OFIAS: A Platform for Exploring In-Network Processing

Ping Du[1], Maoke Chen[1], and Akihiro Nakao[1,2]

[1] National Institute of Information and Communications Technology (NICT), Japan
[2] The University of Tokyo, Japan

Abstract. In-network processing (INP) is being used to cope with the large volume of data streams that need to be analyzed in real-time of data transmission rather than being stored and computed by powerful servers. In this paper, we combine the programmable switch OpenFlow with network virtualization and design the INP platform *OFIAS*, i.e., OpenFlow In A Slice. With the flexibility of OpenFlow and the scalable multiplexing of virtualization, OFIAS can smoothly support multi-party INP with well isolation and attractive performance in comparison to other approaches.

Keywords: Network virtualization, In-network processing, OpenFlow.

1 Introduction

In-network processing (INP), which is able to extract the knowledge and information from the huge volumes of continuous data streams arriving in real-time, has attracting wide interests in both academia [1,2] and industry [3,4]. It can be used not only in scientific processing with huge volume of distributed data, like those from collection of radio telescopes, but also in enterprise, who have data streams generated from manufacturing environment sensors, fabrication units and other real-time management components. In comparison to the traditional "store-and-compute" model, INP can detect critical conditions and respond occasional events in proactive fashion.

In an INP system, data stream is processed at the data processing modules scattered in the network. The topology and processing sequence of the INP modules are determined according to the current data processing objectives. Deploying a successful INP facility faces a set of challenges.

– *Flexibility:* INP requires flexible controls over routing at waypoints in the path of data transmission, depending on the required sequence of processing. Main-stream networking equipments, such as Ethernet bridges and IP routers, are not programmable to users who need to select and order processing modules scattered in the network, on their own demand and sometimes dynamically.
– *Low-cost:* INP is proposed as a substitute of the powerful computing server. It is supposed that each module only does quite simple and light computation, in order that a module can be deployed in low-cost equipments, like

commodity PC servers. For example, some modules do arithmetic operators while some others do FFT (Fast Fourier Transform). Repeating the same module with too many copies is also a waste and therefore modules should be well multiplexed. Even within one processing task, the same module is possibly used several times.

– *Scalability:* Low-cost requirement brings a challenge on the scalability: for a certain task, its processing path may contain a large number of modules, and there might be a large number of tasks who are utilizing these modules simultaneously. Therefore, the platform need to have a (or set of) well-designed controller, efficiently configuring networking facilities to support the demands.
– *Performance:* Data processing modules should be powerful enough so that the data streams could be processed and forwarded without buffering delay in the pipeline.
– *Isolation:* Data processing modules are multiplexed among different tasks, and therefore it is necessary to avoid mutual interference among the tasks. Each task needs to have an independent path of processing, without interference to other tasks.

In this paper, we propose an INP platform architecture named OFIAS – *OpenFlow in A Slice.* OpenFlow [5] is a programmable Ethernet switch, which is designed to overcome the ossification [6] of today's Internet. With OpenFlow, one can deploy a pipeline of data processing modules, and enable or disable any modules to/from the pipeline seamlessly on demand. However, OpenFlow itself doesn't support scalable multiplexing and therefore we integrate it into the facility of slice – a slice is defined as an overlay (virtual) network among a group of virtual machines. The combination of the OpenFlow and virtualization enables multiple INP systems over a shared commodity infrastructure, which is expected to meet the above requirements from both the aspects.

In order to achieve the combination, we design a programmable environment shielding users from common complex operations that are required for deploying their INP systems. In an INP system, instead of the OpenFlow switch hardware, OpenFlow software module is applied with the OS kernel, enabling an OpenFlow-functioning virtual machine – vOFS. The processing modules reside on different physical machines and we connect them with virtual links, defined by the transferring rule installed in those vOFS, for each slice. A virtual link can span multiple hops through underlying physical network even with an on-demand bandwidth reservation.

Our major contribution of this paper is the design of OFIAS, a platform for exploiting INP systems. We integrate the existing network virtualization, GRE tunneling, on-demand bandwidth reservation and OpenFlow technique into an OFIAS platform over CoreLab [7,8], a well-deployed virtualization infrastructure.

