Designing of SDN-Assisted Bandwidth and Latency Aware Route Allocation

Pongsakorn U-chupala[1,a)] Kohei Ichikawa[1,b)] Putchong Uthayopas[2,c)] Susumu Date[3,d)]
Hirotake Abe[4,e)]

Abstract: Bandwidth and latency are two major factors that contribute the most to network application performance. Between each pair of switches in a network, there may be multiple paths connecting them. Each path has different properties because of multiple factors. Traditional shortest-path routing does not take this knowledge into consideration and may result in sub-optimal performance of applications and underutilization of network. We proposed a concept of "bandwidth and latency aware routing". The idea is that we could improve overall performance of the network by separating application into bandwidth-oriented and latency-oriented application and allocate different route for each type of application accordingly. We also proposed a design of this network system implemented using OpenFlow. Routes are calculated from monitored information using Dijkstra algorithm and its variation. To support our design, we show a use case in which our design performs better than traditional routing.

Keywords: Software-Defined Network, OpenFlow, Application-Aware Routing, Networking

1. Introduction

Nowadays, it is clear that networking plays a very important role in our society. The Internet, the biggest interconnected network ever built, has tremendous impact to humanity. As a foundation of information network, a reliable and highly efficient networking infrastructure is of utmost importance.

However, it is becoming increasingly difficult for the traditional networking infrastructure, designed decades ago, to satisfy the requirements of modern application. For example, in general network, to avoid problem caused by information loop such as broadcast storm, STP and many variations of it is implemented [1], [2]. STP is a good solution for the network back when it was originally designed. Unfortunately, STP prevents loops by deliberately ignore some links. This behavior prevents the network (specifically, layer 2 of the network according to the OSI references model) from being fully utilized to its maximum potential.

In any given network system, there usually are various kinds of application running on it concurrently. Application performance is of course affected by network conditions. However, different aspects of network conditions and properties such as bandwidth, latency or distance affect each application differently depend on the application. We define application affinity to different network properties as "requirements".

[1] Nara Institute of Science and Technology, Nara, Japan
[2] Kasetsart University, Bangkok, Thailand
[3] Osaka University, Osaka, Japan
[4] University of Tsukuba, Ibaraki, Japan
[a)] pongsakorn.uchupala.pm7@is.naist.jp
[b)] ichikawa@is.naist.jp
[c)] pu@ku.ac.th
[d)] date@cmc.osaka-u.ac.jp
[e)] habe@cs.tsukuba.ac.jp

Also, in a network, there may be more than one path (route) between any pair of entity. Each of these paths may have different property. For example, one path may be able to provide higher bandwidth while the other may have lower latency. With recent technology such as tunneling, overlay network and virtualized network, it is becoming increasingly difficult to find an appropriate path for the application, as a shortest path is not necessarily always the best path anymore.

The aim of this research is to design a network system, which aligns applications' diverse requirements with different properties of each path in the network and route accordingly. By doing so, we expect to improve the performance of each application running on a shared network. At the same time, we expect that overall utilization of the whole network would be improved as well.

Taking this approach requires deep packet inspection because the information is spread across multiple layers (according to the OSI references model). The ability to manage network flow on a per-application basis is also needed. Unfortunately, these two functionalities are not readily available with traditional network. With OpenFlow, however, it is possible to natively incorporate information from multiple layer during route calculation and allocate route specifically down to each connection of an application [3].

The rest of this paper is organized as follows. Section 2 describes previous researches on optimization for network performance improvement as well as recent technological breakthrough in networking which allows us to materialize our network optimization approach. Section 3 explains our approach for aligning applications' requirements and network properties. Section 4 describes our route allocating system design and justification

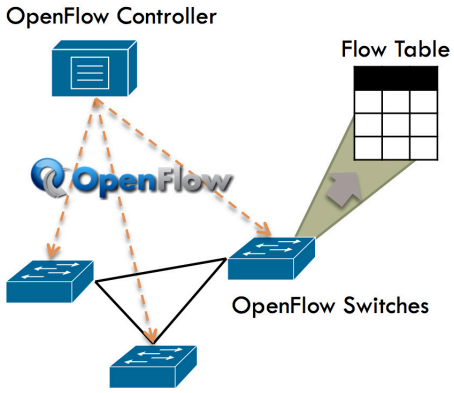

Fig. 1 Relationship between OpenFlow Controller, OpenFlow Switches and Flow Table

for each design decision. Section 5 explains theoretical benefits of our proposed routing mechanism with a use case. Section 6 discusses our planned evaluation strategy. Section 7 is the conclusion and future directions.

