Application-Oriented Bandwidth and Latency Aware Routing with OpenFlow Network

Pongsakorn U-chupala*, Kohei Ichikawa*, Hajimu Iida*, Nawawit Kessaraphong†, Putchong Uthayopas†,
Susumu Date‡, Hirotake Abe§, Hiroaki Yamanaka¶, Eiji Kawai¶
* Nara Institute of Science and Technology, Japan Email: {pongsakorn.uchupala.pm7,ichikawa}@is.naist.jp, iida@itc.naist.jp
† Kasetsart University, Thailand Email: {g5314550091,pu}@ku.ac.th
‡ Osaka University, Japan Email: date@cmc.osaka-u.ac.jp
§ University of Tsukuba, Japan Email: habe@cs.tsukuba.ac.jp
¶ National Institute of Information and Communications Technology, Japan Email: {hyamanaka,eiji-ka}@nict.go.jp

***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 as well as evaluation results.**

I. 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.

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".

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 [1].

II. 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.

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 [1].

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.

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 [2]. With OpenFlow, it is now possible to allocate different route specifically to each connection of an application.

In the paper by Tatsunori et al., OpenFlow is used in conjunction with MPTCP to increase data transfer throughput of a network. [3] 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. [4].

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 [5].

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.

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 [6]. This trait makes Overlord a good monitoring solution for optimized routing.

III. Approach

A. 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 factors 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.

B. 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.

C. 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.

IV. Design

Our proposed bandwidth and latency aware network consists of 4 primary components. Fig.1 shows the relationship and information flow direction between each component.

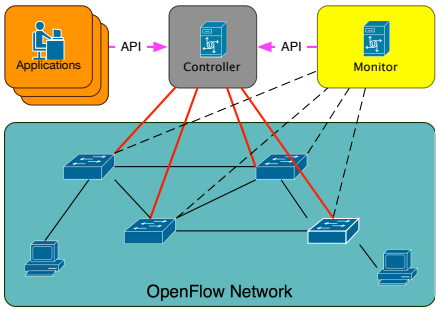

Fig. 1. Structure of Bandwidth and Latency Aware Network

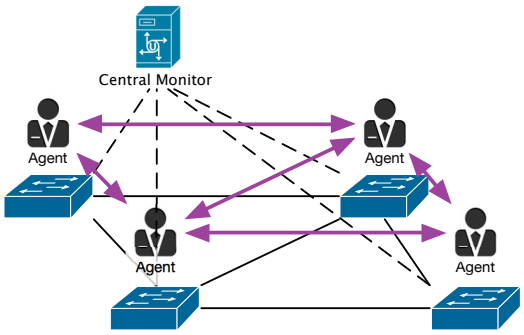

Fig. 2. Structure of Overlord

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

A. 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.

B. 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.

TABLE I
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
(10.0.0.1, *, 10.0.0.2, *)	MIN LATENCY
(*, *, *, *)	DEFAULT

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 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.

C. BW/LAT Supported Application

In order to allocate appropriate route for each application, we have each application registers its preference to the controller. 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. TABLEI shows an example *path preference table* including the use of wild card for preference aggregation and default match.

D. Bandwidth and Latency Aware Controller

Bandwidth and latency aware controller is central to the design 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.

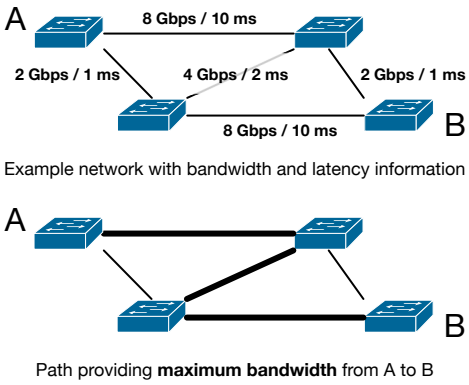

Fig. 3. Example network with corresponding calculated path

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. Fig.3 shows an example network with monitored bandwidth and latency of each link and an example path 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.

In Ichikawa et al. work, they also calculate all-pair maximum bandwidth paths as well [5]. 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. 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 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.

