ovstack : A Protocol Stack of Common Data Plane for Overlay Networks

Ryo Nakamura
University of Tokyo
Email: upa@wide.ad.jp

Kouji Okada
Lepidum
Email: okd@lepidum.co.jp

Yuji Sekiya
University of Tokyo
Email: sekiya@wide.ad.jp

Hiroshi Esaki
University of Tokyo
Email: hiroshi@wide.ad.jp

***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) the topology of existing overlays is essentially full-mesh tunneling topology, 2) dependence of control plane and data plane. The full-mesh tunneling topology cannot enable the overlay routing for performance improvement of networks. The dependence of them causes complication of operations due to the isolation of overlay networks, and increases development costs. To improve the problems, we propose a new abstraction layer provides a common architecture for data planes of overlay networks that can deploy overlay routing. Based on the architecture, we design and implement a protocol stack, called ovstack. In this paper, we describe the architecture, design and implementation, then evaluate the performance of overlays including ovstack. The ovstack can contribute to construct more flexible overlay networks on the current networks easily.**

I. 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 requires 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.

To satisfy various requirements, existing networks constructed by IP and Ethernet lack flexibility. As problems of IP, decoupling location and identification, multi-homing, mobility, simplified renumbering, modularity, and routing quality are remarked in RFC6227 [3]. Some problems of them are attributed to semantics of IP address structure. An IP address block is a part of larger address block. Through this semantics, IP address blocks are assigned to Internet Service Providers (ISP). This scheme provides aggregation of routes in Default Free Zone (DFZ), and it helps compression of global routing table on DFZ. However, this scheme causes coupling of locator information and identification information of IP. Thus, an end node cannot be moved to other networks remaining IP address of a former network. In addition, IP does not have an identifier that denotes a network, thereby multi-tenancy is not be achieved in IP layer. Hence, routing quality that means network multi-tenancy for requirements cannot be realized in existing IP layer.

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 Software Defined Network (SDN). Overlay technologies construct their own networks over the IP network by encapsulating packets with their protocol headers in IP datagrams. This encapsulation enables to add network functionalities to the overlay networks corresponding to the requirements of an application. For example, Virtual eXtensible LAN (VXLAN) [4] encapsulates Ethernet frame with IP, UDP, and VXLN header. VXLAN enables multi-tenancy that is able to separate network up to 1,677,216 segments by introducing new identifier in VXLAN header. In addition, Locator/ID Separation Protocol (LISP) [5] encapsulates IP datagram with IP, UDP, and LISP header. LISP separates two aspects of IP, that are identifier and locator by introducing IP encapsulation, thereby it achieves prefix mobility and multi-homing. Moreover, some other overlays achieve multicast communication in overlay networks [7], [8].

However, the architecture of existing overlay technologies has two problems. First problem is that the topology of existing overlays is essentially 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. Second problem is dependence of control plane and data plane. Existing overlay technologies are specialized in each upper application, and 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 this paper, we introduce a new abstraction layer for overlay networks in the network layering model. And we design and implement a protocol stack of common data plane for overlay networks. The proposed protocol stack, called ovstack, offers common functions of the data plane, that are proper header format, node ID on the overlay network space, the routing table for overlay networks, packet routing and forwarding based on the routing table, and API to be controlled from control plane. Thanks to the ovstack, operators do not have to build their own overlay networks independently, as building IP networks. Furthermore, developers only design a control plane to create a new overlay technology, by utilizing ovstack.

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 paper is organized as follows: we present characteristics of existing overlays, and define problems in Section II. Section III describes requirements and proposed system, and implementation of ovstack is shown in Section IV. Section V shows evaluation environments and its result about performance of ovstack as a data plane. Finally, Section VI summarizes the results of this study, and refers future work.

II. Existing Overlays

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

A. Locator/ID Separation Protocol

Locator/ID Separation Protocol (LISP) is an IP over IP overlay network architecture and protocol. LISP manages two aspects of the IP address, addressing and locator, in isolation. 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 IPv4 or IPv6 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 that multi-homing of the EID prefix. In IP layer, it is necessary to 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. Furthermore, by using an IP address of end host as a EID, LISP achieves host scale address mobility.

