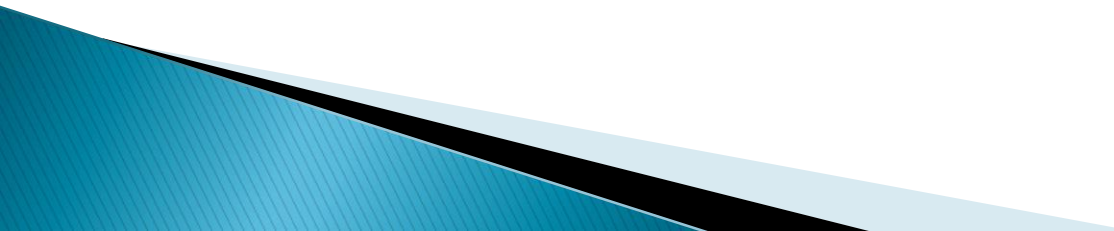



# **EWBridge: East-West Bridge for Heterogeneous SDN NOSes Peering**

# Why SDN?

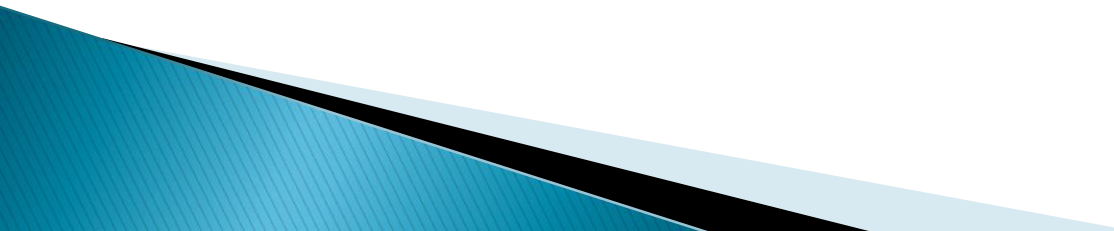
- ▶ Software defined networking (SDN) decouples the vertical and tightly coupled network architecture, and opens up the control plane and the associated protocol.
  - ▶ In this way, SDN promotes the rapid innovation and the evolution of the network.
  - ▶ SDN is considered as a promising way to re-architect the networks.
- 

# Large network partition

- ▶ Large networks are always partitioned into several small networks when deploying SDN:
    - Scalability.
    - Privacy.
    - Incremental deployment.
    - Network faults isolation.
  - ▶ A dedicated network operating system (NOS) or controller is deployed for each network.
  - ▶ Each NOS has the local network view. However, to route data packets in an entire network, a global network is required.
- 


# SDN Peers

- ▶ We divide SDN domains(sub-networks) into 4 categories, referred to as SDN Peers:
  - SDN AS Peers
  - SDN intra-domain sub-network Peers belonging to the same administrative domain
  - DCs of an individual company located at different places
  - Enterprise networks located at different areas connected by the WAN.

- ▶ In order to support communication among different controllers/NOSs, East West(EW) bridge or API is used.
  - ▶ The functions of east/westbound API include import/export data between controllers, algorithms for data consistency models, and monitoring/notification capabilities (e.g., check if a controller is up or notify a take over on a set of forwarding devices).
  - ▶ EW bridge guarantees data integrity between controllers.
  - ▶ To synchronize data between controllers it uses publish/subscribe model.
  - ▶ It has to ensure communication between heterogeneous NOSs.
- 

# Design Goals

- ▶ Define what network information and how such information be exchanged.
- ▶ Design a performance mechanism (EW Bridge) for network view exchange from multi-domain networks.
- ▶ Enable multiple heterogeneous NOSs to work together.
  - EW Bridge should be compatible with different third-party controllers' network view information storage systems.

- ▶ There two types of NOS:
  - ▶ (1) **Single NOS**
    - Floodlight , NOX , Maestro, Beacon, SNAC, and Trema.
  - ▶ (2) **Distributed NOS**
    - Onix, HyperFlow, and DIFANE.
  - ▶ Single NOS improves performance by technologies like multi-thread running on a multi-core server.
  - ▶ However, for large-scale data centers or networks, the capability of a single controller is limited:
    - NOX could process about 30K requests per second;
    - Maestro could process about 600K requests per second.
- 

- ▶ Large scale networks have vast amounts of data flow:
  - i) 1500 server cluster may generate 100K requests/sec
  - ii) 100 switch data center may generate 10000K requests/sec
- ▶ To achieve a scalable control plane, distributed NOSs are proposed.

