Microsoft Research

Microsoft

Network OS in a Cloud

os2atc-2014

Kun Tan
Senior Researcher
Wireless and Networking
Microsoft Research, Asia

Network Operating System

A key component in *software defined networking*

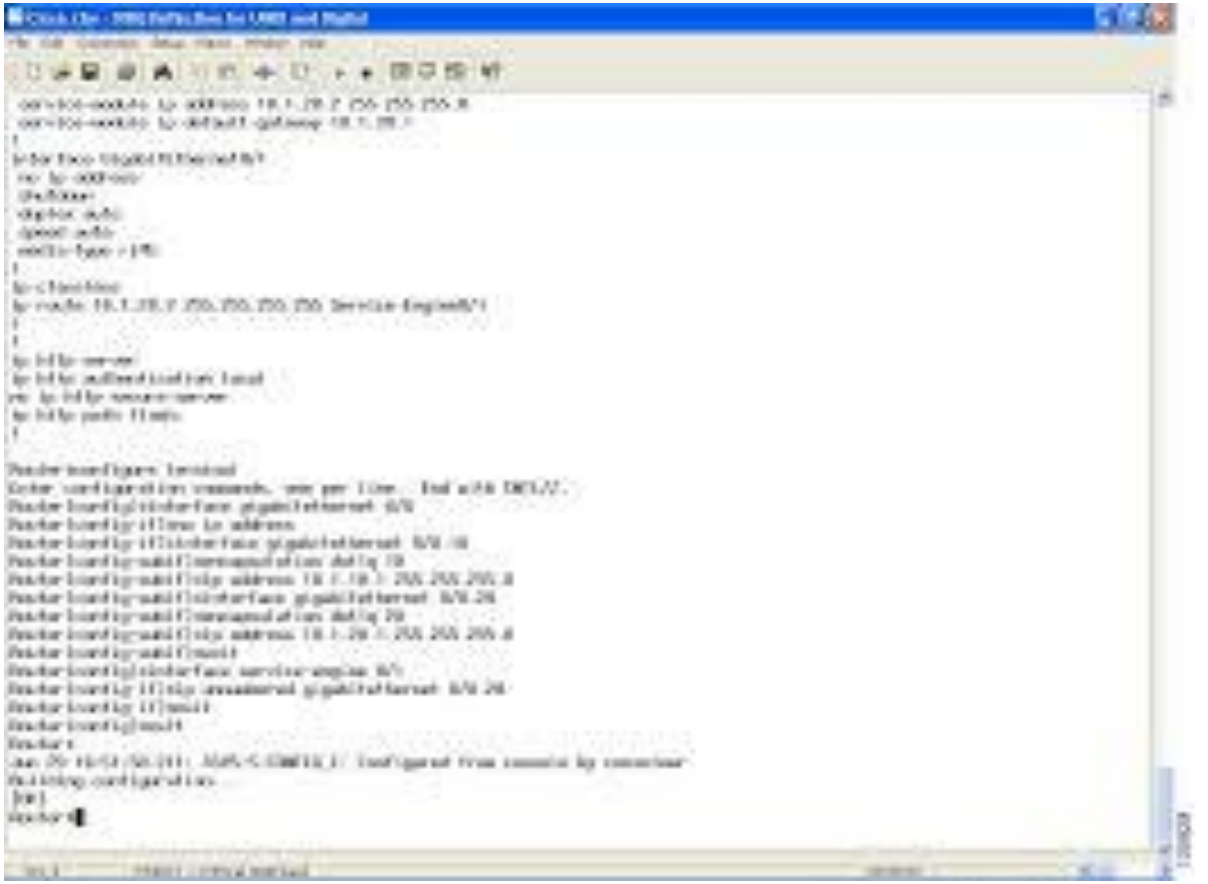
Decoupled programmable control plane in a network

Before SDN, network is controlled by many distributed protocols

Routing – how to forward a packet to a destination
Discovering – who is peering with me?
Managing protocols and control policing – ACL, TE, etc.
Network functions (i.e., middleboxes) – load-balancer, firewalls, etc.

Complexity…

Individual configuration to each distributed network device



Complexity…

Individual configuration to each distributed network device

Time consuming to make changes

Error prone

Misconfiguration contributes major network outage!