2. Background

Each application running on a network infrastructure may have different requirements. For example, the Secure Shell protocol requires as little latency as possible while the FTP (data connection) requires more bandwidth for faster file transfer. To achieve the best performance, the underlying network infrastructure should be able to take each application requirements into account and optimizes itself accordingly. This technique is called application-aware networking. One example of application-aware networking is Predictive and Distributed Routing Balancing (PR-DRB). This is a method for managing network performance by predicting network communication patterns and distributing network communications across multiple paths based on those prediction [4]. However, to actually realize application-aware networking, especially in general cases can be difficult.

A Software-Defined Network (SDN) is a technology that allows researchers to do networking experiment such as testing new network protocol at the actual network line speed using standard devices. OpenFlow, an implementation of SDN, provides standardized interfaces to be implemented by vendors. By supporting OpenFlow, vendors can help researchers doing experiments with devices by the vendor without having to expose the internal structures of the devices. This technology is already being deployed in many universities [3].

Unlike traditional networking paradigm, OpenFlow splits a network into data plane and control plane. Data plane is underlying mechanism of a switch that handles actual data movement from one interface to others. Control plane is the part that handles routing decisions such as which packet goes to where. In traditional network, data plane and control plane reside together in each switch. However, OpenFlow separates control plane out into centralize programmable component called OpenFlow controller. OpenFlow controller and OpenFlow switches communicate with each other through OpenFlow protocol. When any event occurs at the switch, it notifies the controller of the event. The controller then makes a decision according to how it is programmed and sends a response back to the switch to tell it how it should behave. The result is sent back in a form of new entry to flow table in the switch. This flow table will be used for future decision instead of consulting with the controller whenever anything occurs. **Figure 1** illustrates the relationship between OpenFlow controller, OpenFlow switches and flow table.

Using an SDN implementation such as OpenFlow, it is now possible to implement application-aware networking without being limited by traditional paradigm of network analysis, which relied on line-rate packet analysis [5]. With OpenFlow, it is now possible to allocate different route specifically to each connection of an application.

The technique to monitor network status is vital to providing application-aware routing. In order to dynamically align application preference to network properties, very precise near-real-time network information is required.

In the paper by Breitbart et al., bandwidth and latency were major factors for optimization. However, monitoring the current network utilization was not a simple task. Monitoring these values while minimizing the effects of the monitoring on the network was a combinatorial optimization problem that proved to be NP-hard so an approximation algorithm was used instead [6].

In a paper by Tatsunori et al., OpenFlow is used in conjunction with MPTCP to increase data transfer throughput of a network. [7] However, MPTCP cannot be used as a drop-in replacement for existing network as deploying MPTCP is a complicated task and requires modification to client's OS kernel. [8].

MiceTrap is another attempt to use OpenFlow for traffic engineering. This work focused on managing only bandwidth aspect of datacenter network. The author defined elephant flow and mice flow and categorize each flow accordingly for further management. Instead of monitoring from switches, the categorization is done by flagging each flow before it is sent out of the source rather than detecting flow type at the switch. This could be done in datacenter network, which is a closed network where operator has full control over all connected hosts. Unfortunately, this technique does not work with general network which we cannot assume full control over all hosts [9].

Overlord is a monitoring toolkit that emphasizes on preciseness of measured data by actually benchmarking link performance for real value instead of calculating from various factors statistically [10]. This trait makes Overlord a good monitoring solution for optimized routing.

In the paper by Ichikawa et al., they investigated the possibility of allocating routes specific to each connection according to network properties of each path. Their implementation geared toward improving virtual cluster performance. However, many techniques can be generalized and apply for general situations as we presented in this paper [11].

