Hot Topics
Software-defined Networking

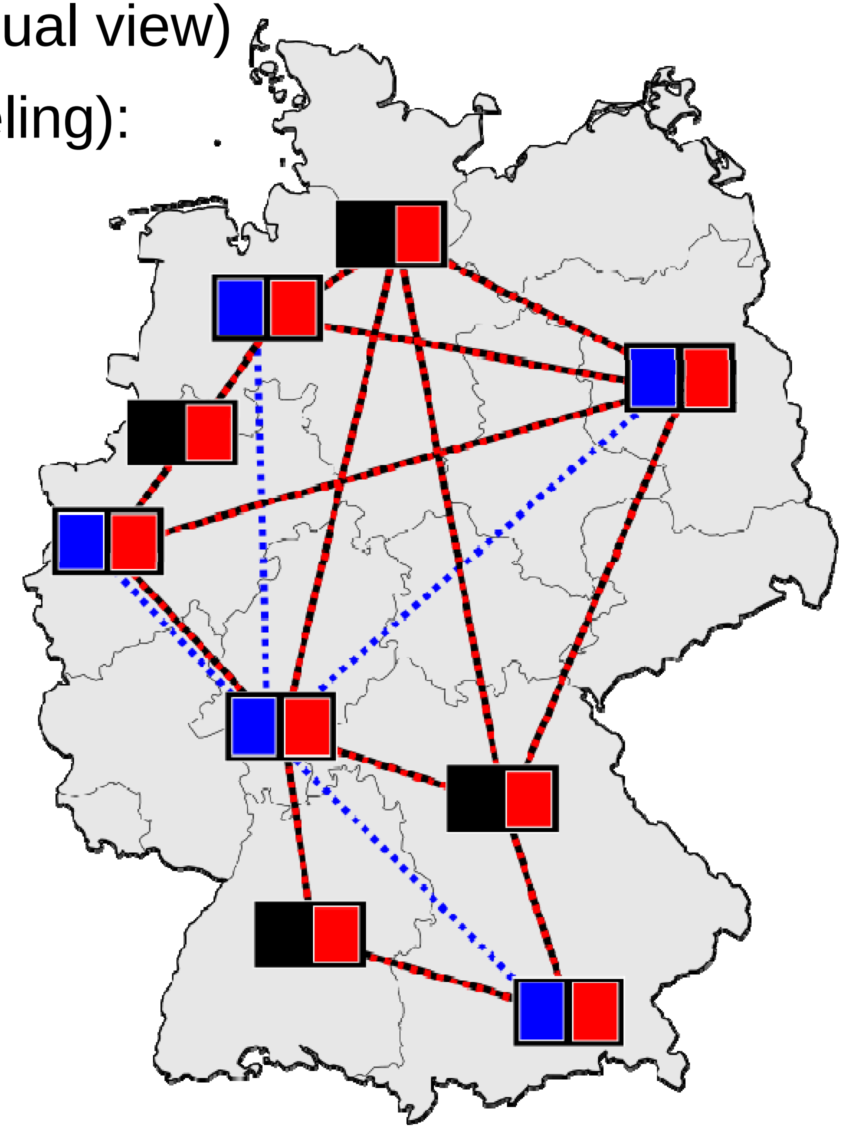
Contents - Hot Topics - SDN

- History
- Control and Data Plane Separation
- Network Virtualization
- Overview of Control Plane
- Network Functions Virtualization

History

Network Virtualization

- representation of one or more logical network topologies on the same infrastructure (network slicing with individual view)
- many different instantiations (using tunneling):
  - Virtual LAN (VLAN)
  - Stacked VLANs (QinQ)
  - Virtual Extensible LAN (VXLAN)
  - NVGRE (Network Virtualization using GRE)



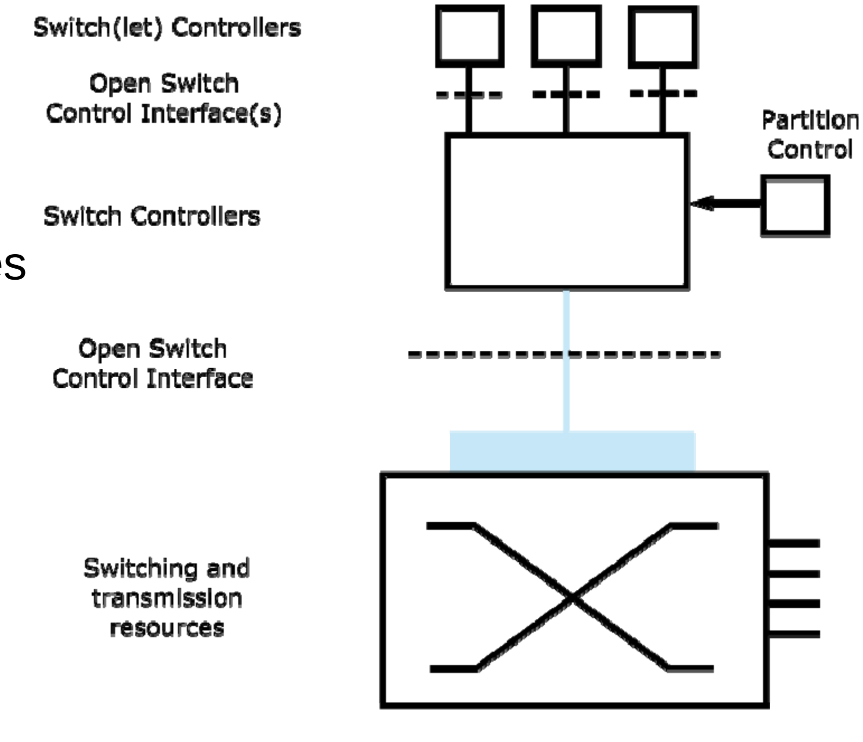
Network Virtualization - Benefits

- Sharing:
  - instantiate multiple logical routers on a single platform
  - requires resource isolation in CPU, memory, bandwidth, forwarding tables, etc.

