Adapting Existing Network Applications to ID-based Communication Networks

Yusuke Fukushima, Ved P. Kafle, Tomoji Tomuro, and Hiroaki Harai
National Institute of Information and Communications Technology, Japan

***Abstract*—Enabling mobility and multihoming over various network protocols improves connectivity and quality of user experiment in the Internet. To realize those functions, Heterogeneity Inclusion and Mobility Adaptation through Locator ID Separation (HIMALIS) has been proposed. However, the special application programing interface (API) requires tiring code modifications. To address this problem, this paper presents an application adaptation method to execute existing network applications in HIMALIS environment without modifications of the existing application source code. We also implement the proposed method and analyze trade-off between the native API and the adaptation method. The experimental results show that throughput degradation led by the adaptation method is negligible, and existing four network applications work with the adaptation method properly.**

Keywords—ID/locator split network, ID-based communication network, existing application apaptation, HIMALIS

I. INTRODUCTION

Enabling mobility and multihoming for various communication devices in the networks consisting of heterogeneous network-layer protocols (e.g. IPv4, IPv6) improves end-to-end connectivity and quality of user experience in the Internet. For example, mobile computing devices having several network interfaces, such as LTE, WiFi, and Zigbee, connects to the Internet with multiple interfaces, and users can freely select the best interface according to available bandwidth and connection stability and move from one network to another when the former one becomes suboptimal due to congestion or mobility. To realize mobility and multihoming native support by the network architecture, ID/locator split-based network architectures [1][2] have been proposed. These new architectures provide new application programing interfaces (API) to the user applications. The application developers can create new user applications exploiting the mobility and multihoming functions. However, introduction of new API requires tiring modifications of the existing application source code for adapting to the new network architecture. Therefore, in this paper, we propose an application adaptation method to execute existing network applications in the ID/locator split-based network architecture. We use the Heterogeneity Inclusion and Mobility Adaptation through Locator ID Separation (HIMALIS) [2] as an example architecture to implement and evaluate the proposed adaptation method.

HIMALIS is an ID/locator split-based network architecture supporting various network layer protocols, such as IPv4, IPv6, and future network layer protocols. In HIMALIS, an additional header field, called identity header, is inserted between transport and network header fields to isolate the transport layer functions from the network layer protocols and to protect data segments, which is referred to as ID-based communication in this paper. Besides, mobility, multihoming, and security functions are controlled with using this header. So, it is easy to work with new network layer protocols in HIMALIS. However, to use existing network applications in HIMALIS, traditional adaptation approaches [3][4][5][6][7] do not work efficiently. Packet inspection and modification-based approaches [3][4][5] are often used for addressing this problem, but the approach offering header re-processing for attaching an identity header in each packet incurs significant throughput degradation. Two open source projects [6][7] address the application adaptation in Host Identity Protocol (HIP) [1], which is also an ID/locator split-based network architecture, by using a DNS proxy module. This approach detects communication mode (HIP-based or IP-based communication) in the proxy according to the destination hostname. In case of OpenHIP [6], when the top-level domain (TLD) of the destination hostname is 'hip', the DNS proxy module starts a HIP-based session initiation process. However, since this approach supports only control plane functions, it is not useful in HIMALIS environment. In case of using custom packets in both control and data planes like HIMALIS, a lightweight runtime application adaptation method is desirable, but it is not available.

The proposed adaptation method is designed on the basis of a dynamic library interposition technique [8] that allows changing behaviors of binary applications dynamically without injection of heavy processing overhead in both control and data planes. We also describe the implementation of the adaptation method. The experimental results show that throughput degradation led by the adaptation method is negligible.

The remainder of this paper is organized as follows. Section II describes the ID-based communication by giving reference to the HIMALIS network architecture.. Design and implementation of the proposed adaptation method are described in Section III. In Section IV, performance evaluation is shown. Section V concludes this paper.