3. Approach

3.1 Bandwidth Oriented and Latency Oriented Application

It is known that bandwidth and latency are two major factors that have the most impact to performance of network application. However, not all applications are affected by those two fac-

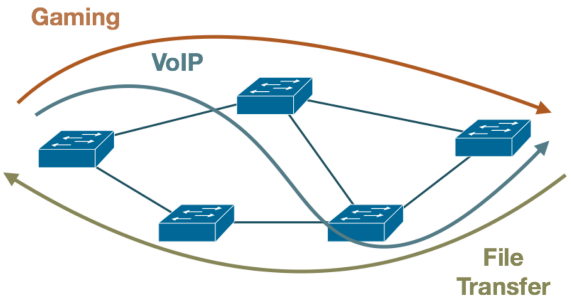

Fig. 2 Concept of Bandwidth and Latency Aware Routing

tors equally. Some application may perform better when more bandwidth is available while others are benefit more from lower latency. Therefore, we categorize application into two types, bandwidth-oriented application and latency-oriented application.

Bandwidth-oriented application is a category of application that achieves higher performance when more bandwidth is available. Most applications that involve transferring a large amount of data fall into this category. Example of application in this category includes HTTP, FTP and media (video/audio) streaming.

Latency-oriented application is a category of application that performs better over low latency network. Usually, applications that involve remote controlling or real-time communication fall into this category. For example, SSH is considered as a latency-oriented application. Most online gaming, although using its own protocol, often fall into this category as well.

3.2 Direct Measurement of Network Properties

A network consists of multiple links connecting devices together. Each of these links has different properties depend on various factors ranging from physical properties of the link such as cable type and condition, congestion caused by real limitation like link over utilization or virtual limitation introduced by congestion control mechanisms of QoS and traffic engineering techniques.

As there are many factors contributing to link properties, the properties are not static and are constantly changing depending on the current status of each link. Dynamicity of link properties is even more varying and unpredictable in more complicating network with tunnels connecting multiple network together or overlay network where each link may not actually connected together directly, but instead, goes through several networks or over the Internet where paths are constantly changing.

Ultimately, as path is a series of links connected together from one point to the other, dynamicity of link properties are also propagated to dynamicity of path properties. Moreover, as there are multiple links forming a path, effect of link property dynamicity gets amplified during propagation to path property dynamicity.

Because of this dynamicity, instead of trying to allocate and track network resources for each application, monitoring network properties by measuring current network properties directly yields a more accurate result. However, direct measurement of network properties is very demanding in term of impact to the performance of the network being measured and has to be done moderately.

