Local Tree Hunting: Finding Closest Contents from In-Network Cache

Hiroshi SHIMIZU†, Member, Hitoshi ASAEDA†, Senior Member, Masahiro JIBIKI†, and Nozomu NISHINAGA†, Members

SUMMARY How to retrieve the closest content from an in-network cache is one of the most important issues in Information-Centric Networking (ICN). This paper proposes a novel content discovery scheme called Local Tree Hunting (LTH). By adding branch-cast functionality to a local tree for content requests to a Content-Centric Network (CCN) response node, the discovery area for caching nodes expands. Since the location of such a branch-casting node moves closer to the request node when the content is more widely cached, the discovery range, i.e. the branch size of the local tree, becomes smaller. Thus, the discovery area is autonomously adjusted depending on the content dissemination. With this feature, LTH is able to find the “almost true closest” caching node without checking all the caching nodes in the in-network cache. The performance analysis employed in Zipf’s law content distribution model and which uses the Least Recently Used eviction rule shows the superiority of LTH with respect to identifying the almost exact closest cache.

key words: ICN/CCN, in-network caching, name-based routing, content dissemination, local tree hunting, branch-cast

1. Introduction

In the Future Internet, various scopes of information handling, such as computing or storage facility, are expected to be implemented into networking resources as well as information transfer. The Information Centric Network (ICN), which is intended to be an advance beyond today’s IP networks [1]–[5], is a method that realizes this. The ICN has a special feature of in-network caching whereby transferred content is cached at intermediate nodes. Therefore, the main scope of the network shifts from how to connect the client to the content server (or publisher) to how to get the content in many caching nodes. In the conventional network, a Domain Name Server (DNS) resolves the IP address of the server from the content name (a URL address, for example). However, it is not practical for a DNS to manage all the locations of the cached contents in the network. Hence, the network itself should have a content hunting or discovery function. To find the desired content, the requester has only to input the content name instead of the destination address. The network then guides the request to the target content. In such a situation, the closest cached content to the request node should be found in order to reduce the response time and save network resources. Hence, in the ICN, it is important to find the closest caching node, a similar problem to determining the shortest path in the Internet.

The Content-Centric Network (CCN) [2] is a prospective ICN architecture. In CCN, the content name is shown with a prefix-based structure in order to achieve name-based routing [5]. Hence, the content request travels through a name-based path. Since this has some disadvantages (described later), some schemes have been developed to overcome them, including flooding requests [6], [7] or advertising caching information throughout neighborhoods [8]. Furthermore, in order to find a caching node efficiently, the breadcrumb scheme has been studied using content transfer trails [9]. However, these studies only evaluated the improvement of the CCN scheme; they are insufficient with respect to the ideal situation in which a user grabs contents from the true closest node.

In this paper, we propose a cached content hunting scheme named Local Tree Hunting (LTH) in which the CCN response node is offered an additional “branch-casting” function for content requests on the caching nodes of a local tree. The main scope of LTH is to find the almost true closest caching node, whose hop-count from the request node is nearly the same as that of the true closest node. This paper improves on our primary design of LTH [10] in terms of the definition of the branch-casting node and a deeper analysis of content eviction.

In Sect. 2, the conventional related schemes are summarized. In Sects. 3 and 4, the proposed scheme and its operational architecture using a location ID for shortest path downloading are described. In Sect. 5, it is shown by simulation that the LTH(S) scheme offers almost the same results as that of the true closest case. Section 6 concludes our work.

2. Related Work

In the CCN scheme [2], the content publisher is located at the root of a name-based tree and name-based routing is used to follow a single name-based path from the origin to any node. The content is cached in the intermediate nodes on the path. The caching node that receives the content request “Interest” first responds as it is the closest node to the request node. However, this is not always the true closest node. Because the hunting area is limited to the name base path and, as a result, only one node responds to the request, the true closest node might not be found.
In the scheme outlined in [6], the request node floods the request. Flooding is repeated until the request reaches a caching node. This scheme can thus identify the true closest node. However, flooding is an encyclopedic process and involves a node that has no caching history related to targeted content. In addition, issues inherent in flooding such as the potential for looping must be considered. Moreover, because message flooding is always expensive, especially in a large network, this study recommends a flooding step restriction.