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.

B. Virtual eXtensible LAN

Virtual eXtensible LAN (VXLAN) achieves Ethernet emulation over IP network. In VXLAN, new 24 bits identifier is added into VXLAN header when the Ethernet frame is encapsulated. By adding this identifier that is named VXLAN Network Identifier, VXLAN is able to isolate enough number of network segments. The control plane of VXLAN is that the unicast frame is encapsulated with IP unicast, and unknown unicast, broadcast and multicast frames are encapsulated with IP multicast. Because using IP multicast, VXLAN nodes (VTEP) can join the VXLAN overlay network without 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 Ethernet frame, it finds FDB entry in accordance with destination MAC address of the frame. If the entry does not exist, VTEP transmits the packet with encapsulation to the given IP multicast address. If the entry is found, VTEP transmits the packet with encapsulation to the IP address that is contained in the found entry. Thus, all of VTEPs joining the IP multicast address can receive broadcast frames. When a VTEP receives an encapsulated packet from the overlay, register 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 utilizing IP multicast and inner source MAC and outer source IP address snooping.

C. Resilient Overlay Network

Resilient Overlay Network (RON) [6] is an overlay network that achieves improvement of network performance such as throughput and delay by utilizing the multihop overlay routing. RON nodes have unique identifier for each node in RON overlay network, and construct overlay routing table to realize the multihop routing overlay. 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 RON 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 overlay routing table using identifier on the overlay network apart from IP routing table. By selecting better quality paths when constructing the overlay routing table, routing overlays achieves to improve the performance of network. To achieve the routing overlays, the RON node adds destination and source node identifiers to encapsulation header.

D. Problem Definition

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

1) Full-mesh Tunneling Topology: First problem is, the topology of most existing overlay technologies is essentially full-mesh tunneling topology. In existing overlays like IP tunneling, LISP and VXLAN, a path between overlay nodes is end-to-end in accordance with IP routing. Thus, performance improvement through relaying nodes in overlay paths cannot be achieved. If realizing routing overlays with tunneling overlay technologies, the overhead of re-encapsulation occurs in relay nodes: Relay nodes decapsulate packets from a overlay link at first, routing 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 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 does not have unique node identifier in 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 is needed.

2) Dependence of Control Plane and Data Plane: Second problem is, the dependence of control plane and data plane. IP tunneling technologies do not have control plane, because tunnel end points are certain given by configuration. LISP utilizes map protocols as control plane, so that data plane (e.g., header formats and routing table) has particular formats specified to LISP map protocol, and control plane has also specific field for the data plane; therefore, the control plane and data plane of LISP cannot be utilized from other overlay network technologies reciprocally. Other existing overlay technologies are similar to that. The header format of RON has some specific fields such as policy tag and flow id for the RON control plane. Moreover, routing overlays for multicast can construct multicast overlay topology, but it cannot make a unicast routing table on the data plane.

As described above, existing overlay networks has mutual dependence of control plane and data plane to achieve the functions. Consequently, all of overlay networks built with each technology are completely isolated, and it causes 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.

III. Approach

In this study, to solve problems as referred to above, we introduce a new abstraction layer for overlay networks and propose a common protocol stack, called ovstack, as a data plane of the abstraction layer. Existing overlays are designed specifically for each upper application and function. Thus, it causes isolation of overlay networks and partial lack of features. In this study, instead of building an overlay network as a single technology, the overlay network itself is abstracted as a function of the network into the network layering model. Figure 1 shows the overview of this abstraction layer. Proposed system provides protocol stack of overlay data plane in the network layering model as a overlay layer at Fig 1.

A. Requirements

The requirements of new abstract layer for overlay networks are shown below.

