Deployable and Scalable Information-Centric Networking

（展開性とスケーラビリティを備えるInformation-Centric Networkingに関する研究）

朱 韵成

ZHU, Yuncheng

(2011年度入学, 49-117402)

指導教員: 中尾 彰宏 准教授

東京大学大学院 学際情報学府 学際情報学専攻
総合分析情報学コース 中尾研究室

Abstract

The current Internet architecture has remained relatively unchanged since its invention for end-to-end data communication. However, it is often posited today that the information objects must be addressed by the information objects themselves, rather than by where they are located in the network as it is done in the current Internet architecture. That is to say, it is information itself that should be central in the network architecture, because information consumers do not care where the information is located, but which information they need. Such a research position has led the research community to redesigning the Internet into so called information-centric networking (ICN), where the network allows users to discover the information directly and not addressed indirectly by which hosts possess it. However, the existing ICN studies leave the following significant problems open, which hinders the deployment of ICN.

1. *Functionality problem*: The existing ICN proposals are most single-strategy, which focuses on only the content retrieval. Hence, they fail to address the critical importance of providing efficient service access, such as publishing user-generated content.

2. *Deployability problem*: Most of the existing ICN proposals are clean-slate approaches hardly deployable in the Internet, as they require substituting the global IP networks. Moreover, some of them are partial solutions still with major drawbacks unresolved, regarding the capability requirements to individual network devices.

3. *Scalability problem*: ICN's key component, route-by-name scheme, brings a challenge to scalability because of a large number of information objects. Feasible and efficient name resolution service is required, but none of the current literatures provide sufficient solutions.

This thesis posits that solving the three problems unresolved by existing ICN research, namely, functionality, deployability and scalability, are the key to the successful migration to an ICN architecture from the current Internet. The benefits of such migration lie in two aspects: for end users, both efficient and fast content access and convenient access to novel services can be provided, and for network providers, low upgrade cost are required provide more values and network resources can be better utilized. Our study reveals the possibility of improving the Internet in an information-centric way by practically deploying ICN to operate over the current Internet infrastructure.

To be concrete, the problems are solved by studying three critical aspects of ICN: caching policies, transport protocol and name resolution, and a new architecture for deployable and scalable ICN is proposed.

First, caching policy is one of the most important research topics in ICN. We propose Upload Caching in Edge Networks (UCEN) to solve *Problem 1*. Our analysis shows that this mechanism reduces upload tether time of 41% end users by more than half and flattens the traffic peak for the access service provider by 49%. To address *Problem 2*, we propose Content-Oriented Caching with In-Network Index (COCINI), a caching scheme to exploit spare storage and bandwidth from end-systems to eliminate redundant traffic and to enable efficient and fast access of contents. Our trace-driven simulation indicates that COCINI can reduce up to 49% traffic volume and can cumulatively reduce about one fourth of the latency of content access.

Second, to resolve *Problem 2*, we propose Information-Centric Transport Protocol (ICTP), a transport layer protocol to support most features of ICN over current Internet infrastructure. The protocol is compatible with current Internet Protocol (IP) and can be incrementally implemented and deployed. In-network processing of information-centric strategies can be benefited by the genuine connection-less feature of the protocol.

Finally, to deal with *Problem 3*, we propose a simple and empirical Distributed Resolution Service (DRS) scheme according to the time and space complexity models we built. The proposed scheme can handle at least 10^{12} name entries using about 3,300 nodes with commodity hardware.

We propose a *Deployable and Scalable Information-Centric Network Architecture* (DSINA), an innovative ICN architecture that incorporates novel route-by-name system into the current Internet infrastructure with register-access-result model, which

integrates all the technologies developed above and enables the migration to ICN from the existing Internet. DSINA can handle not only content retrieval but also user-generated content uploading, notification pushing and other applications. A prototype of DSINA is implemented in C++ with Click programming model, and deployed on the network testbed Emulab. Our evaluation experiments verifies the prototype system works as expected, thus, shows it can be deployed in the real networks and all the functions work correctly there, for example, can perform roughly 7 times more efficiently for content delivery than it has been done in the existing Internet architecture, and is self-scaling with less than 10% average throughput drop when multiple clients request for the same piece of content.

The designs, implementations and evaluations prove that the three problems, functionality, deployability and scalability, have been solved by our proposed technologies and architecture. Our results enable ICN research to be implemented and deployed in a large scale, and enable migration to ICN from the current Internet architecture.

Acknowledgment

I would like to express my deepest appreciation and gratitude to my supervisor, Associate Professor Akihiro Nakao for his outstanding and invaluable direction. He continuously supports my research with attractive ideas and timely assistance throughout the period of my Ph.D. study. He teaches me how to be a good researcher and always encourages me to improve my skills and catch chances for a better academic career. This thesis would not have been possible to be completed without his excellent supervision.

I would like to thank comittee members, Professor Ken Sakamura, Professor Noboru Koshizuka, Professor Jun Rekimoto, and Professor Satoshi Ohzahata (from the University of Electro-Communications) for their valuable suggestions on my research.

I would like to thank my colleagues, as well as research staffs and secretariats in Nakao Laboratory, for their support and kindness during my study and life here. They have been good companions who embrace the spirit of hard working.

I would like to thank Japanese Government Monbukagakusho (MEXT) Scholarship for their finance support during my study as a Ph.D. student in the University of Tokyo.

Finally, I thank to my family and to all my friends, who are always there when I need them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The current Internet architecture has remained relatively unchanged since its invention for end-to-end data communications, which dates back to about three decades ago. The Internet protocol suite [1], commonly known as TCP/IP, emphasizes the end-to-end principle — to put maintenance of state and overall intelligence at the end-systems and to leave the network retaining no state and concentrated on speed and simplicity. The two major routing schemes, unicast and multicast of the Internet Protocol (IP) represent host-to-host and host-to-multiple-hosts transmission models following it. The principle has worked well and has greatly propelled the wide spread of the Internet.

On the other hand, the Internet has become one of the indispensable infrastructures in people's daily life, and as a result, it is facing the problems of the explosion of users and huge traffic, as well as a great diversity of usages. Statistics [2] show that the majority of today's Internet traffic is associated with content retrieval applications using Web or Peer-to-Peer (P2P), and the remaining is related to service access, such as E-mail and online games. The end users do not care at which host the content or service is located any more, but only focus on the information they need.

