A Common Data Plane for Multiple Overlay Networks[☆]

Ryo Nakamura[a,*], Kouji Okada[b], Yuji Sekiya[a], Hiroshi Esaki[a]

[a] *University of Tokyo, Tokyo, 113-8656, Japan.*
[b] *Lepidum Inc, Tokyo, 1-30-3 6F, Japan.*

Abstract

Various overlay networks have been proposed and developed to increase flexibility on networks to address issues of the IP network. However, the existing overlay networks have two problems: 1) performance degradation due to full-mesh tunneling, 2) increasing of development cost due to tightly coupled with control and data plane. To solve the problems, we introduced a new abstraction layer for overlay networks in the existing network layering model. Based on the architecture, we designed and implemented a protocol stack, called `ovstack`, as a common data plane for overlay networks. In this paper, we describe control plane systems to construct overlay networks with dynamic routing protocol on ovstack. We evaluate the performance of overlays including ovstack, and topologies that are built by the control plane systems. The results of performance evaluation shows that delay of packet forwarding is decreased 12 % and jitter is 8 % lower than an existing overlay. Then, our evaluations confirm that the ovstack can contribute to construction of overlay networks specified each requirement on the current networks.

Keywords: Overlay Network, Overlay Routing, Data Plane

1. Introduction

Various applications communicate via an IP network. Every application has its own requirements on networks, such as bandwidth, latency, stability, availability, and operability. For example, e-commerce services require low latency networks [1], and voice communication systems require networks to be low jitter [2]. Furthermore, in Infrastructure as a Service (IaaS) model cloud environments, multi-tenancy for separating networks for each user and prefix mobility are required. On the other hand, as problems of IP, decoupling location and identification, multi-homing, mobility, simplified renumbering, modularity, and

[☆] This work is an extended version of "ovstack : A Protocol Stack of Common Data Plane for Overlay Networks", in Proceedings of IEEE/IFIP NOMS SDNMO Workshop, May. 2014.

[*] Corresponding author
Email address: `upa@wide.ad.jp` (Ryo Nakamura)

routing quality are remarked in RFC6227 [3]. Due to these problems, existing networks constructed by IP and ethernet lack flexibility to satisfy various requirements.

To increase the flexibility of networks, overlay networks have been proposed and developed [4, 5, 6, 7, 8]. Some functions achieved by overlays are also utilized for network virtualization and Software Defined Network (SDN). Overlay technologies construct their own networks over the IP network by encapsulating packets with their protocol headers in IP datagram. This encapsulation enables to add network functionalities to the overlay network corresponding to the requirements of an application. For example, Virtual eXtensible LAN (VXLAN) [4], an ethernet over IP overlay technique, achieves layer 2 multitenancy and is utilized as one of transports on SDN environments. On the other hands, Locator/ID Separation Protocol (LISP) [5] that is IP over IP overlay achieves prefix mobility and multi-homing, and some other overlays achieve multicast communication [7, 8].

However, the architecture of existing overlay technologies has two problems. The first problem is that the topology of existing overlays is essentially a full-mesh tunneling topology. Hence, a path between two overlay nodes follows the path of IP layer. In consequence, it is not possible to build routing overlays without being tied to the routing table of IP layer, and then it causes performance degradation. Second problem is tightly coupled with control plane and data plane. Since existing overlay technologies are specialized in each upper-layer application, the system architectures of control and data plane are interdependent. Thus, overlay networks that are built by each overlay technology are completely independent, which causes complication of network operations and increases of development costs: when developing a new overlay technology, it is necessary to design and develop both of control and data plane.

In our previous work [9], we introduced a new abstraction layer for overlay networks in the network layering model. And we designed and implemented a protocol stack of common data plane for overlay networks. The proposed protocol stack, called ovstack, offers common functions of data plane of overlays including routing table and forwarding packets in overlay layer. In this paper, we extend the work with control plane design, implementation and evaluation. By investigating control plane of ovstack, we show that ovstack helps construction of overlay networks for requirements on the current networks.

Our contributions of this paper include:

- Introducing a new abstraction layer for various overlay networks in the network layering model.
- The design and implementation of ovstack, a framework that provides a protocol stack of data plane for the abstraction layer in Linux kernel.
- The design and implementation of two control planes as routing protocols for ovstack overlays, and its evaluation.

2. Existing Overlay Models

In this section, we summarize characteristics of existing overlays, and clarify problems of the existing overlay.

2.1. Locator/ID Separation Protocol

Locator/ID Separation Protocol (LISP) is an IP over IP overlay network architecture and protocol. LISP isolates two aspects of the IP address, identifier and locator. By introducing this isolation, LISP enables prefix mobility and multi-homing. In LISP, an IP prefix is called Edge ID (EID), and a LISP router that accommodates EIDs is called Ingress or Egress Tunnel Router (xTR). xTRs of a LISP network manages map table that is constructed from entries of EID and the Routing Locator (RLOC) that is the IP address of xTR. xTR forwards IP datagram with IP encapsulation to other xTR in accordance with the map table.

Providing multiple xTRs for an EID enables multi-homing of the EID prefix. In IP layer, it is necessary to have inter-AS connectivity through eBGP to realize multi-homing for a prefix. In contrast, LISP can achieve multi-homing for EIDs easily by multiple xTRs. Moreover, the relation between EID and RLOC does not have particular semantics like IP netmask. Thus arbitrary EIDs can be accommodated by arbitrary xTRs. It enables that prefix mobility in a LISP overlay network.

The LISP map protocol is defined as a control plane of the LISP overlay network. The control plane of LISP is constructed from LISP Alternative Logical Topology (LISP- ALT) based on BGP [9]. xTRs constructs routing table of the overlay network from exchanging map table information using LISP-ALT. Thus, xTRs are able to route and forward packets in accordance with overlay routing table.

2.2. Virtual eXtensible LAN

Virtual eXtensible LAN (VXLAN) provides ethernet emulation over IP network. In VXLAN, new 24 bit identifier is added into VXLAN header when the ethernet frame is encapsulated. By adding this identifier that is named VXLAN Network Identifier (VNI), VXLAN is able to isolate enough number of network segments. The control plane of VXLAN uses unicast frame that is encapsulated with IP unicast, and unknown unicast, broadcast and multicast frames are encapsulated with IP multicast. Because it uses IP multicast, VXLAN nodes (VTEP) can join the VXLAN overlay network without a particular control plane protocol. Thus, the header format of VXLAN is simple to include only some flags and VNI.