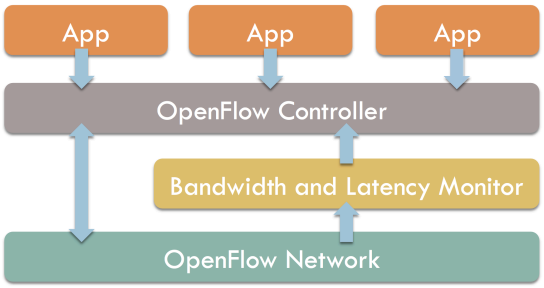

Fig. 3 Architecture of Bandwidth and Latency Aware Network

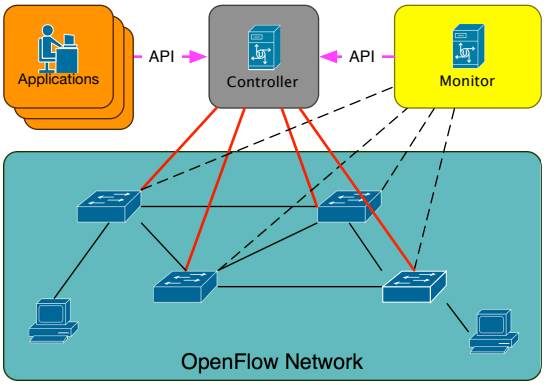

Fig. 4 Structure of Bandwidth and Latency Aware Network

3.3 Bandwidth and Latency Aware Routing

We aim to improve performance of applications running in a shared network as well as overall utilization of the network by aligning application preferences with path properties dynamically. Instead of allocating just one best path (which is usually the shortest path) between each pair of switches (source-destination pair), different paths are allocated to each application according to its preference and current network status. Even though some application may share the same source and destination, each application is allocated with the most appropriate path according to its type. Bandwidth-oriented application will be allocated with a path that provides the most bandwidth. In the same manner, latency-oriented application will be provided with a path with lowest latency between source and intended destination. Allocated path will also be updated periodically as network properties changes to best reflect current situation of the network. **Figure 2** illustrates the concept of bandwidth and latency aware routing.

4. Design

To realize bandwidth and latency aware network, we take advantage of several key technologies including OpenFlow and Overlord. This section explains our proposed design and justification for each design decision in details.

Our proposed bandwidth and latency aware network consists of 4 primary components. **Figure 3** and **Fig. 4** show the relationship and information flow direction between each component.

(1) OpenFlow Network
(2) Bandwidth and Latency Monitor
(3) Bandwidth and Latency Aware OpenFlow Controller (BW/LAT Controller)
(4) BW/LAT Controller Supported Application

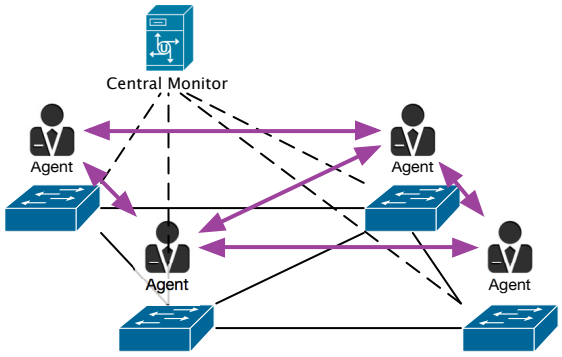

Fig. 5 Structure of Overlord

4.1 OpenFlow Network

In order to implement bandwidth and latency aware network, we need to be able to control route specifically for each application. Flow table of OpenFlow provides the granularity of control we need to achieve this.

The ability to dynamically control and reroute any connection as needed is also required. Centralize programmable controller of OpenFlow allows us to aggregate network information into a single location and allocate route for each location dynamically to best reflect current situation of the network.

4.2 Bandwidth and Latency Monitor

Our routing mechanism relied on accurate near-real-time bandwidth and latency information of each link in the network. With this requirement, Overlord was chosen as the monitoring solution.

Instead of measuring network properties passively by monitoring traffic within the network and calculate the probable condition of each link statistically, Overlord measures network properties by benchmarking each link of the network directly. This measuring method allows for very precise monitoring. However, this method has some effect on link performance during benchmarking. Overlord minimizes this side effect by intelligently decide when each link should be benchmarked and which links could be benchmarked together.

Overlord works by installing an "agent" on each networking device (switch). These agents are controlled from a central monitor controller and periodically communicate with each other to perform benchmarking.

Using Overlord monitoring framework, we track current available bandwidth and current latency between each link of the network. Current available bandwidth is measured using Netperf [12] and latency is measured using ping command.

Monitored bandwidth and latency information are forwarded to OpenFlow controller through REST API. By using the API to decouple monitoring facility from OpenFlow controller, it is open to the possibility of replacing Overlord with other monitoring solution. It is also allowed for users to supply OpenFlow controller with information from his or her own custom monitoring solution.

4.3 BW/LAT Supported Application

In order to allocate appropriate route for each application, we have each application registers its preference to the controller.