1) Node ID in overlay network
2) Routing and forwarding in overlay network
3) Routing tables for each applications
4) Isolation of data plane and control plane

At first, routing on the overlay layer is achieved through that overlay nodes have each node ID in the overlay layer like RON. Thus, by multihop overlay routing achieves to improve performance about specific metrics such as throughput, and multicast routing in an overlay network. On the other hand, this routing table for overlay network have to be able to be constructed multiply. Routing tables should be constructed for each on application, because requirements of each application to network are different from others. Namely, multi-tenancy of overlay network for applications have to be capable. Finally, architectures of control plane and data plane have to be isolated. Instead of especial system for specific requirements or applications, data plane have to be utilized transparently through each control plane corresponding to requirements.

B. Overview of ovstack

In this paper, we propose ovstack as a protocol stack that structures the data plane of the overlay layer. In this section, we describe the overview of ovstack. Existing overlay technologies construct individual overlay networks for each application or requirement on the network layer i.e., IPv4 and IPv6 as shown at Figure 1. Thus, the isolation of overlay networks and overhead of re-encapsulation at relay nodes are occurred. Therefore, ovstack provides common data plane for overlays as a one of layers. Applications utilizing overlay networks can construct their own overlay network for each application through the overlay layer structured by ovstack.

ovstack allows construction of multiple overlay networks for each application or requirement. Figure 2 depicts the overview of it. In ovstack architecture, data plane and control plane are completely isolated by system design. The data plane provides following functions, node identifier, multiple routing tables, packet encapsulation and decapsulation, routing and forwarding encapsulated packets. Multiple routing tables are constructed at ovstack layer, and multiple control planes operate each routing table. Thus, multiple overlay networks for various applications, requirements or specific functions for a flexible structure can be treated at same overlay space.

Moreover, through utilizing ovstack, developers need not to design and implement a data plane by themselves when creating a new overlay technology. In existing overlays, Packet formats and control plane formats are designed for each application, thereby control and data plane do not have transparency. Thus, developers have to design all of entity of an overlay, and complications of operations for individual overlays are increasing. By contrast, ovstack providing common functions of data plane as a abstraction layer, a new overlay technology is able to be developed by only designing a control plane.