The rest of the paper is organized as follows. We introduce the design and implementation of OFIAS in Section 3 and Section 4, respectively. Section 5 demonstrates an example of INP with OFIAS. In Section 6, we evaluate the performance of OFIAS. Finally, Section 7 concludes our work.

2 Relate Works

Network Virtualization. Network virtualization works such as VIOLIN [9], VINI [10] and Trellis [11] are based on container-based virtualization that doesn't provide the forwarding isolation in kernel. Forwarding packets in user space is significantly slower than forwarding packets in the kernel. They don't meet the performance requirement for deploying an INP platform.

OpenFlow. OpenFlow [5] networks are proposed to enable the deployment of new protocols and services. With production OpenFlow switches, a network operator can partition traffic into production and research flows so that experiments can be deployed in a production network. Although a production OpenFlow network makes efforts towards the step of INP (OpenPipes [12]), it still has some limitations. First, a production OpenFlow network has a fixed topology. A user cannot dynamically change her topology so that the user can only define the chain of data processing modules. Second, the production OpenFlow switches are still not commodity products and they are not affordable by common researchers.

Stream Computing. Existing stream computing systems IBM InfoSphere Stream [3,4] and Aurora [1,2] provide an execution platform and services for user-developed applications to handle potentially massive volumes of continuous data streams. They lack of scalability since they are built on a specific platform (e.g., expensive server cluster) so that the data processing modules cannot be distributed into the Internet.

3 INP with OFIAS

OFIAS is built on a network virtualization infrastructure. In an OFIAS platform, substrate providers offer network physical resources and partition them into isolated slices with some virtualization technologies. To achieve on-demand resource allocation, there should be a centralized management (Similar to PLC of PlanetLab [13]) to collect the computing and network resources from all the substrate networks. When the centralized management receives the requirement from a user, it will authenticate the requirement and assign network resources to the user. The user can deploy her INP system with her arbitrary virtual network topology and protocols.

An example usage of INP with OFIAS is shown in Fig. 1, where in each slice, the data processing modules and virtual OpenFlow Switches (vOFS) are chained to a topology with virtual links. The controller configures the vOFS to control the processing and transmission of data streams dynamically.

3.1 Switch

INP switches are the axes of an INP system. An INP system should have its own forwarding and routing. Except for control messages, all data forwarding should

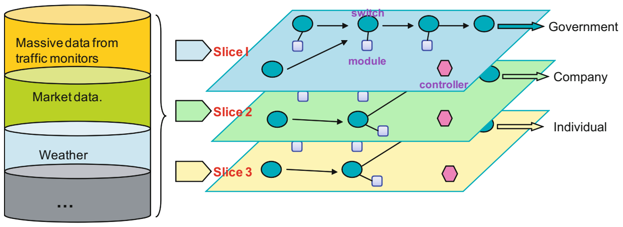


Fig. 1. Example usage of INP with OFIAS

be bounded inside a slice. Since forwarding packets in user space introduces high overhead, each vOFS should be able to define its own flow-table in kernel space. A vOFS's flow-table must be independent of other's running on the same physical node. OpenFlow supports packet forwarding based on the header fields from Layer-2 to Layer-4. The virtual network could be no longer necessarily based on IPv4. Non-IP data streams could be transmitted over an OFIAS network.

3.2 Processing Module

Each INP data module is a virtual machine. It receives data streams from its input port and sends the results out to the output port after computation. The specifications such as functionality, computation capacity, and the connected switch should be registered to the controller. The data processing module, controller and vOFS could share the same substrate node.

3.3 Controller

Controller is the core component of the INP module. As described above, the controller collects the information from the vOFS, processing modules as well as the network resources. Based on the collection and the user's objective, the controller makes the decisions on how to design the virtual topology, forwarding rules to chain the selected the vOFS and processing modules to the target INP system.