Table 1 Example of Path Preference Table

Path Identifier	Preference
(10.0.0.1, 1234, 10.0.0.2, 80)	DEFAULT
(10.0.0.1, *, 10.0.0.80, 80)	MAX BANDWIDTH
(10.0.0.2, 80, 10.0.0.2, *)	MAX BANDWIDTH
(10.0.0.1, *, 10.0.0.2, *)	MIN LATENCY
(*, *, *, *)	DEFAULT

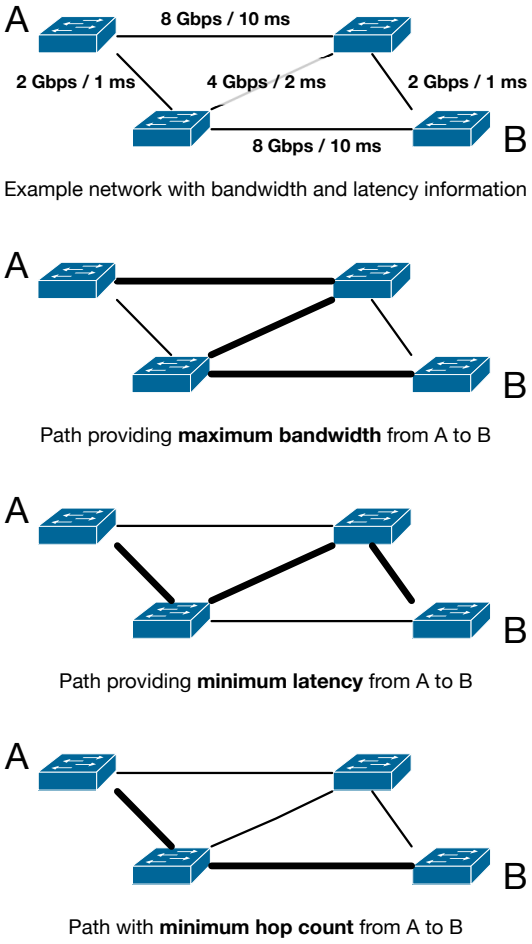

Fig. 6 Example network with corresponding calculated paths

This information is stored in "path preference table" in the controller. Each application will be able to access and modify this table with its preference through REST API provided by OpenFlow controller. Each entry in path preference table consists of *path identifier* and *preference*

Path Identifier is a quadruple of source IP address, source port, destination IP address and destination port. Each field can also be specified as a wild card to accommodate broader match and allow aggregation of multiple connection identifiers into the same rule to reduce matching and improve controller performance.

Possible options for *preference* are DEFAULT, MAX BANDWIDTH and MIN LATENCY. **Table 1** shows an example *path preference table* including the use of wild card for preference aggregation and default match.

4.4 Bandwidth and Latency Aware Controller

Bandwidth and latency aware controller is central to the design

Algorithm 1 Calculation of all-pair maximum bandwidth paths

```
for each vertex s in graph do
    push (bandwidth MAX_VALUE, vertex s) to priority queue
    while priority queue is not empty do
        (bandwidth bw, vertex v) := pop from priority queue
        if v in bandwidths[s] then
            continue /* Visited */
        else
            bandwidth[s][v] := bw
            for each (bandwidth between v and neighber bwn, neighbor n)
of v: do
                candidate_bw := MIN(bandwidths[s][n], bandwidths[s][v] +
bwn)
                if n not in seen or candidate_bw > bandwidth[n] then
                    add n to seen
                    push (candidate_bw, n) to priority queue
                    predecessors[s][n] := v
                end if
            end for
        end if
    end while
end for
```

of bandwidth and latency aware network. The controller takes current available bandwidth and latency information from monitoring facility and pre-calculate appropriate routes for each pair of switches in advance.

For each pair of switches, three routes are calculated. The three routes are maximum bandwidth path, minimum latency path and minimum hop count path. Each route is calculated using Dijkstra algorithm and its variation. **Figure 6** shows an example network with monitored bandwidth and latency of each link and each type of paths calculated for A-B switch pair.

Minimum hop count path and minimum latency path are computed by calculating shortest path with hop and latency as weight respectively. The shortest path is calculated using Dijkstra algorithm as-is. Maximum bandwidth path is, however, computed by finding a path with maximum amount of bandwidth at the bottleneck link. This path is called maximin path. **Algorithm 1** shows our modified version of Dijkstra algorithm used to compute all-pair maximum bandwidth paths.