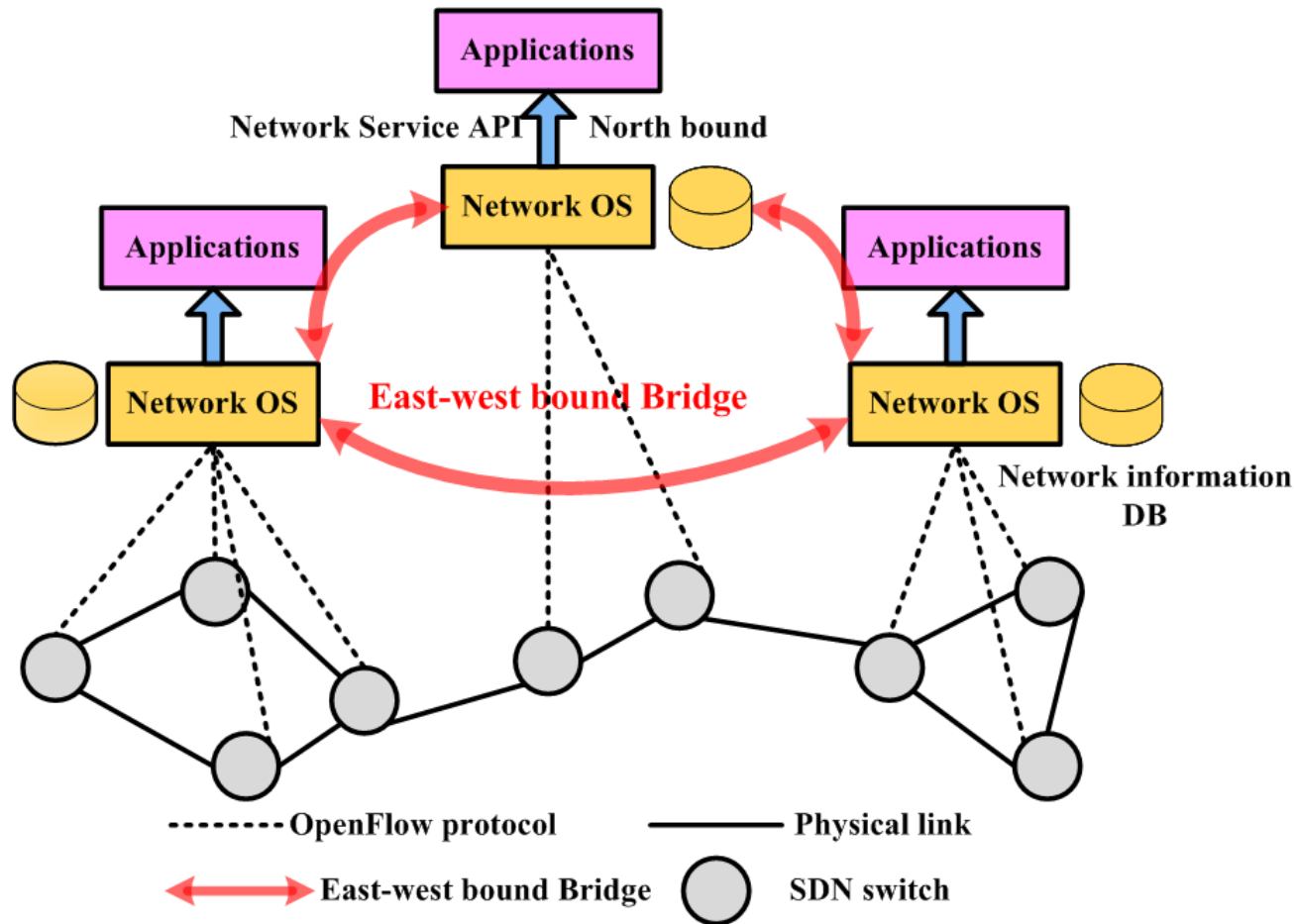
Name	Technology for network View/event sync	Evaluation
HyperFlow	WheelFS (distributed file system )	can only deal with non-frequent events
Onix	DHT (Distributed Hash Table)	A powerful controller; Did not prove the performance of DHT in large scale network

None of them can coexist with others, because the east-west communication interface is private.



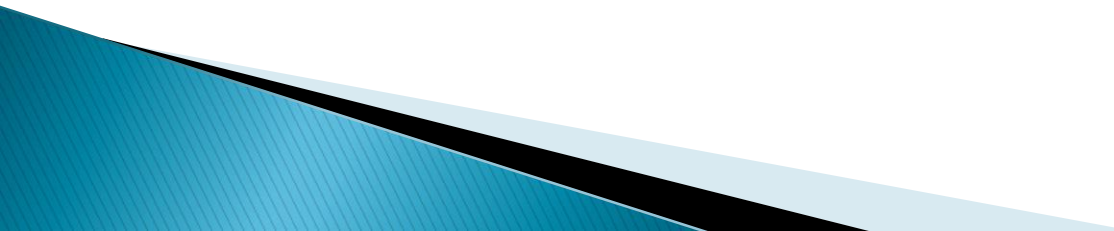
- ▶ Yin et al. had proposed a message exchange protocol SDNi for SDN across multiple sub-network domains.
  - only defined several basic messages such as the reachability information and the flow setup/ tear-down/ update processes.
- ▶ What network information and how such information be exchanged has not been well addressed so far.

# East-West Bridge for SDN Peering



East-west Bridge (EWBridge) for heterogeneous NOSes cooperation in SDN

# **EW Bridge Functionality**

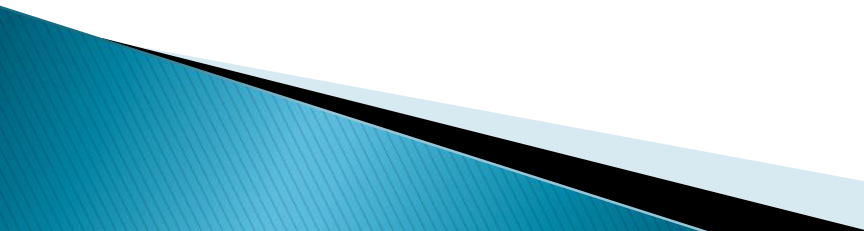
- ▶ Controller/ NOS discovery process
  - ▶ Definition of network view information in different scenarios
  - ▶ Network view information storage and transfer format
  - ▶ High performance network view exchange mechanism
- 

# Controller Discovery

- ▶ Before exchanging the network view, the first task for controllers is to discover each other.
- ▶ The controller discovery algorithms can be:
  - distributed controller discovery algorithm
  - centralized management system such as a registration center
- ▶ A reference for controller list information:
  - ▶ *<Controller\_ID, name, version, IP\_address, is\_TCP, is\_RestAPI, TCP\_port, Rest\_port>*.