The VTEP manages Forwarding Data Base (FDB) that is constructed from pairs of MAC address and IP address of VTEP. This procedure is described below: when a VTEP receives an encapsulated packet from the overlay, it registers a pair of source MAC address of inner ethernet header and source IP address of outer IP header. Thereby, VXLAN constructs FDB for the overlay network with flooding to IP multicast address and source MAC and IP address snooping.

2.3. Resilient Overlay Network

Resilient Overlay Network (RON) [6] is an overlay network that aims to improve network performance such as throughput and delay by utilizing the multihop overlay routing. RON nodes have unique identifiers for each node in overlay network, and construct routing table using the node id to realize the multihop overlay routing. In LISP and VXLAN, the path of encapsulated packets is end-to-end between overlay nodes in accordance with IP routing table. By contrast, in RON, relaying other nodes in the overlay network enables selection of better path between nodes without being tied to the routing table of IP layer.

Existing overlays such as LISP and VXLAN transmit packets through end-to-end path in IP layer. On the other hand, routing overlays such as RON and Scribe [7] that enables multicast routing in overlay network, constructs routing table using an identifier on the overlay network apart from IP routing table. By selecting better quality paths between overlay nodes, routing overlays achieves performance improvement of network. To achieve the routing overlays, the RON node adds destination and source node identifiers to encapsulation header.

2.4. P2P Network

Peer-to-Peer Network (P2P) is also one of overlay networks. In P2P networks, client nodes or proxy servers construct multihop overlay networks with a variety of topologies on the Internet. By transporting application data over overlay networks through relay nodes, P2P networks achieve many applications such as file sharing and video or voice streaming, etc. Furthermore, in research community, many approaches to optimize overlay topologies are proposed to address performance issues of P2P networks.

On the other hand, identifier schemes and data transport architectures of various P2P techniques are optimized for target applications. For example, bittorrent [10] is the one of most popular file sharing P2P applications. Its messaging protocol and node identifier are specialized to find out objective files and share them efficiently. SplitStream [11] is proposed to obtain multicasting high-bandwidth content in a P2P network. It constructs a tree-based overlay topology optimized for grouping nodes with the flat node identifier space using Scribe algorithm [7]. As just described, components of P2P networks are specialized for their target applications.

2.5. Problem Definition

To satisfy various requirements, there are many overlay technologies: each overlay achieves each requirements such as multi-homing, mobility, multi-tenancy by encapsulation, performance improvement, and multicast communication by routing overlays. However, there are two problems by individual technologies are dedicated to specific requirements.

2.5.1. Full-mesh Tunneling Topology

First problem is, the topology of most existing overlay technologies for packet transport is essentially full-mesh tunneling topology that causes degradation of network performance. Overlay routing with relay nodes like RON and P2P networks achieves performance improvement of throughput and multicast communication. However, in existing overlays like IP tunneling, LISP and VXLAN, a path between two overlay nodes is directly connected. If routing overlays are realized by tunneling overlay technologies, the overhead of re-encapsulation occurs in relay nodes: relay nodes decapsulate packets from a overlay link at first, route on basis of inner packet data, and encapsulate packets again to transmit.

In VXLAN, a VTEP transmits ethernet frames with IP encapsulation to an IP address that is contained in FDB. Thus, paths between VTEPs are same as paths of IP layer: the overlay topology between more than two VTEPs is same as full-mesh tunneling topology. LISP also constructs the same overlay topology. The xTR finds a destination locator address from map table using a destination IP address of received IP packets from edge networks, and transmits encapsulated packets to the destination xTR via an IP network. Because existing overlay technologies do not have any unique node identifier on overlay networks, they cannot build the routing overlays without being tied to the IP routing table.

The cause of the problem is that it is not intended that these technologies have the purpose of routing in the overlay network. For that reason, headers of LISP and VXLAN do not contain node identifier on each overlay network. Instead of overlay routing, they achieves various functions such as address mobility and multi-tenancy without complex header formats and control planes. On the other hand, to respond to requirements about the network performance, node identifier and multihop overlay routing like RON are needed.

2.5.2. Tightly Coupled Control Plane and Data Plane

Second problem is, existing overlays has tightly coupled control plane and data plane. LISP utilizes map protocols as control plane, and data plane (e.g., header formats and routing table) has particular formats specified to LISP map protocol, therefore, the control plane and data plane of LISP cannot be utilized from other overlay network technologies, vice versa. Other existing overlay technologies are similar to that. The header format of RON has some specific fields such as policy tag and flow identifier for the RON control plane. Moreover, many P2P systems have been proposed many techniques to optimize overlay networks for performances improvement. However, control planes (i.e. management protocols) and data plane (i.e. addressing and data transport) are completely united.

As described above, existing overlay networks have mutual dependence of control plane and data plane to achieve each function. Consequently, all of overlay networks built with each technology are completely isolated, and both planes do not have transparency. They cause complication of network operations and increase of development cost. When creating a new overlay technology, developers have to design and implement both of control and data plane.

3. Architecture

Existing tunneling protocols and their overlay topologies are designed and proposed for each specific purpose. LISP enables prefix mobility and multihoming by locator/ID separation architecture, VXLAN is designed for segment multiplexing by introducing 24 bit VNI as a new network identifier, and RON is designed for performance improvement by using multihop overlay routing. Thereby, different control or data plane systems can not be used in combination with other planes for various purposes.

In order to solve the problems, we propose an abstraction layer for overlay networks in this paper. As shown in Figure 1, existing overlays are designed specifically for each upper-layer application and requirement, thus it causes isolation of overlay networks and partial lack of features. In proposed architecture, the overlay network is abstracted as a function of network into the network layering model. Developers can construct overlay networks for their requirement on new abstraction layer of overlay.