By contrast, schemes examined in [7]–[9] are studied as operations that complement the CCN scheme. An intermediate node scans neighboring nodes and this scanning is repeated in the same manner as flooding in the scheme described in [7]. Meanwhile, an intermediate node copies and forwards the request through the most recent content trails indicated by the breadcrumb in the scheme in [9]. In the scheme described in [8], neighboring nodes are notified of the event of content caching. Through this notification, an intermediate node shifts the request to the caching neighbor node for short-cutting purposes.

These schemes are compared with the proposed one in Sect. 5.

3. LTH (Local Tree Hunting) Scheme

The proposed LTH scheme is intended to approach the performance of the true closest case in such an environment without having to hunt through all of the caching nodes in order to hunt efficiently, the scheme offers the following features. If the desired content has been disseminated to only a few nodes (or the content dissemination has just started), the number of caching nodes will be small, and this means the discovery area has to be wider in order to find a closer node. On the other hand, if the content has been widely disseminated, a nearby caching node can be found even if the area searched is small. Hence, to make the hunt efficient, the size of the discovery area should vary in accordance with the degree to which the content has been disseminated.

We now describe the basic operation of the proposed scheme handling. The request node sends a request corresponding to an Interest packet in the CCN, as shown in Fig. 1 (a). The first node that receives the request, including the content name on the name based path of the requested content, is called the “Fork node,” and it responds to the request. This is similar to the CCN response operation. An additional aspect of the LTH scheme is that the Fork node casts the request to the branches that have a history of delivering the content. This operation is called “branch-cast.” The caching node on the branch, such as nodes B or C, respond to the request through the shortest path to the request node. If the caching node exists on the response path, it also responds like node D. The request node selects the closest response node by checking the hop-count of the received response signal. The response signal paths belong to the tree rooted at request node R.

In this way, while only the Fork node responds in CCN, the crossing points between the tree whose root is the Fork node and the tree whose root is the request node are searched in LTH. Thus, LTH can find a closer caching node by adding the branch-cast functionality to the CCN response node, i.e., the Fork node. On the other hand, when there are only a few crossing point nodes, such as in a star-hub topology, the LTH benefit is small.

Another characteristic is autonomous discovery area tuning. If the content has not been disseminated widely, because the Fork node is far from the request node (i.e., the branch-cast size of the hunting tree is large, as shown in Fig. 1 (b)) the hunt is performed over many caching nodes. If the content is disseminated more widely, the Fork node shifts closer to the request node, as shown in Fig. 1 (c). The LTH scheme thus hunts efficiently by utilizing the inherent feature of an on-route caching operation wherein the branch size of the Fork node-rooted tree automatically reduces as content is disseminated. With this feature, an almost true closest caching node can be discovered without checking all the caching nodes.

The operational characteristics of the proposed LTH and CCN schemes are compared in detail below.

Similar characteristics:
1) A request node sends the content request through the name based path to the origin.
2) An intermediate node with no requested content forwards it through the name based path.
3) The download content is cached in all of the intermediate nodes; that is, an on-route caching scheme is used.

Points of difference:
1) Each node records a download or forwarding history indicating the outgoing interface for each requested content.
2) The caching node that receives the content request first, called the Fork node, forwards it to the outgoing interface(s) indicated in the history record. This operation, called branch-cast, is repeated until the intermediate node has no record of the history.
3) When the request node is a neighbor (one hop away), branch-cast is not activated because the Fork node is the closest.
4) The caching nodes, including the Fork node, that receive the request send response messages to the request node through the shortest path.
5) The caching node on the shortest path compared to the other response nodes issues a new response message instead of the received message.
6) If the outgoing shortest path for the response message is the same as the interface of the incoming content request, caching nodes other than the fork node do not respond.

Next, we discuss the case when the cached content is evicted, as illustrated in Fig. 2.

Simple expansion LTH(S) and modified expansion LTH(M) are considered. The difference is the location of the fork node that activates the branch-cast. While in LTH(S), the fork node that caches the content activates the branch-cast similarly to original LTH, in LTH(M), the fork node that has the caching history may activate it even if it does not cache. The first caching node which receives the request responds as default response node. The LTH(S) discovery area is wider than LTH(M) in general because the fork node is located closer to the origin in LTH(M).