# Network view abstraction

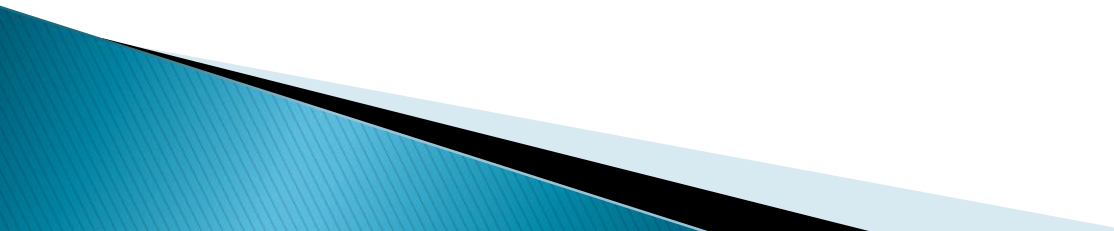
## ► Static state information

- **Reachability**: in carrier network, reachability refers to the IP address prefixes; in DC and enterprise network, it also includes the server/host addresses.
  - **Topology**: node (e.g., switch, server, host, controller, even firewall, balancer, others), link, link bandwidth, port throughput, link connection.
  - **Network service capabilities**: such as SLA, GRE.
  - **QoS parameters**: such as latency, reliability, packet loss rate, availability, throughput, time delay variation, and cost.
- 

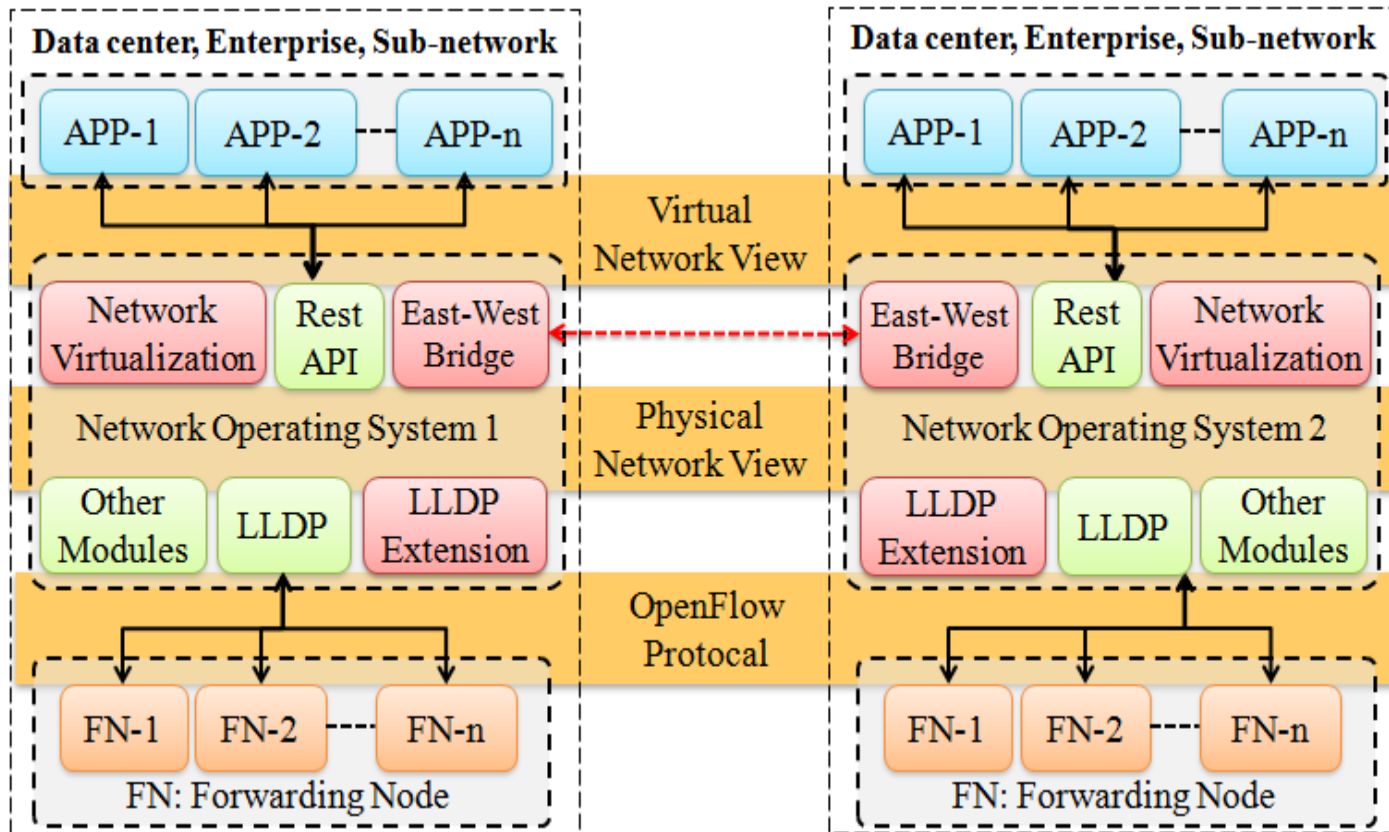
# Network view abstraction

- ▶ Dynamic state information:
  - Mainly includes two aspects:
  - **network state**: such as FlowTable entries information in each switch;
  - **real-time bandwidth utilization** in the topology, and the all the flow paths in the network.