We proposed a protocol stack called ovstack for common data plane for overlay networks [9]. ovstack forms the data plane of the overlay layer. Developers can create their control plane systems to construct overlay networks with ovstack. Figure 2 depicts the overview of relationship of control plane and data plane. ovstack provides common functions to achieve overlay networks: node identifier, routing tables, packet encapsulation, and overlay routing and forwarding. Encapsulated packets are routed without decapsulation in accordance with overlay routing table and node identifier embedded in encapsulation header. When packets are forwarded on overlay layer, outer IP header is swapped to next-hop overlay node like IP routing swaps outer Ethernet header. Control planes just add and delete entries to routing tables, and set node identifier information. Moreover, multiple routing tables can be created in ovstack, so that multiple control planes for each requirement are able to use different routing tables to separate routing information and identifier space at a data plane.

Introducing new identifier space for overlay layer, multihop overlay routing

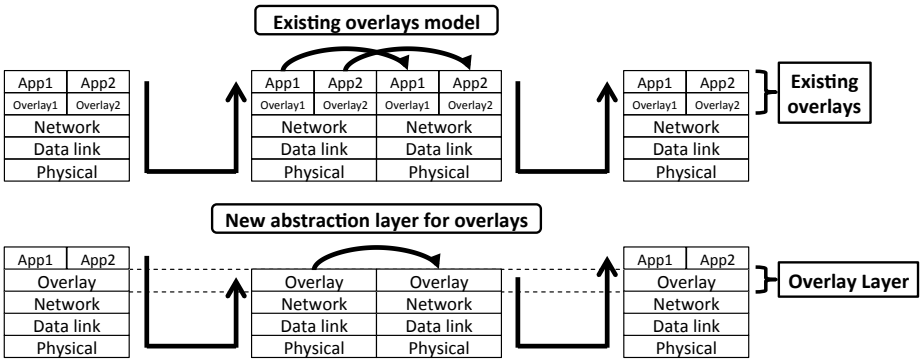


Figure 1: Abstraction layer for Overlay Networks.

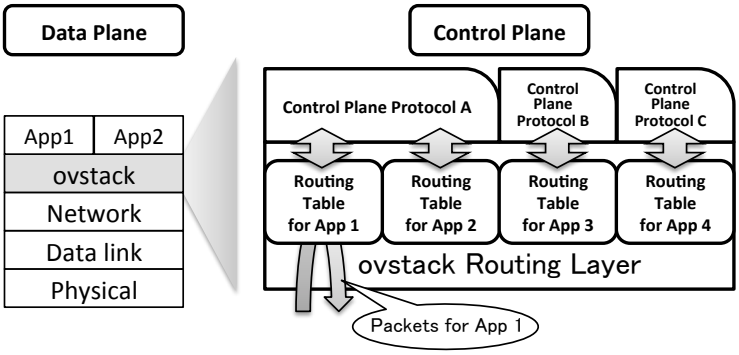


Figure 2: Overview of data plane and control plane of ovstack.

is achieved. Therefore, performance improvement and multicast communication can be realized, as traditional routing overlay (RON) does. On the other hand, control plane, data plane, and upper-layer applications are completely isolated. By providing interfaces between both control and data planes without dependence on upper-layer application systems, transparency of control plane is realized. This isolation helps reduce to development cost, that is one control plane for particular metric such as delay or throughput can be shared for plural upper-layer applications.

The abstraction of overlay networks as the overlay layer eliminates dependencies between control and data planes. It allows us to switch overlay topologies by only exchanging control plane systems. Moreover, by introducing a distinguisher for upper layer applications as a functionality of overlay layer, different topologies for each application requirements such as delay or bandwidth coexist on an overlay network without data plane replacement. Therefore, it helps that reducing development and deployment cost for new overlay topologies for various purposes.

4. Design of Data Plane and Control Plane

In this section, we describe the detail of data plane and control plane of ovstack. The design considers two problems raised in Section 2.5, 1) avoiding full-mesh tunneling topology by introducing node identifier and routing on overlay layer (Section 4.1), and 2) defining a common interface between control and data planes so that both planes are isolated (Section 4.2).

4.1. Data Plane

In ovstack, a node ID is 32 bit identifier in flat space. It is meaning that an ID does not have special semantics as netmask, which is evaluated by longest match policy in IP routing table. Using flat space for node identifier avoids ID and locator binding related problems like IP. Moreover, it has a possibility of introducing Distributed Hash Table (DHT) based algorithms [12, 13] for

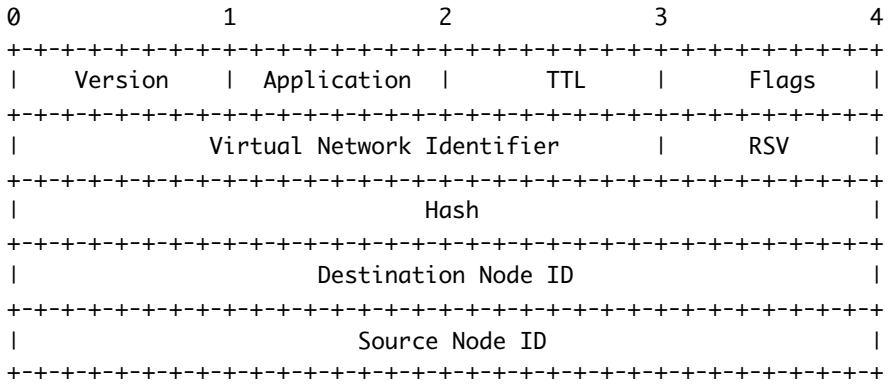


Figure 3: Header format of ovstack.

construction of overlays so that techniques of P2P networks are able to be introduced on ovstack.

ovstack contains routing tables for multihop routing in overlay layer. This routing table is constructed from two entities, Routing Information Base (RIB) and Locator Information Base (LIB). RIB consists of entries of destination ID and next hop node IDs. Overlay routing of ovstack is processed based on this RIB. Arbitrary destination ID can contain multiple next hop node IDs. When a destination has multiple next hop, a packet towards the destination is copied and transmitted to all next hops, that realizes multicast communication in an overlay network. LIB consists of entries of a node ID, IP addresses as locator of the node, and a locator weight. When packets are routed and forwarded on overlay layer, IP is utilized as actual transmission toward next hop. LIB is a map table that is used to look up an actual IP address of next hop. Moreover, LIB allows multiple locator IP addresses for one node, and weight based load balancing like LISP does.

ovstack separates identifier as ovstack node ID and locator as underlay IP address like LISP does. It enables that overlay routing table is isolated from underlay IP routing table. Underlay IP network is treated as just transport between ovstack nodes, then overlay routing tables based on ovstack node ID can construct various overlay topologies which are not bound to underlay IP networks.

