Design and Implementation of OpenFlow Networks for Medical Information Systems

Yusuke Iwasaki †, Satoru Ono †, Shunsuke Saruwatari †, Takashi Watanabe †
† Graduate School of Information Science and Technology, Osaka University
‡ Graduate School of Science and Technology, Shizuoka University

***Abstract*—In current medical information networks, a problem is that the high cost for core switches is increasing to meet the demand for throughput and robustness. To solve the high cost problem, this paper proposes MediFlow, a medical information network using OpenFlow. MediFlow replaces costly high-performance core switches with a MediFlow core network consisting of multiple inexpensive fully connected OpenFlow switches. The MediFlow core network adaptively allocates communication capacity to time-varying traffic in a medical information network. Even when a link fails, MediFlow dynamically assigns a path to provide communication robustness. We implement MediFlow using commercial OpenFlow switches. The experimental evaluation shows that MediFlow improves throughput by allocating individual paths to each flow about 7 seconds after congestion occurs by converging multiple flows. Additionally, even if a link failure occurs in the path used by the flow, MediFlow re-establishes a path with a disconnection time of about 0.7 seconds.**

I. Introduction

In the medical information field, there has been a major change in the last quarter century. In previous times, computer applications were used mainly in administrative fields such as the calculation and output of medical bills. However, over the last ten years computerization has spread to the medical front. The "grand design for informatization of health and medical fields" [1], formulated by the Ministry of Health, Labor and Welfare according to the 2001 e-Japan initiative, had two goals. First, it was to deploy electronic medical records to over 60% of hospitals with 400 beds or more by FY 2006. Second, it was to deploy the medical treatment fee calculation and processing system to more than 70% of hospitals nationwide by FY 2006. In response to these targets, the penetration rate of electronic medical records in hospitals of 400 beds or more reached 70.1% as of 2015. With the introduction of the electronic medical record, almost all medical information is computerized.

In the process of computerization, the amount of content in a medical information network grows year by year. There are many terminals in outpatient booths and staff stations of hospital wards. In hospitals with more than a thousand beds, thousands of terminals may be installed. All medical practices, such as specimen examination, prescribing, X-ray photography, and physiological examination, are constantly ordered as electronic information from these terminals. The electronic information must be reliably transmitted to each departmental system. Especially at outpatient booths, the response time for these sent and received transmissions greatly affects the patient's waiting time and the doctor's business efficiency.

In such a medical information network, the increasing price of a network system has become a problem. The high price is due to the characteristics of medical information networks: time-varying traffic and redundant settings for robustness. The studies [2]–[11] using software-defined networks tackled these problems by installing Multipath TCP (MPTCP) [12] or a path-selection function into terminals. However, these approaches are not appropriate because of a constraint specific to medical information networks, which is that terminal-side software cannot be modified. The details of these problems and constraints are discussed in Section II.

This paper proposes MediFlow, a medical information network using OpenFlow. OpenFlow is a standardized protocol to access the forwarding plane of a network switch or router [13]–[15]. MediFlow is implemented by replacing the expensive core switch of a current medical information network with a MediFlow core network consisting of multiple inexpensive OpenFlow switches. MediFlow preferentially provides communication capacity to heavy traffic whose trend varies from hour to hour by allocating paths that avoid congestion for each flow. Even when a link failure occurs, MediFlow provides robustness by assigning a path to bypass the failed link. We observed that as a result of implementing MediFlow using commercial OpenFlow switches, rerouting time is about 7 seconds when congestion occurs and about 0.7 seconds when link failure occurs.

The remainder of this paper is organized as follows. Section II presents the problems and constraints of current medical information networks. In Section III, we propose MediFlow for medical information networks. In Section IV, we evaluate MediFlow, which is the proposed method, with commercial OpenFlow switches. Finally, our conclusions are summarized in Section V.

II. Medical Information Networks

A. Price increase due to variability of traffic over time

With growing demand for throughput, the variability of traffic over time, which is a characteristic specific to medical information networks, causes an increase in the price of medical information networks. In a medical information network, the traffic is bursty, and rarely is peak performance required at all times. The burstiness depends on the hospital's work being done during that time.

Fig. 1 shows the traffic collected in an actual medical information network operated at one university hospital in Japan. It shows the burstiness of the medical information network traffic. In the figure, we can see two peaks, around noon and 4 pm. The maximum peak exceeds 2 Gbps, but during off-peak time periods the traffic volume is generally less than 500 Mbps.