3.4 Virtual Link

OFIAS must offer the flexibility of customizing the virtual topology. There are two main challenges: (*i*) A virtual link should not only provide the connectivity between two virtual data interfaces, but also be configurable for the link properties such as bandwidth. The performance of any virtual link should ideally be isolated from other virtual links joint at the same physical links. (*ii*) To support flexible topology, a virtual link should be able to be created/withdrawn/modified between any two virtual data interfaces on demand.

4 An OFIAS Prototype on CoreLab

In this section, we describe our prototype implementation of OFIAS on CoreLab, which is shown in Fig. 2. We use Open vSwitch (OVS) [14] as a network switch for the virtualization layer to connect the various vOFS, controller and data process modules.

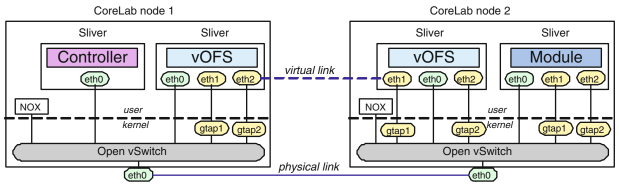


Fig. 2. Prototyping design of OFIAS platform on CoreLab, where each CoreLab node could support multiple physical interfaces, though not shown in this figure

4.1 Virtual Open Switch (vOFS)

All vOFS are implemented as virtual machines with software OpenFlow kernel modules built in. To enable customized kernel in vOFS, we adopt KVM as the virtualization technology. For each vOFS, the virtual interface `eth0` is reserved as the control interface [8]. It allows the user to access it through outside so that it should be attached to at least one global IP address. In Trellis, the guest slivers are assigned with private addresses and multiplexed to the host's IP address by applying network address translation (NAT). Since the NAT approach has well-known drawbacks in performance scalability, OFIAS shares the global IP address among the guests and the host through port-space isolation [8].

As shown in Fig. 2, all vOFS and host are bridged to a datapath (a kind of bridge) of OVS. Besides OVS, we also deploy a NOX [15], which is an open-source OpenFlow controller on each CoreLab node. The flow entries are installed from NOX to OVS. Since the virtual interface `eth0` of each vOFS is written in software, all vOFS can be configured with the same IP and MAC addresses as the host so that any Ethernet frames from outside can be received by a vOFS or host without address translation. For the ARP packets, since all interfaces `eth0` share the same IP and MAC addresses with the host, when a host receive an ARP packet, it will food to all vOFS. To isolate the packets of different vOFS, each `eth0` is assigned with a range of port numbers. The port range of each vOFS can be got from the database of PLC [16] node. As a result, the corresponding flow entries (forwarding rules) are installed after a vOFS is launched. Each `eth0` can only listen on the ports that assigned to it.

When a host has multiple physical interfaces, we create the same number of OVS datapaths. Each datapath is bridged to a physical interface. A vOFS boots with multiple virtual control interfaces bridged to different datapaths. As a result, each vOFS can share one or multiple global IP addresses with host.

4.2 Controller

The OFIAS network may have a flexible topology, the connection between the OpenFlow controller and the vOFS should be independent of topology and the signaling channel is configured as out-of-band control. In our design, the controller connects a remote vOFS through a secure signaling channel attached to the control interface eth0.

OpenFlow provides a set of APIs to enable an OpenFlow controller to control the flow of packets through OpenFlow protocol. Through the signaling channel, the vOFS sends packets to the controller and receives the command from the controller. The controller collects the observation of the OFIAS network and makes the control decision through installing flow entries to the forwarding table of vOFS. A flow entry is defined in the form <header, actions>. When an incoming packet matches a flow entry, the vOFS applies the corresponding actions. Otherwise, the packet is sent to the controller through the signaling channel by default.

4.3 Virtual Link

Virtual links are implemented with GRE [17] tunneling technologies because it has smaller encapsulation overhead than the UDP tunneling mechanism in VINI. The up-to-date Ethernet-over-GRE tunneling mechanism could give a virtual interface the appearance of a direct Layer-2 link, which could be bridged to physical interfaces or other virtual Ethernet interfaces directly. However, the existing Ethernet-over-GRE tunnel interface lacks of flexibility. The local and remote of the tunnel interface must be specified on creation. After been created, the tunnel interface's remote could not be changed so that the virtual links cannot be modified.