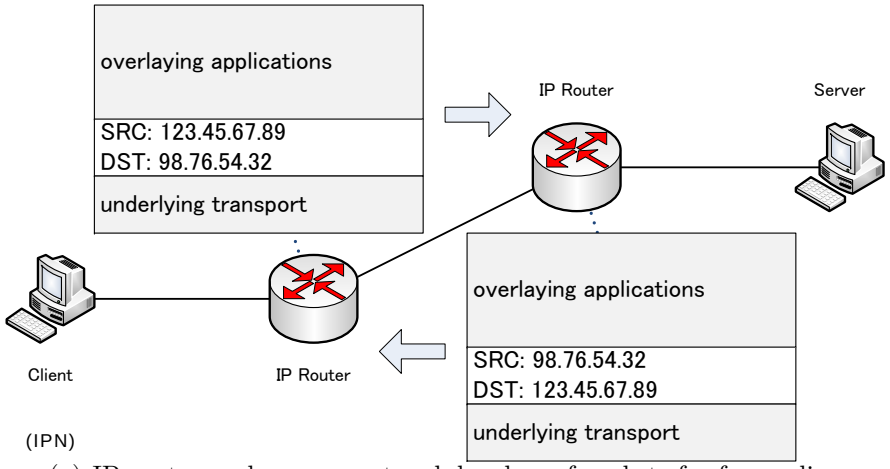
Since the misalignment between the host-centric architecture and the content-centric usage of the current Internet results in inefficient content and service access, Internet Service Providers (ISPs) try to improve the performance by deploying content delivery networks (CDNs) [3] and proxy caches [4]. Although caching facilities greatly improve content retrieve efficiency and thus are common nowadays, these intermediate elements are not integrated with routing and forwarding. As a result, they are

criticized to be indirect, expensive and sometimes conflicting with the interest of ISPs.

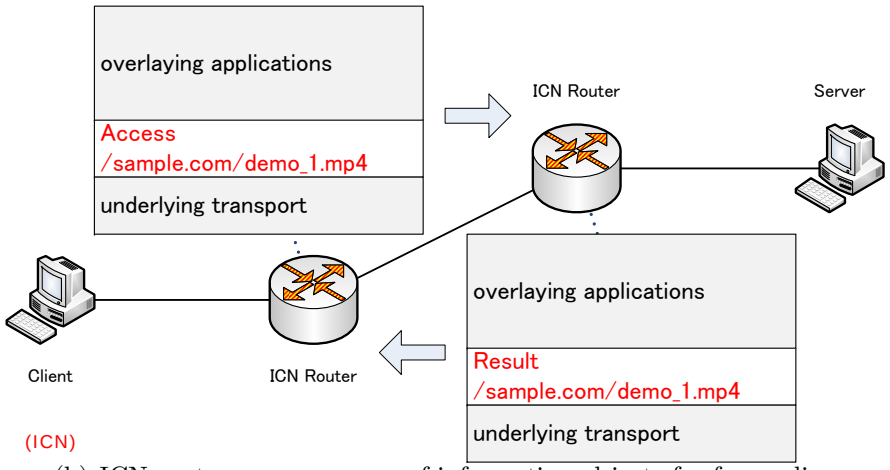
The rapid development of Information-Centric Networking (ICN) concepts in the last few years is one of the significant results of a number of researchers are aiming at redesigning the Internet from the information-centric perspective. In their clean-slate designs [5, 6, 7], content delivery is directly central to the network architectures proposed. Based on the ICN concepts, the principal communication paradigm is no longer end-to-end data delivery between hosts, but focuses straight on retrieving information objects securely, reliably, scalably and efficiently. Therefore, despite the differences in architectural designs of the ICN proposals, all of them share the features such as network devices forward the request for a specific piece of information to wherever the content is available and carry the corresponding content towards the users requesting it, as illustrated in Figure 1.1. In ICN, the most critical information in routers is no longer what records the routes to networks and hosts, but the queries and the results, that is, the information on the existence of the content.

In order to enable the network devices to process information objects, any information object is associated with a unique name in ICN. Upon receiving a request for a specific piece of information, the network device forwards the request according to the locator information of the requested name, which is called "route-by-name." Effectively naming information objects and forwarding requests and responses have been the focus of ICN research efforts. Different designs employ different naming schemes and thus have different route-by-name implementation.

ICN embraces significant advantages compared with the current Internet. First, ICN matches the usage trend of the current Internet coherently and directly, which results in a better utilization of network resources. Second, ICN enforces end-to-end security, where information security is ensured by the information object itself, rather than the channel it is transferred through. Finally, ICN adopts a more flexible forwarding scheme compared to IP, and thus facilitates user choice and competition among ISP. Consequently, ICN is expected to address the network challenges that arise from the increasing demands for highly scalable content distribution, from accelerated growths of mobile devices, and from the deployment of Internet-of-Things (IoT).



(a) IP routers only access network headers of packets for forwarding



(b) ICN routers access names of information objects for forwarding

Figure 1.1: Comparison between packet forwarding in the current Internet and in ICN

1.2 Problem Statement

The existing ICN studies have pointed out a system-level direction to re-designing the major functions of the current Internet. However, they leave the following significant problems open, which hinders the implementation and deployment of ICN. We posit that there are three problems to be solved in order to achieve efficient content and service access.

1.2.1 Functionality Problem

The first problem of existing ICN architectures is that it lacks in essential functionalities such as publication of user-generated contents (UGC) and content cache classification.

ICN ought to support a variety of content and service access, including but not limited to efficient content distribution. Because popularity of mobile network devices such as smartphones has been encouraging more and more people to produce contents and publish them, publication of UGC is one of the features that must be supported by ICN. However, most of the existing ICN proposals are single-strategy, which focuses on only the content retrieval. Especially, the Interest-Data model that is widely used in the existing proposals is not optimal for supporting content pushing. Hence, they fail to address the critical importance of providing efficient service access, such as publishing user-generated content.

Content caching is one of the most important features of ICN. In reality, caching facilities can be classified into two categories, persistent storages, such as the ones used in CDN, and transient caches, such as in-network caching and caches at client hosts. These two sorts of caching are different in many aspects, such as the duration of existence and the purposed scope of access. Therefore, they should be distinguished in registration and propagation of content information, and different strategies should be applied correspondingly.

1.2.2 Deployability Problem

The existing ICN proposals are mostly clean-slate approaches that are not compatible with today's Internet and could not be accepted without a transition process [8]. It is especially unaffordable to replace the current IP with clean-slate ones, because this would result in redesigning addressing, routing and forwarding for the global Internet, which is desirable for neither end users nor network operators.

Besides the impossibility of substituting the global IP networks, some of the current solutions are still with major drawbacks unresolved, regarding the capability requirements to individual network devices. We address specifically two challenges, that is, caching policy proposed requiring high cost for cache storage, and naming and resource management requiring expensive high speed searching hardware such as Ternary Content-Addressable Memory (TCAM).

1.2.3 Scalability Problem

Network routing scalability issues have been driving new architecture designs, recently. Route-by-name, the new mechanism adopted by ICN to forward packets accord-

ing to the names of the information objects they are requesting for, faces a scalability challenge in term of the number of name entries, which is much more critical than it is in the current Internet. The previous study [5] has revealed that the number of registered names is in the order of 10^{12} or more, even under the optimistic expectation, and the requirement of lookup speed is also crucial, extrapolated from the HTTP request rate in the Internet of today, i.e., about 20,000/Gb. These requirements are unlikely to be resolved by a single network device and call for a feasible and scalable distributed solution. However, none of the current literatures [5, 6, 7, 9, 10, 11] provide sufficient solutions to this problem.