Unpredictable

Difficult to diagnosis when things go wrong

Good news: our Internet still works

So we are good at mastering complexity ☺

Complexity does matter

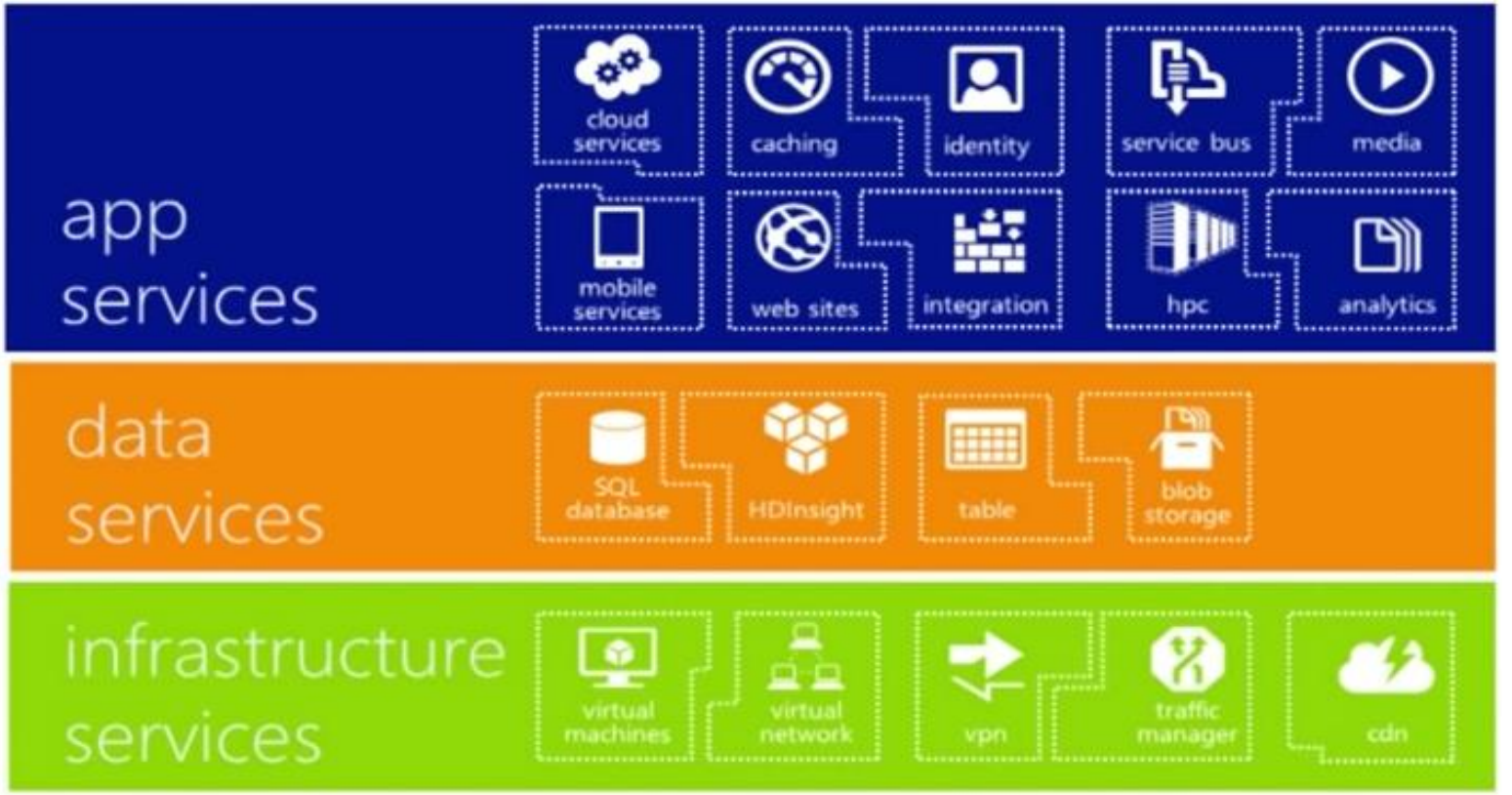
In the era of cloud computing…

Companies move their IT infrastructure to clouds

Example: Windows Azure

Open platform with rich services: computing, storage, networking, etc.