II. HIMALIS NETOWRK ARCHITECTURE

A. Architecture Overview

A HIMALIS network mainly consists of name registries, HIMALIS-capable gateways (HGs), and hosts as shown in Fig.1. HGs and name registries connect to the transit network in which they communicate with others using common IP address, called global locator, where we refer a location address as a locator in this paper. One or more HGs create(s) an access network and provide(s) network layer protocol translation between the access network and transit network. Especially, when an access network has more than two HGs, the access network has site-multihoming capability in which a host connected to the network can select a preferable HG to improve connectivity and to avoid failed communication path during data communications. When a host connects more than two access networks with multiple network interfaces, which is called host multihoming, the host freely select the best access network according to applications requirement. In HIMALIS, a hostname (e.g., host1#himalis.net) and 128-bit long host identifier are uniquely allocated to each host when hosts connect to the network at the first time. After the initial process, a host generates private and public keys and registers its hostname, host identifier, public key, and locators to the pre-authorized name registry to make the host accessible from others.

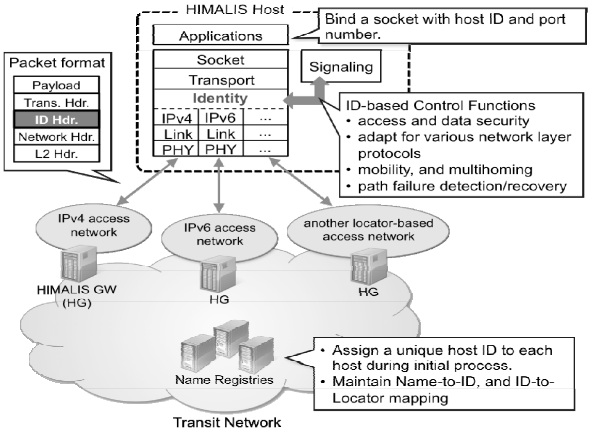

Fig. 1 A HIMALIS network architecture, and the identity layer functions.

To isolate communication from network layer protocols, an identity layer is inserted between the transport layer and network layer. In the identity layer, a mapping table of host identifier and locators is maintained to indicate current destination locator to each destination host identifier and enable access control. The related signaling module provides mobility, multihoming, and security functions with using this mapping table over several locator types. Each packet carries an identity header field consisting of source and destination host identifiers. Since the above identity layer-related functions, namely the ID-based control function, work with condition of network connections and user-defined policy, every application working on the HIMALIS network maintains own communication path from disconnection, automatically.

B. ID-based Communication

Here, we explain the overall communication process with ID-based control functions, namely ID-based communication. For simplicity of explanation, let us consider the case of communication from a host, say host 1, connected to the access network 1 to another host, say host 2, connected to the access network 2 (Fig. 2).

First of all, when host 1 connects to the access network 1, the access network allocates two types of locator; local locator and global locator. A local locator and a global locator are used for network layer packet forwarding in the access network and in the transit network, respectively. In this case, host 1 receives HG1's global locator as own global locator. Then, host 1 registers the global locator to its pre-authorized name registry. After the network connection process completes, host 1 starts to establish a session to host 2 in the identity layer to perform host authentication process. In this process, host 1 obtains the host identifier, global locator, and public key of host 2 from name registries and sends a session initiation request encoded with host 2's public key to the locator. The request is forwarded to host 2 through HG2, and host 2 decodes this request by using its private key. In the similar way, host 2 obtains host 1's information from the name registries and sends the response encoded with host 1's public key back to host 1 (see [9] for detail). After an identity layer session established, the session is maintained by exchanging keepalive packets. Finally, host 1 can send data packets to host 2 after making a TCP session to host 2.

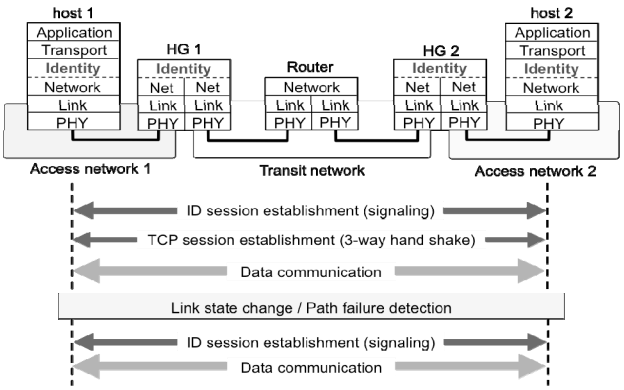

Fig. 2 A session initiation and path management in the identity layer.