To enable flexible virtual link, we implement a new Ethernet-over-GRE tunnel interface, called half-open GRE-Taps (gtap for short in the Fig. 2), which only specifies the local and ikey on creation. If we want to connect two gtaps, we only need to fill the value of local and ikey of peer's gtap as the value of the remote and okey of the local gtap. A flexible virtual link can be realized through dynamically changing the fields of the gtap.

The tunnel interface connects the host's physical interface through the OVS's datapath. Since the Ethernet frames from the virtual data interface are encapsulated in GRE packets at the tunnel interface, each OFIAS network can use overlapped IP address space or even non-IP protocol.

When a host has multiple physical interfaces, a tunnel interface can bridged to different physical interfaces through different OVS datapaths. As a result, a virtual link can specify its underlying links across different ISPs.

The most previous researches are focused on the isolation between the computation resources, few has addressed the isolation between network resources. Dynamic Circuit Network (DCN) [18] can provide dedicated bandwidth for the demanding applications. With DCN supported switches, each virtual link could be attached an optical path with bandwidth on demand.

5 Demonstration of INP with OFIAS

The potential applications of INP could be complex such as network service composition, transportation (medical) monitoring and any other services. In this demonstration, we will use an arithmetic operation network as an example scenario as shown in Fig. 3, where the input is a random sequence of integers. Each data processing module is an arithmetic operator that calculates the received the data and sends out the results.

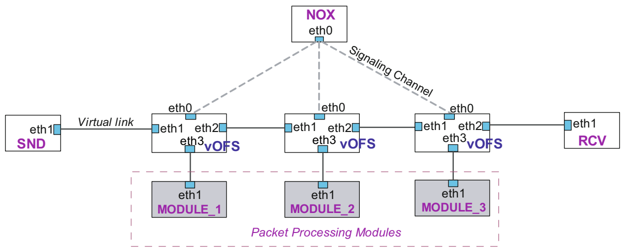


Fig. 3. Demonstration of INP with OFIAS, where a slice consists of 9 slivers

We develop a demonstration operation via a GUI controller which is implemented via GUESS [19]. The GUI controller hosts on a laptop and displays the demonstration topology of the switches, processing modules, the controller, the sender and the receiver. Each component runs a daemon at the background to receive the command from the GUI controller. The GUI controller takes the user requirements and policies and displays the arithmetic operation results. The record of the demonstration could be found at [20]. The GUI controller's functionalities include: (*i*) insert/remove any module, (*ii*) enable/disable any calculator module, and (*iii*) change the virtual topology as well as calculator sequence.

6 Performance Evaluation

In this section, we evaluate our prototype of OFIAS on CoreLab. Our evaluation focuses on two parts: (1) how the virtual link of OFIAS performs; and (2) how the vOFS of OFIAS performs comparing to previous approaches VINI and Trellis. Here, we use user-space Click as a reference to the performance of VINI and Trellis that forward packets in user space. The evaluation about scalability of CoreLab could be referred to [7]. Figure 4 shows the experimental environment, which is configured with three CoreLab nodes.

Each node is with a 2.67GHz Intel CPU and 4GB memory. Node 1 and Node 2 are with two physical interfaces, while Node 3 is with one physical interface. They connect with each other over 1Gbps Ethernet link according to the topology shown in Fig. 4. The host OS is Fedora 8 with kernel 2.6.31. Each VM is with 512M memory and its virtual interface driver is para-virtualized driver `virtio` [21].