1.3 Thesis Statement

This thesis posits that solving the three problems unresolved by existing ICN research, namely, functionality, deployability and scalability, are the key to the successful migration to an ICN architecture from the current Internet. Generally speaking, the benefits of such migration lie in two aspects. For end users, deployable caching strategies make efficient and fast content access possible, and convenient access to fresh services can be provided by new access models. For network providers, deployable and scalable architecture reduces required upgrade cost to provide more values, and network resources can be better utilized with novel transport. Our study reveals the possibility of improving the Internet in an information-centric way by practically deploying ICN to operate over the current Internet infrastructure. Our findings are not constrained to some specific architecture proposal but they are expected to contribute to many networks that adopt information-centric concepts.

1.4 Thesis Summary

Chapter 1 provides a general introduction of ICN and the problems it is facing.

Chapter 2 introduces three famous research efforts of ICN and analyzes the pros and cons of the three architectures proposed.

Chapter 3 proposes *Upload Caching in Edge Networks* (UCEN). Our analysis shows that this mechanism reduces upload tether time of 41% end users by more than half and flattens the traffic peak for the access service provider by 49%.

Chapter 4 proposes *Content-Oriented Caching with In-Network Index* (COCINI), a

caching scheme to exploit spare storage and bandwidth from end-systems to eliminate redundant traffic and to enable efficient and fast access of contents. Our trace-driven simulation indicates that COCINI can reduce up to 49% traffic volume and can cumulatively reduce about one fourth of the latency of content access.

Chapter 5 proposes *Information-Centric Transport Protocol* (ICTP), a transport layer protocol to support most features of ICN over current Internet infrastructure. The protocol is compatible with the current IP and can be incrementally implemented and deployed. In-network processing of information-centric strategies can benefit from the genuine connection-less feature of the protocol.

Chapter 6 proposes a simple and empirical *Distributed Resolution Service* (DRS) scheme according to the time and space complexity models we built. The proposed scheme can handle at least 10^{12} name entries using about 3,300 nodes with commodity hardware.

Integrating all the technologies developed above, Chapter 7 proposes *A Deployable and Scalable Information-Centric Network Architecture* (DSINA), which incorporates novel route-by-name system into the current Internet infrastructure with register-access-result model. DSINA can handle not only content retrieval but also user-generated content uploading, notification pushing and other applications. Our prototype implementation deployed on Emulab verifies the prototype system works as expected, thus, shows it can be deployed in the real networks and all the functions work correctly there.

Chapter 8 concludes this thesis and discusses its future work.

Chapter 2

Related Work

2.1 Introduction

ICN has been extensively studied, especially in cases of content-oriented network architectures. Data-Oriented Network Architecture (DONA) [5] and Named Data Networking (NDN) [12] are two notable examples of clean-slate content-oriented network architectures. Pursuing a Pub/Sub Internet (PURSUIT) [13] is another clean-slate architecture based on the publish-subscribe communication paradigm. In this section, the popular research efforts mentioned above are introduced and there pros and cons are briefly discussed.

2.2 Data-Oriented Network Architecture

The Data-Oriented Network Architecture (DONA) proposed by Teemu Koponen *et al.* [5] is the one of the earliest ICN attempts that argues a "clean-slate" technique in naming and name resolution, aiming at providing persistence, availability and authenticity in content access.

In DONA, an information objects is associated with a flat and self-certifying name in the form P:L, where P is a cryptographic hash of the public key of a principal (i.e., a registered information publisher) and L is a label chosen by the principal, who ensures the names are unique. Based on the naming scheme, DONA proposes a *route-by-name* paradigm for name resolution with two basic primitives, FIND and REGISTER, and requires every network provider to own a *resolution handle* (RH). In DONA, principals announce information objects and their locations using REGISTER messages sent to RHs, while clients send FIND messages to resolve the names and initiate the transport

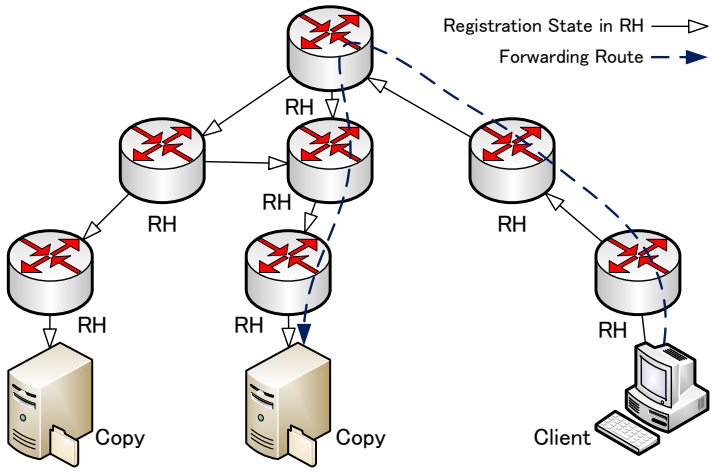


Figure 2.1: RHs route a client-issued FIND message (dashed arrow) to a nearby copy

exchange, as shown in Figure 2.1. The packet exchanges that occur after a FIND message reaching its target server are not handled by the proposed system but using standard IP routing and forwarding.

Designed as a replacement to DNS naming resolution scheme, DONA is inadequate in taking into account the scalability problem. It is neither possible to store nor to resolve all unaggregated names for all the content available in the Internet for any network device in a tier-1 network. Aggregation for flat names [9] requires publishers to provide aggregation information, but it is hard to justify both incentives for such scheme and its effectiveness. Moreover, only hosts authorized to serve an information object with name P:L can send a REGISTER message to their local RHs in DONA. This constrains the available benefit provided by the architecture to the same extent of that is done by CDN, and hinders the architecture to serve user-generated contents (UGC).

2.3 Named Data Networking

The Named Data Networking (NDN) project led by Lixia Zhang [12] is one of the four projects under NSF's Future Internet Architecture Project [14], which is also known as Content-Centric Networking (CCN) [6]. The project proposes hop-by-hop content-centric routing based on hierarchical content names.

One of the key contributions of NDN is the forwarding engine model. As described in Figure 2.2, instead of host prefixes, content prefixes are stored in forwarding nodes

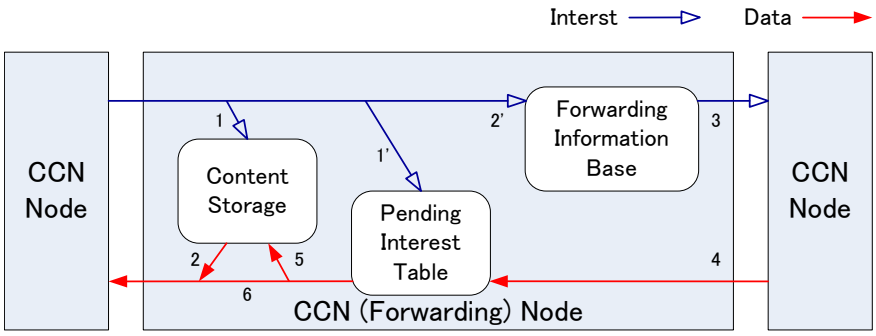


Figure 2.2: Content retrieval among three NDN nodes

in NDN. The forwarding engine consists of three components, *Content Storage* (CS), *Pending Interest Table* (PIT) and *Forwarding Information Base* (FIB). When a forwarding node receives an Interest message sent from another node, it (1) first compares the requested content name against CS. If the content is cached in the local CS, the node (2) transfers the content from CS to the requesting node directly. Otherwise, the forwarding node (1') checks the content name against PIT to make sure it is not a duplicated request and records the incoming interface, and then (2') looks up the name in FIB and (3) sends the Interest message to a neighboring node. A Data message responded by a serving node is forwarded exactly in the reverse direction to reach the client. When a forwarding node receives a Data message, it (4) looks up the content name in PIT for any pending request to it. If there is any, it (5) puts the content to the CS for caching and (6) sends the Data message to corresponding interfaces.