- Customizability:
  - customizable routing and forwarding software
  - general-purpose CPUs for control plane
  - network processors and FPGAs/ASICs for data plane

Network Virtualization - Examples

- Tempest: Switchlets (1998)
  - virtualization of switches
  - separation of control framework from switches
  - similar to OpenFlow
  - problem: requires standardization, adoption
    and deployment of new hardware

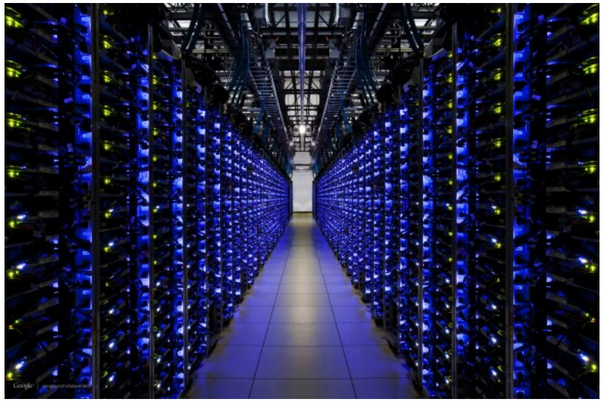


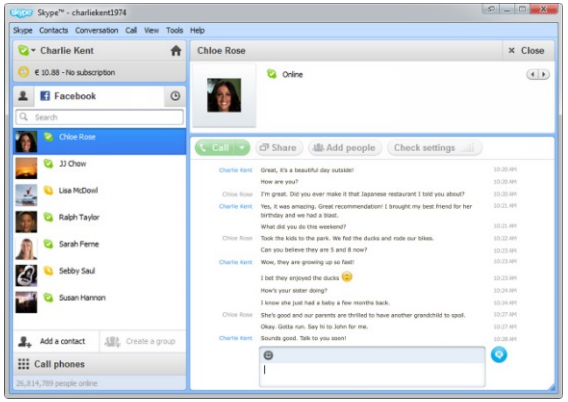
Open Signalling in the Tempest

- VINI: Virtual Network Infrastructure (2006)
  - virtualization of the network infrastructure
  - separation of data and control planes
  - bridging the gap between small-scale experiments and live deployment at scale

Network Virtualization - Examples

- Cabo: Concurrent Architectures are Better than One (2007)
  - virtualization of services
  - infrastructure providers (maintaining routers, links, datacenters) can operate independently from service providers (offering end-to-end/over-the-top services, e.g. VPN, Videotelephony)





Legacy of Virtual Networks for SDN:

- separating services from infrastructure (later: NFV)
- logic network topologies on top of physical infrastructure

Plane Separation - Examples

1) Separate control channel: IETF FORCES (RFC5810, 2003)

•forwarding elements (FE) can be controlled by multiple control elements (CE) via a standard control channel (FORCE):

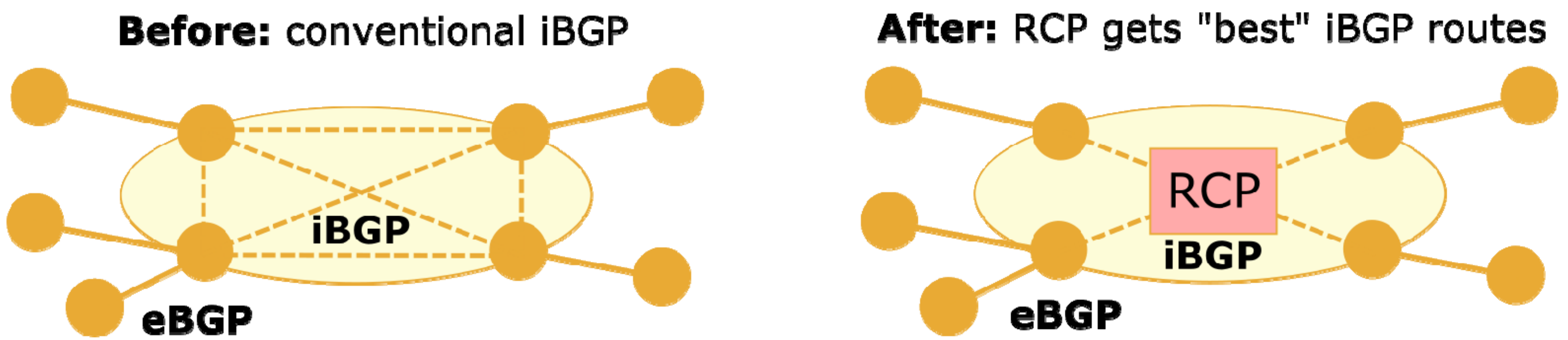
- → forwarding packets, metering, shaping, NATing, traffic classification
- Problem: requires standardization, adoption and deployment of new hardware

2) In-band signaling: Routing Control Platform (2004)

•reuse existing protocols as control channels

•one RCP per AS compute routes on behalf of the routers

•reuse existing routing protocol (BGP) to signal those routes



Plane Separation - Examples

2) In-band signaling: Routing Control Platform (2004)

•to routers it looks like a normal route announcement from another router (in fact to RCP)

•deployment easier using this approach → no standardization, adoption, deployment necessary

•Problem: control constrained by what existing protocol (i.e. BGP) supports

3) Customized hardware: Ethane (2007)

•allows direct enforcement of a single, fine-grained network policy

•domain controller computes flow table entries (to be installed in switches) based on access control policies

•Problem: requires custom switches supporting Ethane protocol: OpenWrt, NetFPGA, Linux

Plane Separation - Examples

4) Hardware with open interfaces: OpenFlow (2008)

•best of both worlds: operate on existing protocols without customizing hardware → OpenFlow

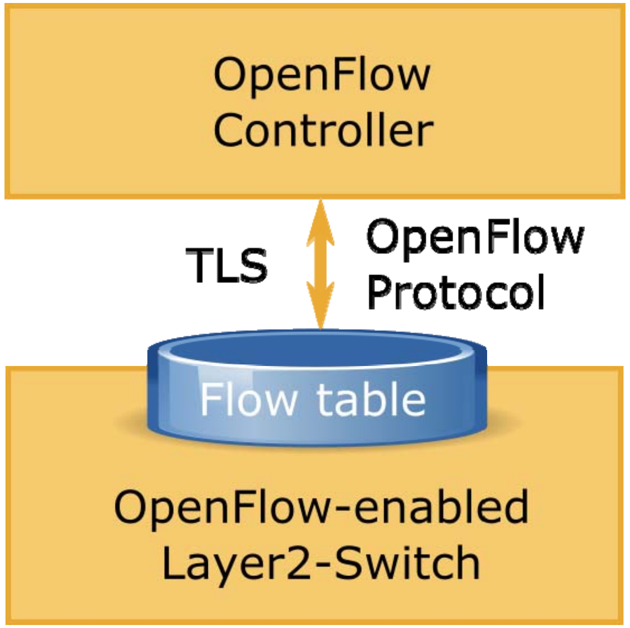
•taking existing capabilities of hardware switches (flow tables already implemented)

•switch exposes flow table through simple OpenFlow protocol

•vendors can keep platform closed, but expose an open interface to control forwarding table

•Switch matches subset of packet header fields:

- Switch port
- MAC src/dst
- Ethernet type
- VLAN ID/priority
- IP src/dst + IP protocol + IP ToS bits
- TCP/UDP sport/dport



Control and Data Plane Separation

Control and Data Plane Separation

1) control plane:

•logic for controlling forward behavior

•usually implemented in general-purpose hardware for easy modification/adaptation

•"brain" of the network

•examples: routing protocols, network middlebox configuration

2) data plane:

•forward traffic according to control plane logic

•often implemented in special hardware for increased throughput: ASICs, FPGAs

•examples: IP forwarding, Layer 2 switching

Plane Separation - Reasoning

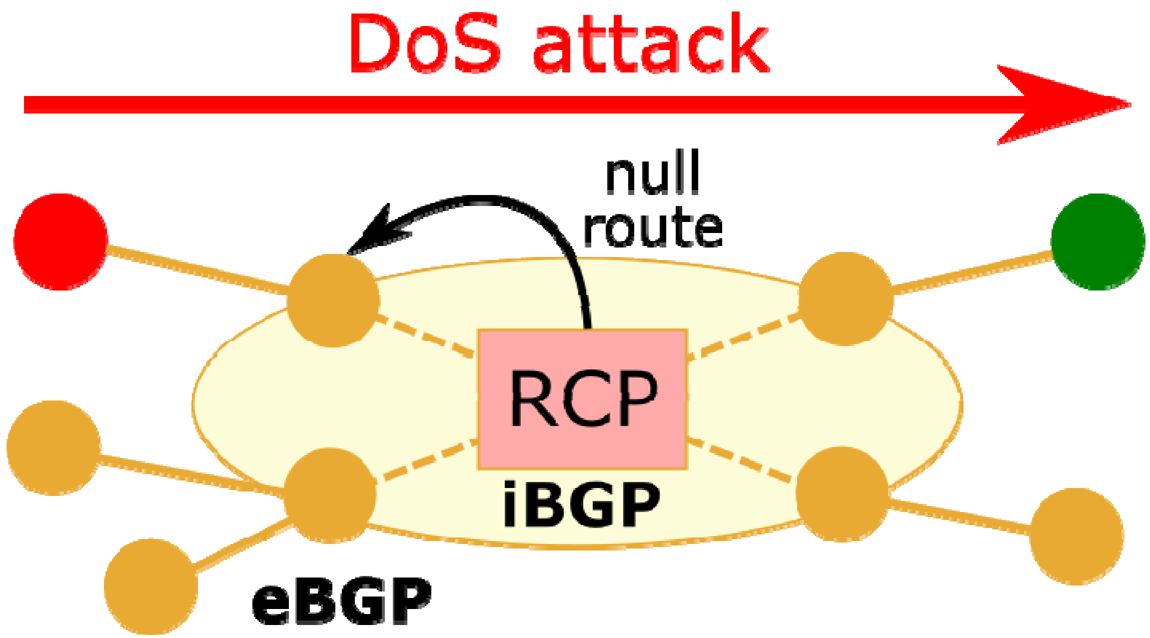
Why separation?

- greater flexibility: new services introduced more easily
- independent evolution and development: software can evolve independently of the hardware
- accelerating innovation in existing networks:
  - control logic is not tied to hardware → can be patched in software
  - introducing technologies more rapidly without consensus standardization
  - user driven innovation
- control from high-level software programs: high level of abstraction, debug/check more easily
- network-wide view of controller: easier to infer (and reason) about network behaviour

Plane Separation - Opportunities

1) mitigating DoS (Denial-of-service) attacks by filtering attack traffic:

- measurement system detects attack and identifies entry point and victim of attack
- control plane installs a null route at the entry point
- → data plane drops offending traffic at the entry point
- e.g. AT&T IRSCP (commercial RCP)



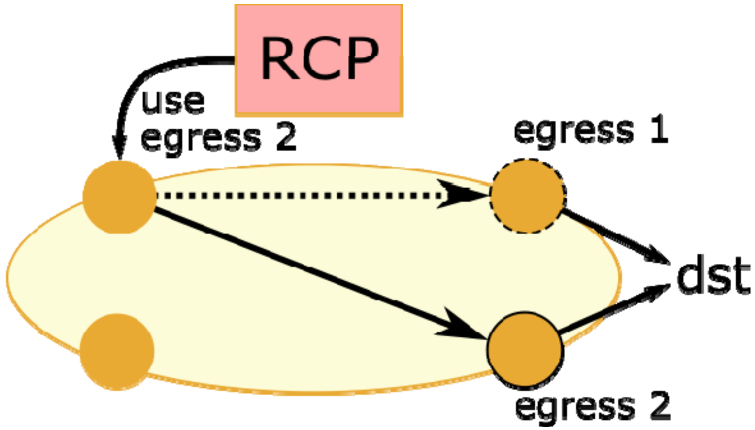
Plane Separation - Opportunities

2) Interdomain routing: Constrained policies

•artificially constrained routes of todays interdomain routing protocol BGP

- route selection based on a fixed set of steps
- limited knobs to control in-/outbound traffic
- incorporating additional information (reputation of route, time of day) problematic

•→ route selection on a richer set of policies: route controller can directly update state in forwarding elements



2.1) Maintenance dry-out

•e.g. planned maintenance on an edge router: directly tell ingress router to use egress2 instead of egress1

•much more difficult using traditional routing to adjust route of a single router: tuning route weights (indirect way)

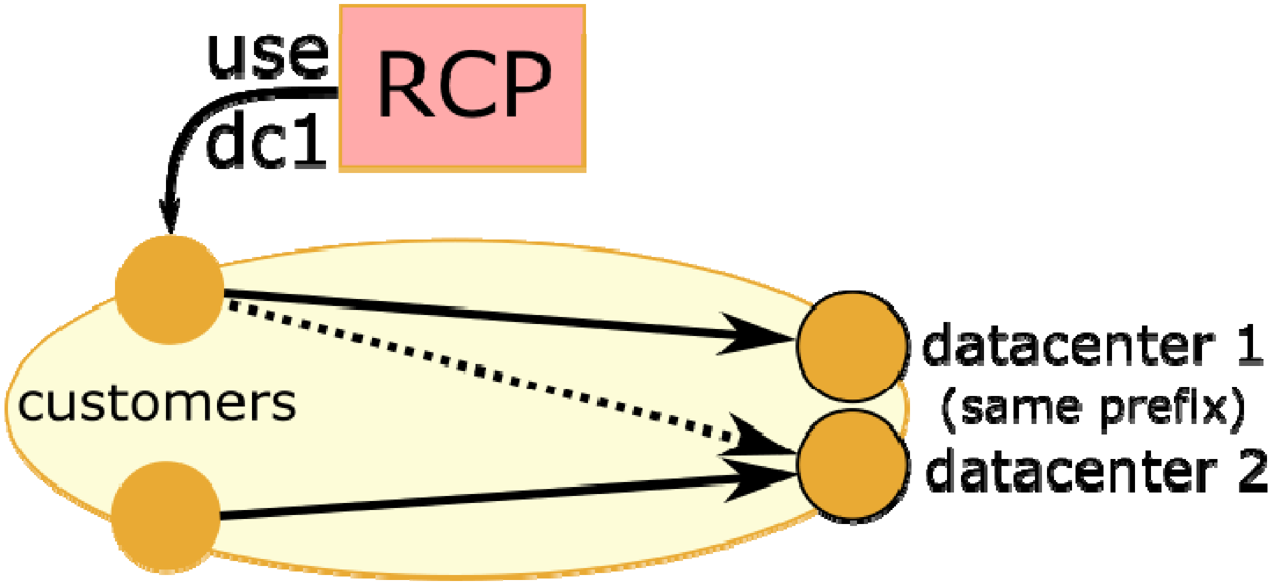
Plane Separation - Opportunities

2.2) Egress selection

•customer-controlled egress selection if multiple paths to reach same destination (depending on source rather then destination)