4.1.1. ovstack Routing Layer

We describe routing table look up and forwarding decision processes of ovstack. Figure 3 shows the header format of ovstack. ovstack encapsulates packets with IP, UDP, and ovstack header. Functions and roles of each field are described below.

- **Application Field**

Application field represents the type of encapsulated upper-layer application. When packets are routed on overlay layer, ovstack uses this field to distinguish an ovstack overlay network that packets belong to, and decides a routing table to look up.

- **TTL (Time To Live)**

  When ovstack node routes encapsulated packet on overlay layer, outer IP header is swapped toward next ovstack node, and IP TTL of outer IP header is reset. Thus, when overlay routing tables comprise directed cycle graph due to miss-configuration or control plane failure, packets caught in the routing loop do not disappear. To avoid this, we introduce TTL field into ovstack header.

  TTL field represents the remained hop count that packets are forwarded. ovstack nodes decrease a hop count when forwarding a packet. And if the field is 0, the packet is dropped. Since the hop count between hosts on overlay network is likely less than the Internet, the default hop count for ovstack should be less than 64.

- **Virtual Network Identifier**

  Virtual Network Identifier (VNI) field is 24 bit identifier for multi-tenancy. Applications can isolate networks on an ovstack overlay by utilizing this VNI field. An application driver fills the VNI field when encapsulating a packet. Thus, receiver nodes can distinguish the network that the packet belongs to.

  Multi-tenancy and network isolation on both data center and Wide Area Network (WAN) environments are needed. Although VLAN is used for network isolation most practically, 12 bit VLAN ID is not enough for recent use cases such as large scale IaaS cloud [4]. Therefore, VXLAN introduces 24 bit VNI field for network isolation on data centers, and it is also reused into wide area VPN uses as a data plane for E-VPN [14]. Hence, we introduce 24 bit VNI field into ovstack header for multi-tenancy on large scale environments.

  As an example, how to separate layer 2 networks for ethernet over ovstack is described in Section 4.1.2.

- **Hash**

  Hash field is utilized to decide a locator address of a next hop node when the next hop node has multiple locators. If a node has multiple locators, traffic is balanced for locators on IP layer. In this case, a flow that has to be prevented packet reordering can be distinguished by only upper layer application. Then, by filling the hash field associated with a flow of the application distinguished, reordering caused by load balancing on the layer is avoided.

The process flow of a packet by ovstack routing layer is shown at Figure 4. First of all, application drivers that communicate upper-layer applications and

the ovstack routing layer have to be prepared for each application. The data path to be transferred is decided by only upper-layer application. It is similar to case of IP. An application driver encapsulates data with an ovstack header filled destination ID, source ID, application number and hash.

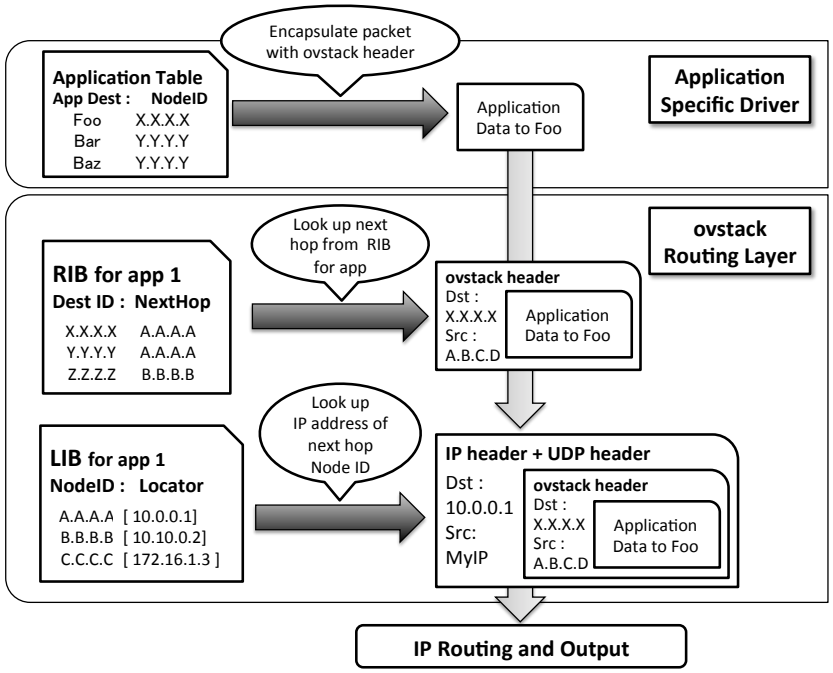


Figure 4: Process flow of a packet form an application driver to an overlay network through the ovstack routing layer.

ovstack routing layer receives encapsulated packets from application drivers. Then, next hop entries are found from routing table corresponding to application number of the packet: ovstack looks up a next hop from RIB at first, and look up the locator address of the next hop node from LIB. If the next hop or its locator address is not found, the packet is dropped. If the next hop has multiple locator addresses, a locator address is decided in accordance with the hash value of ovstack header. When a locator address of next hop is decided, the packet is encapsulated with UDP and IP headers. A source IP address is decided from own locator addresses by hash value. At last, the packet encapsulated up to IP header is passed to IP routing stack, and transmitted to the next hop. When a relay node receives an encapsulated packet, remove outer IP and UDP headers at first. Next, the packet with ovstack header is processed by ovstack routing layer in the same manner as received from application drivers. In this way, ovstack avoids the performance degradation due to re-encapsulation on relay node by multihop overlay routing without decapsulation of ovstack header.