An example of a time-specific task is a backup of the medical information system. The national academic university hospitals in Japan started the Gemini project [16] on the occasion of the Great East Japan Earthquake in 2011. Each national university hospital needs to back up two kinds of medical information via a wide area network. The first backup is a proprietary system file used for disaster recovery. The second backup is a standardized file, to which all hospitals participating in the project can refer in the event of a disaster. Since these data are large in capacity, the backup traffic occupies the network for a long time with a high bandwidth.

In order to support such peak performance, it is currently necessary to use high-priced network equipment. In the medical information networks to date, the network administrator tries to predict the traffic-concentrated link, and the link is multiplexed using Link AggreGation (LAG) [17]. However, the multiplexed link is not always used, and the port used for the LAG is idle most of the time. Furthermore, the backup task also increases the price of network equipment. The timing of backup task execution has to be planned carefully because of the huge traffic. However, since the size of the backup source increases day by day, the increase induces an increase in the cost of re-designing the network system and task schedule.

B. Price increase due to robustness requirement

In a medical information network, although the failure rate of the network equipment is relatively low compared to that of terminals, whose numbers are rapidly increasing, the high cost for network equipment is increasing because of the demand for robustness. In recent years, various information terminals, such as the smartphone and tablet, have been used in the medical field. Since these terminals are highly portable, they are used for three-point authentication (patients, medical practices, medical staff) in almost all kinds of medical actions. Additionally, the electronic medical record system has various kinds of networked equipment, such as MRI devices and X-ray photography devices. Since the failure of a medical information system may affect human life, the information terminals, medical devices, and network equipment need high robustness.

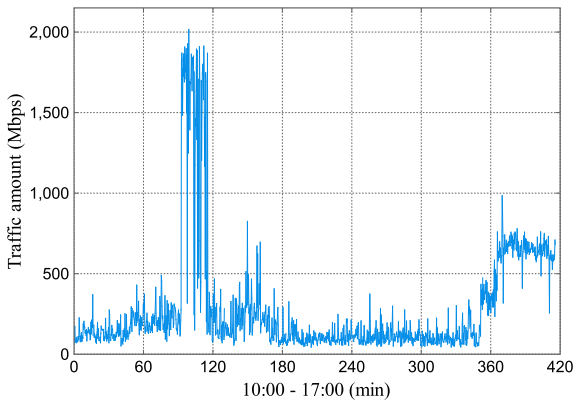

Fig. 1. Traffic trend on medical information network.

To address ever-increasing traffic and the demand for robustness, medical network systems use expensive network equipment. In order to improve robustness, network equipment at present uses redundancy of major components. Redundancy improves the overall robustness, but at the same time it involves a high cost. In particular, the core switch steadily improves the performance, and the use of special equipment such as a chassis-type core switch is the cause of the price increase.

However, in recent years, since the reliability and durability of each component has been improved, these redundant settings have not been effectively utilized. In our survey, 50% of all failures are failures of laptop computers. On the other hand, the failure rate of network equipment is relatively small, less than 1% of all failures.

An example of a redundant setting is the use of the Virtual Router Redundancy Protocol (VRRP) [18]. When we use VRRP, although the backup side uses the same components as the master side, the backup side will not be utilized unless a failure occurs on the master side. Therefore, the performance of expensive and high-performance components are not sufficiently utilized.

C. Restriction of medical information network

As described above, the problem of the medical information network is that the network equipment is expensive because of the demand due to the time-varying traffic trends and the redundant settings for robustness. As discussed, such a problem may be solved by installing MPTCP [12] or a path-selection function into the terminals in multimedia data transfer, a data center network, and so on [2]–[11].

However, in the medical information network, it is difficult to adopt this approach of changing the terminal-side system because the medical information system is not constructed by a single vendor but by multiple vendors. Many applications provided by each vendor are installed in an information terminal for browsing the electronic medical record provided by that vendor's system. Before the installation, the applications are verified not to mutually interfere. Additionally, there are many vendor-specific devices provided by each vendor. If we make uniform changes to all of these devices, much time and cost is required for installation and verification. Furthermore, some of these devices are approved as medical devices authorized by the Pharmaceutical Affairs Law. The authorized devices have a medical device registration number, and the device configuration cannot change after authorization.

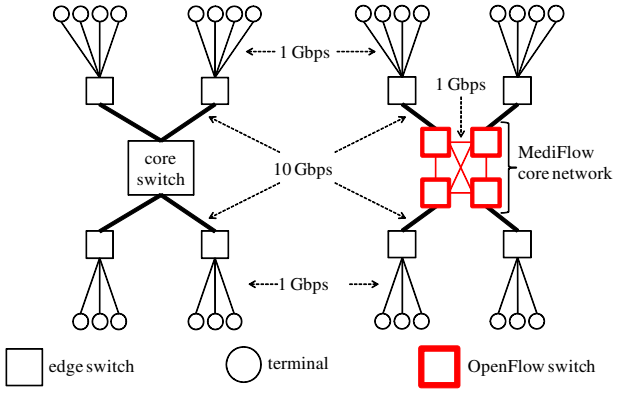

Fig. 2. Current medical information network (left) and MediFlow (right).