In case of mobility event at host 1, host 1 sends a location update request that contains new locator to host 2 through newly connected access network. After receiving this request at host 2, host 2 sends data to the new locator (See [10] for actual handover performance). In each ID session, communication path connectivity is also maintained by sending keepalive packets from the session initiator host (host 1, in this example). When hosts exchanged several keepalive packets without data sending and receiving, hosts close the ID session for security reason. On the other hand, when the session initiator host detects a failure in the current communication path based on keepalive timeout. After detecting the failure, the host 1 sends location update request to maintain the connectivity (See [11] for detail).

It is notable that another benefit establishing the identity layer session is simple connectivity management of transport layer sessions to the same destination host because mobility support of multiple transport layer sessions is useful to improve quality of response.

C. HIMALIS Socket API

The available HIMALIS software was developed based on Ubuntu 10.04 LTS 32-bit with kernel version 2.6. To realize higher throughput in data plane and to give flexibility in the control plane, the header processing in the identity header and host identifier-to-locator mapping mechanism are implemented in kernel space, and other control modules are implemented in user space. For application developers, a HIMALIS-capable special socket API is available. This API wraps well-known BSD socket API, similar socket functions have the same programing interfaces. For example, developers can use *socket()*, *connect()*, *send()*, and so on as they developed application with BSD socket API. The main differences from BSD socket API, HIMALIS socket API provides a session initiation function to establish an identity-layer based session and new data structures to store host identifiers instead IP addresses. Here, let us make a brief explanation of the difference of application development with two APIs by using example source codes shown in Fig. 3.

```
int sock;
struct sockaddr_in serv;

int sock = socket(AF_INET, SOCK_STREAM, 0); // open an IPv4 socket

// connect to the destination host who has 192.168.100.20
srv.sin_family = AF_INET;
srv.sin_port = htons(5001);
srv.sin_addr.s_addr = inet_addr("192.168.100.20");
connect(sock, (struct sockaddr*)&serv, sizeof(serv));
```

(a) a client application with BSD socket API

```
int sock;
struct sockid_idl serv, clnt;                          /* For storing host ID */
struct idlconn_result res;      /* For storing result of idlconn_contohost() */

/* Obtain own and the server's host ID and establish an ID session */
idlconn_conntohost("server#himalis.net", 0, &res);

sock = socket(AF_IDL, SOCK_STREAM, 0);  /* Open HIMALIS socket */

/* Bind own host ID */
clnt.sidl_family = AF_IDL;
clnt.sidl_port = 0;
clnt.sidl_id = res.src_id;
bind(sock, (struct sockaddr *)&clnt, sizeof(clnt));

/* Bind the server host ID */
serv.sidl_family = AF_IDL;
serv.sidl_port = htons(5001);
serv.sidl_id = res.dst_id;

connect(sock, (struct sockaddr*)&serv, sizeof(serv));
```

(b) a corresponding client application with HIMALIS socket API

Fig. 3 An example of client programs.

In case of a TCP/IPv4 network client application with BSD socket API (Fig. 3a), *socket()* is called with *AF_INET* domain parameter to create a socket configured for TCP/IPv4 communication, and the pointer of this socket is returned to *sock* variable. Then, information to the server application including the destination address (192.168.100.20) and port number (5001) is stored into *sockaddr_in* data structure, and *connect()* registers the information into the socket and connects to the server application. In contrast, a network client application with HIMALIS socket API (Fig. 3b) firstly calls *idlconn_conntohost()* with the server hostname and *idlconn_result* data structure to establish an identity-layer session to the server. After the session establishment succeeded, own and the server host identifiers and the server port number are stored into the *idlconn_result* data structure. Then, *socket()* is called with *AF_IDL* domain parameter to create a socket configured for TCP over the identity layer communication, which is referred to as IDL socket, and the pointer of this socket is returned to *sock* variable. The own host identifier and client service port number (zero port number indicates automatic port allocation) is stored into *sockid_idl* data structure to set into the socket using *bind()*. To establish TCP connection on the identity session, *connect()* is called with the server host identifier and port number stored into *sockid_idl* data structure.