However, the architecture has difficulty in supporting data uploading, while user-generated content is becoming popular nowadays [15, 16]. Since host locating information is completely eliminated from the architecture, the architecture suggests a workaround that asks a client to send an Interest message including the publishing name for uploading data to trigger the server initiating another content retrieval process towards the client itself. This requires the client announce the name for its uploading data beforehand so that it is accessible from the server, while such behavior hampers the aggregation of names and the mobility of client hosts. In addition, people doubt the viability and efficiency in FIB units resolving a large number of content names and in integrating CS into forwarding nodes in the networks, especially for the top-tier network service providers.

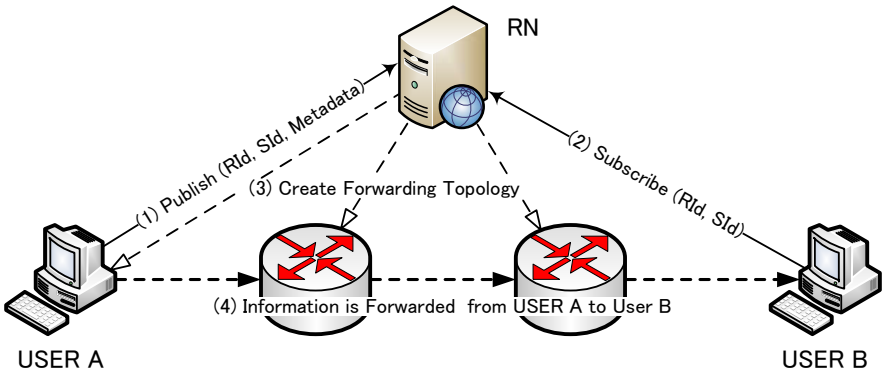


Figure 2.3: Publication and Subscription in PURSUIT

2.4 Pursuing a Pub/Sub Internet

Pursuing a Pub/Sub Internet [13] is a FP7 EU project inheriting the vision and results of their former project published as The Publish-Subscribe Internet Routing Paradigm (PSIRP) by Nikos Fotiou *et al.* [7] that focuses on providing layered information access control using rendezvous nodes.

In this architecture, the identifier of an information object has two parts, a *rendezvous identifier* (RId) and a *scope identifier* (SId). RId is a flat global unique identifier such as it is in DONA, while SId specifies a specific scope within which the information object is accessible. As illustrated in Figure 2.3, USER A sends a publish message containing RId, SId along with meta-data of the information object to a correspondent *rendezvous node* (RN). A RN implements *Rendezvous*, *Topology* and *Forwarding* (RTF) functions in the architecture. When someone having interest in such information object, e.g. USER B, sends a subscribe message to the RN, the interest is matched with actual publications in RN using rendezvous function. The topology function monitors the network topology and detects changes, and finally the forwarding function implements information object forwarding through creating a delivery path from the publisher to the subscriber using an MPLS-like label switching protocol and initiating the actual data transfer.

Nevertheless, this architecture does not specify how publish and subscribe messages are forwarded to RN in charge of a specific scope, which is one of the critical challenges to information-centric networking. Besides, adopting label switching for content forwarding is impractical to be done in the whole Internet, because RN has to operate network devices of other network providers when the subscription comes from an ex-

ternal network, which is the very same problem that Multiprotocol Label Switching (MPLS) [17] has been facing when it comes to wide-area routing. Moreover, PURSUIT shares a similar problem with NDN that users must own some constantly connected servers so that they can get their content accessed with those servers.

Chapter 3

Upload Caching in Edge Networks

3.1 Introduction

The common argument for the current caching systems and the existing ICN proposals is that they aid end users within edge networks in retrieving content from always-on servers, while little research has focused on uploading user-generated content (UGC). Recent studies show that the traffic uploading UGC accounts for a large portion of the current Internet traffic and brings new challenges in the Internet of today [15, 16].

In the light of this observation, we propose *Upload Caching in Edge Networks* (UCEN), a new mechanism assisting upload of UGC of end users within edge networks. In this mechanism, an end user will not directly upload his/her pieces of content to the remote servers, but put them to a gateway server located within the same edge network with the user. The gateway caches the received content and schedules its delayed upload to the destination servers without incurring much extra delay in the total elapsed time for the upload, without the involvement of the end user in the latter step.

Deploying upload caches in edge networks brings benefit for both end users and service providers. First, for end users, it shortens the duration while user must stay online for uploading their generated content, thanks to the short round-trip time (RTT) and ample bandwidth between users and gateway. In addition, from the service provider point of view, traffic peak of the edge networks or the destination servers can be flattened because the inter-network upload from the gateways to the destination servers may be scheduled at later time. Finally, the cached piece of uploaded content may be reused for populating download cache so that even the traffic of the first content retrieval of the edge networks may be reduced.

3.2 Related Work

We recognize that a similar feature of proposed architecture can be achieved by Delay-Tolerant Networking (DTN) Architecture [18], where routers take a "store and forward" approach and data are incrementally moved and stored throughout the network until they eventually reach their destination. However, our solution is different from DTN in several aspects. At first, the assumption of underlying infrastructure is different. DTN targets occasionally-connected networks where conventional approaches are impractical, while our proposed architecture is based on the Internet today and the current user behaviors. Furthermore, DTN introduces a new bundle layer overlay, which requires routing functionality and persistent storage in all the nodes, while our proposed solution is light-weight and stateless with only ephemeral cache placed on the gateway and all functionalities implemented in a single layer.

3.3 System Design

Unlike the traditional content upload schemes where only a pair of client and server is involved in an upload procedure, a gateway in the edge network plays an important role in the proposed mechanism. In our proposal, an end user uploads its generated content to the gateway located within the same edge network with the user. The gateway caches the received content and uploads it to the corresponding destination server.

There are two major goals of the proposed mechanism. The first goal is to shorten the duration while an end user must stay online for uploading their generated content, making use of the short RTT and ample bandwidth within the edge networks. The second goal is to flatten the traffic peak in the network of service providers including edge networks and the destination servers.

In this section, we introduce our design decisions to construct a full-fledged mechanism satisfying the goals, including upload time shortening, scheduled upload, server delegation and coordination of entities. Finally, we present a typical upload process following our design decisions.

3.3.1 Upload Time Shortening

Shortening the upload time of user-generated content for the end users is the first goal of upload cache in edge networks. It is achieved by dividing the upload process into two segments: uploading from the client to the gateway server in the same edge network and upload from the gateway to the destination server.