Less expensive than on-premises DC

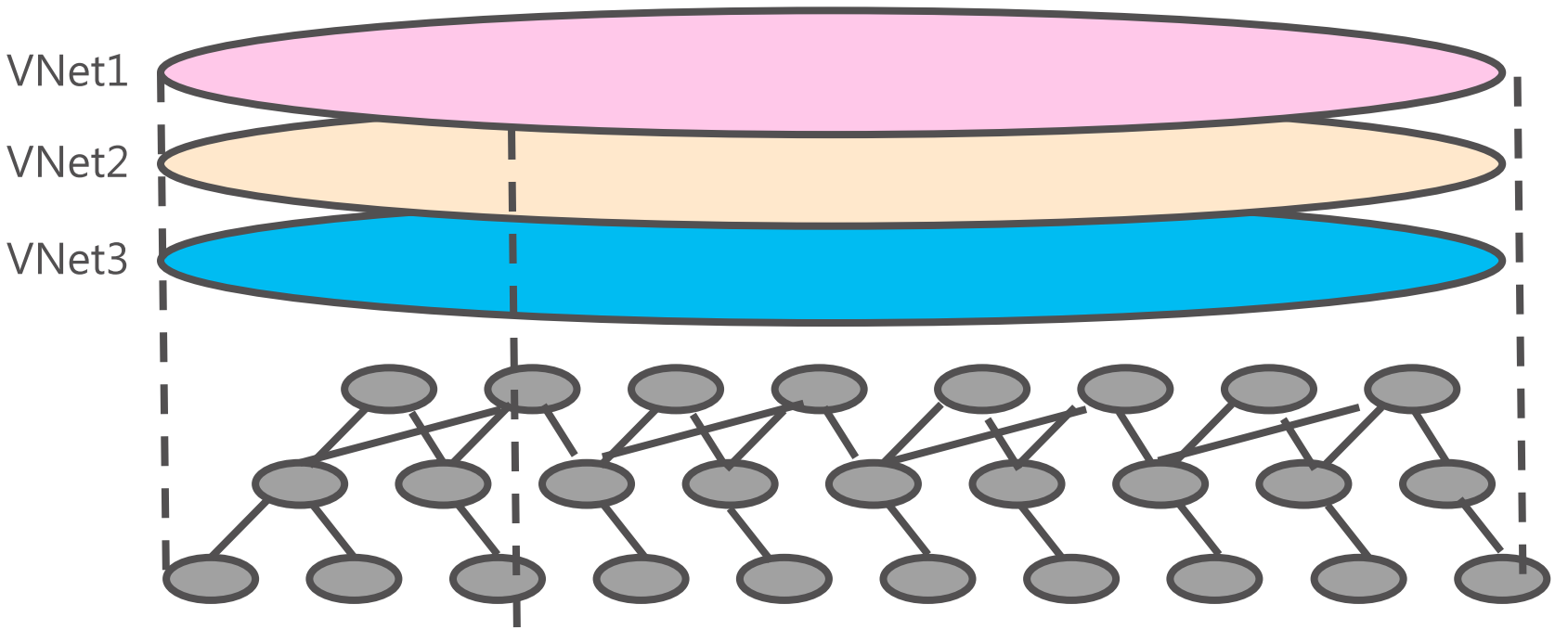


Example: Windows Azure

25 DCs world-wide hosting 20 million tenants

All tenants are running in virtualized network environments

Example: Windows Azure

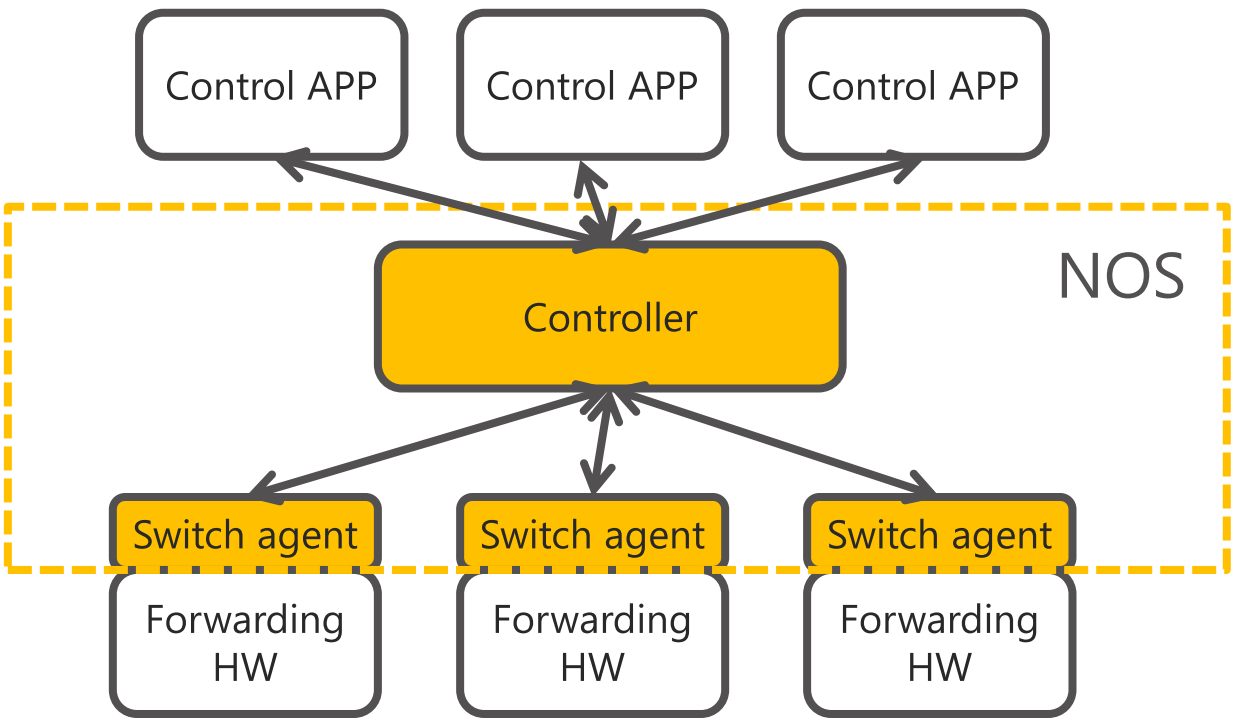


Require network to be agile, highly available and robust

Paradigm shifting to SDN

Traditional approach simply cannot catch the evaluation of cloud computing

We need new network architecture



Major changes

Distributed protocols → logically central control programs

Low-level primitives → High-level abstractions

(Semi-)manual configuration → software automation

Key design issues in NOS

Abstraction

What is the *right* view, or APIs, of the network to control apps and switches?

Scalability

Support 50K+ VNets in a DC with 100K+ servers and 2K+ switches

Availability and reliability

How to handle failures

Abstraction

Data plane abstraction: flow tables

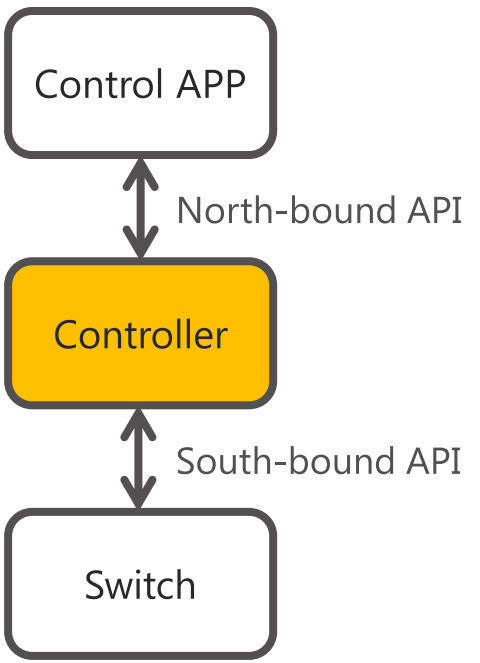
Simple, but general enough to specify per-packet operation of a switch

The element in a flow table is a *rule*

Two components: a match list and an action list

For any incoming packet, the switch simply looks up in flow tables to find matching rules.

If any found, the corresponding actions are taken



Ex.

	Match	Action
Rule 1	Dst ip: 10.0.0.1	forward to port 0
Rule 2	Src ip: 132.23.0.0/16	drop
Rule 3	Src ip: 10.0.0.0/16; dst ip: 10.1.0.0/16	Rate limit = 10Mbps