•possibly even giving customers control over the decision

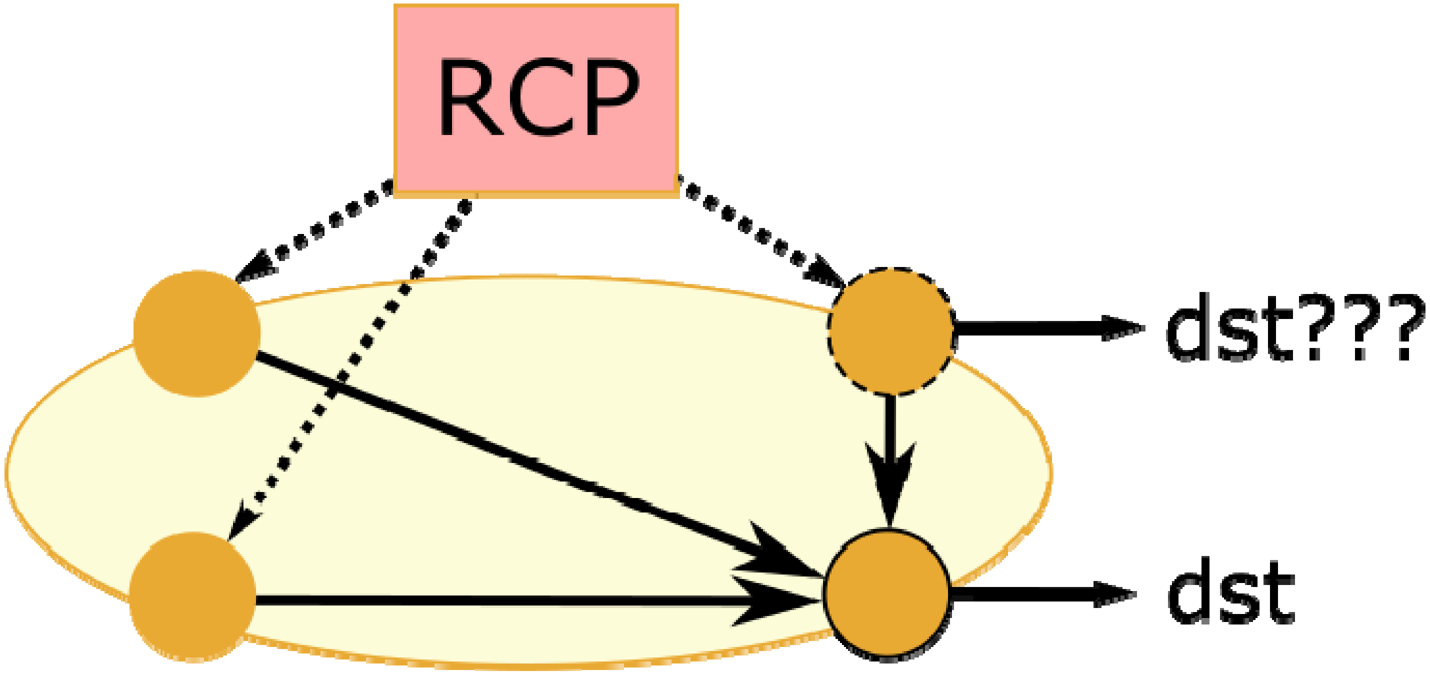
•today: routing traffic based on destination prefix



Plane Separation - Opportunities

2.2) Enhanced security

- anomaly detection: detecting suspicious/bogus routes
- prefer "familiar" routes over unfamiliar ones using reputation of routes
- today: no easy way to incorporate route-reputation into route selection



Plane Separation - Opportunities

3) Datacenters

•cost: expensive vendor switched vs. inexpensive commodity/off-the-shelf switches

•flexible control: tailor network for services, quickly improve and innovate

Other opportunities:

•dynamic access control

•seamless mobility/migration → decreased latency due to VM migration

•server load balancing

•network virtualization

•using multiple wireless access points

•energy-efficient networking

•adaptive traffic monitoring

•DoS attack detection/mitigation (see example 1)

Plane Separation - Challenges

1) Scalability

•one control element responsible for many (thousands) forwarding elements

•RCP: storing route and computing route decisions for every (potentially thousands) router

•→ single point of failure

•Solution: aggregation, hierarchy, redundancy, distributed coordination

2) Reliability/Security: What happens when a controller fails or is compromised?

•multiple identical controllers ("hot spare"), connected to same nodes, performing same route calculation → implicit consistency

•takes over if primary fails

Plane Separation - Challenges

3) Consistency

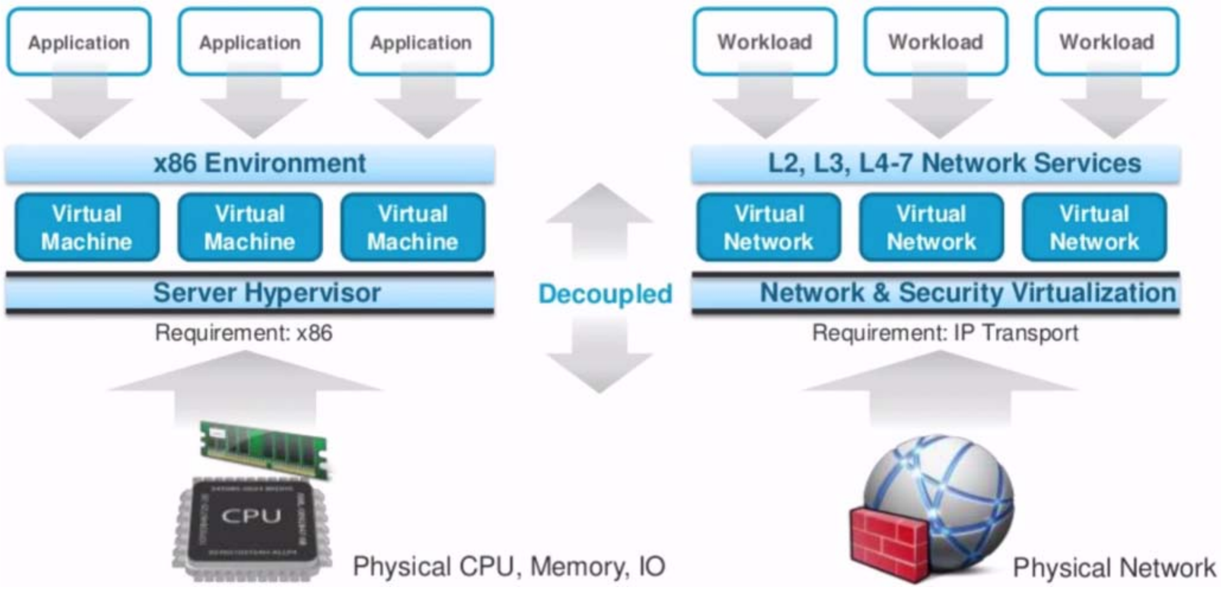
•implicit consistency

•syncing backup controllers repeatedly, possible inconsistency/loss of state using long intervals