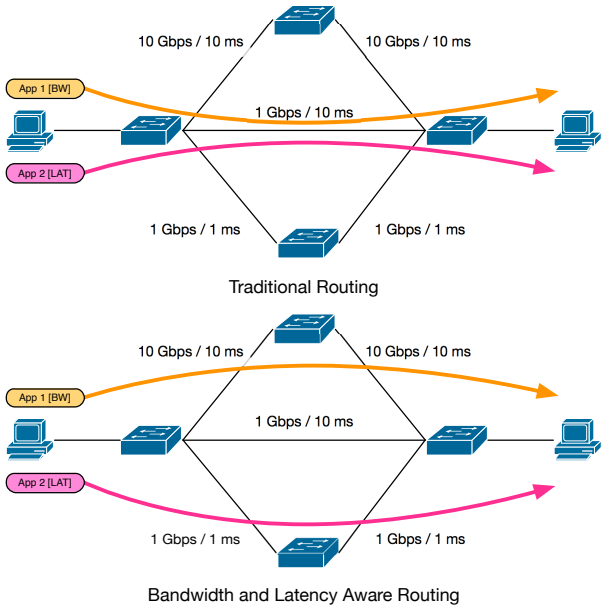

Fig. 4. Emulated Network Topology and Path Selected by Traditional Routing and Bandwidth and Latency Aware Routing

V. Evaluation

A combination of evaluation methods using emulation and experiments with virtual environment is performed to confirm the eligibility and feasibility of bandwidth and latency aware network.

A. Emulation using Mininet

The objective of this evaluation is to confirm that our implementation of the controller works properly. That is, the controller must be able to properly select the most appropriate paths when there are multiple paths with different properties to choose from according to information in path preference table.

To confirm this point, we emulate a network using Mininet, a popular network emulator among software-defined network researchers [7]. The network is manually crafted so that there are multiple paths from designated source and destination. Some path has longer distance than the others. Since we are only evaluating the controller, monitoring component was bypassed. Instead, we set properties of each link manually through the controller's API to create a situation where longer paths could provide higher bandwidth and lower latency. Fig.4 shows topology of the emulated network as well as resulting path allocated for bandwidth-oriented application and latency-oriented application. Traditional learning switch with STP is also employed as a comparison.

With the traditional routing method using shortest-path selected by STP, all communication between leftmost host and

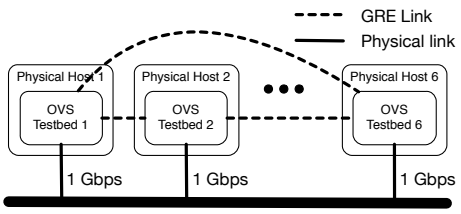

Fig. 5. Structure of the Testbed in Virtual Environment

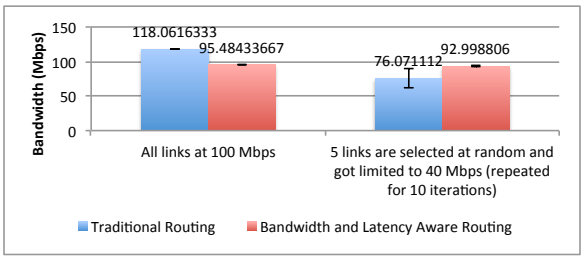

Fig. 6. Average Bandwidth of the networks from Bandwidth Experiment

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

B. Experiments using Virtual Environment

To evaluate the performance of bandwidth and latency aware routing, we create a network using our private testbed. Mesh topology is used in our evaluation to evaluate bandwidth and latency aware routing under extreme condition.

Our testbed consists of 6 virtual machine hosts. Each host has a gigabit ethernet connection that is shared among all virtual machines running on the same host. In order to force the communication to go through physical network, we deployed six virtual machines, one per host. Each virtual machine runs CentOS 6.5 with Open vSwitch 1.11 and Overlord monitoring agent.

Each host is connected to all other hosts using GRE tunnel, forming mesh topology. The tunnel is then connected to Open vSwitch instance on each host. Traffic control policy will be applied to these tunnels to simulate bandwidth and latency fluctuation during the experiment. An extra tap interface is also connected to Open vSwitch on each host with an ip address set up to expose the network to each virtual machine and enable network level connectivity between each host required by Overlord. This also allows us to evaluate the performance directly from the virtual machine. Fig.5 illustrates the structure of the testbed.

The controller and monitor is installed on another virtual machine running Ubuntu 14.04 with the latest version of POX checked out from carp branch as of July, 2014.

When no controller is available, Open vSwitch can be configured so that it falls back to behaving like a traditional learning switch. Open vSwitch also comes with STP. This function of Open vSwitch is used to represent traditional routing as a comparison to our bandwidth and latency aware routing.

Using the prepared testbed, we perform two experiments, bandwidth experiment and latency experiment.