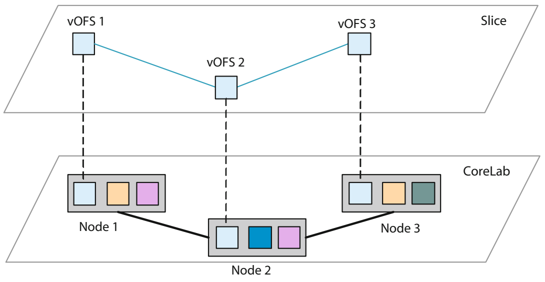


Fig. 4. Experimental environment for evaluating OFIAS

First, we check the performance of virtual link comparing to other possible communication channels. Figure 5 shows the average TCP throughput and standard deviation measure by Linux tool `iperf`. In this figure, the TCP throughput of vanilla is measured between the physical nodes 1 and 2 directly. Others are measured between the vOFS 1 and 2. The first result is that OVS can work much better than NAT (as a reference to Trellis) as a virtualization layer switching technology. The second result is that although applying full virtualization technology KVM may sacrifice the performance to some extent, the TCP throughput between two guest machines can achieve about 750 Mbps ("OFIAS control channel"), which is only 20% less than that of Vanilla. It is expected the developing back-end driver `vhost` [22] for `virtio` can reduce the overhead of KVM further. The virtual link ("OFIAS data channel") can achieve about 600 Mbps average throughput, which indicates that the GRE tunneling introduces about 15% additional overhead.

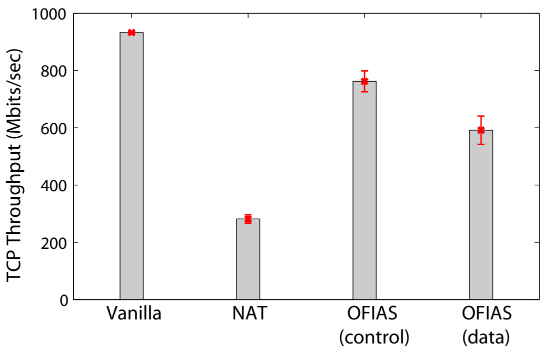
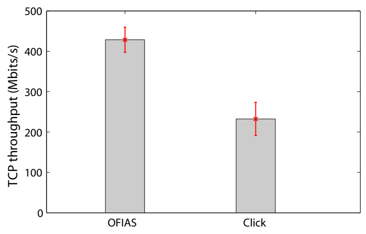
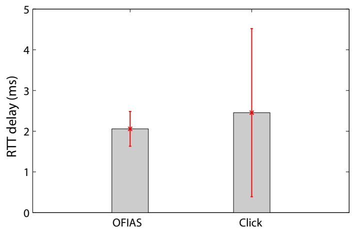


Fig. 5. OFIAS (control and data) channels versus other possible communication channels

Second, we evaluate the forwarding technologies in a slice. In the following experiments, the packets are sent between the vOFS 1 and the vOFS 3, forwarded at the vOFS 2 over the built virtual links. The virtual link between the vOFS 1 and the vOFS 2 is over the underlying link Node 1–Node 2. The virtual link between



(a) Packet forwarding capability



(b) Packet forwarding delay

Fig. 6. Packet forwarding of OFIAS versus Click

the vOFS 2 and the vOFS 3 is over the underlying link Node 2–Switch–Node 3. These two virtual links are physically disjoint and isolated from each other.

Figure 6(a) measures packet forwarding capability through the average TCP throughput and its standard deviation. OFIAS can achieve a TCP throughput around 430Mbps, which is much more than that of the user-space Click (230Mbps). As a comparison, we also list the experimental results of VINI. VINI can achieve a TCP throughput of 195Mbps on DETER and a throughput of 86.2Mbps on PlanetLab under the same topology. Both of them are even smaller than our experimental result with user-space Click. We think the reason is that the virtual links in VINI are created by UDP tunneling mechanism, which is heavier than GRE tunneling mechanism.

Figure 6(b) measures the packet forwarding delay through packet round trip time using the Linux `ping` tool. Each test run sends 10000 ICMP ping packets. The results show Click introduces more forwarding delay and jitter than OFIAS due to the overhead of packet forwarding in user-space.

In summary, the flexible virtual link and in-kernel forwarding mechanisms enable OFIAS a faster packet forwarding rate than existing network virtualization architectures such as VINI. Moreover, a programmable OFIAS network can run a much wider range of services and protocols than VINI and current Internet infrastructure.