Network Virtualization

Network Virtualization

- abstraction of the physical network
- support for multiple logical networks running on a common shared physical substrate
- container of network services
- aspects of networks that can be virtualized:
  - nodes: virtual machines (fully-fledged VMs - Xen, KVM, VMware, VirtualBox; LXC)
  - links: tunnels (VLAN, VXLAN, GRE, Open vSwitch etc.)
  - storage:
    - block (Fibre Channel, iSCSI, SAS)
    - file (NFS, CIFS/SMB, AFS)



Network Virtualization - Motivation

- 2000s: lots of work on overlay networks on top of IP (here to stay)
- realization that one-size-fits all architectures are difficult
- Why not allow easier evolution?
- Promises:
  - rapid innovation: services delivered at software speeds
  - new forms of network control
  - vendor independence
  - simplified programming and operations
- Distinction:
  - SDN does not inherently abstract the details of the physical network
  - it does separate control and data plane
  - network virtualization provides the separation of logical and physical networks
  - SDN can be a useful tool for implementing virtual networks

Network Virtualization - Design goals

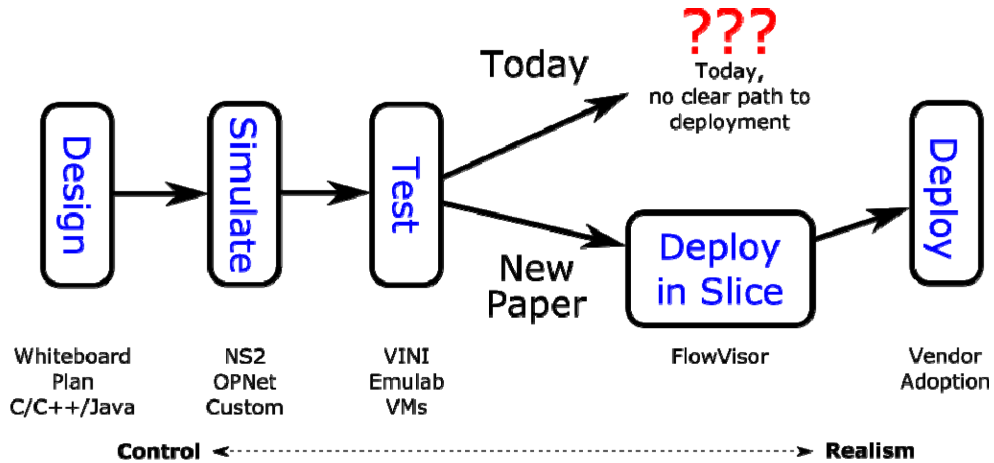
1. Flexibility: topologies, routing and forwarding architecture; independent configuration
2. Manageability: separate policy and mechanisms
3. Scalability: maximize number of co-existing virtual networks
4. Security and Isolation: isolate both the logical networks and resources
5. Programmability: customizable/reprogrammable router, etc.
6. Heterogeneity: technology-agnostic, support for different technologies

Network Virtualization - Examples and Applications

1) Experimentation on production networks

•How to test and deploy a "paper" design? → Goal: Realism

•Ideally: Deploy in parallel on production network



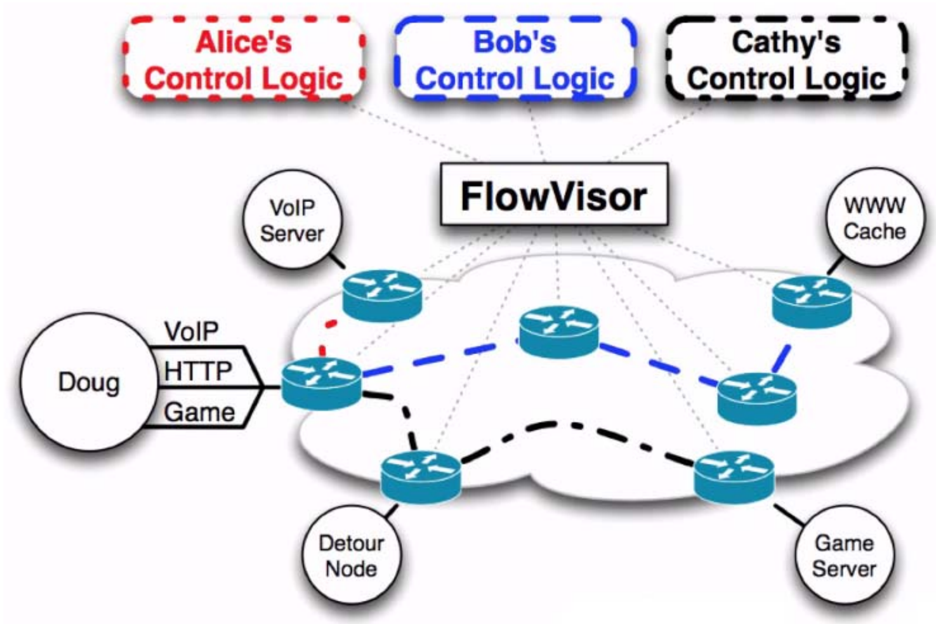
•FlowVisor: experimental traffic runs in parallel to production network

•different flows of user ("Doug") can be controlled by different controller designs (i.e. experimental protocols/architectures)

•a subset of flows (uncritical) can be controlled by new designs

•→ virtualization on flow space (5 tuple: IP addresses, ports, protocol)

Network Virtualization - Examples and Applications



2) Dynamic scaling of resources

•own datacenter with limited resources

•fluctuation on demand, disaster, DDoS → additional resources required

•dynamically provision additional resources on demand as a leased service (dynamic scaling)

•Example: Amazon Virtual Private Cloud, EC2

- allows customer to define own network, address space, etc.
- extend existing enterprise data center

Network Virtualization - Examples and Applications

3) Network function virtualization

•unification of middlebox function

•today: purchase a variety of middleboxes separately: firewall, load-balancer, DPI, IDS/IPS

•instead:

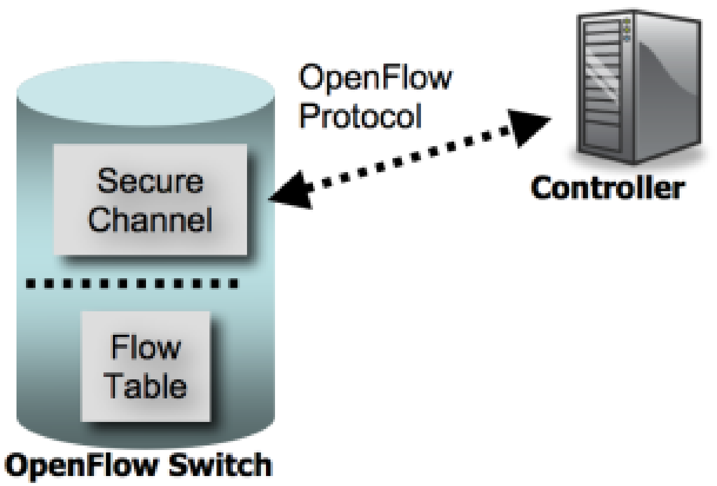
- distributed compute pool
- dynamically install this functions as software (potentially on VMs)
- interlinking via Virtual Networks