# Network view storage

- ▶ We can **formalize** the network view by directed graph with entity (node, virtual node, link, virtual link).
  - ▶ Considering that the network storage should have a higher scalability, availability and data IO speed, EWBridge suggests the ‘**key-value**’ database plus caching systems.
  - ▶ Databases with **transactional** function should be adopted to guarantee data integrity.
- 

# Network view learning



- ▶ Enable EWBridge in all kinds of NOSes by adding three modules:
  - Network Virtualization, East-West Bridge, and LLDP Extension



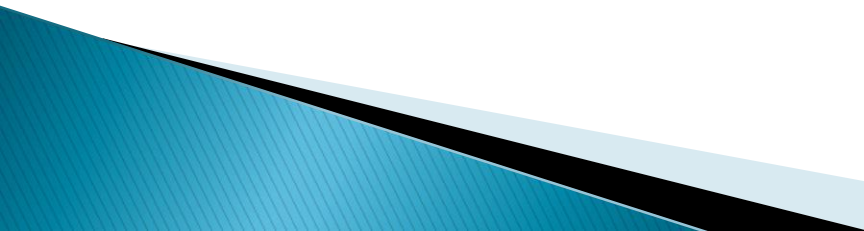
# Network view learning

- ▶ The basic information such as node, node\_capability, port, and link information usually can be learned by the LLDP .
- ▶ To learn more network view information such as
  - OpenFlow version, number of the FlowTables on each node, link utilities, and flow entries.
- ▶ We can extend the NOS by adding a network view driver module named ‘**LLDP extension**’.
  - By counting the total number of packets related to a port in all the FlowTables, the driver can learn the link utilities.
  - By certain commands (OpenFlow switches provide those commands) to switches, the OpenFlow version, number of FlowTables, and flow entries can be learned.

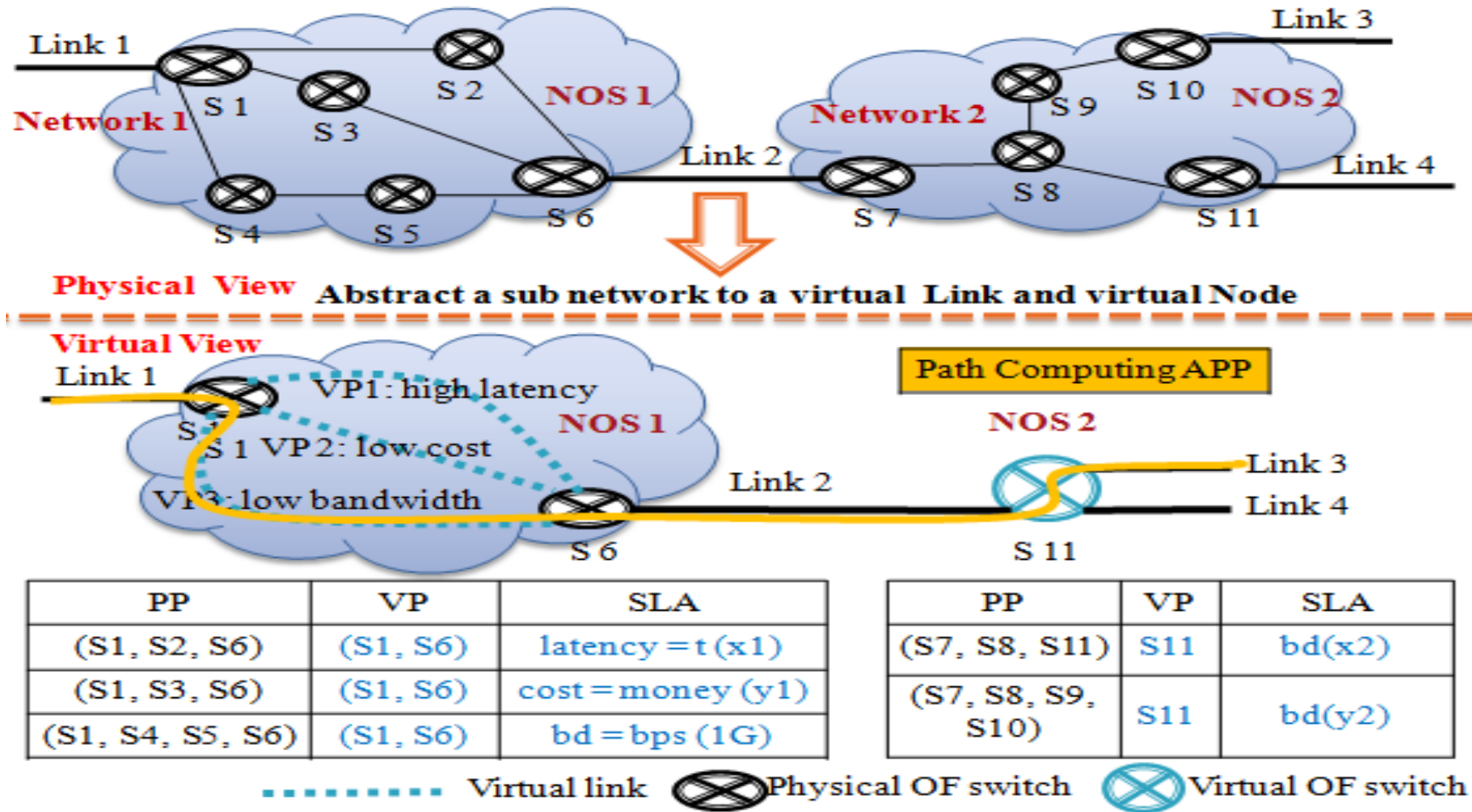
# Network view transfer format

- ▶ **JSON** is suggested as a basic implementation, and the XML, YANG, YAML as alternatives.
- ▶ Those languages have the ability to enable EW Bridge with the following advantages:
  - (1) vendor and application-independent, thus the network view transfer format is independent with the storage systems;
  - (2) allow explicit definition of the inherent structure according to the requirements; such features make the network view message format flexible and easy to extend;
  - (3) They are files and not a data packet format. The elements are easy to extend.