7 Conclusion and Future Work

In this paper, we describe the design of OpenFlow In A Slice (OFIAS), which applies network virtualization to extend OpenFlow and enable multiple in-network processing (INP) overlay networks running on the same physical infrastructure. We have implemented OFIAS in CoreLab with KVM-based virtualization environments with our modified Open vSwitch and GRE tunneling technologies. Following our demonstration, researchers are expected to develop and test their INP systems on our scalable OFIAS platform.

On the other hand, INP still faces a variety of challenges in cost and efficiency. OFIAS provides not only a facility to practice INP but also a testbed for optimize it. The coordination among processing modules, network infrastructure and controller is of the future work.

References

1. The aurora project, `http://www.cs.brown.edu/research/aurora/`
2. Arvind, D., Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Nishizawa, I., Rosenstein, J., Widom, J.: Stream: The stanford stream data manager. IEEE Data Engineering Bulletin (2003)
3. Ibm infosphere streams, `http://www-01.ibm.com/software/data/infosphere/streams/`
4. Gedik, B., Andrade, H., Wu, K.-L., Yu, P.S., Doo, M.: Spade: the system s declarative stream processing engine. In: ACM SIGMOD (2008)
5. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: Openflow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. 38, 69–74 (2008)
6. Clark, D., Wroclawski, J., Sollins, K., Braden, R.: Tussle in cyberspace: defining tomorrow's internet. IEEE/ACM Transactions on Networking 13, 462–475 (2005)
7. Nakao, A., Ozaki, R., Nishida, Y.: Corelab: An emerging network testbed employing hosted virtual machine monitor. In: ROADS (2008)
8. Du, P., Chen, M., Nakao, A.: Port-Space Isolation for Multiplexing a Single IP Address through Open vSwitch. In: Magedanz, T., Gavras, A., Thanh, N.H., Chase, J.S. (eds.) TridentCom 2010. LNICST, vol. 46, pp. 113–122. Springer, Heidelberg (2011)
9. Jiang, X., Xu, D.: Violin: Virtual internetworking on overlay infrastructure. In: Parallel and Distributed Processing and Applications (2005)
10. Bavier, A., Feamster, N., Huang, M., Peterson, L., Rexford, J.: In vini veritas: realistic and controlled network experimentation. In: ACM SIGCOMM (2006)
11. Bhatia, S., Motiwala, M., Muhlbauer, W., Mundada, Y., Valancius, V., Bavier, A., Feamster, N., Peterson, L., Rexford, J.: Trellis: A platform for building flexible, fast virtual networks on commodity hardware. In: ACM ROADS (2008)
12. Gibb, G., Underhill, D., Covington, A., Yabe, T., McKeown, N.: Openpipes: Prototyping high-speed networking systems. In: ACM SIGCOMM, Demo Session (2009)
13. Planetlab, `http://www.planet-lab.org/`
14. Pfaff, B., Pettit, J., Amidon, K., Casado, M., Koponen, T., Shenker, S.: Extending networking into the virtualization layer. In: ACM HotNets (2009)
15. Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., Shenker, S.: Nox: towards an operating system for networks. SIGCOMM Computer Communication Review 38, 105–110 (2008)
16. The trutees of princeton university. myplc, `http://www.planet-lab.org/doc/myplc`
17. Farinacci, D., Li, T., Hanks, S., Meyer, D., Traina, P.: Generic Routing Encapsulation (GRE), RFC 2784 (Proposed Standard), updated by RFC 2890 (March 2000), `http://www.ietf.org/rfc/rfc2784.txt`
18. Internet2 dynamic circuit network (2008), `http://www.internet2.edu/network/dc/`
19. Adar, E.: Guess: A language and interface for graph exploration. In: ACM CHI (2006)
20. Ofias demostration, `http://plc119.nvlab.org/demo/OFIAS.mp4`
21. Russell, R.: virtio: towards a de-facto standard for virtual i/o devices. ACM SIGOPS Operating Systems Review 42, 95–103 (2008)
22. vhost-net: a kernel-level virtio-net server, `http://www.linux-kvm.org/page/VhostNet`