From the above, the overall programing manner of HIMALIS socket API is similar to one of BSD socket API. So, it is not difficult to make source code modifications of existing network applications for application developers.

III. Adaptation Method

A. Adaptation Mechanism

Using HIMALIS socket API is the only way to develop optimum network applications for ID-based communication. As mentioned in Section II-C, since HIMALIS API provides BSD socket API like programming interfaces, it is not difficult to develop network applications with HIMALIS socket API. However, this way is not always the best choice in some cases. For example, in case that you like to use all the Linux network applications on the identity layer functions, it is apparently hard to re-write all the source codes one-by-one. Also, in case that you have only binary program for some reason, you cannot even re-write its source code. In addition, some of network applications for the mobile devices are either bandwidth-intensive or delay-sensitive (i.e. video streaming, video call, and online games). Therefore, the processing overhead in the data plane should be minimized. However, available solutions [3][4][5][6][7] may incur larger overhead into data communication in the HIMALIS architecture because they require the packet header re-processing.

The goal of our adaptation method is making existing socket applications work on the ID-based functions without modifications of their source codes as well as achieving negligible throughput degradation. The basic idea to address the above requirements is to employ both the dynamic library interposition technique and locator translation module. The dynamic library interposition is an old technique but solid technique to enable runtime modifications of application's behavior. By using this, for example, *socket()* shown in Fig. 3a can create an IDL socket instead of a BSD socket. This is because that the interposition program can replace the

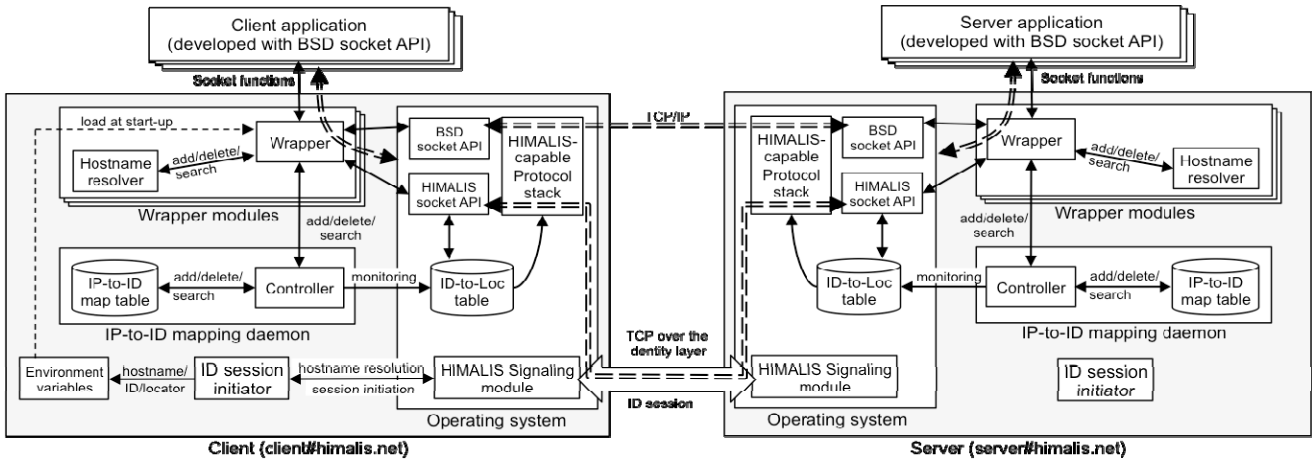

Fig. 4 A simplified block diagram of the implementation.

AF_INET domain parameter in *socket()* by *AF_IDL* domain during the execution. Notice that the interposition method may not incur processing overhead because socket functions for the data plane (e.g. *send()* and *recv()*) do not require any modifications. Using HIMALIS socket API provides locator-independent ID-based communication for network applications. However, most of existing applications do not expect and support any locator changes during data communication. The locator translation module resolves this problem by maintaining the mapping between the locators used in the existing application and the destination host IDs.