4.1.2. Example: an Application Driver for Ethernet

We designed Overlaid Ethernet (oveth) as an application driver to transport ethernet frames on an ovstack overlay. oveth is implemented as a device driver for pseudo ethernet interface. oveth pseudo interfaces and its FDB are created for each VNI, which means layer 2 segments of each VNI are completely isolated. A sender node encapsulates packets from an oveth interface with a VNI corresponding the interface, receiver nodes can identify the layer 2 segment that packets belong to. Thus, oveth achieves multi-tenancy on an ovstack overlay by utilizing VNI field of ovstack header. Moreover, when packet is transmitted to ovstack routing layer from oveth, oveth driver fill the hash field of the ovstack header. The number of hash is calculated from source MAC address, destination MAC address and ether type of an inner ethernet frame to prevent packet reordering from load balancing of ovstack layer.

FDB of oveth is constructed in the same way as VXLAN does. When receiving encapsulated packets from ovstack routing layer, oveth driver learns the pair of i) source MAC address of inner ethernet frame and ii) source node ID of outer ovstack header. If a destination node ID is not found on FDB when transmitting packets, the destination node ID is set to a given node ID which is configured as a destination of broadcast MAC address. By building a route of the given ID as a multicast route, broadcast frames are transmitted to all of ovstack nodes that join an oveth overlay network.

4.2. Control Plane

As described in Section 2.1, locator/ID separation architecture achieves multi-home and mobility like LISP does. However, as described in Section 2.5.2, control and data plane systems of LISP depends mutually, and so it causes development and deployment cost. On the other hand, VXLAN data plane does not have these dependencies, then E-VPN [14] reuses VXLAN header into wide area networks with BGP based control plane system. In E-VPN, VXLAN becomes just a data plane system, and BGP handles pairs of destination MAC address and VTEP IP address. To be more deployable, data plane system including encapsulation header format and control plane system shall be isolated, and the definition of API between both planes is needed.

ovstack is a data plane, and its routing table and forwarding architecture is isolated from any control plane systems. Therefore, ovstack as a data plane provides APIs for only operations of routing tables to control plane systems. Moreover, in order to cooperate with ovstack we designed and implemented two control plane protocols to demonstrate that ovstack can accommodate multiple types of overlay networks for each requirement. They construct a unicast routing table and one multicast route for transporting ethernet frames by oveth.

4.2.1. API of Data Plane for Control Plane

We designed an API of ovstack routing layer to configure RIB and LIB, and control plane systems can control ovstack routing tables through this API. To achieve the isolation of control plane and data plane, ovstack routing layer

offers only APIs to configure LIB, RIB and own node ID information. Thus, data plane does not have to consider how control plane systems to calculate routing tables.

Table 1 shows supported operations of the API. The operation column indicates each command name, and the attributes column indicates necessary attributes of a command name. All operations must specify the application that is the same number of application field of ovstack header described in Section 4.1.1 to isolate operations among ovstack overlays. ovstack provides multiple overlay networks for each applications, so that all identifier spaces such as own node IDs and routing tables must be separated for each control planes and overlays. NODE operations modify or get the own node ID, and LOCATOR operations modify locator IP addresses of own node. NODE operations modify a LIB that contains information of node IDs and locator addresses of other nodes, and ROUTE operations modify a RIB that is an actual routing table of an overlay. By using this API, control plane systems can construct tailored overlay topologies through controlling RIB and LIB of the ovstack data plane.

Table 1: Operations that are offered to control plane systems by ovstack routing layer.

Operation	Attributes
NODE_ID_SET	application, node ID
NODE_ID_GET	application
LOCATOR_ADD	application, locator address, weight
LOCATOR_DEL	application, locator address
LOCATOR_GET	application
NODE_ADD	application, node ID, locator address, weight
NODE_DEL	application, node ID, locator address
NODE_GET	application
ROUTE_ADD	application, destination node ID, next hop node ID
ROUTE_DEL	application, destination node ID, next hop node ID
ROUTE_GET	application

4.2.2. End-to-End LAN

End-to-End LAN (e2LAN) is a control plane system of ovstack. It creates end-to-end full-mesh topology. Thus, the topology built by e2LAN is similar to topologies of VXLAN and LISP. e2LAN constructs RIB from the information of LIB. When e2LAN starts, e2LAN daemon obtains information of nodes and its locator addresses from LIB through NODE_GET operation. Then, e2LAN installs new route entry *"to Node A via Node A"* through ROUTE_ADD command.

e2LAN daemon is executed with following arguments: an application number, an own node ID, and an ID for multicast route. e2LAN checks a LIB and modifies a RIB that are specified by the application number. Moreover, e2LAN constructs one multicast route that reaches to all of ovstack nodes that join an overlay network. This multicast route is used for transporting broadcast frames

via oveth. The multicast route constructed by e2LAN is star topology with one reflector node, where a node with biggest node ID in LIB is elected as a reflector node.

4.2.3. ovstack Resilient Overlay Network

ovstack Resilient Overlay Network (vRON) is an userland software of a control plane protocol for ovstack. It is implemented a part of RON algorithm. vRON constructs delay shortest path topology. Each ovstack nodes with vRON measures delay between own node and other nodes, and exchanges delay information between all of nodes. Using delay as cost, a node calculates overlay route to other nodes with Dijkstra algorithm. vRON knows list and change of ovstack nodes in same overlay network through similar way to e2LAN. vRON checks the ovstack LIB through the API, and when node is added or deleted, vRON nodes recalculate overlay routing table.

vRON daemon is executed with following arguments: an application number, an own node ID, and an ID for multicast route. vRON checks a LIB and modifies a RIB that are specified by the application number. vRON also constructs a multicast route that also reaches to all of ovstack nodes in a same overlay network. Different from e2LAN, a multicast route of vRON is calculated with Prim's algorithm. Thus, a multicast route becomes minimum spanning tree between ovstack nodes.

4.3. Alternative data plane protocol for overlay layer

We propose ovstack as a data plane for overlay layer. Similarly, other data plane protocols can be designed certainly. The functionalities of overlay layer are defined by following items:

- node identifier and routing on overlay layer,
- identifier for encapsulated application, and
- isolation of control and data planes by defining API.