In addition, an optional operation of LTH is to act as a proactive breadcrumb or content broadcast[8]. This is called “LTH+BC.” Figure 3 illustrates the operation. When caching the content, node X notifies its neighboring nodes of this activity using the proactive breadcrumb. The request node or intermediate node such as node A, shifts the request forwarding based on the received breadcrumb in order to produce a shortcut. When caching, node X initiates the branch-cast operation as a fork node. In LTH(S)+BC, when the content in node X has been evicted, the request is returned to node A and the breadcrumb in node A is deleted. The returned request is forwarded through the name base path to caching node C since node B also does not cache. When the request reaches caching node C, the branch-cast operation begins. Node C corresponds to node F in Fig. 2 (a).

By contrast, in LTH(M)+BC, Node X starts the branch-cast operation as a fork node even if it has been evicted. In this case, the request is also returned to the name base path. Then, node C responds to the request without activating the branch-cast operation as a default response node.

4. Protocol Design

Referring to Fig. 4 (a), the LTH protocol is composed of four phases: content request, response, acknowledgment, and content download. The node numbers correspond to those of Fig. 1 (a). Figure 4 (b) shows the protocol header. The phase is specified by the Type field. Each node is allocated a unique node ID that indicates its location. When a content request occurs, a content label that is temporarily defined and locally unique to each node is generated. The original content name is mapped using the content label and node ID. Although schemes using a labeled content name have already been studied in ICN/SDN [12], the content label can be issued in a distributed manner in this scheme such that centralized management is not always needed.
1) Content Request: When the content request sent from the request node reaches the fork node, the first caching node that receives it is forwarded to the other caching nodes by performing the branch-cast operation on a local tree.
2) Response: A caching node receiving the request responds. The response message is routed using the request node ID as the destination address for the shortest path transfer to the request node while the forwarding route of the content request is based on the original content name.
3) Acknowledgment: The request node checks the hop-count in the response message and sends the acknowledgment through the interface that received the smallest hop-count response message. The content label and request node ID pairs recorded in the intermediate nodes are used for routing the acknowledgment message to the closest response node.
4) Content Download: The caching node that receives the acknowledgment message first becomes the download node. The request node ID is used for the shortest path download routing while the content label and request node ID pair is used for routing the backward path. The content label is only effective until the content download is complete. After completing the content download, the intermediate nodes become the caching nodes. Regarding the request node ID, the IPv6 address would be considered as a candidate for the
unique node ID.

5. Performance Analysis

The four schemes of LTH(S), LTH(M), LTH(S)+BC, and LTH(M)+BC previously mentioned are analyzed in this study. Figure 5 shows the 48-node network model that was used as a reference model for future photonic network in Japan [12]. Two nodes were located in Tokyo and 46 nodes were located in other prefectures. Figure 5 also shows the simulation parameters of mean request frequency, content catalog size and buffer size of each node. Regarding the origin nodes, we considered two location cases, in which nodes S1 and S2 having the smallest and largest mean hop-counts to other nodes, respectively.

Content eviction was based on the Least Recently Used (LRU) rule. In addition, we employed Zipf’s law model [13] for the distribution of requester populations for content ranking. The request probability \( p(i) \) of the \( i \)-th most popular content is given by the following equation.

\[
p(i) = \frac{1}{i} \sum_{k=1}^{N} \frac{1}{k}
\]

where \( N \) is the number of content catalogs.

The accumulated probability of the distribution in the case in which \( N=12,800 \) is shown in Fig. 6. The graph shows the accumulated value is nearly proportional not to the the number of content catalogs but to its logarithm.

We analyzed the following metrics.

a) Mean content hop-count: This is the content download hop-count between the request node and the selected caching node among response nodes. It is the target for finding the almost true closest caching node necessary to reduce the hop-count.

b) Relative ratio of hop-count to true closest case: This is obtained by normalization of the content hop-count with that of flooding scheme that finds the true closest node. It is used to indicate the proximity of the true closest case.

c) Mean hunting hop-count: This is the total hop-counts for content requests that are travelling for content hunting, including the proactive breadcrumb operation described previously. This is used as a cost metric to obtain the small content hop-count.

When a request node has already cached the requested content, no hunting operation occurs. At that time, both hop-count metrics are set to zero.