The adaptation mechanism consists of the following three modules; a *session initiator*, *dynamic library wrapper*, and *IP-to-ID mapping daemon*. The role of session initiator is to establish an identity layer session to the destination host. As explained in Section II-B, this process consists of three processes; hostname resolution, host authentication, and session initiation. Unlike session establishments in the transport layer, the identity layer session is host-to-host basis rather than application-to-application. So, after having established an identity layer session, this module does not start a session initiation process again until the session exists.

The dynamic library wrapper replaces socket functions in BSD socket API to ones in HIMALIS socket API. Since this approach allows interrupting Linux system call invoked by binary applications, it can lightly change binary application's behavior dynamically without incurring heavy process (e.g., header re-processing). To use this mechanism, *LD_PRELOAD* environment variable is used in this wrapper module as done in [8]. When a binary program is executed with this wrapper module, the wrapper module interrupts socket function calls, such as *socket()*, *connect()*, *send()*, and so on, and call the same function with another parameters again. For example, when *socket()* is called with *AF_INET* domain parameter as shown in Fig. 3a, the wrapper module interrupt this function call and call socket() with *AF_IDL* domain parameter as shown in Fig. 3b to create socket configured for TCP over the identity layer communication. After creating the socket, the wrapper module registers the file descriptor, represented by a signed integer value, pointing to the socket. Then, when other socket functions, such as *connect()* and *send()*, called with the file descriptor, the wrapper can recognize whether the called function is related to HIMALIS socket API or not according to the file descriptor.

The IP-to-ID mapping daemon provides a mapping function of an IP address and a host identifier.

In case that an application consisting of several modules, they need to share same mapping information. By implementing the mapping table management mechanism as a daemon process, each process can share the same mapping entries.

B. Implementation

To verify the function of the each module and to evaluate overall performance, we have implemented the three modules using C language. Fig. 4 describes a simplified data flow diagram of the adaptation modules. The three modules are deployed in both the client and server hosts. Let us explain how the adaptation modules work with existing network applications. At first, the IP-to-ID daemon is launched in both the client and server hosts to accept a connection from the wrapper module. To execute an application with the remaining adaptation modules, a special script, namely *idl* command, is provided for users. This command has two options as follows.

$ idl -d {destination hostname} -e {application and arguments}

When the former option is specified with the destination hostname, the session initiator module establishes an ID session to the destination host before executing the application. After the ID session establishment completes, the destination's hostname, and host identifier are recorded into environment variables. This option is usually required for the client application. If the latter option is specified with executable command and the corresponding arguments, the *idl* command executes the application with the dynamic library wrapper module. More precisely, since the wrapper module has been provided as a shared library, the *idl* command sets the wrapper module to *LD_PRELOAD* environment variable and executes the application with the corresponding arguments.

Let us consider the case of executing iperf program, which is often used for bandwidth estimation, between the client and the server hosts shown in Fig. 4. Firstly, an iperf program is executed with TCP/IPv4 server mode at the server host as follows.

```
$ idl -e iperf -s
```

After this command, the iperf program becomes ready for receiving HIMALIS packets. Secondly, an iperf program is executed with TCP/IPv4 client mode at the client host as follows.

```
$ idl -d server#himalis.net -e iperf -c 192.168.100.200
```

To ensure consistency of the original arguments used for the iperf program, the arguments contain a fake destination address. After this command, the session initiator module is launched to establish an ID session to the server host and store the destination hostname and identifier into the environment variables, and then the client application is executed with the wrapper module. The wrapper module obtains the destination host information from environment variables. Since the fake address is used during the application is running, the wrapper module register a pair of the destination host identifier and the fake address into the IP-to-ID mapping daemon. The wrapper module also caches this mapping entry for quick translation, but the registered information is referred from other wrapper module for solving inconsistency problem mentioned in Section III-a. When the client application calls an original socket function (e.g., *socket(AF_INET, ...)*), the wrapper interrupts it and call corresponding socket function (e.g., *socket(AF_IDL, ...)*) with specific parameters. In this way, the wrapper module changes the behavior of each socket function for adapting the application into the HIMALIS network environment.