III. MediFlow

Based on the discussion in Section II, we designed and implemented a medical information network, "MediFlow," using OpenFlow.

A. Overview

Fig. 2 shows the basic idea of the study. MediFlow replaces the expensive core switch in the conventional medical information network (Fig. 2 left) with a MediFlow core network (Fig. 2 right). The MediFlow core network is a network constructed with multiple inexpensive OpenFlow switches that are connected to each other.

MediFlow improves the throughput of each flow by dynamically allocating flows to paths while avoiding congested links. Fig. 3 shows the operation of the MediFlow core network when congestion occurs. The OpenFlow switches S1–S4 are connected to each other via a 1-Gbps link. As shown on the left in Fig. 3, when there are three 1-Gbps flows flowing from S1 to S3, the links from S1 to S3 are congested, and the throughput becomes only about 0.33 Gbps for each flow. In MediFlow, total throughput can be improved to 1 Gbps for each flow by reallocating paths that avoid the congested link from S1 to S3 as shown on the right in Fig. 3.

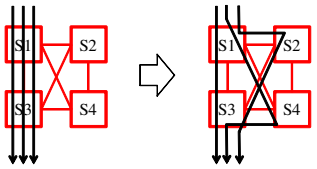

Fig. 3. Operation of MediFlow when congestion is detected.

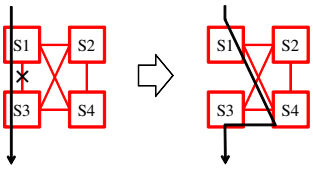

Fig. 4. Operation of MediFlow when link is lost.

TABLE I
Database in MediFlow

ofswitchTable	OpenFlow switch list
linkTable	Relation among OpenFlow switches
hostTable	Relation between terminals and OpenFlow switches
pathTable	All paths in MediFlow core network
hopTable	Hops constructing each path
flowTable	Relation between flows and paths
trafficTable	Traffic volume of each link in MediFlow core network

MediFlow can also dynamically allocate a path to avoid a link where a failure has occurred. Fig. 4 shows the operation when a failure occurs. In Fig. 4, there is one flow of 1 Gbps from S1 to S3. If the link from S1 to S3 fails, MediFlow automatically reallocates the path from S1→S3 to S1→S4→S3.

Fig. 5 shows an overview of MediFlow. Each OpenFlow switch is connected to a controller. The controller consists of three elements:

1) automatic topology detection mechanism,
2) path allocation mechanism,
3) congestion detection and avoidance mechanism.

Since all the elements can be realized using only OpenFlow functions, it is unnecessary to change the terminal side. Each element is implemented using Ryu 4.9, which is a Python-based OpenFlow framework [19].

Table I shows the tables that the controller has. MediFlow manages ofswitchTable, linkTable, hostTable, pathTable, hopTable, flowTable, and trafficTable. Each table is implemented using MySQL 14.14.

B. Automatic topology detection mechanism

The automatic topology detection mechanism is a mechanism for ascertaining the topology of the MediFlow core network. MediFlow automatically obtains the topology using the OpenFlow function, so we can easily extend the MediFlow

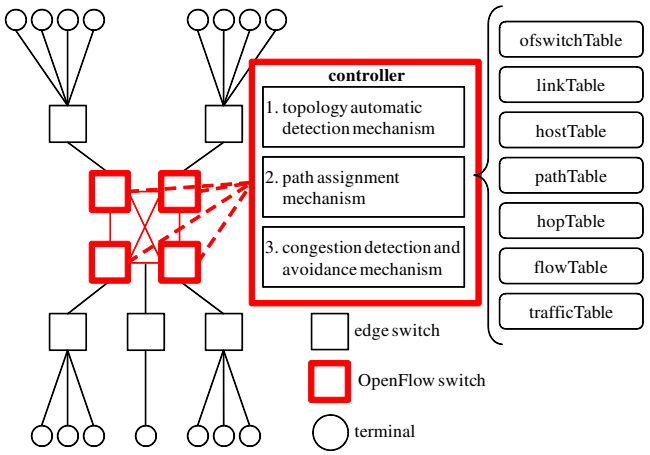

Fig. 5. Overview of MediFlow.

core network simply by connecting a new OpenFlow switch. Using the topology information acquired by the automatic topology detection mechanism, paths are allocated to avoid congestion and link failure. In order to calculate the path, it is necessary to collect the following information:

1) interconnection status among OpenFlow switches,
2) host connection status,
3) disconnection of links.