5.1 Comparison Models

The comparison models using the schemes mentioned in Sect. 2 are illustrated in Fig. 7.

a) CCN scheme [2]: Request node R sends the request that includes the content ID to the name base path. Caching node C, which first receives the request on the name base path, responds and transmits the content to the request node. Node C is called the “default response node.” The hunting hop-count is equal to the content hop-count. The hunting area of the caching nodes is limited to the name base path. In other words, the response node is only the default response node.

b) CCN scheme + Proactive Breadcrumb [8]: Node A shifts the request to node X based on the proactive breadcrumb, as in the operation shown in Fig. 3. When the content is evicted in node X, the request is returned to node A. The hunting hop-count includes the proactive breadcrumb operation overhead such as in LTH+BC.

c) Scanning [7]: In addition to the CCN-like default response by node C, intermediate nodes A and B scan their neighbors by the requests in a flooding manner. When a node scans N nodes, it is counted as N hops as hunting hop-counts.
count. The scan operation stops when the request reaches a caching node. The difference from the flooding scheme described in the following paragraph is that the request node does not participate in the flooding.

d) Multiple breadcrumbs: We examined the scheme that uses a reactive breadcrumb [9] generated by the forwarding history. The multiple intermediate nodes A and B forward the copied request to the most recent trails. In addition, the request is transferred to the default response node C in the same manner as in the CCN scheme. The breadcrumb guides the request until it encounters a caching node. When an unevicted caching node is not encountered, the request disappears at the final breadcrumb node on the trail.
e) Flooding [6]: Request node R floods the request in all directions. The flooding is repeated until a caching node is found. If the node floods in N directions, it is counted as N hunting hops. Because this realizes the true closest hunting, the hop-count performance of this scheme is used as a reference of the true closest hunting case.

5.2 Simulation Analysis

Figure 8 shows performances in cases of 32, 64, 128, 256, and 512 buffers when an origin node locates in S1 and S2 (shown in Fig. 5). Although the performances in the S1 case are better than those in the S2 case, both performance behaviors are similar with respect to buffer size dependency.

In terms of the content hop-count performance and relative ratio of content hop-counts, which are shown in graphs (a) and (b) in Fig. 8, the scanning scheme was inferior to the flooding scheme. The flooding scheme identifies the true closest node because the encyclopedic-type hunting mechanism is employed. Following these two schemes, LTH(S) + BC and LTH(S) performed extremely well while suppressing the hunting hop-count to approximately one-
fourth of those of the scanning and flooding schemes, as shown in graphs (c) in Fig. 8. By contrast, although the content hop-count performance of the LTH(M)+BC scheme was slightly inferior to that of LTH(S)+BC and LTH(S), the hunting hop-count performance was reduced to nearly one-tenth of that observed in the flooding and scanning schemes. In other words, LTH(M)+BC obtained the almost true closest performance with a content hop-count increase of between 5 and 10%, as shown in graphs (b) in Fig. 8, whereas the hunting hop-count was suppressed to approximately one-tenth, as shown in graphs (c).

The reason that the hunting hop-counts of LTH(S) and LTH(S)+BC are larger than that of LTH(M)+BC is explained as follows. The content request is branch-casted to the entire network when the origin node becomes the fork node. Because of content eviction, this situation often occurs in LTH(S) and LTH(S)+BC schemes. By contrast, in LTH(M)+BC, because a node with caching history may become a fork node, the origin node rarely conducts the branch-cast operation, even if much content is evicted.

As for the relative ratio of content hop-counts, as shown in graphs (b) in Fig. 8, LTH(S)+BC, LTH(S), LTH(M)+BC, and scanning schemes provide a flat performance for buffer size increase, whereas other schemes show an increase in buffer size.

Regarding the proactive breadcrumb operation (+BC), it reduces the content hop-count when LTH(S)+BC, LTH(M)+BC, and CCN+BC are employed.

Compared to the multiple breadcrumbs scheme, LTH(M)+BC provided the better content hop-count performance and maintained the same hunting hop-count performance in cases involving any buffer size.

Accordingly, LTH(M)+BC is preferable for use in hunting the almost true closest caching node when considering the hunting hop-count as a cost.