C. Current Limitation and Possible Solution

The proposed adaptation method enables existing network applications working with ID-based communication. However, this method cannot provide an optimal solution and may not work in some cases. The former is apparently correct because this method does not optimize the communication sequence in the existing application because the adaptation module needs to fake IP related functions all the time. For the latter case, some applications carry the current locator value in payload to make 2-way communication (e.g., FTP active mode) and to address problems led by network address translation (e.g., SIP-based applications). The basic idea to support those applications is to exchange own locator value during session initiation process. This function can be simply implemented in the session initiation process.

IV. Evaluation

A. Experimental Setup

To understand the implementation overhead, we conducted experiments with two networks as shown in Fig. 5. In each node in the networks, HIMALIS software is installed. Each node is connected with Gigabit Ethernet cable. To obtain pure processing overhead, host 1 and host 2 are connected using IPv4 or IPv6 protocol (Fig. 5a). To evaluate throughput in the full HIMALIS environment, we also set up all the components required in HIMALIS (Fig. 5b), and two access networks are constructed different network layer protocol (IPv4, or IPv6). A domain registry (DNR) and host name registry (HNR) are required to construct name registries. Each access network requires authentication agent/registrant (AAR) and local name server (LNS) to provide access security.

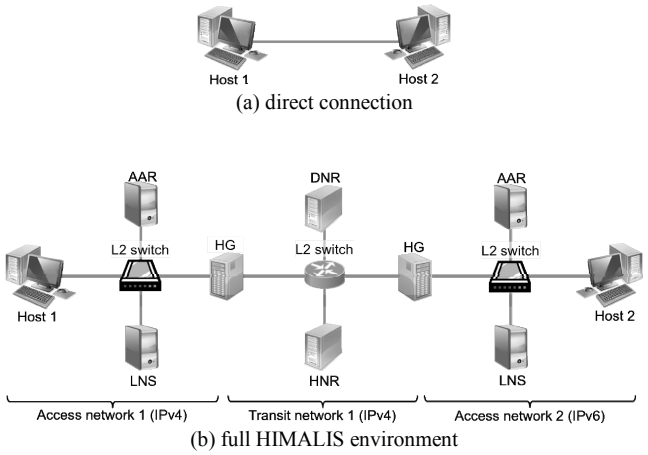

Fig. 5 Experimental network environments.

B. Tested Applications

To make verifications of the adaptation mechanism, we have tested the implemented module with different types of four binary applications; video lan client (v.1.0.6), iperf (v.2.0.4), apache2 (v.2.2.14), and wget (v.1.12). The video lan client (VLC) is selected as a representative delay sensitive video streaming application, and apache2 and wget are selected to test communication between web server and client. In this experiment, As a result, those applications above are work property with adaptation method.

C. Throughput Comparison

To understand processing overhead of the adaptation method, we have conducted TCP throughput measurements between host 1 and host 2 in two network environments (Fig.5). In this experiment, iperf or idperf program is used, where idperf is an iperf program developed using HIMALIS socket API. Since the size of ID header is 40 bytes, ideal throughput degradation of ID-based communication is around 2.7% compared with IP-based communication where MTU size is set to 1,500.

TABLE I. Throughput in case of direct connection (Mbits/sec)

	BSD Socket	HIMALIS Socket	Adaptation
IPv4	934.84	909.49 (▼2.71%)	908.48 (▼2.81%)
IPv6	921.82	894.99 (▼2.91%)	895.05 (▼2.90%)

Table I shows 1-minute average throughput in case of direct connection environment (Fig. 5a). For both IPv4 and

IPv6 configurations, throughput degradations of both HIMALIS socket and the adaptation method are close to the ideal value.