1) Interconnection status among OpenFlow switches is collected by features messages and Link Layer Discovery Protocol (LLDP) packets. When receiving a hello message issued when an OpenFlow switch is connected to the controller, MediFlow acquires information of the connected OpenFlow switch using the features message. The features message includes the MAC address and each port number of the OpenFlow switch. Based on the acquired information, MediFlow sends packet-out messages to the OpenFlow switch for sending LLDP packets from each port. When another OpenFlow switch receives the LLDP packet, a packet-in message is issued. The packet-in message contains the MAC address of the OpenFlow switch that sent the LLDP packet and the port number from which the LLDP was received.

By combining the MAC addresses and port numbers that are collected with the feature messages and LLDP packets, MediFlow apprehends how the OpenFlow switches are connected to each other. Information acquired by the features message is recorded in ofswitchTable, and information acquired by the LLDP packet is recorded in linkTable. To process messages and packets, our implementation uses Ryu's event handlers: event.EventSwitchEnter, ofp_event.EventOFPStateChange, and event.EventLinkAdd.

2) Host connection status is obtained using OpenFlow packet-in messages. In MediFlow, hosts include information terminals and medical devices. A packet-in message is issued to the controller when a packet arrives at an OpenFlow switch whose flow table does not have an entry that matches the packet. When the packet-in message arrives, the MediFlow controller acquires the identifier and port number of the OpenFlow switch and the source MAC address of the packet that corresponds to the packet-in message. If the source MAC address arrives at the controller for the first time, it is recorded in the hostTable that the terminal exists in the direction of the acquired OpenFlow switch identifier and port number. These operations are implemented in Ryu's event handler event.EventHostAdd.

3) Disconnection of links is detected by port-status messages from OpenFlow switches and by LLDP packets periodically exchanged between OpenFlow switches. A port-status message is issued when the state of the port in an OpenFlow switch changes. Transmission of LLDP packets from each OpenFlow switch is controlled from the controller. Since the flow entry corresponding to the LLDP packet is designed not to be added, a packet-in message is always sent to the controller when each OpenFlow switch receives the LLDP packet. When the controller receives notification of a link down with a port-status message or determines a link is down by not having received any LLDP packet for a certain period of time, the link down event handler is called. In our implementation, the link down event handler corresponds to Ryu's event handler event.EventLinkDelete.

Algorithm 1 packet-in event handler

```
1: Input m: packet-in message
2: d_now ← getDatapathId(m)
3: a_src ← getSourceMacAddress(m)
4: a_dst ← getDestinationMacAddress(m)
5: p ← flowTable[(a_src, a_dst)]
6: if p = NULL then
7:     d_src ← hostTable[a_src]
8:     d_dst ← hostTable[a_dst]
9:     p ← getShortestFromPathTable(d_src, d_dst)
10:    flowTable[(a_src, a_dst)] ← p
11: end if
12: q ← getPortFromHopTable(d_now, p)
13: addFlowEntry(d_now, q, m)
14: sendPacketOut(d_now, q, m)
```

C. Path allocation mechanism

The purpose of the path allocation mechanism is to determine a path for transferring packets in the MediFlow core network. It is desirable for the path calculation cost not to affect the communication performance. For example, if a path search is being performed at a time when a packet-in message arrives, the path search cost will diminish the communication performance. In order that it should not affect communication performance, the path allocation mechanism employs the following two ideas:

1) separation of paths and terminals,
2) pre-calculation of all paths among OpenFlow switches.

1) Separation of paths and terminals is realized by hostTable and flowTable. The hostTable associates the host MAC addresses with datapath IDs. The datapath ID is a mechanism in OpenFlow to identify an arbitrary OpenFlow switch. The flowTable associates flows with paths. In MediFlow, a path is represented by a list of pairs consisting of a datapath ID and a port. Because of the combining of hostTable and flowTable, MediFlow enables the path calculation cost to depend only on the number of OpenFlow switches.

2) Pre-calculation of all paths among OpenFlow switches reduces the runtime path allocation cost. The pre-calculated paths are stored in pathTable and hopTable. Using the information acquired by the automatic topology detection mechanism described in Section III-B, all the paths among OpenFlow switches are discovered by a breadth-first search. Because MediFlow assumes that the number of OpenFlow switches is at most 10, the computational complexity does not matter even if all paths are listed by the breadth-first search. Each path is recorded in pathTable together with the number of hops so that the controller can easily acquire the shortest path.