Node identifier and routing on overlay layer provide topology flexibility for specific requirements such as delay or bandwidth. We uses full flat ID as node identifier for ovstack, however longest match will also be used when considering scalability. Identifier for upper layer application, called application field in ovstack, is needed for that multiple applications use single data plane system like protocol number of IP header. Furthermore, defining API between control and data plane systems to isolate both planes is significant. This isolation achieves that operators and developers can utilize appropriate control plane systems for their purposes without data plane system replacement.

By adding or implementing these functionalities, same benefits can be provided out of the existing solutions. For instance, an identifier for encapsulated packet type on VXLAN is proposed [15].

4.4. Implementation

We implemented all of ovstack components as a Linux kernel module called `ovstack.ko`, and implemented two control plane systems as userland daemons. ovstack is one of network protocol stack above transport layer, so that we implemented ovstack as a part of network stack of Linux. When ovstack.ko is loaded, it create UDP socket to send and receive encapsulated packets, and it exports two functions to kernel space to inject packets to ovstack routing layer and to receive packets. Furthermore, We implemented oveth as an ethernet device driver of Linux. oveth uses the two functions exported by ovstack to send and receive encapsulated ethernet frames between ovstack.

The API to communicate control plane systems and ovstack is implemented as a extension of Netlink API [16]. Netlink has a mechanism called Generic Netlink to add new family dynamically. ovstack.ko add a ovstack family to Netlink as one of Generic Netlink family. ovstack family provides operations described in Section 4.2.1, so that control plane systems that calculate actual topologies and routing tables can install route entries to ovstack routing layer through this Netlink API like Quagga [17] software suites.

As control plane systems, we implemented vRON and e2LAN as userland software of Linux. They checks and modifies RIB and LIB through the Netlink API for ovstack. Moreover, in order to maintain and confirm RIB and LIB easily in actual environments, we implemented a couple of iproute2 extensions. Users and developers can operate ovstack through a similar way to operating IP layer of Linux kernel. The extension support the following operations: add/delete/show node and locator address information of LIB, add/delete/show route entries of RIB, and set own node ID and locator address. The command syntax of extension is modeled after `ip` command. For example, you can add new route by `ip ov route add app A to X.X.X.X via Y.Y.Y.Y` command, and you can see routing table by `ip ov route show` command.

5. Evaluation

In this section, we present our evaluation on the performance of our proposed method. More specifically,

- we evaluate the data plane (ovstack) performance to reveal additional overhead on packet encapsulation, and
- two control planes, e2LAN and vRON, are evaluated focusing on the variations of path construction between them.

The first evaluation is to figure out data plane performance. ovstack does not need packet encapsulation and decapsulation when forwarding packets. It will improve overlay packet forwarding performance compared to using existing tunneling protocols. The second evaluation is to demonstrate that ovstack can accommodate various overlay topologies by exchanging control plane systems. In order to show that, we evaluated two control plane systems, e2LAN and vRON, focusing on the delay as an example metric on a simulator environment.

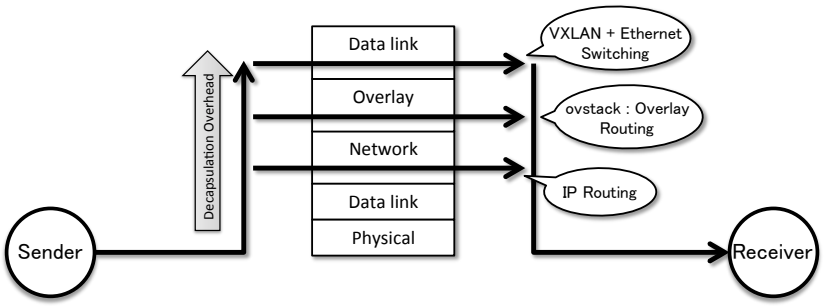


Figure 5: Packet forwarding on upper layers requires decapsulation end encapsulation overhead.

Table 2: Evaluation Equipment.

	CPU	Memory	Linux Kernel
ovstack node	Core i7 3770K 3.5GHz	32GB	3.8.0-19-generic
tester node	Xeon E5 2420 1.9GHz	16GB	3.0.93-netmap

5.1. Data Plane

Our first evaluation is about forwarding performance with throughput, delay, and jitter measurement to figure out performance degradation or improvement because of encapsulation beyond layer boundaries. ovstack forwards encapsulated packets above network layer. As show in Figure 5, 1) ovstack forwarding performance will degrade due to overhead of decapsulation and encapsulation for IP and UDP headers. However, 2) ovstack performance will be better than overlay forwarding with existing tunneling protocols because ovstack does not require decapsulation and encapsulation for overlay headers (e.g., VXLAN). In this evaluation, we figure out these degradation and improvement by comparing packet forwarding performances on each layer.

The computer nodes used in this performance tests are show in Table 2, and test topologies are shown at Figure 6. All of inter connect of nodes are Intel X520 10 Gbps Ethernet cards with direct attach cables. The tester software used in tests is implemented using netmap [18]. Because of ovstack forwards packets in kernel space, the maximum bandwidth of test traffic generated by most software traffic generators (e.g, iperf or pktgen) may be forwarded by ovstack without packet loss. Thus, we used netmap for generating test traffic. Then, we measured the performance with IP routing, ovstack routing, and switching between two different VXLAN networks as shown in Figure 5. All test traffic of IP, ovstack and VXLAN packets are generated by a tester node.

At first, we tested the throughput of implementations changing packet size of test traffic from 64, 128, 256, 512, 1024, 1500, 2048 to 4096 bytes. At cases of each packet size, the test traffic was transmitted for 180 sec.