TABLE II. THROUGHPUT IN CASE OF FULL HIMALIS ENVIRONMENT (MBITS/SEC)

	HIMALIS Socket	Adaptation
IPv4-to-IPv6	733.91	733.30

Table II also shows 1-minute average throughput results when host 1 and host 2 connect to the full HIMALIS environment. This result shows that the throughput degradation by using the adaptation module is negligible.

D. Verification of Handover Process

To make verification that the adaptation mechanism works with ID-based control functions, we have also conducted a make-before-break handover experiment. In this experiment, each L2 switch (Fig.5b) connects to a wireless base station to provide wireless connection to host 1. host 2 works as an apache server, and host 1 switches two base stations during data download using *wget* command. Fig. 6 shows the received data size in case that host 1 moves from the access network 2 to the access network 1. As we can see, the adaptation method allows both original *apache* server and *wget* command programs working with handover process properly.

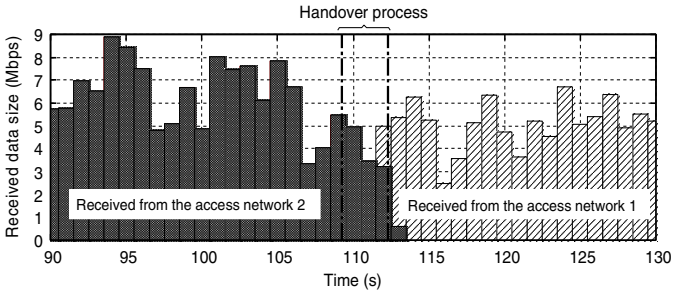

Fig. 6 Handover experiment with using an apache server and wget command.

V. CONCLUSION

In this paper, we proposed an adaptation method that enables existing network applications working with the ID-based communication networks without modification of their original source code. We verify the adaptation function for the four applications, and our experiments indicate that the throughput degradation by using the adaptation method is negligible.

REFERENCES

[1] R. Moskowitz, T. Heer, P. Jokela and T. Henderson, Host Identity Protocol Version 2 (HIPv2), RFC 7401, April 2015.

[2] V. P. Kafle, and M. Inoue, "HIMALIS: Heterogeneity Inclusion and Mobility Adaptation through Locator ID Separa- tion in New Generation Network", *IEICE Trans. Commun.*,Vol.E93-B, no.3, pp.478–489, Mar. 2010.

[3] G. Tsirtsis, and P. Srisuresh, "Network Address Translator Protocol Translator (NAT-PT)", RFC 2766, Feb. 2000.

[4] netfilter, http://www.netfilter.org

[5] H. Kitamura, "A SOCKS-based IPv6/IPv4 Gateway Mechanism", RFC 3089, Apr. 2001.

[6] OpenHIP project, http://www.openhip.org

[7] InfraHIP pro ject, http://infrahip.hiit.fi

[8] J. Salz, A. C. Snoeren, H. Balakrishnan, "TESLA: A Transparent, Extensible, Session-Layer Architecture for End-to-end Network Services", *Proc. of USENIX Symposium on Internet Technologies and Systems (USITS)*, pp.211–224, Mar. 2003.

[9] V. P. Kafle, R. Li, D. Inoue, and H. Harai, "Design and Implementation of Security for HIMALIS Architecture of Future Networks," *IEICE Trans. Inf. Sys.*, vol.E96-D, no.2, pp.226–237, 2013.

[10] V. P. Kafle, Y. Fukushima, and H. Harai, "ID/Locator Split-based distributed mobility management mechanism," *Wireless Personal Communications*, Springer, January 2014.

[11] Y. Fukushima, V. P. Kafle, T. Tomuro, and H. Harai, "Implementation of Communication Path Recovery Mechanism in a Multihomed ID/Locator-split Network," *The Sixth International Conference on Ubiquitous and Future Networks (ICUFN 2014)*, Shanghai, China, July 2014.

[12] V. P. Kafle, H. Tazaki, T. Tomuro, Y. Kobari, and H. Harai, "Prototype Implementation and Evaluation of ID/Locator-Split Protocol Stack", *IEICE Tech. Rep.*, vol.112, no.27, NS2012-26, pp.53–58, May 2012.