1) Bandwidth Experiment: The purpose of this experiment is to evaluate the performance of traditional routing and our implementation of bandwidth and latency aware routing in term of bandwidth. Since the network is shared across multiple running virtual machines, in order to reduce interference from external factors, bandwidth of all links are limited to 100 Mbps. Note that the limit set by Linux kernel is not absolute and some fluctuation may occur. Netperf is then used to measure bandwidth between all-pair of switches, each in both directions. The average value of all measured bandwidth is then used to represent the overall bandwidth of the network.

To simulate bandwidth fluctuation due to path dynamicity, 5 links are selected at random. Bandwidth limit of each link is further reduced to 40 Mbps. All pair bandwidth measuring is then performed and average values are calculated again for both traditional routing and bandwidth and latency aware routing. This process is repeated for 10 times. Finally, we calculated the average results from all iterations.

For the baseline case where all links are limited at 100 Mbps, traditional routing performs better than our implementation of bandwidth and latency aware network. This is because the difference in bandwidth between each link is not big enough for the system to benefit from selecting better part. Bandwidth and latency aware routing becomes an unnecessary overhead in this case. However, we believe that our implementation is imperfect as it is implemented as a proof of concept and there are still some rooms for improvement.

In case which 5 links have bandwidth limited, however, bandwidth and latency aware routing outperform traditional routing because lower bandwidth links are avoided. With traditional routing, the result is vary as depicted by standard deviation in the figure. The reason is that for traditional routing, bandwidth vary depend on the number of links in the spanning tree that is being bandwidth-limited. Bandwidth and latency aware routing do not have this problem.

2) Latency Experiment: This experiment aim is to evaluate the performance of traditional routing and our implementation of bandwidth and latency aware routing in term of latency. Using the same testbed as the previous experiment (with no bandwidth limit). Again, we compare Open vSwitch STP

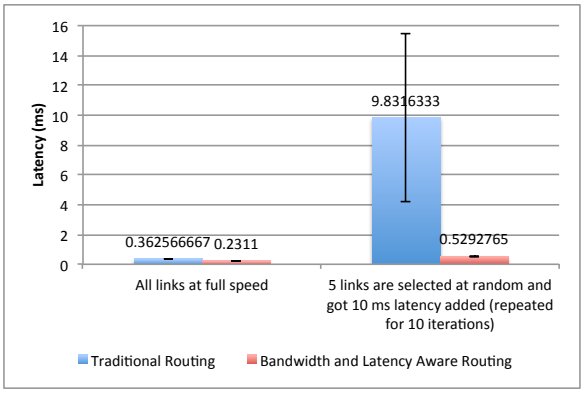

Fig. 7. Average Latency of the networks from Latency Experiment

representing traditional routing and bandwidth and latency aware routing. For each routing technique and for each pair of switches, latency is measured using the average latency obtained from pinging ten times. Then, the average value of average latency between each link is used to represent the average latency of the network using each routing technique.

At full speed, bandwidth and latency aware routing performs better than traditional routing. This is because bandwidth and latency aware routing always using shortest path where as traditional routing always routes through root node of the spanning tree, increasing packet's traveling distance. The difference is even more drastic when 5 links were randomly selected and have 10 ms latency added. Bandwidth and latency aware routing notices the change and avoids using those links. Traditional routing, however, has no knowledge of this and continue using the link.

Again, with traditional routing, the result is highly variable depends of which links are being latency-limited. The variation is even more severe as added latency in our experiment is very high resulting in high penalty for not taking the best route. Bandwidth and latency aware routing achieve very low and stable latency.

C. Future evaluation plan

PRAGMA-ENT is a global-scale OpenFlow testbed that is still being developed [8]. 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.

VI. 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.

Acknowledgment

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] N. McKeown and T. Anderson, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1355746

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

[3] K. Tatsunori, Maeda, Hirotake, Abe, Kazuhiko, "MPTCP with path selection mechanizm based on predicted throughput on OpenFlow-enabled environment," *IPSJ SIG Notes*, vol. 2013, no. 7, pp. 1–5, 2013. [Online]. Available: http://ci.nii.ac.jp/naid/110009633037/en/

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

[5] K. Ichikawa and H. Abe, "A network performance-aware routing for multisite virtual clusters," in *19th IEEE International Conference on Networks (ICON)*. Ieee, Dec. 2013, pp. 1–5. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6781935 http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6781935

[6] N. Kessaraphong, P. Uthayopas, and K. Ichikawa, "Building a Network Performance Benchmarking System Using Monitoring as a Service Infrastructure," in *The 18th International Computer Science and Engineering Conference*, 2014, pp. 2–5.

[7] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," ... *Workshop on Hot Topics in Networks*, pp. 1–6, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1868466

[8] "Scientific Expeditions - PRAGMA." [Online]. Available: http://www.pragma-grid.net/expeditions.php