Algorithm 1 shows the algorithm of the packet-in event handler. When a packet-in message arrives at the controller, it extracts the source MAC address and the destination MAC address included in the packet-in message. The pair of the

Algorithm 2 port statistics reply event handler

1: **Input** m: port statistics reply message
2: $d \leftarrow$ getDatapathId(m)
3: $p \leftarrow$ getPortNumber(m)
4: $b \leftarrow$ getTxBytes(m)
5: $t \leftarrow$ now()
6: $(t_{\mathrm{pre}}, b_{\mathrm{pre}}) \leftarrow$ getPreviousFromTrafficTable(d, p)
7: $r \leftarrow (b - b_{\mathrm{pre}})/(t - t_{\mathrm{pre}})$
8: **if** $r > \alpha$ **then**
9: call congested_event_handler(d, p)
10: **end if**
11: insertToTrafficTable(d, p, t, b)

Algorithm 3 congested_event_handler function

1: **Input** d: datapath ID, p: port number
2: $F \leftarrow$ getAllocatedFlowSet(d, p)
3: **if** $|F| = 1$ **then**
4: **return**
5: **end if**
6: $(a_{\mathrm{src}}, a_{\mathrm{dst}}) \leftarrow$ choose a flow randomly from F
7: $Q \leftarrow$ getPathSetFromPathTable($a_{\mathrm{src}}, a_{\mathrm{dst}}$)
8: **while** $|Q| > 0$ **do**
9: $q_{\mathrm{new}} \leftarrow$ get a path randomly from Q
10: **if** q_{new} does not include (d, p) **then**
11: $q_{\mathrm{old}} \leftarrow$ flowTable[$(a_{\mathrm{src}}, a_{\mathrm{dst}})$]
12: flowTable[$(a_{\mathrm{src}}, a_{\mathrm{dst}})$] $\leftarrow q_{\mathrm{new}}$
13: add flow entries related to q_{new}
14: delete flow entries related to q_{old}
15: **return**
16: **end if**
17: $Q = Q - q$
18: **end while**

extracted source MAC address and destination MAC address is set as a flow, and a check is performed to ascertain whether or not a path has already been allocated to the flow. If the path has already been allocated, it acquires the OpenFlow switch to send next, registers it in the flow entry, and transmits a packet-out message.

If the path has not yet been allocated, the controller acquires the final destination datapath ID using the extracted destination MAC address and hostTable. Next, from the pathTable, the controller gets the shortest path from the datapath ID that issues received the packet-in message to the datapath ID acquired from hostTable. Finally, based on the acquired path, the controller acquires the OpenFlow switch to which to send next, registers it in the flow entry and flowTable, and transmits a packet-out message.

When a link failure occurs, the relevant information is deleted from ofswitchTable, linkTable, pathTable, hopTable, and flowTable. Normally, the next operation is to delete the associated flow entries from the OpenFlow switches. However, if the associated flow entry is deleted, a packet-in message is generated again, resulting in overhead. In order to reduce overhead, MediFlow deletes the flow entry related to the link failure after registering the alternative path.

D. Congestion detection and avoidance mechanism

The congestion detection and avoidance mechanism detects congestion occurring in the link in the MediFlow core network and reconfigures the path so as to avoid the congested link. Assuming that the number of flows is F and the number of OpenFlow switches is N, the variation of allocating a flow to a path is $O(N^F)$. Since the number of flows in a medical information network changes from moment to moment, a lightweight congestion detection and avoidance mechanism is preferable.

To realize lightweight congestion detection, MediFlow monitors only links among OpenFlow switches and provides a simple threshold-based congestion detection mechanism. MediFlow obtains the amount of communication per unit time of each port using an OpenFlow function and judges congestion on the basis of a threshold. First, the controller periodically issues a port statistics request to each OpenFlow switch using a multipart message. At this time, by requesting information on only the ports connecting the OpenFlow switches, the load for acquiring the traffic information is reduced. When the port statistics reply comes back from each OpenFlow switch, the port statistics reply event handler is executed.

Algorithm 2 shows the operation of the port statistics reply event handler. When a port statistics reply is returned, the controller gets the datapath ID, port number, and the number of bytes transmitted by the OpenFlow switch that sent the reply message. The number of bytes transmitted is the cumulative value up to that point. To calculate the traffic volume of the port at this moment, the event handler acquires b_{pre} and t_{pre} from trafficTable via the getPreviousTrafficRecord function: b_{pre} is the number of bytes transmitted, and t_{pre} is the timestamp acquired, at the previous time.