Challenge: diversity

Flow table schemas on switches are diverse

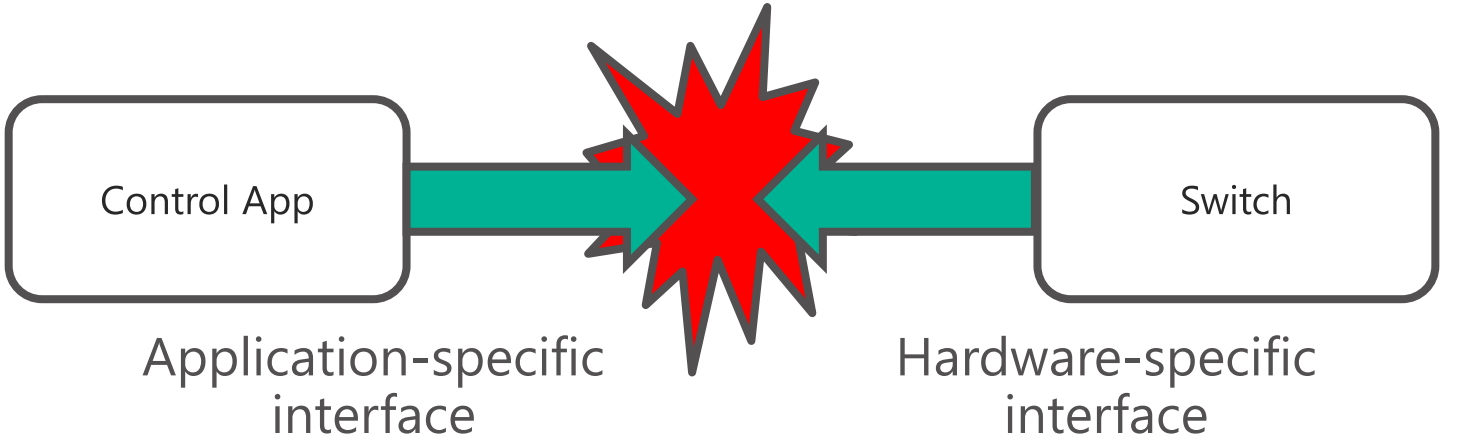
Each switch vendor may bring different flow table definitions and processing order.

Complex and optimized for hardware implementations: e.g., Broadcom T2 contains 800+ various flow tables.

The control applications are diverse

Most of apps only touch a small portion of switch functionalities: e.g., routing, ACL, QoS, etc.

Simple, intuitive, portable, optimized for application logics



Dealing diversity

Manually dealing diversity does not scale

#1 Exposing hardware API to SDN App dev: rewriting the app for each hardware platform

#2 Adding an intermediate "driver" to map APP API to HW API: needs $O(NM)$ such drivers

Sometimes, such drivers are not easy to write: non-trivial human efforts or suboptimal utilization of hardware resources

Examples

#1 Routing in a rack: a single L2/L3 table

VLAN	DMAC	DIP (*)	SMAC(*)	DMAC(*)	OPORT
1	F8:e5:10	*	*	*	1
1	F8:e5:01	10.0.0.10	F8:e5:01	F8:e5:03	2

A single table is intuitive. A careless "driver" may only map it to TCAM resource of a switch with limited capacity (4K).

An optimal mapper can fully utilize the L2/L3 resources of a switch

#2 Managing ACL in groups

SIP	DIP	grpID
10.0.0/24	*	1
10.0.1/24	10.0.0/24	1

grpID	ACLOPT
1	Drop
2	accept

May not find any simple map to any hardware resources

Our approach: flow translator

Goal: automate the translation from the *virtual flow tables* defined by app to the *physical flow tables* on switch hardware

s.t.

Correctness:

A packet that passes the physical flow tables is applied the same actions as if it passes the virtual flow tables.

Efficiency:

All hardware resources that can accommodate a VFT translation are efficiently exploited.

Algorithm overview

Multi-table schemas

For both virtual flow tables and physical flow tables

Global vocabulary for match fields and actions

So, we can match a field between VFT and PFT easily

Two-step translation process

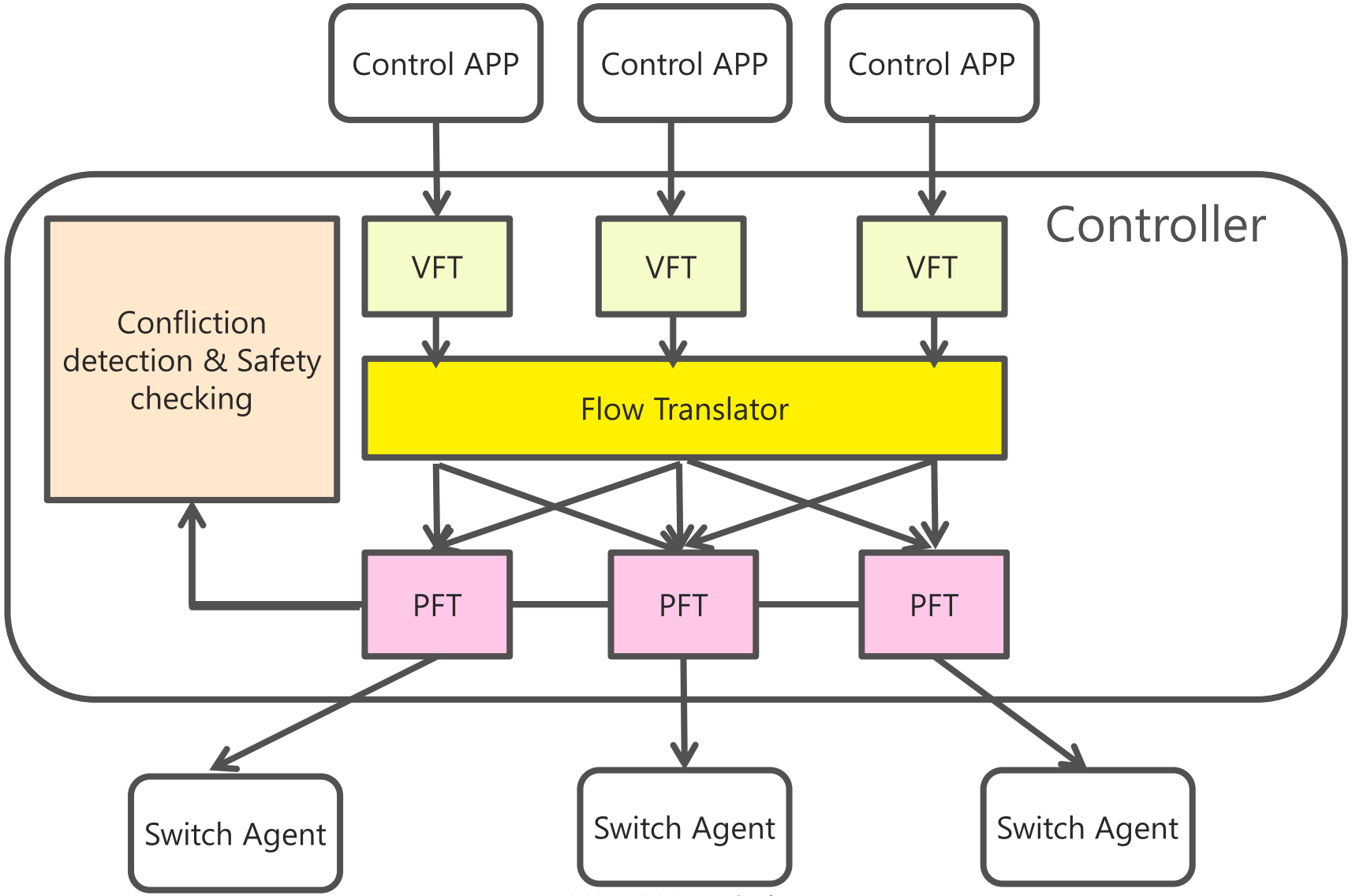
#1 Offline matching.