# Virtual network view for privacy and scalability

- ▶ Some domains may be willing to expose only a part of the network view due to their privacy concerns.
  - ▶ EWBridge supports **abstracting** a physical network to a virtual network for such domains.
  - ▶ For the virtual network view, there is a **mapping table** between the physical network and the abstracted virtual network views.
- 

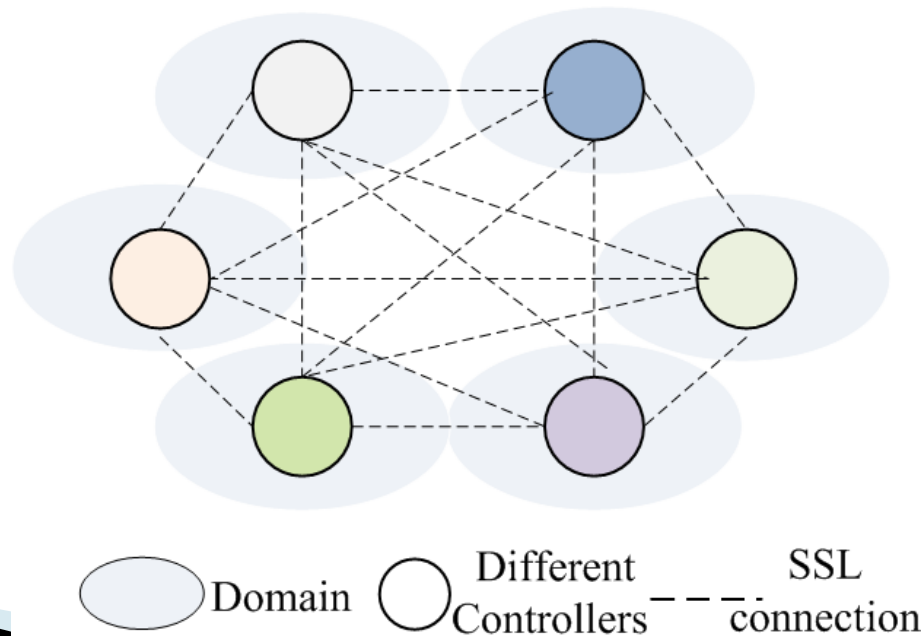
# Virtual network view for privacy and scalability



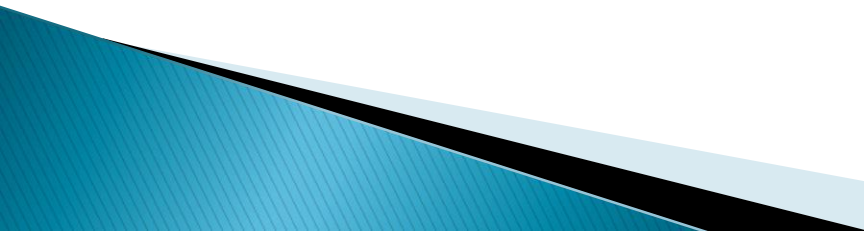
Physical view to virtual view (PP: Physical Path; VP: Virtual Path; OF: OpenFlow; S: Switch; bd: bandwidth; t: time; bps: bits per second)

# High Speed Bridge Exchange Mechanism

- ▶ After the controller discovery process, each controller learns all the addresses of their peers.
- ▶ Then all the controllers can establish a virtual **full mesh** topology based on TCP/ SSL.



# High Speed Bridge Exchange Mechanism

- ▶ All the SDN peers are equal to each other.
  - ▶ For the network event such as link failure, adding/ deleting switch, adding/ deleting IP prefixes, each controller can subscribe to other controllers' events.
  - ▶ **Publish/ subscribe system** to deliver update messages:
    - Once an event is triggered, the corresponding controller will push the event to all the subscribers simultaneously.
    - Each controller can get the update message directly from the controller it cares.
- 

# High Speed Bridge Exchange Mechanism

- ▶ We can design a FSM with 5 messages: OPEN, UPDATE, NOTIFICATION, KEEPALIVE, VIEW-REFRESH.
- ▶ Compared with BGP FSM, EWBridge mainly **changes the UPDATE message into JSON file format**, and simplifies the finite state machine.
- ▶ To achieve high speed data exchange:
  - In the normal condition, all the NOSes **keep in connections**, by sending KEEPALIVE messages.
  - Once network view updates, EWBridge can send UPDATE file out to its peers in parallel without re-setup TCP connections.
  - **Each UPDATE file carry multiple UPDATE messages.**

# Scalability Evaluation

Symbol	meaning
$x$	Requests a controller can deal in one second
$y$	Requests a computer generates in one second
$z$	Number of computers one person has
$N$	Number of people in an AS/DC/enterprise

- ▶ Then there will be about  $Nzy/x$  domains.
- ▶ The total number of connections is:  $Nzy/x \times (Nzy/x - 1)$
- ▶ Normally, one client uses one computer at a time and we assume each computer generate 1 request per second.