In Ichikawa et al. work, they also calculate all-pair maximum bandwidth paths as well [11]. Their work use a modified version of Floyd-Warshall algorithm which has the complexity of $\Theta(|V|^3) \Rightarrow \Theta(n^3)$. However, we noticed that in a graph representation of bandwidth in a network, there is no negative edge. In this case, Dijkstra algorithm could also be used and will achieve better theoretical performance since executing Dijkstra algorithm repeatedly for each vertex has the complexity of $\Theta(|V|(|E| + |V|log|V|)) \Rightarrow \Theta(n^2)$.

We have investigated several OpenFlow controller frameworks including Trema, POX, Ryu and Floodlight [13], [14], [15], [16]. POX was chosen as our framework of choice because its simplicity, availability of documentation at the time of research and our familiarity with Python programming language.

With pre-calculated routes and path preference table, for each new connection, OpenFlow controller will try to match the connection with entries in path preference table. If there are multiple

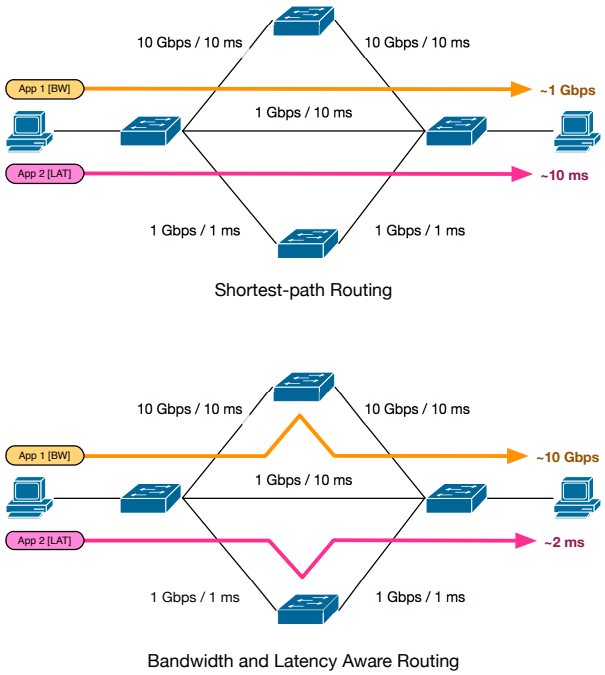

Fig. 7 Example Use Case Comparison between Shortest-path Routing and Bandwidth and Latency Aware Routing

matches, the most specific rule (the rule with the least number of wild card) is used. In case of a tie, the tie is break using order of priority of path identifier components. This is the order from top priority to least priority respectively, source IP address, destination IP address, source port and destination port. Finally, a flow entry is installed on each switch along the most appropriate path to set up the path for the particular connection according to the preference in path preference table. The allocated route for a connection is fixed for a certain configurable amount of time to allow re-routing to another path as network conditions continuously changing.

5. Use Case

Let us consider an example network shown in Fig. 7. This is an overlay network. Each link in the network is connected together not physically with real cables but virtually through multiple tunnels over the Internet. Unfortunately, various factors can affect performance of communication over the Internet in an unpredictable manner. This point makes tunnel performance vary also in an unpredictable manner. The variation of tunnel performance then, in turn, propagates to the unpredictability of performance of each link in the overlay network and finally results in the uneven performance of each link as shown in the figure.

With the traditional routing method using shortest-path, all communication between leftmost host and rightmost host in this network will take the middle path, which is the shortest. However, this is not the best path both in term of bandwidth and latency and results in poor performance of both application 1 and application 2.

With the bandwidth and latency aware routing, each application, although sharing the same source and destination, is routed separately according to its requirement. With the bandwidth and

```
from mininet.topo import Topo


class Usecase(Topo):
    def __init__(self):
        # Initialize topology
        Topo.__init__(self)

        # Add hosts and switches
        h1 = self.addHost("h1")
        h2 = self.addHost("h2")
        s1 = self.addSwitch("s1")
        s2 = self.addSwitch("s2")
        s3 = self.addSwitch("s3")
        s4 = self.addSwitch("s4")

        # Add links
        self.addLink(h1, s1)
        self.addLink(h2, s4)
        self.addLink(s1, s2)
        self.addLink(s1, s3)
        self.addLink(s1, s4)
        self.addLink(s2, s4)
        self.addLink(s3, s4)

topos = {
    "usecase": (lambda: Usecase())
}
```

Fig. 8 Mininet Script for Creating Topology in **Fig. 7**

latency information and application requirement information, application 1 is allocated with upper path while application 2 is allocated with lower path. These two paths result in higher performance in term of bandwidth and latency for both application 1 and application 2.