4) Rapid deployment and development of new network services

Overview of Control Plane

Overview of Control Plane - OpenFlow Specification

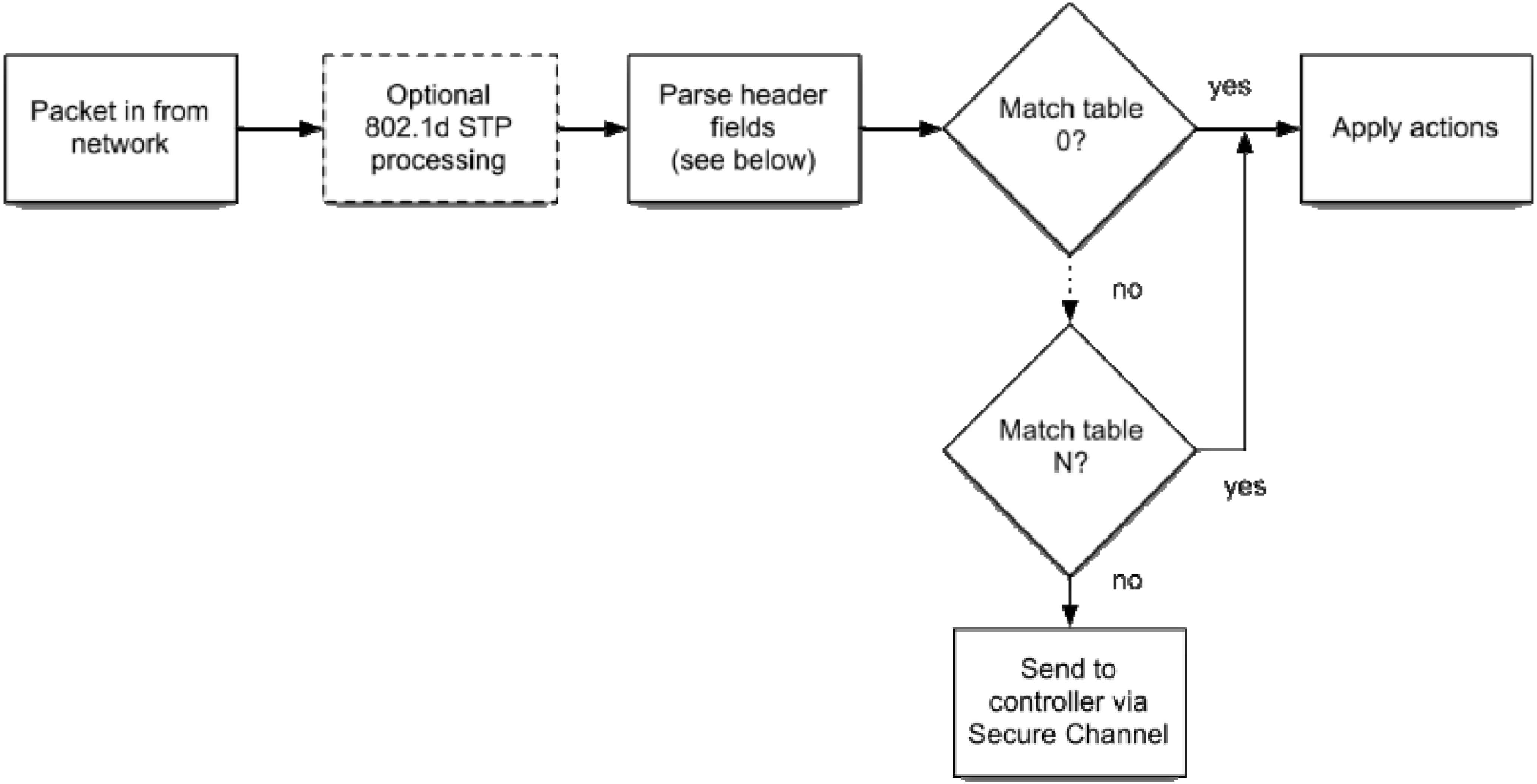
- OpenFlow controller communicates with switch over a secure channel
- OpenFlow protocol defines components of the switch, message format, types of actions the flow table should be able to perform
- purpose of control channel: update flow table entries in switch
- controller centric: logic executed at controller
- switch only forwards traffic based on flow table (data plane)



Overview of Control Plane - OpenFlow Switch Components

- Flow table: Performs packet lookup
  - matching the headers of incoming packets to flow table
  - performing action based on match being found (forward, drop, modify, enqueue)
  - no match → traffic is sent to controller
- Switch matches subset of packet header fields (12-tuple, wildcards):
  - Switch port
  - MAC src/dst
  - Ethernet type
  - VLAN ID/priority
  - IP src/dst
  - IP protocol
  - IP ToS bits
  - TCP/UDP sport/dport
- Secure channel: Communication to external controller
  - listens on control port 6634
  - inspecting flow table entries, modify flows, etc. (e.g. dpctl userspace program)

Overview of Control Plane - Matching



Overview of Control Plane - Mandatory Actions

- must be supported by OpenFlow v1.0 compliant switches

- Forward:
  - ALL: send out on all interfaces, except the incoming interface
  - CONTROLLER: encapsulate and send to controller
  - LOCAL: send to switch's local networking stack
  - TABLE: perform actions in flow table (packet-out messages)
  - IN PORT: sent packet out to the port, that packet was incoming from
  - optional: normal forwarding (based on routing table), spanning tree

- Drop:
  - flow-entry with no specified action → drop all matching packets

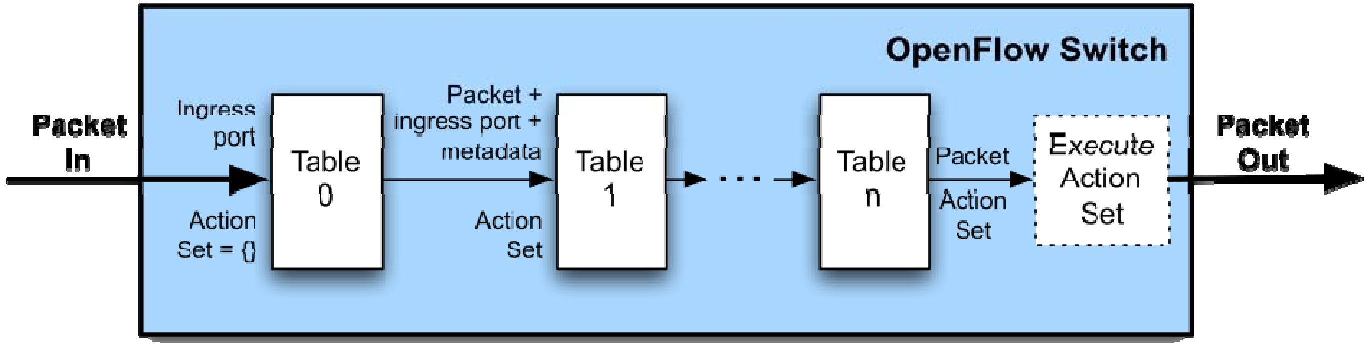
Overview of Control Plane - Optional Actions