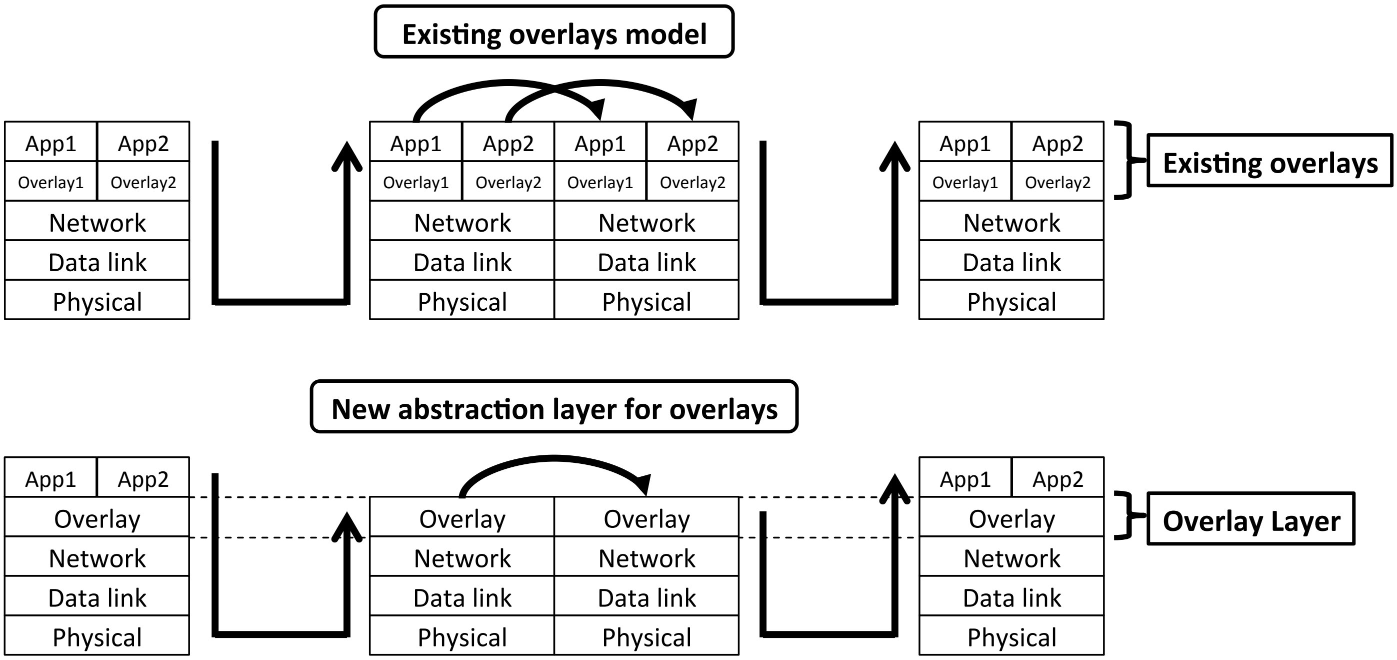

Fig. 1. Abstraction layer for Overlay Networks.

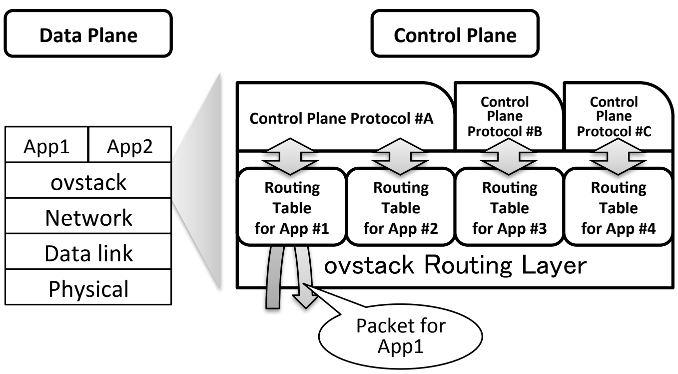

Fig. 2. Overview of data plane and control plane of ovstack.

C. Design of Data Plane of ovstack

ovstack is an protocol stack as a data plane of overlay networks. In this section, we describe the design of aspect as a data plane of ovstack.

In ovstack, node ID is 32 bits identifier in flat space. It is meaning that ID does not have especial semantics like netmask and longest match of IP. Using flat space for node identifier avoids ID and locator binding related problems like IP. Moreover, it has possibility of introducing methods of Distributed Hash Table (DHT) such as Chord [10] and Routing on Flat Label [11] for construction of overlays.

ovstack contain routing tables for multihop routing in overlay layer. This routing table is constructed from 2 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. A 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. Multicast forwarding is achieved by this method. Moreover, as referred to above, ID space of ovstack does not have semantics, so that specific IDs for multicast do not exist.

LIB consists of entries of a node ID and IP addresses as a locator of a node. When routing packets on overlay layer, next hop node is found from RIB. And IP is utilized as actual transmission of lower layer of the overlay. LIB is map table that is used when finding an actual IP address of next hop nodes. ovstack LIB allows multiple locator IP addresses for one node. it enables that load balancing like LISP locator balancing that cannot be achieved on IP layer. Each locator address has weight value, so that weight based locator address load balancing is achieved. Thus, if a node has multiple links in IP layer, ovstack can utilize bandwidth of these links efficiently.