Also, by having each application take different routes, there is no congestion in the middle path. This point leads to better utilization of available network resources.

To confirm the eligibility of this use case, we emulated the topology of the network using Mininet, a popular network emulator among software-defined network researchers [17]. **Figure 8** shows the script we used to emulate the topology. As for bandwidth and latency information, we bypassed bandwidth and latency monitor and set the properties on each link directly through the controller's API. Our implementation of the controller correctly allocated paths for each application as in the use case.

6. Evaluation Plan

The aforementioned use case shows the benefits of bandwidth and latency aware routing in theory. However, actual experiment and evaluation are still needed to confirm the feasibility of this routing technique. This section describes our planned experiments and evaluation.

We divide the evaluation into three phases.

(1) Emulation
(2) Experiment using Virtual Environment
(3) Real World Experiment with PRAGMA-ENT Testbed

6.1 Emulation

To evaluate how bandwidth and latency aware routing performs under extreme conditions, we plan to emulate mesh topology networks with various number switches. In each network, latency and congestion will be introduced to a various percentage of random numbers of links. The traditional shortest path routing and the bandwidth and latency aware routing will be used as routing strategies for comparison. Then, bandwidth and latency of all links will be measured using Netperf and ping respectively. As for the emulator, Mininet is chosen due to its popularity among SDN researchers and ease of use [17].

6.2 Experiment using Virtual Environment

We plan to use several virtual machines running OpenVSwitch [18] to create a virtual OpenFlow network and repeat some selected cases from emulation phase.

In addition, to evaluate improvement in user experience, we plan to test the networks for HTTP file transfer speed, representing bandwidth-oriented application, and response time of communication over telnet, representing latency-oriented application.

6.3 Real World Experiment with PRAGMA-ENT Testbed

PRAGMA-ENT is a global-scale OpenFlow testbed that is still being developed [19]. When the testbed is ready, we plan to deploy bandwidth and latency aware controller to a sliced portion of this testbed and measure its performance using the similar method as previous phases.

If the controller is deemed stable enough, we wish to deploy bandwidth and latency aware routing on a production network and collect real usage performance statistics.

7. Conclusions and Future Work

In this paper, we discussed the idea of improving application performance in any network as well as overall utilization of the network by aligning application requirements with network properties. We show that this technique could be archived using OpenFlow.

We focused on two network properties, bandwidth and latency, as they are the two major factors contributing to network performance. We then categorized applications into bandwidth-oriented application and latency-oriented application using application aptitude to these two properties as criteria.

We also show that there can be several paths between any pair of switches in a network. Each of these paths has different properties due to many factors such as instability of the Internet or traffic engineering technique deployed on underlying layer of the network.

With this knowledge, we proposed a routing technique, which aligns application property orientations to paths with appropriate properties. We called this method "bandwidth and latency aware routing".

To realize bandwidth and latency aware routing, we designed a network system with four components, OpenFlow network, bandwidth and latency monitor, bandwidth and latency aware OpenFlow controller and bandwidth and latency aware controller supported application.

To support our idea, we presented a use case in which our proposed method achieved better performance than traditional shortest-path routing mechanism.

We discussed proper evaluation methods that we have planned

for this work. The evaluation is divided into three phases, emulation, experiment using virtual environment and real world experiment with PRAGMA-ENT testbed.

We are still implementing the prototype system of bandwidth and latency aware network. After the implementation is completed, we will perform the evaluation as discussed and present further progress in the subsequent report.

Acknowledgments This work was partly supported by JSPS KAKENHI Grant Number 25730075. The first author also gratefully acknowledges the support from Japanese government scholarship.

References

[1] Di Benedetto, M., Mellacheruvu, R., Finn, N. W. and Mahajan, U.: Multiple instance spanning tree protocol (2012).