- may be supported by OpenFlow v1.0 compliant switches

- Modify-Field : modify packet header values of the packet, e.g.
  - VLAN ID → redirect to logically separate networks
  - destination IP address → load balancing

- Enqueue: send packet through a queue attached to a port
  - apply QoS
  - traffic shaping

Overview of Control Plane - OpenFlow v1.3 Enhancements



- Action set: perform a set of actions on each matching packet
- Group: a list of action sets, allows switch to refer to a common set of actions performed on multiple sets of matching flows
- each table can update fields, modify action set → "Execute Action Set" performed before packet leaves
- recent version: v1.4 (Oct. 2013), OpenFlow is evolving

Overview of Control Plane - Action Group Options

- execute all action sets in a group
- e.g. implementing multicast: one packet is cloned for each action set in the group

Indirect group:

- one action set in a group is executed: performing same set of operations on multiple flow entries

Example actions:

- TTL: decrement, copy
- MPLS: apply (push) MPLS tags to a packet
- QoS: apply QoS actions (e.g. set_queue) to a packet
- metering and traffic monitoring

Overview of Control Plane - Other SDN Architectures

- Juniper's Contrail Controller
  - Linux-based
  - XMPP as control plane
  - L2 and L3 virtual networks
  - Contribution to OpenDaylight (FOSS implementation of various SDN control architectures)

- Cisco's Open Network Environment
  - centralized software controller
  - programmable data plane
  - ability to provide virtual overlays

- OpenFlow by far the most common SDN control architecture

Overview of Control Plane - OpenFlow Controllers

- only cover "Southbound interface": control channel between SDN controller and switches
- "Northbound interface": higher level of abstraction, policy layers on top of lower level SDN channels

1) NOX/POX

- first-gen OpenFlow controller
- FOSS, stable, widely used
- high-performance, clean codebase, well maintained and supported
- users implement control logic C++
- supports OpenFlow v1.0 (fork CPqD up to v1.3)
- model: controller registers for events → programmer writes event handlers
- low-level facilities and semantics of OpenFlow (low level of abstractions)
- POX (Python equivalent): easy to read/write code, worse performance

Overview of Control Plane - OpenFlow Controllers

2) Ryu

- implemented in Python → low performance
- supports OpenFlow up to v1.3
- works with OpenStack
- aims to be "Operating System" for SDN

3) Floodlight

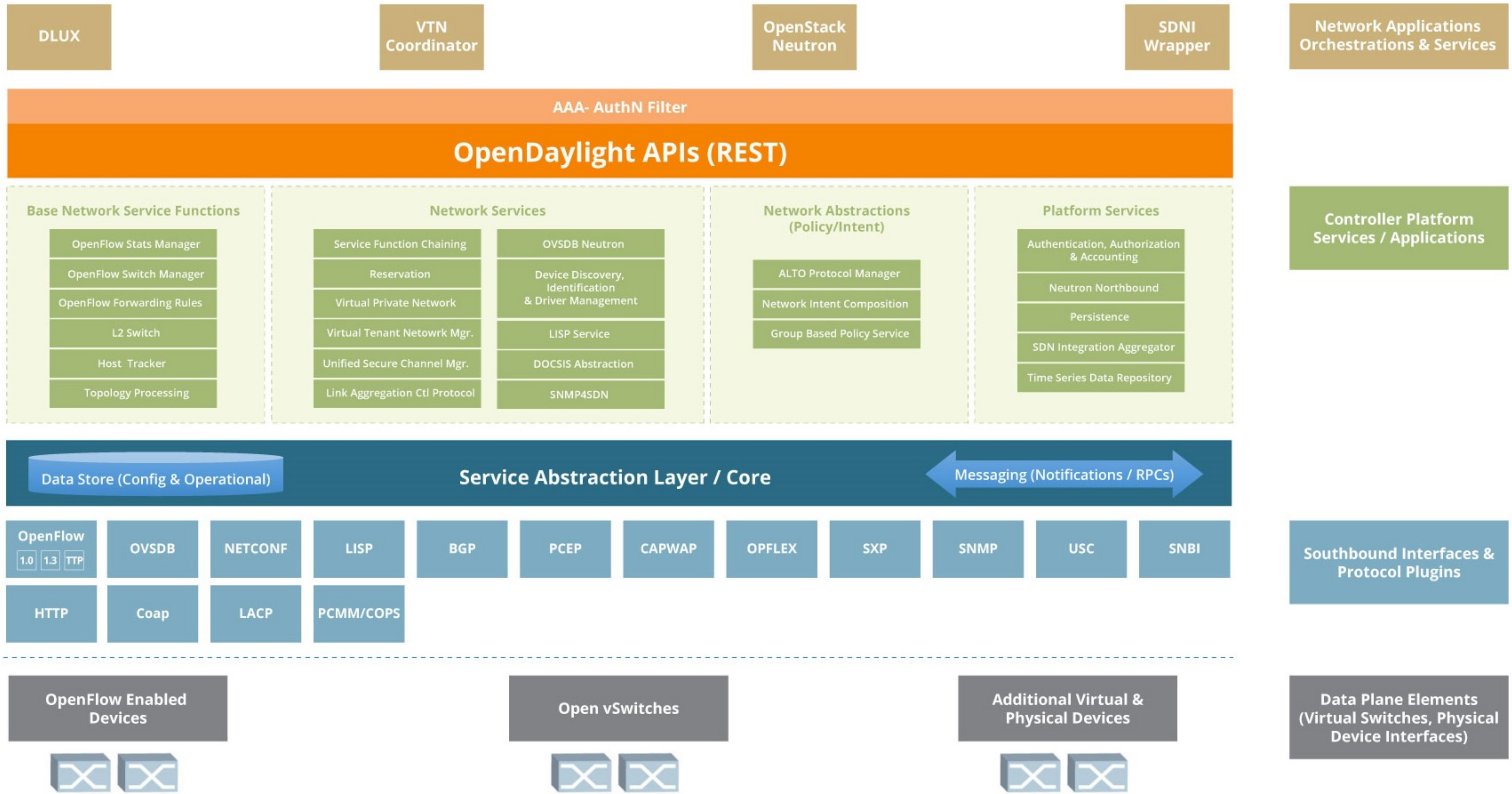
- OpenFlow v1.0
- implemented in Java
- good documentation
- integration with REST API (RPC)
- production-level performance
- OpenStack integration
- disadvantage: steep learning curve