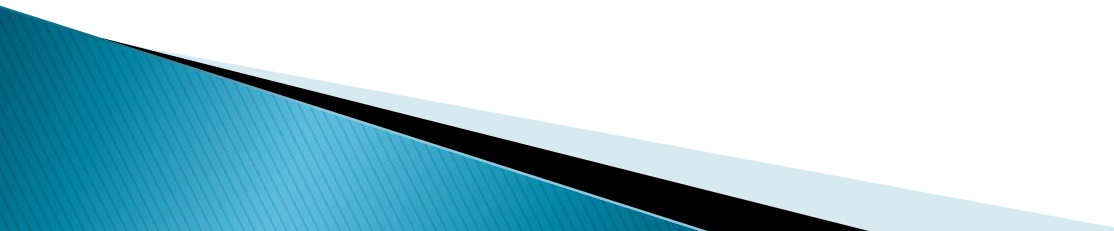


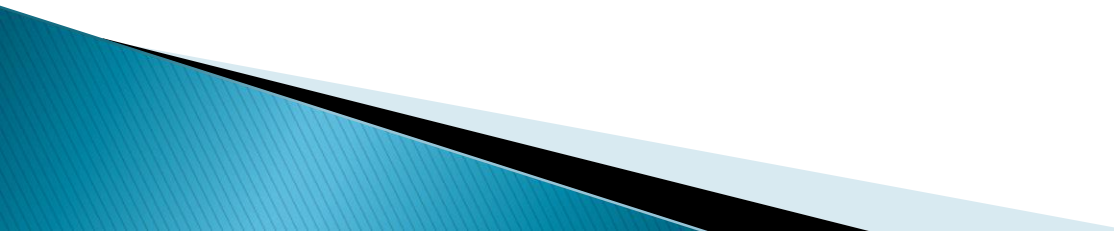
# Scalability Evaluation

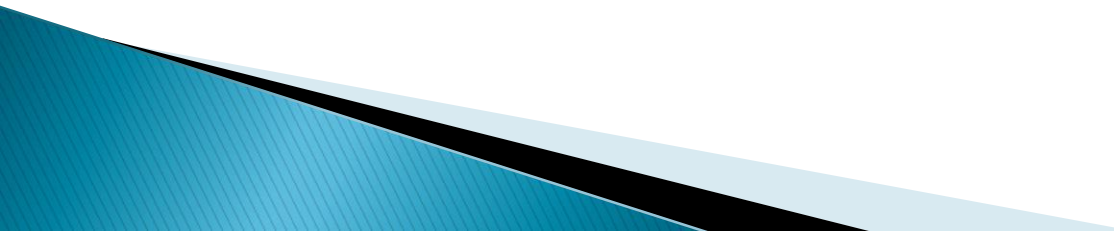
- ▶ According to the well-known NOX as controller, a NOX could process about 30K request per second.
- ▶ In DC/AS, the online people are usually less than 3,000,000 (from Caida)
- ▶ Number of domain and connection degree:

	<b>People/Customer (Online)</b>	<b>Domain Number</b>	<b>Connection Degree</b>
<b>Campus</b>	$\leq 100,000$	4	3
<b>Enterprise</b>	$\leq 500,000$	17	16
<b>DC or AS</b>	3,000,000	100	99

# ONIX

- ▶ Onix, a distributed NOS is a platform on top of which a network control plane can be implemented as a distributed system.
  - ▶ Control planes written within Onix operate on a global view of the network, and use basic state distribution primitives provided by the platform.
  - ▶ Thus Onix provides a general API for control plane implementations, while allowing them to make their own trade-offs among consistency, durability, and scalability.
- 

- ▶ Onix consists of roughly 150,000 lines of C++ and integrates a number of third party libraries.
  - ▶ At its simplest, Onix is a harness which contains logic for communicating with the network elements, aggregating that information into the NIB, and providing a framework in which application programmers can write a management application.
  - ▶ A single Onix instance can run across multiple processes, each implemented using a different programming language, if necessary.
- 

- ▶ In this model, supporting a new programming language becomes a matter of writing a few thousand lines of integration code, typically in the new language itself.
  - ▶ Onix currently supports C++, Python, and Java.
  - ▶ Processes are interconnected using the same RPC system that Onix instances can use among themselves.
  - ▶ Independent of the programming language, all software modules in Onix are written as loosely-coupled components, which can be replaced with others without recompiling Onix as long as the component's binary interface remains the same.
- 

- ▶ Components can be loaded and unloaded dynamically and designers can express dependencies between components to ensure they are loaded and unloaded in the proper order.