Next, we describe packet routing and forwarding processes. Figure 3 shows the header format of ovstack. ovstack encapsulates packets with IP, UDP, and this ovstack header. Functions and roles of each field are described below.

```
ovstack header format :
0                   1                   2                   3                   4
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Version    |  Application  |       TTL       |      Flags      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Virtual Network Identifier         |       RSV       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                Hash                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Destination Node ID                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Source Node ID                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 3. Header format of ovstack.

- **Application Field**
  Application field presents the type of upper layer application. When routing packets, a routing table to be referred to is indicated by this number of application field of received packets.
- **TTL (Time To Live)**
  TTL field presents the residual hop count. Nodes decrease a hop count when forwarding the packet. And if the field is 0, the packet is dropped. Thus, packet increases caused by routing loop on overlays is avoided.
- **Virtual Network Identifier**
  Virtual Network Identifier (VNI) field is utilized for multi-tenancy by upper layer applications. Applications can isolate networks on ovstack overlay utilizing this VNI field. The routing layer of ovstack doesn't do anything about this field.
- **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 in response to a flow of the application distinguished, reordering caused by load balancing on the layer is avoided.
- **Destination Node ID**
  Destination Node ID field presents the node ID that a packet has to be arrived at last. A node receiving a packet finds a next hop by this field from routing table, and transmits to the next hop.
- **Source Node ID**
  Source Node ID field presents a node that sent the packet.

The process flow of a packet from an application to an overlay network through the ovstack routing layer is shown at Figure 4.

At first, application drivers have to be prepared for each application that utilizes ovstack. Where data have to be transferred to can be decided by only the application. It is similar to a case of IP. Due to this, applications must resolve the ID of destination node by using some protocols (e.g., application table shown at Figure 4, or some DNS like systems). An application driver encapsulates data with an ovstack header filled destination ID, source ID, application number and hash.

ovstack routing layer receives encapsulated packets from application drivers. Then, it finds a next hop of a packet in accordance with routing table in response to application number of the packet. At first, look up the next hop node ID from RIB. if it is not found, the packet is dropped. At second, look up the locator address of the next hop node from LIB. if it is not found, the packet is also dropped. if the next hop has multiple locator addresses, a locator address is decided in accordance with hash value of ovstack header of the packet. When decided a locator address of next hop, the packet is encapsulated with UDP header, and IP header. A source IP

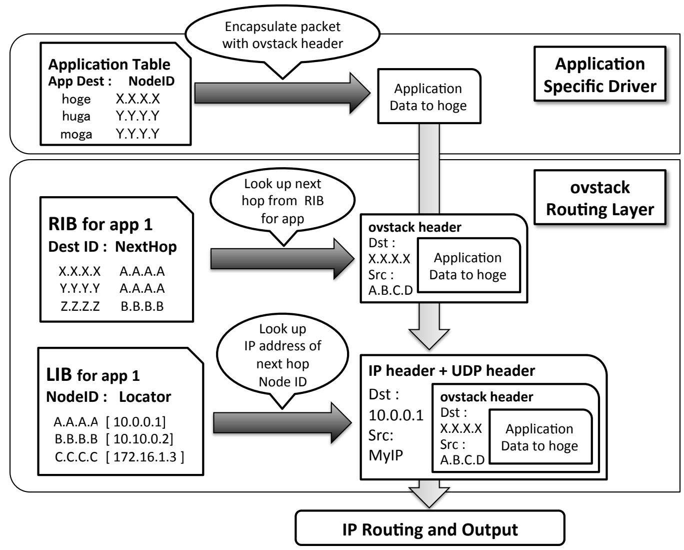

Fig. 4. Processing flow of a packet form an application driver to an overlay network through the ovstack routing layer.

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 of own node, and transmitted to the next hop through an IP network.