Figure 7(a) shows the result of throughput measurement. The result shows that the performance of ovstack is worse than the one of IP due to packet

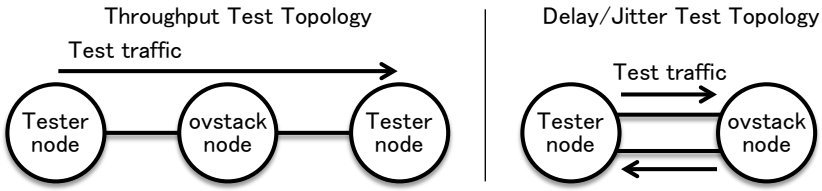


Figure 6: Experimental Topology to evaluate performance of data plane.

encapsulations. However, ovstack performance is comparable to IP routing when packet size is above 1500 bytes. On the other hand, the performance of ovstack is better than VXLAN's result. This is because, in VXLAN case, decapsulation packets, switching, re-encapsulation with VXLAN, UDP, IP headers processes are required. Thus these wasted processes cause performance degradation. This provides the fact, as described Section 2.5.1, routing and forwarding on overlay layer can reduce the overhead of overlay routing that was constructed from existing overlays having full-mesh tunneling topology.

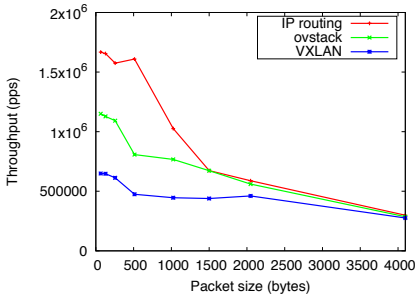
Figure 7(b) shows the result of forwarding delay test, and Figure 7(c) show the jitter of packet forwarding. In case of both tests, we used 128 bytes packet size because when the timestamp is inserted to payload of VXLAN encapsulated packet, the packet length is longer than 100 bytes (outer ethernet header, outer IP header, UDP header, VXLAN header, inner ethernet header, inner IP header, and struct timespec). The jitter denotes the variance of the difference in round trip time between successive packets.

With the results, performance of ovstack is worse than IP routing without encapsulation as we expected. The cause of this degradation is also encapsulation obviously. However, it is better than VXLAN's one, delay decreased 12 % than VXLAN and jitter is 8 % lower than VXLAN. These results means that the routing on overlay layer is effective against routing using tunneling technologies, as well.

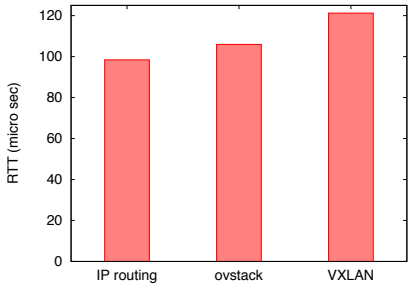
5.2. Control Plane

To demonstrate that ovstack can accommodate different overlay topologies on one data plane system, we designed and implemented two control plane systems as a proof of concept. e2LAN constructs full-mesh topology as existing overlay networks, and vRON constructs delay based shortest path topology. Then, through comparing delay between all of nodes in each overlay topology constructed by e2LAN and vRON, we show that ovstack can meet specific requirements for applications by only exchanging control plane systems.

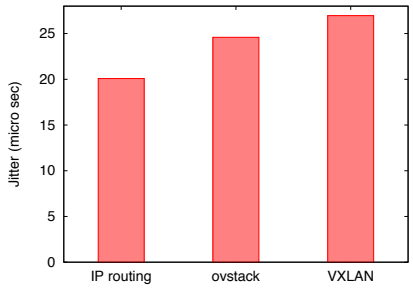
In order to demonstrate control plane implementations in a large network, we simulated the network using ns-3 [19] and Direct Code Execution (ns-3-dce) [20] extension. ns-3-dce can emulate Linux kernel and userspace on ns-3 simulation environment. We simulated Linux nodes including ovstack kernel module and control planes on ns-3 simulated network. The underlay IP network topology of simulated environment is AS-Level eBGP topology. Quagga is used



(a) Throughput of packet forwarding.



(b) Delay of 128 bytes packet forwarding.



(c) Jitter of 128 bytes packet forwarding.

Figure 7: Routing and forwarding packet performances about throughput, delay, and jitter.

for routing software of eBGP. Therefore, in simulated environment, one Linux node is equal to, one ovstack node, one eBGP router, and one AS. ASes used in simulation are top 30 ASes from JPNIC AS, which are selected from CAIDA AS relationship [21] by anybed tools [22].

Figure 8 shows cumulative distribution function of delays between all node pairs in underlay IP network, e2LAN overlay network, and vRON overlay network. In this simulation, ethernet frame is used as overlayed application through oveth driver, and delays of links between simulated node are randomly changed from 10 to 50 ms. Simulations are performed 5 times changing delays for each link. Delays between nodes are measured by 126 bytes size UDP packets that timestamp is inserted to payload. With the result, delays of e2LAN overlay and eBGP based IP network are roughly equal. This is because e2LAN constructs full-mesh tunneling topology as VXLAN and LISP, thus paths between nodes are equal to IP routing. The reason why delays of e2LAN are longer than IP routing is overhead of encapsulation and decapsulation for measurement packet. On the other hand, the delay at vRON topology is lower than e2LAN and IP routing. Since vRON can select shorter delay path between nodes on overlay layer without being tied to routing table of IP layer. As the result, this proof of concept shows that, different overlay topologies specialized particular metric such as delay can be constructed on the ovstack data plane.

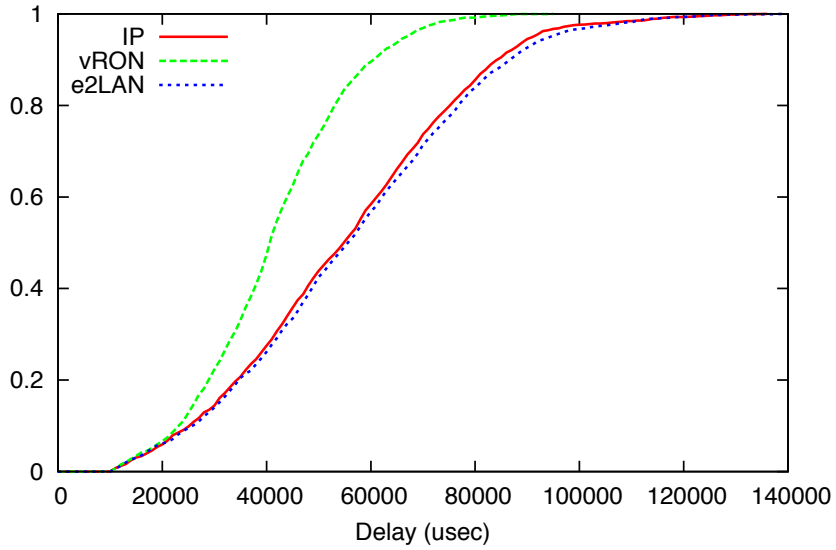


Figure 8: CDF of delays between nodes in each topology.