When b is the current number of bytes transmitted and t is the current time stamp, the current traffic load r is calculated with line 7 in Algorithm 2. When the traffic load exceeds the threshold α, it is judged that congestion has occurred, and the congested_event_handler function is called. The congested_event_handler function will be described later. The threshold α uses 90% of the wire rate in the current implementation. For example, for a wire rate of 1 Gbps at 1000BASE-TX, α is 900 Mbps. Finally, b and t are recorded in trafficTable via the insertTrafficStats function.

The congested_event_handler function is a function to reallocate paths to avoid congested links. We do not calculate the path in the function but only select a path at random from paths previously calculated as described in Section III-C. The randomness is introduced to avoid congestion by lightweight processing. If congestion occurs again after the path is selected, the congested_event_handler function is called again. The path reassignment continues until finally the appropriate path allocation is completed or the traffic causing the congestion disappears.

Algorithm 3 shows the behavior of the congested_event_handler function. The congested_event_handler function is given the information of a congested link, which is a datapath ID d and a port number p, as input arguments. The congested event handler acquires all flows that are already allocated to paths that include (d, p). If there is only one allocated flow, it is not necessary to change the path,

so the handler ends with return. If there are two or more flows, one of them is randomly selected as a path change candidate. At line 7 in Algorithm 3, the handler acquires all paths that can be allocated to the candidate flow with the getPathSetFromPathTable() function. In lines 8–17, one path is taken out from the set Q, which are the acquired paths, and if the path does not include a current congested link (d, p), the path is allocated to the flow $(a_{\mathrm{src}}, a_{\mathrm{dst}})$.

As with the case of a link failure as described in Section III-C, a mechanism to reduce path switching overhead is introduced when congestion occurs. In lines 13–14, MediFlow registers the new path as flow entries before deleting the flow entry related to the old path.

IV. Experimental Evaluation

We implemented MediFlow using commercial OpenFlow switches and evaluated the basic performance.

A. Evaluation environment

Fig. 6 shows the experimental evaluation environment. It consists of three OpenFlow switches (S1–S3), one L2 switch, one controller, one traffic generator, and two packet capture PCs (PC1, PC2). The three OpenFlow switches S1–S3 are connected to each other, and each OpenFlow switch is a Pica8 P-3297 [20], which equips 48 ports of a 1000BASE-T interface. Pica8 P-3297 supports OpenFlow versions from 1.0 to 1.4 and stores flow entries in the Ternary Content Addressable Memory (TCAM). The controller is connected to each OpenFlow switch via the L2 switch, an HP ProCurve Switch 2610-48-PWR. The controller is an HP Pavilion HPE h9 (CPU: Core i7-3770 3.4 GHz, RAM: 16 GB, HDD: 2 TB $\times$ 3), and Ubuntu 16.04 is running on the PC. PC1 and PC2, which conduct packet capture, are a Microsoft Surface Pro 3 (CPU: Core i7-4650U 2.3 GHz, RAM: 8 GB, SSD: 256 GB) and a NEC LaVie ZERO (CPU: Core i7-6500U 2.5 GHz, RAM: 8 GB, SSD: 256 GB), respectively. PC1 monitors the link from S1 to S2, and PC2 monitors the link from S3 to S2 using mirror ports. The traffic generator is a SmartBits 600B [21] manufactured by Spirent Communications Inc. The SmartBits has two line cards. Each line card has two 1000BASE-T Ethernet ports (src1, src2, dst1, dst2). SmartBits can generate an arbitrary flow rate with an arbitrary frame length and arbitrary protocol. In the evaluation, the frame length was set to 1518 B, and the protocol was set to TCP.

B. Price

A core switch used in current medical information networks is priced at several hundred thousand USD. There is a case in which a core switch of about 600,000 USD was introduced at a 600-bed hospital. On the other hand, the OpenFlow switch is several thousand USD per switch. The Pica8 P-3297 used in the actual machine evaluation for this paper was priced at about 3500 USD as of March 2017. Even if 10 OpenFlow switches are used, the price is about 35,000 USD, which shows that the cost is drastically reduced.

C. Throughput

In order to evaluate the throughput of MediFlow, we monitored the throughput trend when there were multiple flows. In this evaluation, 800 Mbps traffic from src1 to dst1 (Fig. 6) was generated immediately after the start of the experiment, and 800 Mbps traffic from src2 to dst2 were generated 6 s after the start.

Fig. 7 shows the evaluation results. The horizontal axis shows time elapsed from the start of the experiment, and the vertical axis shows traffic. The traffic trends for the links S1→S2 and S3→S2 are plotted. Immediately after the start of the experiment, only 800 Mbps traffic from src1 to dst1 can be observed on the link S1→S2. When 800 Mbps traffic from src2 to dst2 occurs 6 seconds after the start of the experiment, the traffic given to the link S1→S2 totals 1600 Mbps, which shows that congestion has occurred. The path S1→S3→S2 is allocated to the flow src2 to dst2 by the congestion detection and avoidance mechanism 14 s after the start of the experiment, and it can be seen that the link S1→S2 is 800 Mbps and the link S3→S2 is also 800 Mbps.