The end users may note endure a long duration of time while they must stay online for uploading their generated content in mobile environments. In this case, clients can benefit from the edge networks usually having short RTT and ample bandwidth compared to inter-network links. They can leave the network as soon as finishing uploading their content to the gateway, which is often much faster than uploading it to the destination server. The clients also provide enough auxiliary information to the gateway server so that it may autonomously finish the remaining upload to the destination server without the involvement of the end users.

3.3.2 Scheduled Upload

In most cases, end users do not need the instant availability of uploaded content. In fact, instant availability is sometimes impractical to be provided, such as uploaded video content usually need to be re-encoded to some specific formats. Therefore, we can schedule the actual upload from the gateway to the destination server to a specific time negotiated, rather than let the gateway upload the content to the destination server as soon as it has received the complete content, so that we can avoid creating a large amount of upload traffic when the edge network or the destination server already has a high volume of network traffic.

There are two cases to consider regarding the trade-off between instant availability and traffic control. Access service providers may choose to schedule actual upload to flat the traffic peak for the edge network (that is, itself) or the destination servers (in which case there should be some contract so that it also benefits the access service provider). In the former case, the gateway asks the client for the deadline indicating that the client can tolerate the actual upload process begins by that time, while it accepts the content from the client, and then schedules the upload to sometime before the deadline according to its traffic prediction. In the latter case, the client and the destination server will negotiate for an upload deadline, usually proposed by the client,

and a recommended upload time, which is provided by the destination server and must be earlier than the deadline suggested by the client. Consequently, when the client uploads its piece of content to the gateway, it will also piggy-back the suggested upload time, so that the gateway may begin the upload process to the destination server according to the indicated time.

3.3.3 Server Delegation

In traditional caching systems, edge networks need to retrieve the content at least once to populate the cache, which limits the efficiency of download cache in reducing downstream traffic. In contrast, our proposed mechanism saves even the first content retrieval by means of cache reuse.

With the proposed mechanism, after data of the content is uploaded to the destination server, the server can reply to the gateway with the *publish address* of the content, such as the URI where the content will be accessible, if the content is cache-able. With the publish address, the gateway can to respond to the requests to the cached content in delegation of the destination server, or reuses the cached upload data for the download caching system in the same edge network. As the past research [19] on the geographic distribution of content request proves spatial locality in content references, the cache reuse feature is highly effective in improving the efficiency of reducing downstream traffic.

3.3.4 Coordination of Entities

Our proposed mechanism of upload cache in edge networks involves three entities, clients operated by end users, gateways run by access service providers and servers hosted by content service providers. In the current practice, an end user is authenticated by the access service provider via access control and is authenticated by content service provider by specific service application. However, there is generally no direct relationship between access service providers and content service providers. Moreover, end users are usually not willing to disclose their account information to their access service providers for privacy reasons. Even if they would, it would become an overhead for access service providers as they need to store authentication information of end users in their gateways. The new mechanism providing upload cache in edge networks solves this problem by introducing a temporary token in the service application.

In addition to the trust problem, in order to achieve such an indirect upload process, the client also needs to know the availability of destination server supporting such a feature and the existence and the location of the gateway in the same edge network. The information on the servers and the gateways may be provided on the fly through the access service provider or via an automatic configuration service such as Dynamic Host Configuration Protocol (DHCP) [20].

3.3.5 Typical Upload Processes

With the necessary information on the destination server and the gateway, the client can initiate a typical upload process with upload caching in edge networks, as illustrated in Figure 3.1.

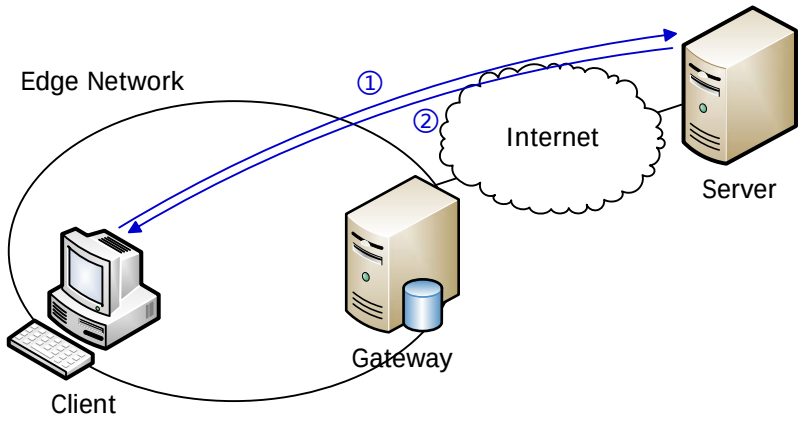
First, as shown in Figure 3.1(a), the client requests for an indirect upload token for a piece of content from the destination server. It submits the length and a fingerprint value (for example, the MD5 hash) of the specific content it wants to upload to the destination server. The server authenticates the privilege of user uploading such content and provides the token to the client that can be used for indirect upload only once and for the specific content.

When the client has received the token, it communicates with the gateway to upload the piece of content to the gateway, as shown in Figure 3.1(b). Besides the actual data of the content, the client will also transfer both the destination of the upload and the token received from the destination server to the gateway. However, no privacy information, such as end user's account or password on the destination server, is provided to the access service provider.

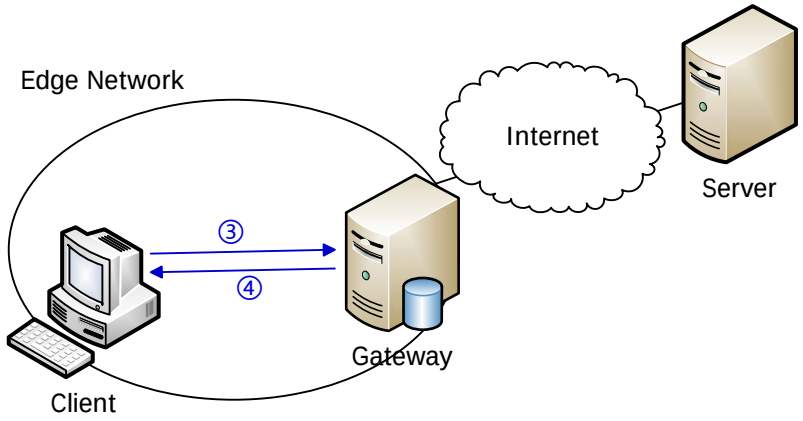
Finally, the gateway verifies the token with the destination server. Once the indirect upload is acknowledged by the server, the gateway uploads the temporary stored data of the content to the server. This step, as shown in Figure 3.1(c), doesn't involve the client at all, so the client can leave the network as soon as it finishes uploading the content to the gateway.