As shown in graphs (a) in Fig. 8, the content hop-count performance was nearly linear to the logarithms of buffer sizes. Figure 6 shows that the accumulated volume of cached content is proportional to the logarithm of the content catalog size when the buffer is occupied based on a high ranking order. However, because high ranking content generally in an LRU eviction environment remains in a buffer.

| Group | Rank | Request Frequency | Mean Request Frequency Per Content Categories of Each Group |
|-------|------|-------------------|----------------------------------------------------------|
| 1     | 1    | 46,927            | 46927                                                    |
| 2     | 2 - 4| 50,799            | 16933                                                    |
| 3     | 5    | 43,685            | 6269.29                                                  |
| 4     | 12 - 31| 47,037        | 2351.85                                                  |
| 5     | 32 - 85| 46,733          | 865.46                                                   |
| 6     | 86 - 231| 46,741       | 320.14                                                   |
| 7     | 232 - 631| 46,978         | 117.44                                                   |
| 8     | 632 - 1720| 46,832         | 42.65                                                    |
| 9     | 1721 - 4693| 46,863       | 15.76                                                    |
| 10    | 4694 - 12800| 47,205       | 5.82                                                     |

Fig. 9 Simulated performances in cases of 128 and 512 buffer sizes (origin node is S1) (The horizontal axis indicates content group described in Table 1. The mean request frequency per content catalog of each group is shown in right column of the table).
for a longer period than does low ranking content, many buffer parts are occupied with high ranking contents. Thus, the buffer size affects performance in a logarithmic manner.

A detailed performance behavior is then analyzed based on ranked groups having the same requester population. To this purpose, the content catalogs are bunched into several groups so that the request frequency (i.e., the requester population of each group) becomes nearly identical. Because the highest ranking group 1 consisting of the single highest ranking content occupies 10%, as shown in Fig. 6, they are divided into 10 groups. (The groupings are solely for analysis and do not modify Zipf’s law distribution.)

Table 1 shows the groupings in the simulation. Because the request frequency corresponds to the requester population, each group possesses nearly 10% of the 470,000 total population, as indicated in the middle column. By contrast, the right column indicates the mean request frequency per content catalog in each group. For example, because Group 1 possessed only one highest ranking content, the mean value in the right column is the same as the request frequency. In Group 10 (having catalog rankings of 4,694 to 12,800, the mean value is calculated as: 47,205/(12,800 – 4,693) = 5.82.

Figure 9 shows the simulation results for the models having 128 and 512 buffers per node in the origin node S1 case. The horizontal axis of all graphs indicates the content group described in Table 1.

In all groups, we found that LTH(S), LTH(S)+BC, and LTH(M)+BC schemes offer better performance next to that of the scanning scheme, as shown in graphs (a) and (b) in Fig. 9. The peak position of the relative ratio of content hop-count to the true closest case depends on the buffer size and shifts from Group 4 with a 128 buffer size to Group 6 with a 512 buffer size.

Among LTH(S), LTH(M)+BC, and LTH(S)+BC schemes, a smaller performance gap is observed in the relative content hop-count ratio up to Group 3 in the 128 buffer case and up to Group 5 in the 512 buffer case. Therefore, we determined that selecting the hunting scheme contributes to an improvement in performance primarily in middle and low ranking content catalogs.

6. Conclusion

This paper proposed Local Tree Hunting (LTH) scheme to find a caching node beyond the content name base path without using a cyclopedic method like flooding and its usefulness was verified by simulation. By the branch-cast operation for content request in a local tree, an almost true closest caching node can be discovered while suppressing the hunting hop-count because the search area is autonomously adjusted according to the content dissemination rate. The performance can be further improved by cooperating with proactive breadcrumb operation for neighbor nodes.

Based on a quantitative comparison of CCN, CCN+BC, multiple breadcrumbs, scanning and flooding schemes, we obtained the following results. After scanning and flooding schemes, LTH(S) and LTH(S)+BC can locate the closest caching node while suppressing the hunting hop-count to approximately one-fourth that of the encrypted schemes. By contrast, although LTH(M)+BC is only somewhat inferior to LTH(S) and LTH(S)+BC schemes in terms of content hop-count performance, the hunting hop-count is further reduced to nearly one-tenth of that in these cyclopedic schemes. Hence, LTH(M)+BC is preferable for closest node hunting when considering the hunting hop-count as cost.

Acknowledgments

The authors would like to thank Prof. M. Murata of Osaka University and the colleagues of the New Generation Network Laboratory for their constructive suggestions.