[2] Fine, M., Gai, S. and McCloghrie, K.: Shared spanning tree protocol (2004).

[3] McKeown, N. and Anderson, T.: OpenFlow: enabling innovation in campus networks, *ACM SIGCOMM Computer Communication Review*, Vol. 38, No. 2, pp. 69–74 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1355746⟩ (2008).

[4] Castillo, C. N. n., Lugones, D., Franco, D., Luque, E. and Collier, M.: Predictive and Distributed Routing Balancing, an Application-Aware Approach, *Procedia Computer Science*, Vol. 18, pp. 179–188 (online), DOI: 10.1016/j.procs.2013.05.181 (2013).

[5] Koldehofe, B. and Dürr, F.: The power of software-defined networking: line-rate content-based routing using OpenFlow, *MW4NG '12 Proceedings of the 7th Workshop on Middleware for Next Generation Internet Computing*, pp. 1–6 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2405181⟩ (2012).

[6] Breitbart, Y., Garofalakis, M., Rastogi, R. and Silberschatz, A.: Efficiently monitoring bandwidth and latency in IP networks, *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, Vol. 2, Ieee, pp. 933–942 (online), DOI: 10.1109/INFCOM.2001.916285 (2001).

[7] Tatsunori, M., Hirotake, A. and Kazuhiko, K.: MPTCP with path selection mechanizm based on predicted throughput on OpenFlow-enabled environment, *IPSJ SIG Notes*, Vol. 2013, No. 7, pp. 1–5 (online), available from ⟨http://ci.nii.ac.jp/naid/110009633037/en/⟩ (2013).

[8] Raiciu, C., Paasch, C., Barre, S., Ford, A., Honda, M., Duchene, F., Bonaventure, O. and Handley, M.: How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP., *NSDI'12 Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, No. 1, pp. 29–42 (online), available from ⟨http://elf.cs.pub.ro/soa/res/lectures/mptcp-nsdi12.pdf⟩ (2012).

[9] Trestian, R.: MiceTrap: Scalable traffic engineering of datacenter mice flows using OpenFlow, *2013 IFIP/IEEE International Symposium on Integrated Network Management*, pp. 904–907 (online), available from ⟨http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6573108⟩ (2013).

[10] Overlord Network Monitoring Framework, , available from ⟨https://github.com/mekpro/overlord⟩ (accessed 10/06/14).

[11] Ichikawa, K. and Abe, H.: A network performance-aware routing for multisite virtual clusters, *19th IEEE International Conference on Networks (ICON)*, Ieee, pp. 1–5 (online), DOI: 10.1109/ICON.2013.6781935 (2013).

[12] Jones, R.: NetPerf: a network performance benchmark, *Information Networks Division, Hewlett-Packard Company*, (online), available from ⟨http://www.netperf.org/netperf/⟩ (1996).

[13] Trema: Full-Stack OpenFlow Framework in Ruby and C, , available from ⟨http://trema.github.io/trema/⟩ (accessed 10/06/14).

[14] POX Wiki, , available from ⟨https://openflow.stanford.edu/display/ONL/POX+Wiki⟩ (accessed 10/06/14).

[15] Ryu SDN Framework, , available from ⟨http://osrg.github.io/ryu/index.html⟩ (accessed 11/06/14).

[16] Project Floodlight — Open Source Software for Building Software-Defined Networks, , available from ⟨http://www.projectfloodlight.org/⟩ (accessed 10/06/14).

[17] Lantz, B., Heller, B. and McKeown, N.: A network in a laptop: rapid prototyping for software-defined networks, ... *Workshop on Hot Topics in Networks*, pp. 1–6 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1868466⟩ (2010).

[18] Pfaff, B., Pettit, J. and Koponen, T.: Extending networking into the virtualization layer, *Proc. HotNets (...*, (online), available from ⟨http://www.icsi.berkeley.edu/pubs/networking/extendingnetworking09.pdf⟩ (2009).

[19] Scientific Expeditions - PRAGMA, , available from ⟨http://www.pragma-grid.net/expeditions.php⟩ (accessed 15/06/14).