- SDN: How to get it?
- Distributed control

Own literature:

- Optimizing Long-Lived CloudNets with Migrations
  Gregor Schaffrath, Stefan Schmid, and Anja Feldmann.

- Toward Transparent SDN Deployment in Enterprise Networks
  Dan Levin, Marco Canini, Stefan Schmid, and Anja Feldmann.
  Open Networking Summit (ONS), Santa Clara, California, USA, April 2013.

- Towards Unified SDN Control
  Stefan Schmid and Jukka Suomela.
  ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN), Hong Kong, China, August 2013.

- Exploiting Locality in Distributed SDN Control
  Stefan Schmid and Jukka Suomela.
  ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN), Hong Kong, China, August 2013.

- Software Transactional Networking: Concurrent and Consistent Policy Composition
  Marco Canini, Petr Kuznetsov, Dan Levin, and Stefan Schmid.
  ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN), Hong Kong, China, August 2013.

Thank you!

But wait! Next slide!

Internship in Berlin? We are hiring…!

Also help with open-source CloudNet prototype welcome 😊

Backup Slides

Policy Composition
The Problem of Policy Composition

- Existing controller platforms: “northbound API” forces programmers to reason manually about low-level dependencies between different parts of their code.

- Multiple tasks (e.g., routing, monitoring, …) how to ensure that packet-processing rules installed to perform one task do not override the functionality of another? Monolithic applications...

Solution: modularity! Programmer constructs complex application out of multiple modules that each partially specify the handling of the traffic. Modules that need to process the same traffic could run in parallel or in series.

Frenetic/Pyrethic

Modules with actions count, fwd, rewrite IP:

Parallel composition: Monitor and Route

Automatically composed (ordered wrt. priorities):

Compiled Prioritized Rule Set for “Monitor | Route”

| Action | srcip | dstip | Rule
|--------|-------|-------|-------------------|
| count  | 5.6.7.8 | 10.0.0.1 | datip=10.0.0.1 → fwd(1)
| count  | 5.6.7.8 | 10.0.0.2 | datip=10.0.0.2 → fwd(2)
| fwd    | 5.6.7.8 | 10.0.0.1 | datip=10.0.0.2 → fwd(2)

Sequential composition: Balance and Route

Automatically composed (ordered wrt. priorities):

Compiled Prioritized Rule Set for “Load-balance >> Route”

| Action | srcip | dstip | Rule
|--------|-------|-------|-------------------|
| count  | 5.6.7.8 | 10.0.0.1 | datip=10.0.0.1 → fwd(1)
| count  | 5.6.7.8 | 10.0.0.2 | datip=10.0.0.2 → fwd(2)
| fwd    | 5.6.7.8 | 10.0.0.1 | datip=10.0.0.2 → fwd(2)

Topology Abstraction with Network Objects

- Modular programming requires way to constrain what each module can see (information hiding) and do (protection).

- Network Objects (NO): Give familiar abstraction of a network topology to each module.

- Network Object consists of an abstract topology and a policy function applied to the abstract topology.

- For example, the abstract topology could be a subgraph of the real topology, one big virtual switch, or anything in between.

- May consist of a mix of physical and virtual switches, and even be nested!

- Example: MAC learning switch