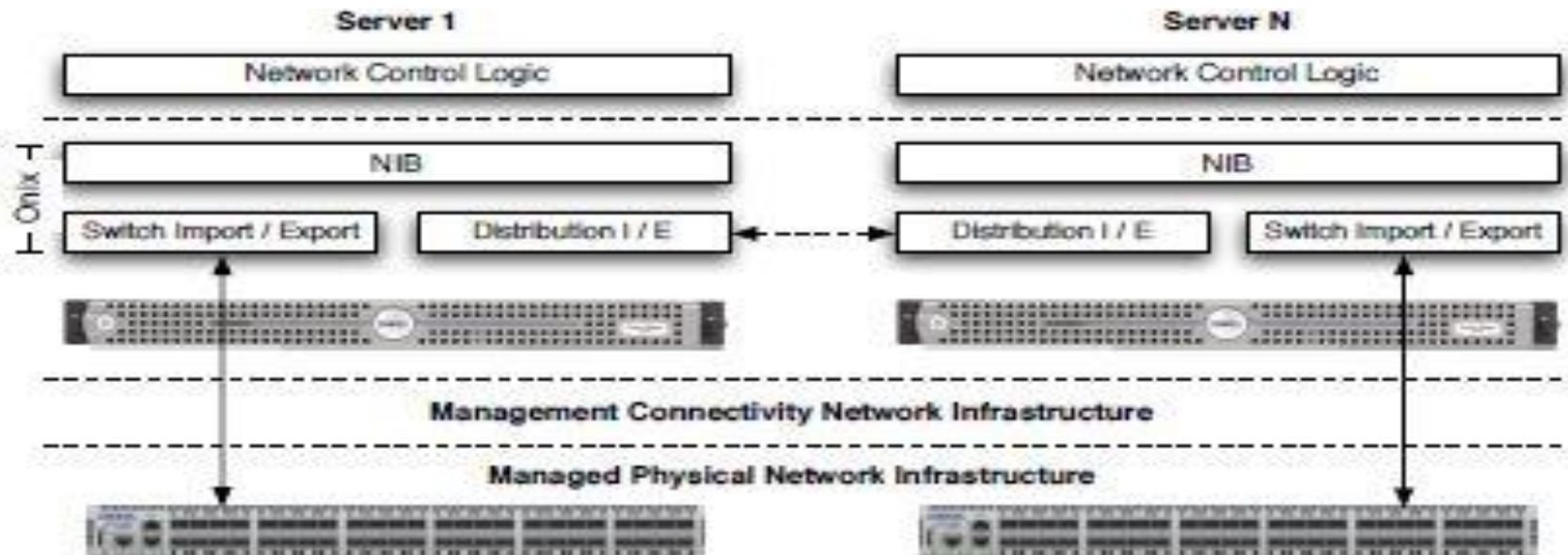
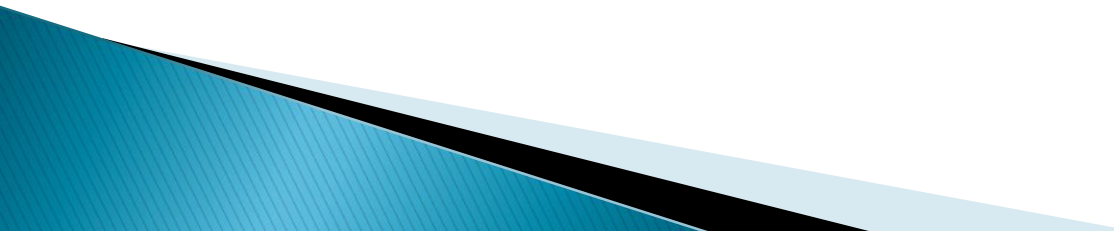



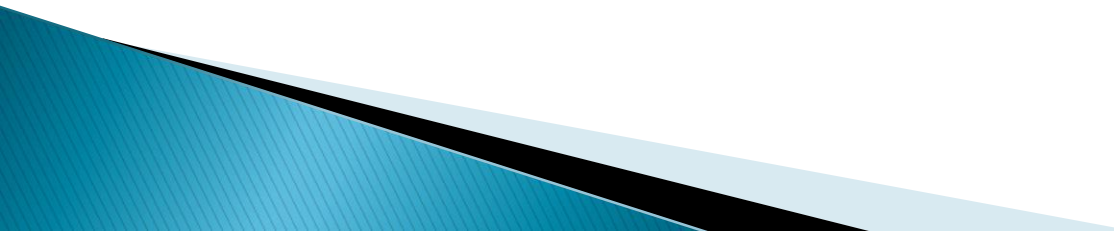
Figure 1: There are four components in an Onix controlled network: managed physical infrastructure, connectivity infrastructure, Onix, and the control logic implemented by the management application. This figure depicts two Onix instances coordinating and sharing (via the dashed arrow) their views of the underlying network state, and offering the control logic a read/write interface to that state.

# ONIX Components

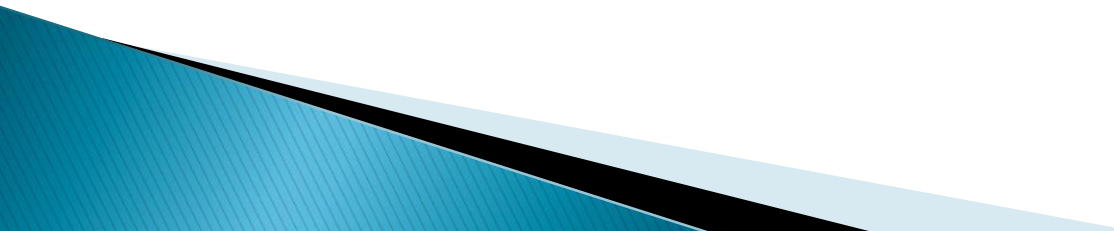
- ▶ **Physical infrastructure:** This includes network switches and routers, as well as any other network elements (such as load balancers) that support an interface allowing Onix to read and write the state controlling the element's behavior (such as forwarding table entries).
  - ▶ These network elements need not run any software other than that required to support this interface and achieve basic connectivity.
- 