When a relay node receives an encapsulated packet, remove outer IP and UDP headers at first. Next, the packet with ovstack header is processed in the same manner as received from an application driver. Look up a next hop from RIB in response to the application number of the packet, look up a locator address of the next hop, encapsulating the packet with UDP and IP headers, and transmitting to next hop. In this way, ovstack achieves hop-by-hop overlay routing.

D. Design of Control Plane of ovstack

ovstack is data plane, and its routing table and forwarding architecture is isolated from control plane systems to achieve transparency. Therefore, ovstack as a data plane provides APIs for operation of routing tables to control plane systems.

RIB and LIB that routing table of ovstack is constructed from are prepared multiply for each applications. Therefore, multiple overlay networks for each requirement can be built on one data plane system of ovstack. Thus, instructions to routing table from control plane are also operated per routing table individually. Instructions for a single routing table are show below.

- Add node ID, locator address and weight to LIB, delete and look up them.
- Add route to RIB, delete and look up it.

By adding locator address information of a node to LIB, packets in ovstack header is transmitted through an IP network. Moreover, by adding entries of route to RIB, routing table is built by control planes. ovstack provides these APIs, and control planes maintain routing tables through them. Instructions for LIB and RIB are also isolated, so that proper control planes for LIB and RIB is designed and selected in response to operational requirements.

IV. IMPLEMENTATION

In this paper, we implemented ovstack as a part of network stack of Linux kernel. In addition, we implemented Overlaid Ethernet (oveth) as an application driver for Ethernet on ovstack.

A. ovstack

ovstack is implemented as a kernel module of Linux, and it runs as a part of network stack. ovstack provides a function of ovstack routing with RIB and LIB as a referred to above. At the initializing, ovstack creates a UDP socket in kernel. To receive encapsulated packets in UDP, `encap_rcv` callback of `struct udp_sock` is used in similar way of VXLAN driver of Linux kernel. When received packets through UDP socket or application driver, ovstack processes routing and forwarding in accordance with RIB. If the destination of ovstack packets is own node ID, packets are passed to application driver in response to application filed of ovstack header. If it is not own node ID, packets are transmitted to a locator address of next hop node that is found from LIB without decapsulation of ovstack header.

In addition, we implemented APIs to operate RIB and LIB via Netlink. Netlink is an API for communication between userland application and Linux kernel. Through Generic Netlink, a part of Netlink API, developers are able to make a user defined instruction and data structure for communication to with kernel. Instructions that are add/del/look up RIB and LIB are implemented with this Generic Netlink extension.

B. oveth

oveth is application driver to transport Ethernet frame on an overlay network of ovstack. oveth is implemented as a Linux device driver for an Ethernet interface. Users are able to create pseudo Ethernet interface that is connected to an overlay network by ip link add type oveth command of customized iproute2. These pseudo NICs can be created for each VNI. oveth creates Forwarding Data Base (FDB) that constructed from pairs of destination MAC address and destination node ID for each VNI, so that multi-tenancy on ovstack overlay is achieved. Packets transmitted from a pseudo NIC are encapsulated with ovstack header in accordance with FDB of a VNI of the NIC, and passed to ovstack routing layer.