Overview of Control Plane - OpenFlow Controllers

3) OpenDaylight

- implemented in Java
- robust, extensible FOSS codebase
- heavy industry involvement and backing (8 platinum, 1 gold, 42 silver)
- common abstraction for northbound capabilities
- focus: open framework for building upon SDN/NFV innovations
- → not limited to OpenFlow
- advantage: industry acceptance, integration with OpenStack, cloud applications, etc.
- disadvantage: complex → steep learning curve, rather poor documentation

Overview of Control Plane - OpenDaylight Architecture



Overview of Control Plane - Controller Comparison

	NOX	POX	Ryu	Floodlight	ODL
Language	C++	Python	Python	Java	Java
Performance	Fast	Slow	Slow	Fast	Fast
Distributed	No	No	Yes	Yes	Yes
OpenFlow	1.0	1.0	1.0, 1.1, 1.3, 1.4	1.0	1.0, 1.3
Multi-tenant clouds	No	No	Yes	Yes	Yes
Learning curve	Moderate	Easy	Moderate	Steep	Steep

Control Plane - Motivation for "Northbound" Interface

OpenFlow programming is not easy:

•low level of abstraction (channel to control flow table entries)

•difficult to implement higher level task (e.g. security policies, load balancing) using OpenFlow

•difficult to perform multiple independent tasks (e.g. interfering/conflicting routing and access control)

•only unhandled packets inspected by controller → incomplete view

•race conditions, if flow tables rules are not installed properly, e.g. "routing" loops

→ "Northbound" interface:

•programming interface allows applications and orchestration systems to program the network

•higher level of abstraction

•→ policies are being "compiled" into OpenFlow rules

Control Plane - Motivation for "Northbound" Interface

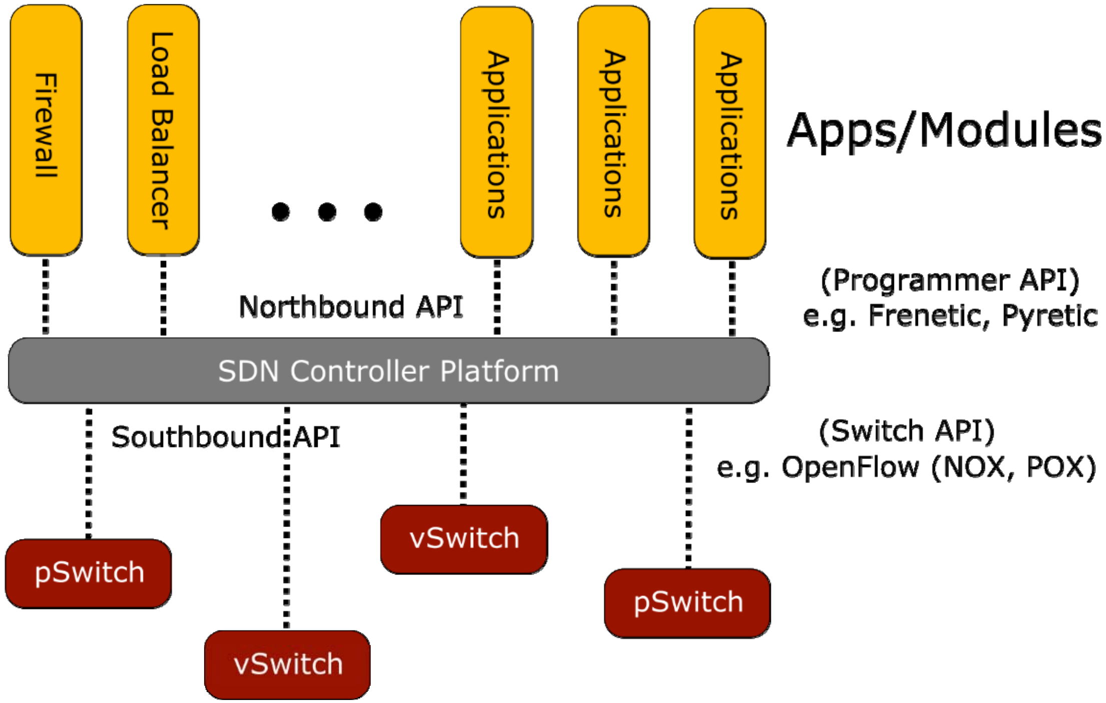
Use cases:

•path computation

•routing

•recover from failures

•security policies



Users of the "Northbound" interface:

•sophisticated network operators

•service providers: value-added services

•vendors: create services on top of a switch/controller

•researches

•anyone, who wants to develop capabilities on top of OpenFlow

Control Plane - Motivation for "Northbound" Interface

Benefits:

- vendor independence
- quickly modify/customize control through popular programming languages

Examples:

- large virtual switch
- security applications
- resource management and control (traffic engineering/load balancing)
- middlebox integration

Network Functions Virtualization

Network Functions Virtualization

- place arbitrary functions in VMs and distribute them across the network
- status quo: middlebox functions (firewall, load-balancer, DPI, IDS/IPS) placed in separate, monolithic middleboxes
- NFV: functions distributed in VMs/containers across the network
- decoupling functions from hardware → increased flexibility how packet processing is performed

Benefits:

- reduced CAPEX/OPEX, reduced time to market
- elastic scaling
- vendor agnostic

New use cases:

- virtualized services for enterprises, virtual CDNs
- virtualized mobile core network (decreased latency)
- integration of production and testing

Network Functions Virtualization

- fine-grained functional elements, instead of monolithic middleboxes, e.g.
- WAN optimizer = Caching + Deduplication + Compression + Encryption + FEC + Rate Limiter
- Application firewall = IP-Defragmenter + Application Detection Engine + Logger + Blocker
- → placing individual functional elements in virtual containers and chain them according to middlebox functionality, reuse of functional elements
- Problem: Orchestration and customizability
  - enable network operators to implement modular network functions without worrying about Placement and Steering
  - add custom middlebox functions inside network data plane
- Future work:
  - (better) algorithms for placement and steering
  - high-speed data plane implementation: throughput, instantiation, migration
  - better ways to resolve policy conflicts

NFV - Difficulties in Placement and Steering

Placement: Where to place functions in the network? Goals:

- minimize the number of locations to place the elements
- minimize the bandwidth utilization for implementing the policy
- minimize the latency of flows for implementing the policy

Steering: How to route traffic through these functions? Goals:

- not only shortest path
- consistent ordering
- chaining in a certain order
- dynamic chaining: enable path changes for packet flow based on certain conditions (e.g. stateful firewalls)
- load awareness

NFV - Difficulties in Placement and Steering

Mapping traffic flows and demands to:

- available network resources (i.e. paths)
- available processing capacity (i.e. middleboxes)

- need for a unified abstraction for control, data and storage
- continuously monitor link and machine loads
- create new element instances in case of overload
- steer some traffic through new instances
- reclaim unused element instances