It takes about 7 seconds to detect congestion and avoid congestion depending on our implementation of MediFlow. Since a flow such as a backup in a medical information network continues for several tens of minutes to several hours, there is no practical problem even with a switching time of 7 seconds. The delay of about 7 seconds is caused by the querying of traffic information of each OpenFlow switch at intervals of 5 seconds. By shortening the interval of traffic

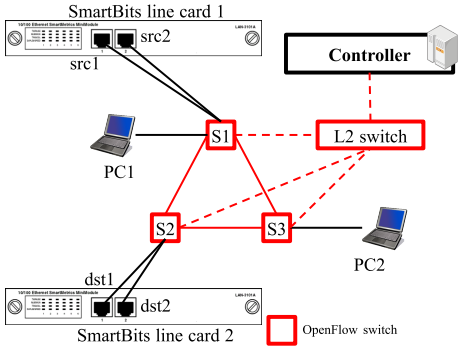

Fig. 6. Experimental evaluation settings.

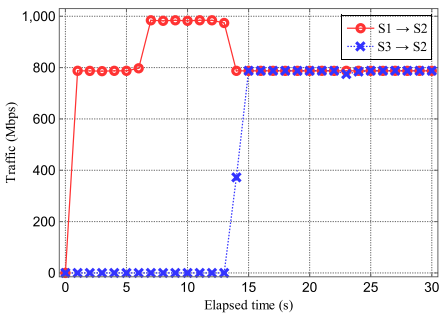

Fig. 7. Traffic trend on congestion.

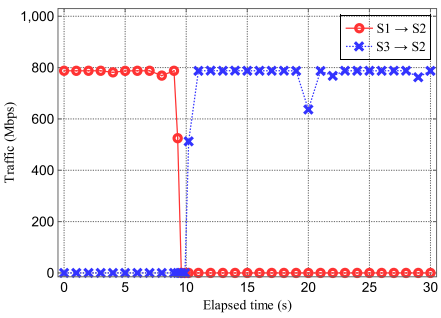

Fig. 8. Traffic trend on link lost.

information queries to the OpenFlow switch, it is possible to shorten the time required for congestion avoidance to several seconds. The lower limit of the switching time depends on the implementation of the Management Information Base (MIB). In the Pica8 P-3297 used for this implementation, the update interval of MIB is 1 second because it assumes use by SNMP(Simple Network Management Protocol).

D. Robustness

In order to evaluate the robustness of MediFlow, we monitored the throughput trend when there was a single flow and a link failure. In this evaluation, 800 Mbps traffic from src1 to dst1 (Fig. 6) was generated immediately after the start of the experiment, and the cable connected to S1→S3 was removed 9 seconds after the start.

Fig. 8 shows the evaluation results. The horizontal axis is the time elapsed from the start of the experiment, and the vertical axis is the traffic. As for the evaluation in Section IV-C, the traffic of S1→S2 link and the S3→S2 link is plotted. Immediately after the start, the path S1→S2 is allocated to the flow from src1 to dst1, so the 800 Mbps traffic on S1→S2 can be observed in the figure. When we remove the cable connecting S1 and S2 9 seconds after the start, the traffic on S1→S2 becomes zero. About 0.7 seconds after removing the cable, MediFlow reallocates the path S1→S3→S2 to the flow from src1 to dst1. Therefore, the 800 Mbps traffic of S3→S2 can be observed in the plot.

We believe that the approximate 0.7 seconds recovery time is dependent on the implementation of the OpenFlow switch. The OpenFlow switch sends a port-status message that the state of the port has changed. MediFlow adds and removes flow entries for new paths as the port-status message arrives. We estimate that about 0.7 seconds is the summation of the time required for detection of the link break, notification, registration of the flow entry, and deletion of the flow entry. Since the default value for TCP session timeout is 21 seconds in Microsoft Windows and 189 seconds in Linux, even if the link is lost for about 0.7 s, there is no practical problem.

V. Conclusion

In this paper, we have proposed MediFlow, a medical information network using OpenFlow. MediFlow solves the problem of high price by replacing the expensive core switch of the conventional medical information network with the MediFlow core network. We have implemented MediFlow using using commercially available OpenFlow switches. Experimental evaluation shows that MediFlow allocates paths to flows while avoiding congested or failed links.

References

[1] Ministry of Health, Labor and Welfare, "Informationization in the field of health care (in Japanese)." http://www.mhlw.go.jp/shingi/0112/s1226-1.html.