FDB of oveth is constructed in the same way of VXLAN. When receiving encapsulated packets from ovstack routing layer, oveth driver learns the pair of source MAC address of inner Ethernet frame and source node ID of outer ovstack header. If destination node ID is not found on FDB when transmitting packets, destination node ID is set 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 transferred to all of ovstack node that join an oveth overlay network.

V. EVALUATION

Our proposed method, ovstack, is a protocol stack of data plane for overlays. Thus, in this paper, we evaluated about performance of packet forwarding. Due to encapsulation, degradation of forwarding performance against native IP routing and Ethernet switching is expected. Thereby, it must be evaluated whether the performance is acceptable or not. In addition, ovstack avoid the overhead of re-encapsulation by routing on overlay layer. It must be also evaluated the improvement of this avoidance.

TABLE I. EVALUATION EQUIPMENTS.

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

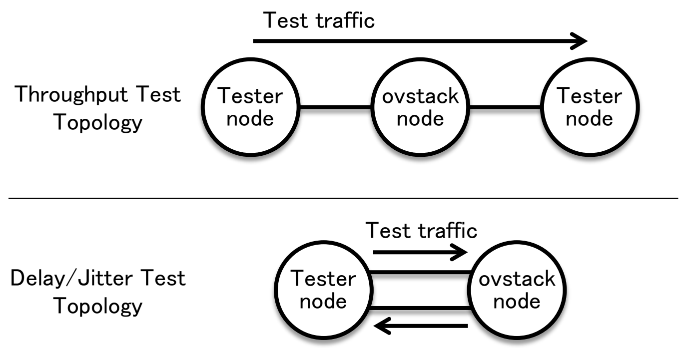

Fig. 5. Experimental Topology to evaluate performance of ovstack data plane.

We evaluated forwarding performance about throughput, delay, and jitter. The computer nodes used in this performance tests are show in Table I, and test topologies are shown at Figure 5. All of inter connect of nodes are Intel X520 10 Gigabit Ethernet Cards and direct attach cables. The tester software used in tests is implemented using netmap [12]. Because of that ovstack forwards packets in kernel, the max 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. In tests, ovstack routing, switching between 2 different VXLAN networks, IP routing, and Ethernet switching are evaluated. All test traffic of ovstack, VXLAN, IP and Ethernet packets, are generated by a tester node. Openvswitch[13] version 1.9.0 was used for Ethernet switching.

We tested the throughput of implementations changing packet size of test traffic from 64, 128, 256, 512, 1024, 1500, 2048, 4096, to 9000 bytes. At cases of each packet size, test traffic was transmitted for 180 sec. Figure 6 shows the result of throughput test.

With the result, the performance of ovstack is worse than performance of IP and Ethernet. It is caused by packet encapsulation of ovstack. However, performance does not change much after 1500 bytes and 64, 126, 256, bytes compared to the Ethernet switching. It means that ovstack performance and its degradation is within the feasible range compared to Ethernet switching. On the other hand, the performance of ovstack is better than VXLAN's result. In VXLAN case, decapsulation packets from a link, Ethernet switching, encapsulation with VXLAN, UDP, IP headers again, and transmitted processes are required. Thus these wasted processes cause performance degradation. It means that routing and forwarding on overlay layer can reduce the overhead of overlay routing that was constructed from existing overlays having full-mesh tunneling topology.

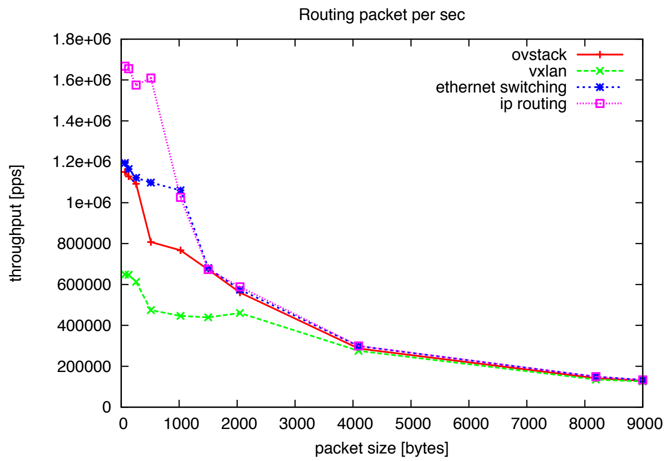

Fig. 6. Throughput of packet forwarding.

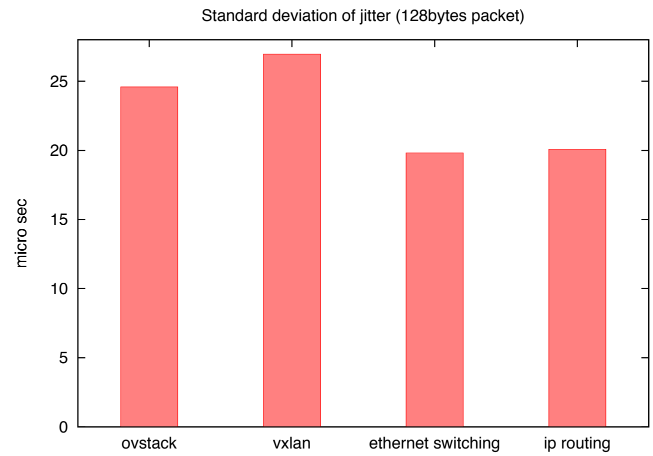

Fig. 8. Jitter of packet forwarding.

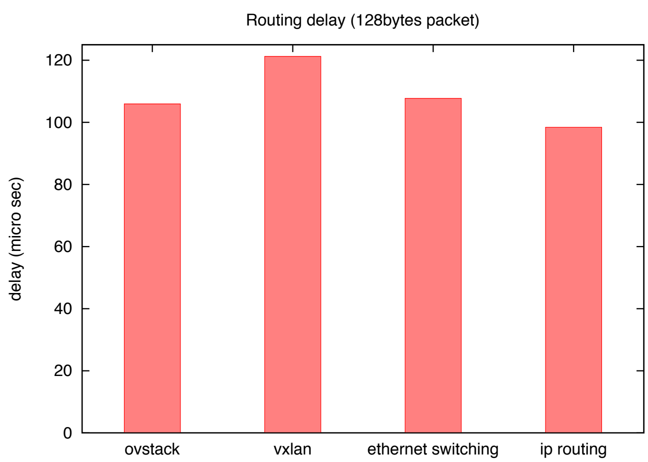

Fig. 7. Delay of packet forwarding.

Figure 7 shows the result of forwarding delay test, and Figure 8 show the jitter of packet forwarding. In case of both tests, packet size was 128 bytes, because when inserted a timestamp to VXLAN encapsulated packets, the packet length had to be 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 packet forwarding without encapsulation as we expected. The cause of this degradation is also encapsulation obviously. However, it is better than VXLAN's one. These results means that the routing on overlay layer is effective against routing using tunneling technologies, as well.