3.4 Evaluation

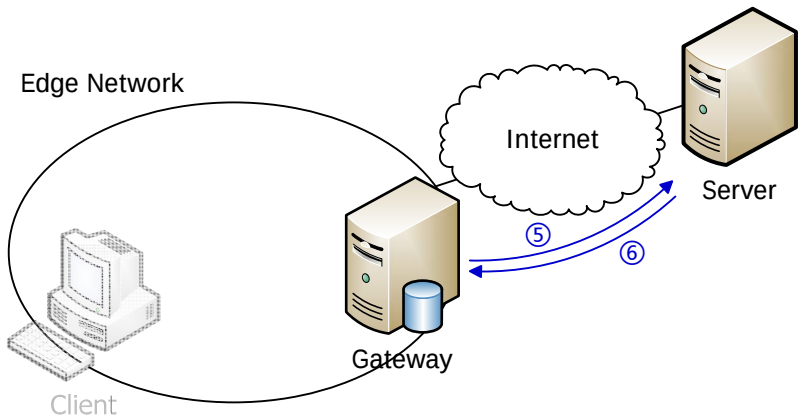
UCEN can be applied to a variety of network architectures including but not limited to ICN. In this section, we evaluate UCEN using a prototype implementation



(a) client requests for token from server



(b) client upload token and content to gateway



(c) gateway upload content to server

Figure 3.1: Typical upload process with upload caching in edge networks

of our proposed on top of Hypertext Transfer Protocol (HTTP) [21]. There are two advantages of such a decision:

1. HTTP is one of the most utilized protocols in today's Internet [22], thus we believe that its extension bring a broad impact on the online applications without much obstruction in deployment.

2. HTTP has many functionalities that have been proposed in the future Internet architecture literature [23], so that we can expect the implementation on top of HTTP to be migratable to other ICN architectures.

3.4.1 HTTP Cached-POST Implementation

Following the protocol specification of HTTP/1.1 [21], we propose an extension to the protocol with a new type of request method, called *Cached-POST* (C-POST), that implements upload caching in edge networks. The format of a C-POST request is similar to that of a traditional POST request, and such requests also utilizes the most of the message header fields as they are used in POST requests. However, as shown in Figure 3.1, there are three steps for uploading a piece of content in the new mechanism, thus C-POST requests are issued three times to complete content upload. Moreover, there is additional information to be transferred during the indirect upload procedures, such as the upload token. HTTP cookie, a standard type of message header fields, is used to carry the additional information in the C-POST request headers.

In the first C-POST request issued by the client to the destination server, in contrast to the traditional POST request, no message body is carried after the message header. Instead, such a request always carries a Cookie that indicates the length and the fingerprint of the specific content. If the end user is authenticated and the C-POST is accepted, the server replies with a standard "202 Accepted" response. The response messages always carries a Set-Cookie field that indicates the upload token with an expiration time and sometimes another Set-Cookie field indicating the suggested upload time if the scheduled upload feature is enabled.

The second C-POST request that goes from the client to the gateway and the third one that goes from the gateway to the destination server are almost identical to the traditional POST requests except that the requests carry Cookie fields including the token information the client has received from the destination server. The standard

successful response to these requests are "200 OK" as they are to the traditional POST requests. The final response replied by the destination server may include a Content-Location field to tell the gateway the publish location of the uploaded content to support the cache reuse feature.

3.4.2 Client Upload Acceleration

As introduced above, one of the most important parameters in the upload cache mechanism in edge networks is *client tether time*, which indicates the duration the client has to keep online while uploading a piece of content. We analyze the client tether time (including TCP establishment and deconstruction) according to the typical HTTP upload procedures as illustrated in Figure 3.2.

Let τ_1 denote the RTT between the client and the gateway and τ_2 denote the RTT between the gateway and the destination server. Assuming the piece of content to be uploaded has size of L bytes, and the throughput between the client and the server is b. Therefore, the client tether time in the traditional HTTP upload, as shown in Figure 3.2(a), is

$$t_1 = 3(\tau_1 + \tau_2) + L/b \tag{3.1}$$

On the other hand, let the throughput between the client and the gateway be b', the client tether time in HTTP upload with C-POST, as shown in Figure 3.2(b), is

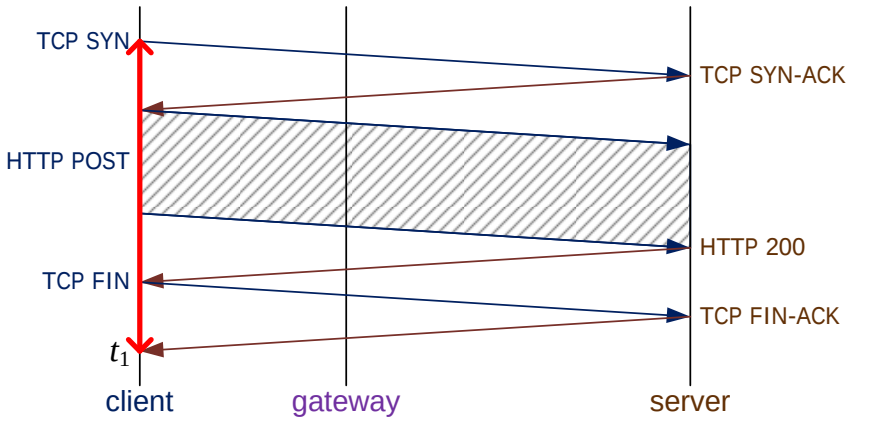
$$t_2 = 2(\tau_1 + \tau_2) + L/b' + 2\tau_1 \tag{3.2}$$

In edge networks, we usually have RTT $\tau_1 < \tau_2$ and throughput $b' > b$ according to the TCP throughput equation [24]. Thus, the client tether time is calculated according to Eq. (3.1) and Eq. (3.2), that is

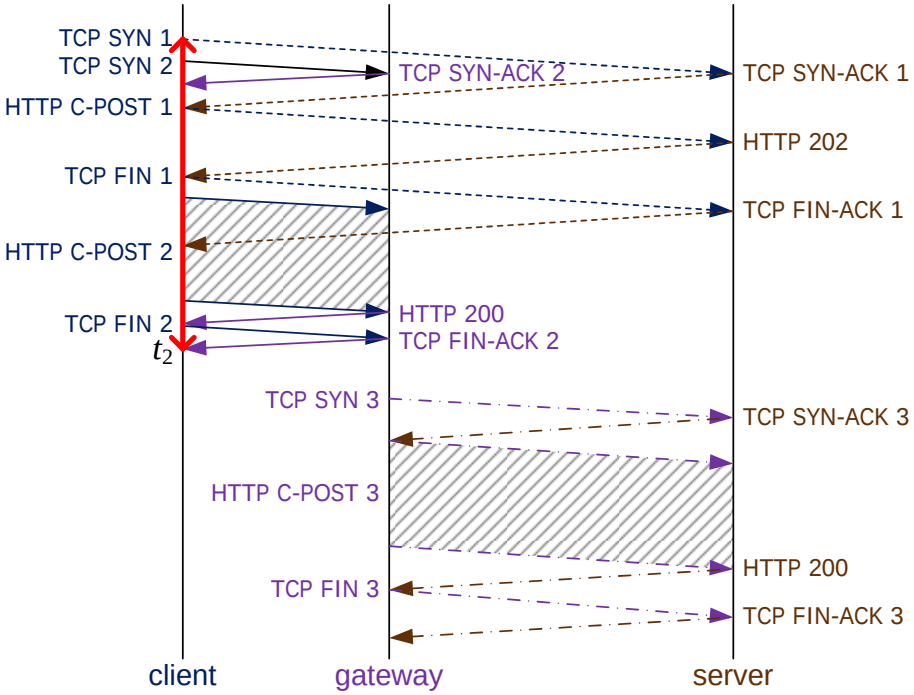
$$\Delta t = t_1 - t_2 = L(1/b - 1/b') + (\tau_2 - \tau_1) \tag{3.3}$$

As RTT can be simply probed, client decides whether utilize upload cache service or not according to its estimation of throughput and the size of content with its own criteria. For example, some clients may decide to use upload cache when $\Delta t \geq t_1/2$, while some will use it when Δt is larger than some constant time.