6. Conclusion

We have proposed ovstack, a data plane for overlay networks by introducing new abstraction layer between existing network layer and the applications. The data plane aims to decouple any control planes, which have different requirements, and provide a flexible path configuration. Then, we have designed and implemented the proposed system. We evaluated the performance of ovstack as a data plane, and the result is acceptable. Furthermore, we implemented and tested two control plane protocols for ovstack. The result of control plane simulation shows that ovstack can accommodate multiple overlay networks for each requirement. As a result, we conclude that, the overlay network is abstracted as a layer of the network layering model. Through using the ovstack, developers will be able to create new overlay routing systems by developing control plane only, utilizing common data plane that has acceptable performance.

Acknowledgement

This research has been supported by the Strategic International Collaborative R&D Promotion Project of the Ministry of Internal Affairs and Communication, Japan, and by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 608533 (NECOMA). The authors wish to thank to H.Tazaki and Y.Kuga for encouragement and helpful discussions.

References

[1] J. Loveless, ACM-QUEUE:High-frequency Trading and Exchange Technology, http://queue.acm.org/detail.cfm?id=2536492 (2013).

[2] K.-T. Chen, C.-Y. Huang, P. Huang, C.-L. Lei, Quantifying skype user satisfaction, SIGCOMM Comput. Commun. Rev. 36 (4) (2006) 399–410. doi:10.1145/1151659.1159959.
URL http://doi.acm.org/10.1145/1151659.1159959

[3] T. Li, Design Goals for Scalable Internet Routing, RFC 6227, IRTF (May 2011).

[4] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, C. Wright, Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks, RFC 7348, IETF (Aug. 2014).

[5] D. Farinacci, V. Fuller, D. Meyer, D. Lewis, Locator/ID Separation Protocol (LISP), RFC 6830, IETF (Jan. 2013).

[6] D. Andersen, H. Balakrishnan, F. Kaashoek, R. Morris, Resilient overlay networks, SIGOPS Oper. Syst. Rev. 35 (5) (2001) 131–145. doi:10.1145/502059.502048.
URL http://doi.acm.org/10.1145/502059.502048

[7] M. Castro, P. Druschel, A.-M. Kermarrec, A. I. T. Rowstron, Scribe: a large-scale and decentralized application-level multicast infrastructure, Selected Areas in Communications, IEEE Journal on 20 (8) (2002) 1489–1499. doi:10.1109/JSAC.2002.803069.

[8] Y. Chawathe, Scattercast: an adaptable broadcast distribution framework, Multimedia Syst. 9 (2003) 104–118. doi:10.1007/s00530-002-0082-z.
URL http://dl.acm.org/citation.cfm?id=957991.958006

[9] R. Nakamura, K. Okada, Y. Sekiya, H. Esaki, ovstack : A Protocol Stack of Common Data Plane for Overlay Networks, in: SDNMO-Workshops, 2014, IFIP/IEEE NOMS, 2014.

[10] BitTorrent, http://www.bittorrent.com/.

[11] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, A. Singh, Splitstream: high-bandwidth multicast in cooperative environments, SIGOPS Oper. Syst. Rev. 37 (2003) 298–313. doi:http://doi.acm.org/10.1145/1165389.945474.
URL http://doi.acm.org/10.1145/1165389.945474

[12] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for internet applications, in: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '01, ACM,

New York, NY, USA, 2001, pp. 149–160. doi:10.1145/383059.383071.
URL http://doi.acm.org/10.1145/383059.383071

[13] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, Rofl: Routing on flat labels, SIGCOMM Comput. Commun. Rev. 36 (4) (2006) 363–374. doi:10.1145/1151659.1159955.
URL http://doi.acm.org/10.1145/1151659.1159955

[14] R. Shekhar, A. Lohiya, J. Rabadan, S. Sathappan, W. Henderickx, S. Palislamovic, F. Balus, A. Sajassi, D. Cai, draft-ietf-bess-dci-evpn-overlay-00, ID, IETF (Jan. 2015).

[15] P. Quinn, R. Manur, L. Kreeger, D. Lewis, F. Maino, M. Smith, P. Agarwal, L. Yong, X. Xu, U. Elzur, P. Garg, D. Melman, draft-quinn-vxlan-gpe-04.txt, ID, IETF (Feb. 2015).

[16] J. Salim, H. Khosravi, A. Kleen, A. Kuznetsov, Linux Netlink as an IP Services Protocol, RFC 3549, IETF (Jul. 2003).

[17] Quagga routing suite, "http://www.quagga.net/".

[18] L. Rizzo, Netmap: A novel framework for fast packet i/o, in: Proceedings of the 2012 USENIX Conference on Annual Technical Conference, USENIX ATC'12, USENIX Association, Berkeley, CA, USA, 2012, pp. 9–9.
URL http://dl.acm.org/citation.cfm?id=2342821.2342830

[19] NS-3 project, http://www.nsnam.org/.

[20] H. Tazaki, F. Uarbani, E. Mancini, M. Lacage, D. Camara, T. Turletti, W. Dabbous, Direct code execution: Revisiting library os architecture for reproducible network experiments, in: Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, CoNEXT '13, ACM, New York, NY, USA, 2013, pp. 217–228. doi:10.1145/2535372.2535374.
URL http://doi.acm.org/10.1145/2535372.2535374

[21] CAIDA Project, http://www.caida.org/home/.

[22] H. Hazeyama, M. Suzuki, S. Miwa, D. Miyamoto, Y. Kadobayashi, Outfitting an Inter-AS Topology to a Network Emulation TestBed for Realistic Performance Tests of DDoS Countermeasures, in: Proceedings of Workshop on Cyber Security Experimentation and Test (CSET'08), 2008.