Given VFTs, finding all possible mappings for each VT to a set of PTs, s.t. the correctness condition

#2 Online translation.

When a new rule is inserted in VT, populating the rules to corresponding PTs based on the offline mappings. If one resource is exhausted, another possible mapping is picked

Architecture



Conflict detection & safety check

Safe-guard the shared resource in network

Confliction resolution

Last-writer-wins
Transaction-based operation
Priority-based updating

Safety checking

Keep certain invariant of the network
#1 Connectivity – network should remain connected
#2 Capacity reservation

Scalability

#1 Scale number of many *virtualized networks* in a cloud

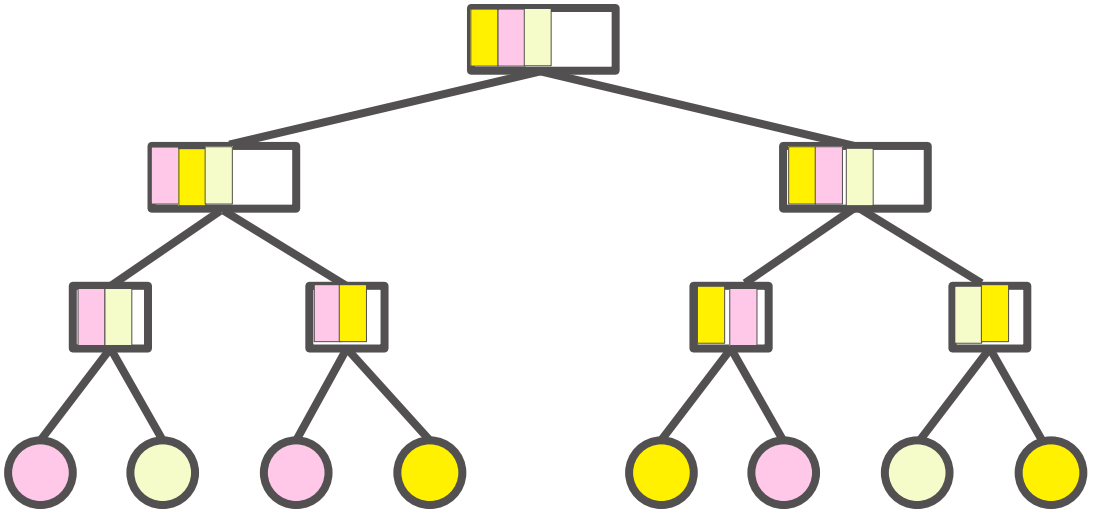
50K+ VNet in one DC