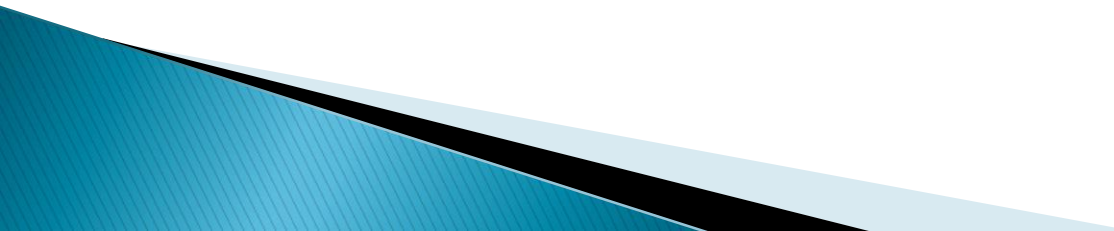


- ▶ **Connectivity infrastructure:** The communication between the physical networking gear and Onix (the “control traffic”) transits the connectivity infrastructure.
  - ▶ This control channel may be implemented either in-band (in which the control traffic shares the same forwarding elements as the data traffic on the network), or out-of-band (in which a separate physical network is used to handle the control traffic).
  - ▶ The connectivity infrastructure must support bidirectional communication between the Onix instances and the switches, and optionally supports convergence on link failure.
  - ▶ Standard routing protocols (such as IS-IS or OSPF) are suitable for building and maintaining forwarding state in the connectivity infrastructure.
- 

- ▶ **Onix:** Onix is a distributed system which runs on a cluster of one or more physical servers, each of which may run multiple Onix instances.
  - ▶ As the control platform, Onix is responsible for giving the control logic programmatic access to the network (both reading and writing network state).
  - ▶ In order to scale to very large networks (millions of ports) and to provide the requisite resilience for production deployments, an Onix instance is also responsible for disseminating network state to other instances within the cluster.
- 



- ▶ **Control logic:** The network control logic is implemented on top of Onix's API.
  - ▶ This control logic determines the desired network behavior; Onix merely provides the primitives needed to access the appropriate network state.
  - ▶ The copy of the network state tracked by Onix is stored in a data structure we call the Network Information Base (NIB), which we view as roughly analogous to the Routing Information Base (RIB) used by IP routers.
- 

- ▶ NIB gives a graph of all network entities within a network topology.
  - ▶ Onix provides scalability and resilience by replicating and distributing the NIB between multiple running instances.
- 

Category	Purpose
Query	Find entities.
Create, destroy	Create and remove entities.
Access attributes	Inspect and modify entities.
Notifications	Receive updates about changes.
Synchronize	Wait for updates being exported to network elements and controllers.
Configuration	Configure how state is imported to and exported from the NIB.
Pull	Ask for entities to be imported on-demand.

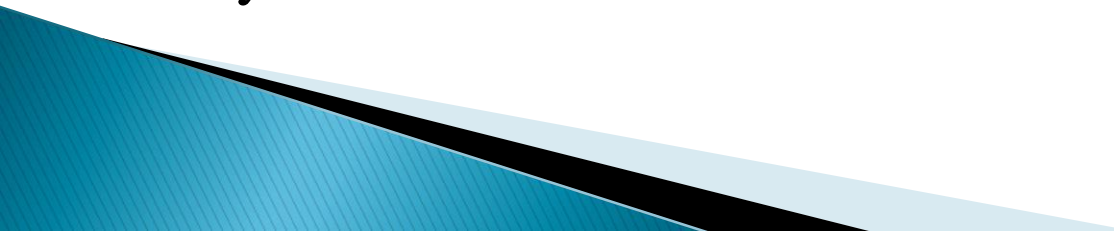
Table 1: Functions provided by the Onix NIB API.

## **Onix supports three strategies that can be used to improve scaling.**

First, it allows control applications to partition the workload so that adding instances reduces work without merely replicating it.

Second, Onix allows for aggregation in which the network managed by a cluster of Onix nodes appears as a single node in a separate cluster's NIB. This allows for federated and hierarchical structuring of Onix clusters, thus reducing the total amount of information required within a single Onix cluster.

Finally, Onix provides applications with control over the consistency and durability of the network state.




**Partitioning:** The network control logic may configure Onix so that a particular controller instance keeps only a subset of the NIB in memory and up-to-date.

Further, one Onix instance may have connections to a subset of the network elements, and subsequently, can have fewer events originating from the elements to process.

**Aggregation:** In a multi-Onix setup, one instance of Onix can expose a subset of the elements in its NIB as an aggregate element to another Onix instance. This is typically used to expose less fidelity to upper tiers in a hierarchy of Onix controllers.

For example, in a large campus network, each building might be managed by an Onix controller (or controller cluster) which exposes all of the network elements in that building as a single aggregate node to a global Onix instance which performs campus-wide traffic engineering. This is similar in spirit to global control management paradigms in ATM networks.

- ▶ **Consistency and durability:** The control logic dictates the consistency requirements for the network state it manages.
  - ▶ This is done by implementing any of the required distributed locking and consistency algorithms for state requiring strong consistency, and providing conflict detection and resolution for state not guaranteed to be consistent by use of these algorithms.
  - ▶ By default, Onix provides two data stores that an application can use for state with differing preferences for durability and consistency. For state applications that favor durability and stronger consistency, Onix offers a replicated transactional database and, for volatile state that is more tolerant of inconsistencies, a memory-based one-hop DHT.
- 

**Thank You**