[2] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. H, "Improving datacenter performance and robustness with multipath TCP," *ACM SIGCOMM Computer Communication Review*, vol. 41, pp. 266–277, August 2011.

[3] Y. Cao, M. Xu, X. Fu, and E. Dong, "Explicit multipath congestion control for data center networks," in *Proceedings of the ACM 9th Conference on Emerging Networking Experiments and Technologies (ACM CoNEXT'13)*, pp. 73–84, December 2013.

[4] S. Hu, K. Chen, H. Wu, , W. Bai, C. Lan, , H. Wang, , H. Zhao, and C. Guo, "Explicit path control in commodity data centers: Design and applications," in *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (USENIX NSDI'15)*, pp. 15–28, May 2015.

[5] M. Bredel, Z. Bozakov, A. Barczyk, and H. Newman, "Flow-based load balancing in multipathed layer-2 networks using OpenFlow and multipath-TCP," in *Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking (ACM HotSDN'14)*, pp. 213–214, August 2014.

[6] H. E. Egilmez, S. Civanlar, and A. M. Tekalp, "A distributed QoS routing architecture for scalable video streaming over multi-domain OpenFlow networks," in *Proceedings of the IEEE 19th International Conference on Image Processing (IEEE ICIP'12)*, pp. 1–4, September 2012.

[7] H. E. Egilmez, B. Gorkemli, A. M. Tekalp, and S. Civanlar, "Scalable video streaming over OpenFlow networks: An optimization framework for QoS routing," in *Proceedings of the 18th IEEE International Conference on Image Processing (IEEE ICIP'11)*, pp. 1–4, September 2011.

[8] S. Laga, T. V. Cleemput, F. V. Raemdonck, F. Vanhoutte, N. Bouten, M. Claeys, and F. D. Turck, "Optimizing scalable video delivery through OpenFlow layer-based routing," in *Proceedings of the IEEE Network Operations and Management Symposium (IEEE NOMS'14)*, pp. 1–4, May 2014.

[9] M. Karl, J. Gruen, and T. Herfet, "Multimedia optimized routing in OpenFlow networks," in *Proceedings of the 19th IEEE International Conference on Networks (ICON'13)*, pp. 1–6, December 2013.

[10] C. Nakasan, K. Ichikawa, H. Iida, and P. Uthayopas, "A simple multipath OpenFlow controller using topology-based algorithm for multipath TCP," in *Proceedings of the PRAGMA Workshop on International Clouds for Data Science (PRAGMA-ICDS'15)*, pp. 1–8, October 2015.

[11] Y. Li and D. Pan, "OpenFlow based load balancing for fat-tree networks with multipath support," in *Proceedings of the IEEE 12th International Conference on Communications (IEEE ICC'13)*, pp. 1–5, June 2013.

[12] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP extensions for multipath operation with multiple addresses." IETF RFC 6824, January 2013.

[13] N. Mckeown, S. Shenker, T. Anderson, L. Peterson, J. Turner, H. Balakrishnan, and J. Rexford, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 69–74, April 2008.

[14] A. Kolasani and B. Ramamurthy, "Network innovation using OpenFlow: A survey," *IEEE Communications Surveys and Tutorials*, vol. 16, pp. 493–512, August 2014.

[15] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and OpenFlow: From concept to implementation," *IEEE Communications Surveys and Tutorials*, vol. 16, pp. 2181–2206, May 2014.

[16] Sience Information NETwork 5 Web Site, "The Gemini Project (in Japanese)." http://w4a.sinet.ad.jp/case/tokyo-med/.

[17] IEEE 802.3 Working Group, "Aggregation of multiple link segments," *IEEE Std 802.3ad-2000*, 2000.

[18] S. Knight, D. Whipple, R. Hinden, D. Mitzel, P. Hunt, P. Higginson, M. Shand, and A. Lindem, "Virtual router redundancy protocol." RFC 2338, April 1998.

[19] Ryu SDN Framework Community, "Ryu web site." https://osrg.github.io/ryu/.

[20] PICA8 Inc, "PICA8 open networking." http://www.pica8.com/.

[21] Spirent Federal Systems, "SmartBits 600b." http://www.spirentfederal.com/ip/products/smartbits/datasheets/.

[22] M. Rose and K. McCloghrie, "Structure and identification of management information for TCP/IP-based internets." IETF RFC 1155, May 1990.

[23] K. McCloghrie and M. Rose, "Management information base for network management of TCP/IP-based internets: MIB-II." IETF RFC 1158, March 1991.

[24] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A simple network management protocol (SNMP)." IETF RFC 1157, May 1990.