Connectivity and isolation

VLAN does not scale

up to 4096 VLAN tags

L2 solution, too many states in each switch



Scalability

#2 Scale the network function with the all cloud traffic

~ Tbps

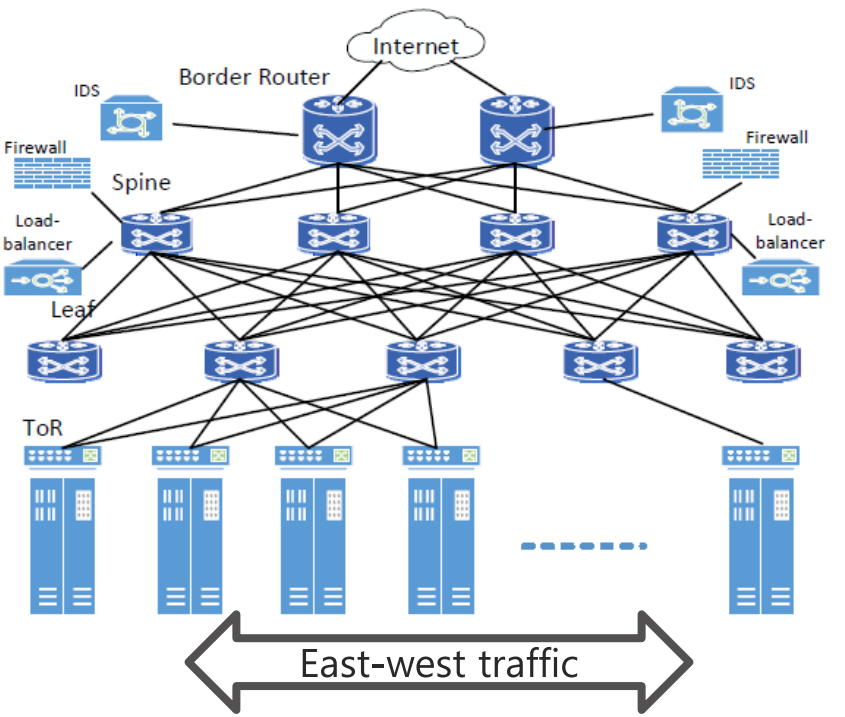
Fixed NF deployment does not scale

Specialized boxes at selected "choke" points

Inflated routing

Shared, little performance isolation

Limited types



Solution: scale-out

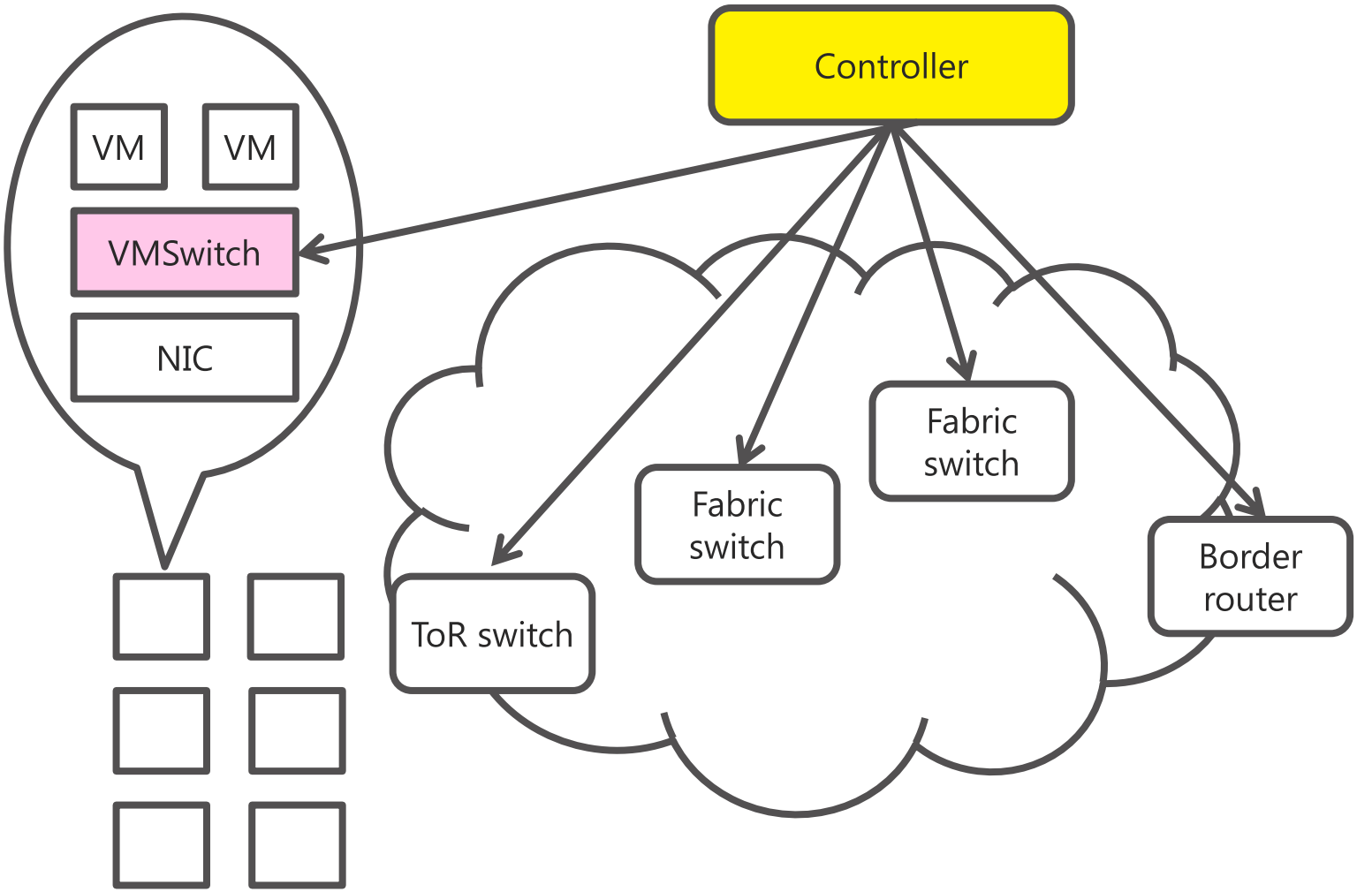
Push states to hosts

Host networking – VMSwitch is part of the network!

Embrace network function virtualization

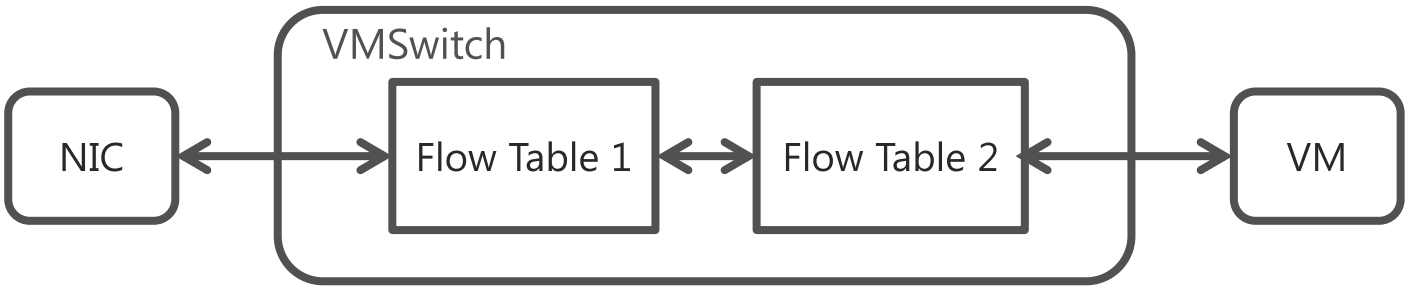
Use software and commodity servers for NF processing
Scale NF processing with the cloud
Offloading whenever possible to improve the performance