References

[1] http://www.named-data.org/ndn-proj.pdf, “Named data networking (NDN) project,” NDN-0001, Oct. 31, 2010.
[2] V. Jacobson, D.K. Smetters, J.D. Thornton, M.F. Plass, N.H. Briggs, and R.L. Braynard, “Networking named content,” Proc. CoNEXT 2009, pp.1–12, Dec. 2009.
[3] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, “A survey of information-centric networking,” IEEE Commun. Mag., pp.27–36, July 2012.
[4] M.F. Bari, S.R. Chowdhury, R. Ahmed, R. Boutaba, and B. Mathieu, “A survey of naming and routing in information-centric networks,” IEEE Commun. Mag., pp.44–53, Dec. 2012.
[5] H.S. Jeon, I.S. Choi, B.J. Lee, and H.Y. Song, “A closer look at content-centric internet research projects,” ICACT2012, pp.698–702, Feb. 2012.
[6] S. Wang, J. Bi, J. Wu, Z. Li, W. Zhang, and X. Yang, “Could in-network caching benefit information-centric networking?,” AINTEC ’11 Proc. 7th Asian Internet Engineering Conference, pp.112–115, 2011.
[7] M. Lee, K. Cho, K. Park, T.T. Kwon, and Y. Choi, “SCAN: Scalable content routing for content-aware networking,” IEEE ICC 2011, pp.1–5, June 2011.
[8] L. Dong, D. Zhang, Y. Zhang, and D. Raychaudhury, “Optimal caching with content broadcast in cache-and-forward networks,” Proc. IEEE ICC 2011, pp.1–5, June 2011.
[9] E.J. Rosenweig and J. Kurose, “Breadcrumbs: efficient, best-effort content location in cache networks,” Proc. IEEE INFOCOM 2009, pp.2631–2635, 2009.
[10] H. Shimizu, H. Asaeda, M. Jibiki, and N. Nishinaga, “Content hunting for in-network cache: Design and performance analysis,” Proc. IEEE ICC 2014, pp.3178–3183, June 2014.
[11] S. Salsano, N. Blefari-Melazzi, A. Detti, G. Morabito, and L. Veltri, “Information centric networking over SDN and OpenFlow: Architectural aspects and experiments on the OFELIA testbed,” Comput. Netw., vol.57, Issue 16, pp.3207–3221, Nov. 2013.
[12] T. Sakano, Y. Tsukishima, H. Hasegawa, T. Tsuritani, Y. Hirota, S. Arakawa, and H. Tode, “A study on a photonic network model based on the regional characteristics of Japan,” IEICE Technical Report, PN2013-1, June 2013.
[13] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, “Web caching and zipf-like distributions: Evidence and implications,” Proc. IEEE INFOCOM ’99, vol.1, pp.126–134, 1999.
Hiroshi Shimizu is with National Institute of Information Communications Technology from 2011. He joined to Communication Research Laboratory of NEC Central Research Laboratories in 1975 and worked for NEC Corporation until 2009. From 2009 to 2011, he was a professor of Tokai University. He received his B.E, M.E. in Electronics Engineering and Ph.D. in Informatics from Kyoto University, Japan, in 1973, 1975 and 2008, respectively.

Hitoshi Asaeda is a planning manager of Network Research Headquarters, National Institute of Information Communications Technology. From 1991 to 2001, he was with IBM Japan, Ltd. From 2001 to 2004, he was a research engineer specialist at INRIA, France. He was a project associate professor at the Graduate School of Media and Governance, Keio University, where he worked from 2005 to 2012. He chairs the ICN working group of Asia Future Internet Forum. He holds a Ph.D. from Keio University.

Masahiro Jibiki received the Ph.D. degree in systems management from the University of Tsukuba, Japan, in 2003. In 1992, he joined NEC Corporation, and he is a researcher in the Central Research Laboratories. From 2006 to 2009, he was also a visiting professor at the University of Wakayama, Japan. Currently, he is also holding the post of an expert researcher in the National Institute of Information and Communications Technology. His research interests include networking, distributed systems and software science.

Nozomu Nishinaga received his B.S. and M.S. in Electronics Engineering and his Ph.D. in Information Engineering from Nagoya University, Japan in 1994, 1996, and 1998, respectively. From November 1998 to March 1999, he was a research assistant at the Information Media Education Center, Nagoya University. From 1999 to the present, he has been a researcher with National Institute of Information and Communications Technology (formerly, Communications Research Laboratory). Since April 2011, he is director of New Generation Network Laboratory, Network Research Headquarters. His current research interests include internet architecture and wireless communications. He is a member of IEICE.