VI. Conclusion and Future work

In this paper, we have presented a new abstraction layer for overlay networks between the existing network layer and application in the network layering model. The new layer has abstracted the overlay network as a function of the network stack. Then, we have proposed ovstack as a protocol stack of a data plane for overlay networks, and we have designed and implemented proposed system as a part of network stack of Linux kernel. We evaluated the performance of ovstack as a data plane. The performance is degraded due to encapsulation, however it is acceptable. Furthermore, it is better than overlay routing using existing overlay technology because of the routing on overlay layer. As a result, we conclude that, the overlay network is abstracted as a layer of the network stack, and its data plane is provided. Through using the ovstack, developers will be able to create new overlay routing systems freely with common data plane that has acceptable performance.

Although we proposed ovstack as a protocol stack for common data plane, the overlay network cannot be built by only data plane. Now, we are designing and implementing some protocols for the control plane to show the proof that overlay networks are able to be built on the ovstack abstraction layer with various control planes. In addition, we are now considering a scalable way to negotiate node ID not to cause the Locator/ID binding problems in multiple overlays environment.

Acknowledgment

This research is a part of GINEW Project funded by National Institute of Information and Communication Technology in Japan. Also, 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).

Code Availability

The source code described in this paper is available at https://github.com/upa/ovstack. Linux kernel module implementations of ovstack and oveth, and iproute2 including extensions for them are contained in this repository.

References

[1] "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, and C.-L. Lei, "Quantifying skype user satisfaction," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, pp. 399–410, Aug. 2006. [Online]. Available: http://doi.acm.org/10.1145/1151659.1159959

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

[4] M.Mahalingam, D.Dutt, K.Duda, P.Agarwal, L. Kreeger, T. Sridhar, M.Bursell, and C.Wright, "draft-mahalingam-dutt-dcops-vxlan-00.txt," IETF, ID, Aug 2011.

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

[6] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 131–145, Oct. 2001. [Online]. Available: http://doi.acm.org/10.1145/502059.502048

[7] M. Castro, P. Druschel, A.-M. Kermarrec, and A. I. T. Rowstron, "Scribe: a large-scale and decentralized application-level multicast infrastructure," *Selected Areas in Communications, IEEE Journal on*, vol. 20, no. 8, pp. 1489–1499, 2002.

[8] Y. Chawathe, "Scattercast: an adaptable broadcast distribution framework," *Multimedia Syst.*, vol. 9, pp. 104–118, July 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=957991.958006

[9] V. Fuller, D. Farinacci, D. Meyer, and D. Lewis, "Locator/ID Separation Protocol Alternative Logical Topology (LISP+ALT)," IETF, RFC 6836, January 2013.

[10] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and 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*, ser. SIGCOMM '01. New York, NY, USA: ACM, 2001, pp. 149–160. [Online]. Available: http://doi.acm.org/10.1145/383059.383071

[11] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, and I. Stoica, "Rofl: Routing on flat labels," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, pp. 363–374, Aug. 2006. [Online]. Available: http://doi.acm.org/10.1145/1151659.1159955

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

[13] "Openvswitch," http://openvswitch.org/.