Host networking



VMSwitch

Still a switch



A piece of software that is scale-out

Flexible and easy to maintain large states

Network function virtualization

Cloud provider offers explicit middle-box services to tenants

An ***appstore*** for middleboxes: *Middlebox-as-a-Service (MaaS)*

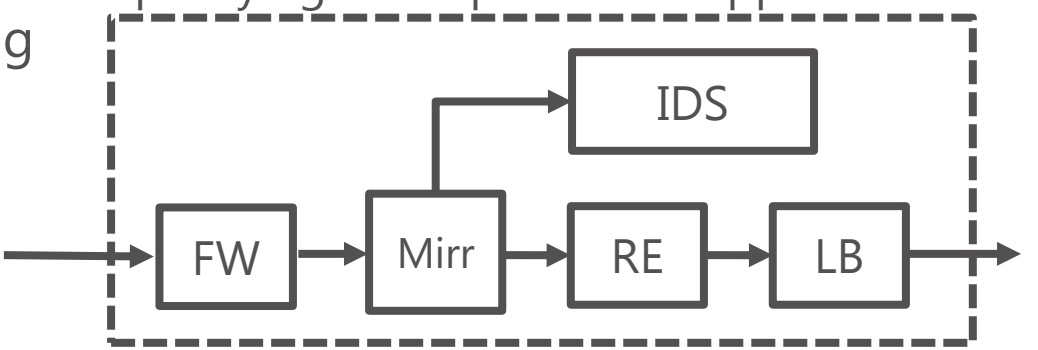
Tenant specifies what they need and the provider takes care the deployment and management of MB apps

Composite a middlebox service from elementary middlebox applications

Middlebox specification

Each elementary MB app is running inside a virtual machine.