Since the reduction of the client tether time is contributed from two parts, the difference in throughput and RTT, we evaluate the reduction and examine effects of



(a) traditional HTTP upload



(b) HTTP upload with C-POST

Figure 3.2: Complete HTTP upload procedures

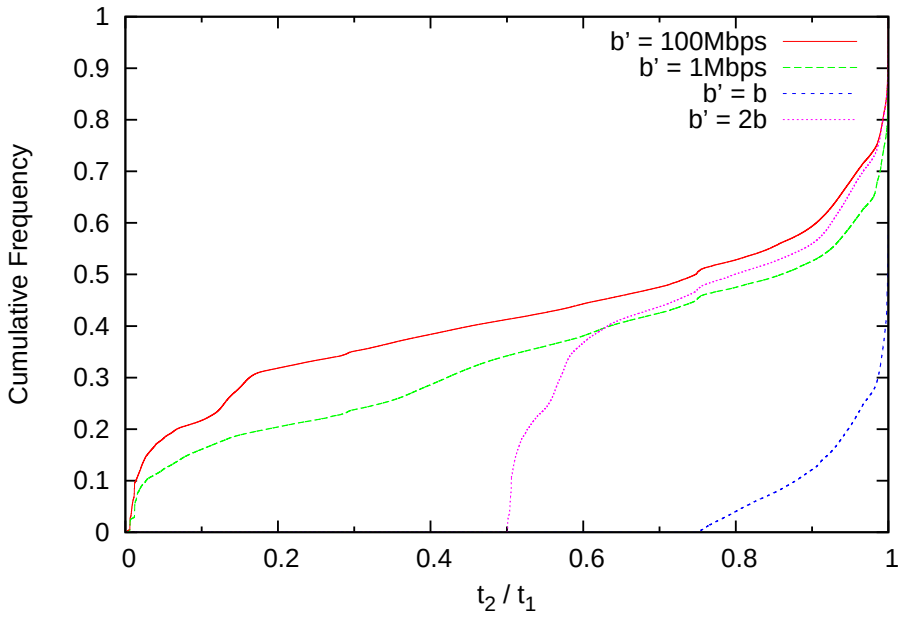


Figure 3.3: Cumulative frequency distribution of the reduction of client tether time

these two factors on the result by replaying the actual traffic trace captured from a campus network.[1]

In the evaluation, assuming the edge router playing the role of the gateway in UCEN, from the captured traffic trace, we can profile both RTT values τ_1 and τ_2, as well as the throughput between clients and servers b. However, we cannot infer the throughput between the clients and the gateway, b'. Thus, we use different values of b' and the result distribution of the reduced tether time reduction compared to the original time is illustrated in Figure 3.3.

Even if the throughput between clients and the gateway is the same as that between clients and servers (in which case the system only benefits from the differences of RTT), UCEN still provides a little benefit on client tether time. More than 10% of traffic reduces its client tether time by 10% or more. Doubling the throughput, nearly half of the traffic reduces its client tether time by 20% and the maximum reduction ratio is about 50%.

If the throughput in edge networks is assumed to be constant, result shows curves dissimilar from the two described above. If the throughput is 100Mbps, 41% of traffic reduces their tether time by half, and even if it is 1Mbps, 34% of traffic reduces theirs

[1]Please refer to Appendix A for the details of data preparation, similarity hereinafter.

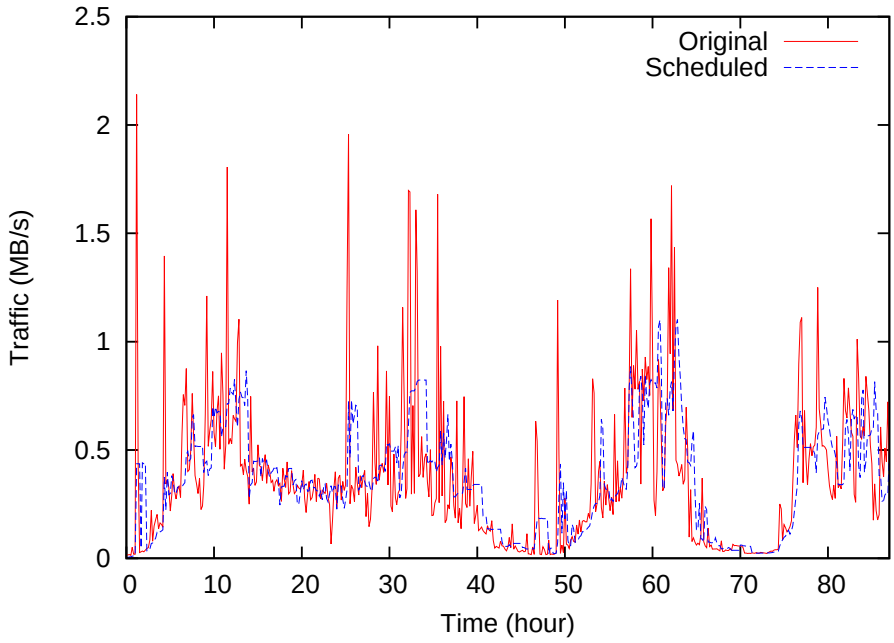


Figure 3.4: Traffic on the edge router before and after scheduling

by half. The curve is flat and the maximum reduction ratio approaches to 1 in the constant throughput case, which implies UCEN is beneficial for clients uploading their contents to very slow servers.

3.4.3 Traffic Peak Reduction

We also evaluate the impact of flattening traffic peak for the gateway as we do not have much information about the traffic at the server side. Let the deadline of delaying the actual upload be an hour, we can have a simple heuristic algorithm on scheduled upload. When the gateway has received a C-POST request from the client, it schedules the actual upload to the time slot that has least scheduled traffic currently. Upload scheduling simulation is done according to the captured traffic on 10-minute periods with no background traffic assumed.