Specifying a composite MB App

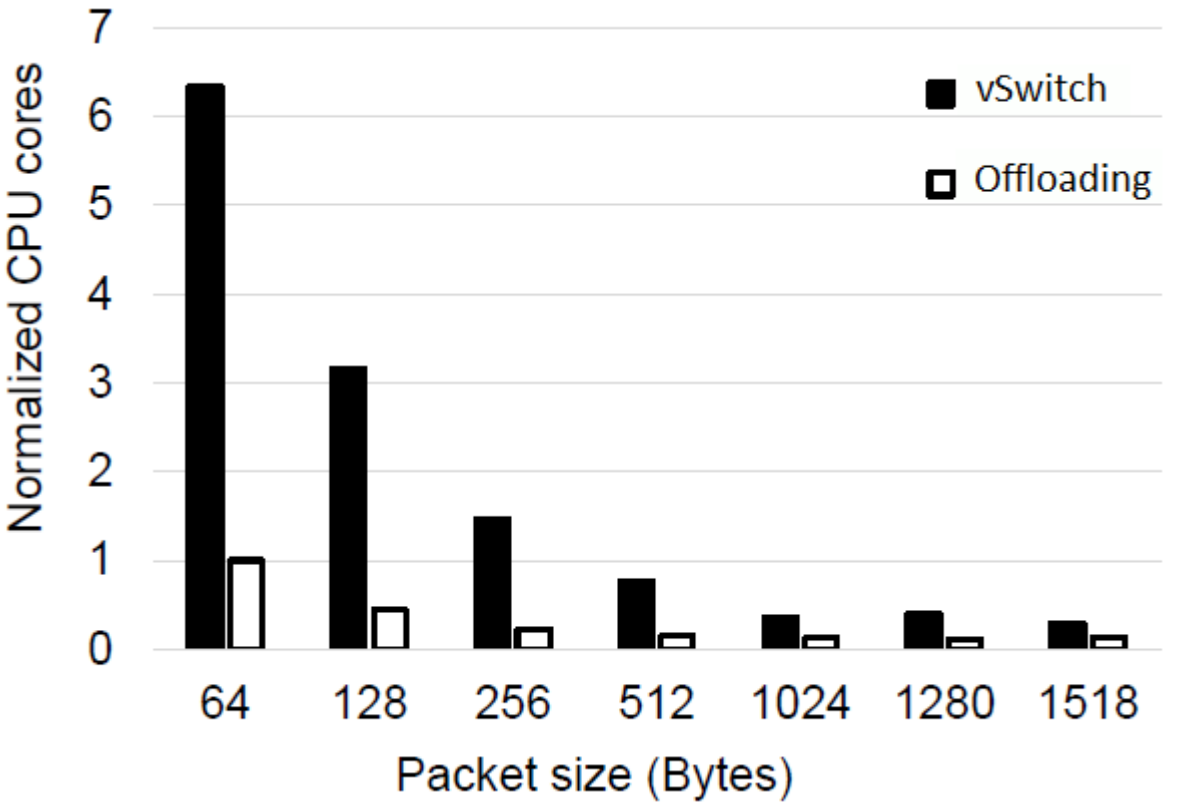


Each solid box is an elementary MB app

Performance optimization

Offload processing to hardware resource whenever possible

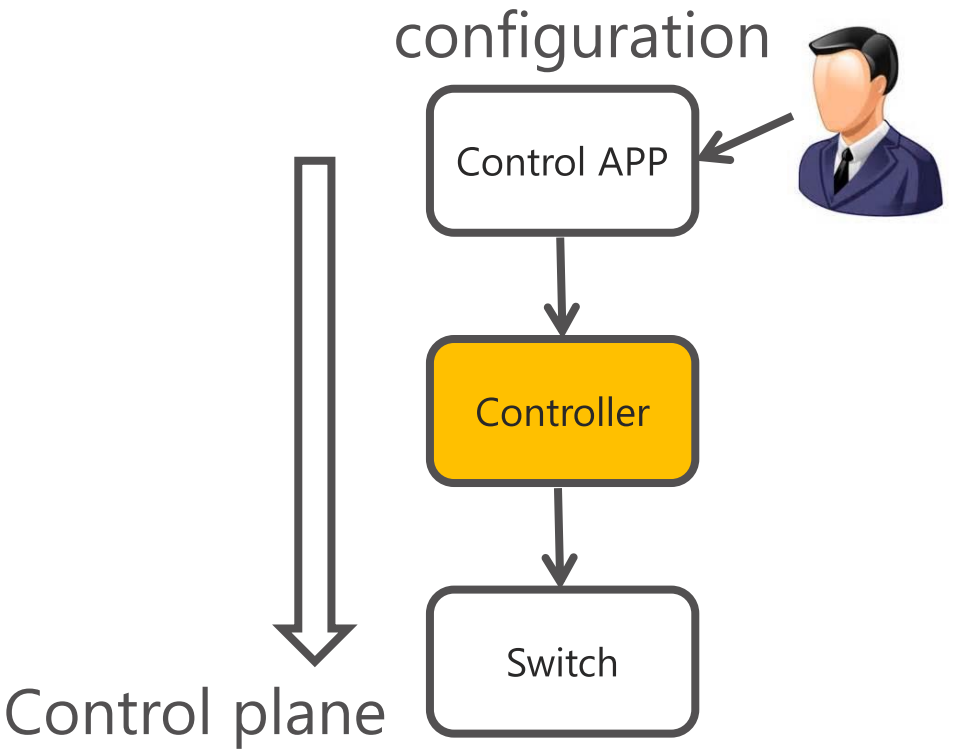
Ex. Offloading policy checking to ToR switch



Beyond SDN

We already see a huge momentum moving towards SDN

But, this not enough…



Tough questions

- What is the root reason that causes our application slow down?
- What can I do NOW to improve it?
- How should different virtual networks share the same network capacity?
- How we can guarantee their performance?

Why these are difficult?

Data are difficult to obtain

Scattered in each server, switch or network link

Some are hard to acquire

Data may be erroneous

There are many, many data, multi-TGb per day

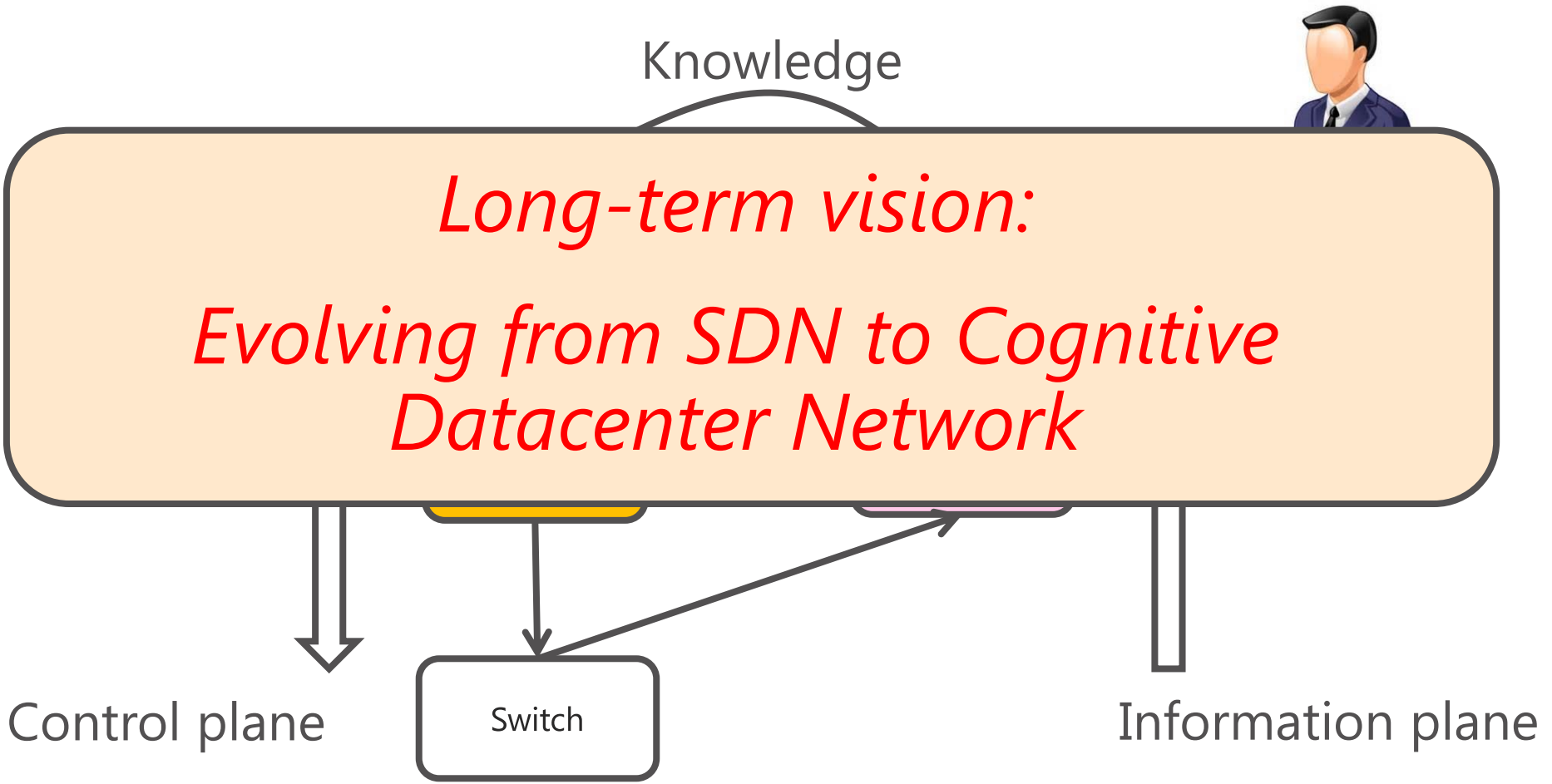
Real-time detection, reasoning and reaction

Catching dynamic events: attacks, performance drops, violation of SLA

Some need real fast response

Some may tolerant more delay but should be as fast as possible: a few second application performance degradation may raise a ticket

Network information plane



Thanks!