The result of scheduled upload is illustrated in Figure 3.4 and Figure 3.5. Without upload scheduling, the highest traffic on the edge router is 2.14 MB/s. This peak, as shown in Figure 3.4, is reduced by scheduled upload by 49% to 1.10 MB/s. The trade-off results in delayed upload to the destination server. Figure 3.5 shows the distribution of the delay, where the median delay is 48 minutes. Because only a simple heuristic algorithm is used in the simulation, we can expect further reduction and less delay

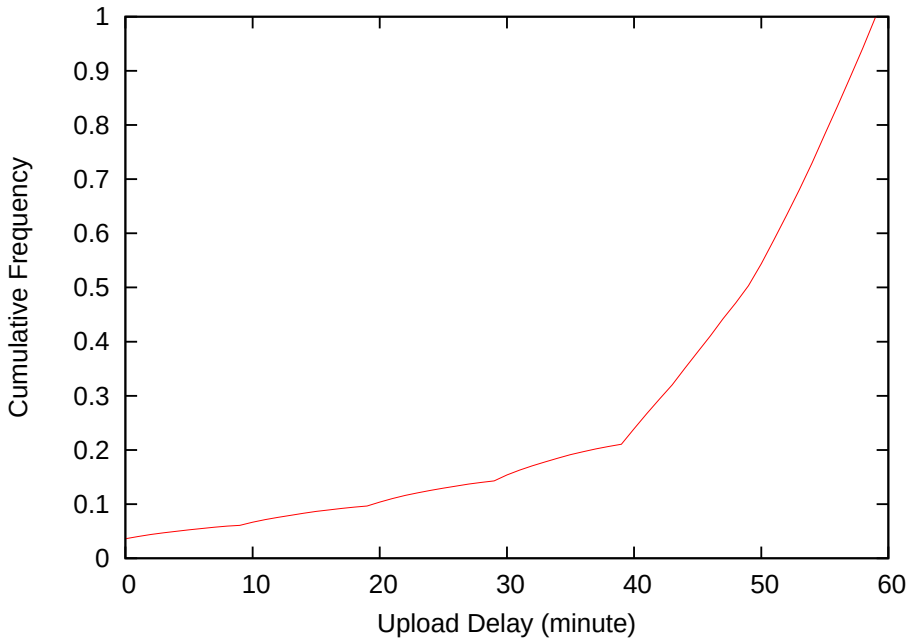


Figure 3.5: Distribution of upload delay due to scheduled upload

time resulting from the scheduled upload feature.

3.5 Summary

In this chapter, we have presented a new cache mechanism – Upload Caching in Edge Networks (UCEN). UCEN utilizes light-weight in-network caching to assist publishing of user-generated contents (UGC), and achieves benefits for both end users and service providers without exposing privacy information of the end users to access service providers.

For end users, UCEN significantly shortens the duration while the end users must stay online for uploading their generated content. For service providers, UCEN flattens the traffic peak significantly (as much as half, as shown in our simulation) for the edge networks or the destination servers. Such caching functionality is achieved by indirectly uploading to the gateways in edge networks with the token retrieved from destination servers and leaving gateways to complete the actual upload to the destination servers. The gateway schedules its delayed upload to the destination servers without incurring much extra delay in the total elapsed time for the upload, and without the involvement of the end users. The gateway also delegates the destination server for serving the uploaded content in the edge network, if it is cache-able.

Besides being a caching policy for ICN, this design can also be implemented as an extension to HTTP request, called Cached-POST. In order to quantify the effectiveness of the proposed architecture, we have conducted a series of simulations based on the actual traffic trace. Our evaluation shows that 41% of upload traffic has the holding time reduced by half by using C-POST method, so that users can leave a network quickly after the uploading, and peak traffic is reduced by 49% via scheduling upload time, thus, saves the capital expense for improving bandwidth.

Uploading support, including upload caching, is a functionality missing in the current ICN architecture proposals. We believe the upload caching mechanism proposed, UCEN, is an indispensable feature for meeting the requirements for today's communication patterns, especially dealing with growing UGC traffic, and without this feature, ICN would not be widely deployed as a future Internet architecture.

Chapter 4

Content-Oriented Caching with In-Network Index

4.1 Introduction

In this chapter, we propose *Content-Oriented Caching with In-Network Index* (COCINI), a *deployable* and *self-scaling* scheme exploiting spare storage and bandwidth from end-systems, to eliminate redundant traffic and to enable efficient and fast access of content in ICN. In contrast to the existing approaches lacking the perspective of deployability, COCINI is designed to be incrementally implemented and deployed, for example (but not limited to), from the edge of the Internet.

Also, In COCINI, while the content itself is cached on individual clients, the functionalities of redirection and corresponding indexing are integrated into ICN routers. This design eliminates the cost for storage space and the drawback of single-point-of-failure, as observed in peer-to-peer's *self-scaling* architecture: the more users donate storage capacity, the more efficient the system becomes.

In a nutshell, while a COCINI-capable client issues the request for content to the origin server as they usually do, the request is redirected to the nearest client that holds the cache of the content if it exists, according to the index of COCINI-capable routers. Since the redirection decision is made within the COCINI-capable routers, the topology information they have already collected for forwarding traffic can be utilized to optimize the decision. As a result of the proposed content-based redirection and caching, COCINI can eliminate the redundant traffic and to enable faster access to the cached content localized around the users.

4.2 Related Work

In response to the content-oriented trend of the modern usage of the Internet, many research efforts have attempted to adjust and redesign our communication model. However, we posit that the existing proposals are either a clean-slate approach hardly deployable in a wide area and in a short term or a partial solution still with major drawbacks unresolved, such as high cost for resource management. While the concepts such as name-based routing [25] and hash-based addressing [26] have been proposed earlier, several content-oriented schemes have been recently proposed [27, 28]. However, it is hard to put such clean-slate thinking into practice in a large scale and in a short period of time. The existing work also lacks the perspective for incremental deployment of such clean-slate designs.

On the other hand, a large body of work has been proposed to remove redundant traffic from the Internet through cache proxies such as web proxies [29], object caches [30], packet caches [31, 32], wide-area content delivery networks [33, 3], and etc. However, these techniques are typically developed for the sake of servers and their deployment and administration are often in the hands of the servers or third parties such as CDN providers, and not controlled by clients at the very edge of the network. Thus, the existing work lacks the design for edge network operators to be able to deploy effective elimination of redundant traffic by themselves.

4.3 System Design

One of the most important goals in COCINI is to enable clients to retrieve content from the nearest possible location. We face two challenges in achieving this goal:

1. to make the content and its copies discoverable and retrievable for clients,
2. to realize the content retrieval as a deployable mechanism over the current Internet infrastructure.

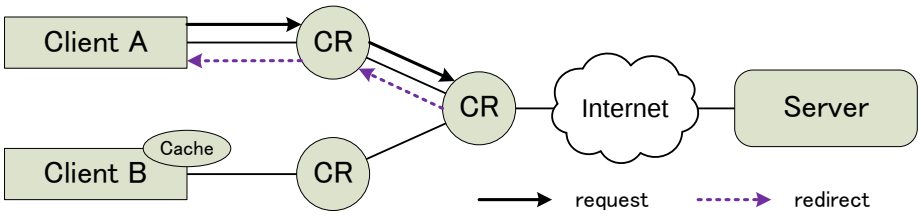
This part first describes the scheme for the content discovery and retrieval and then introduces modifications required on both routers and clients for the COCINI design.

4.3.1 Scheme Overview

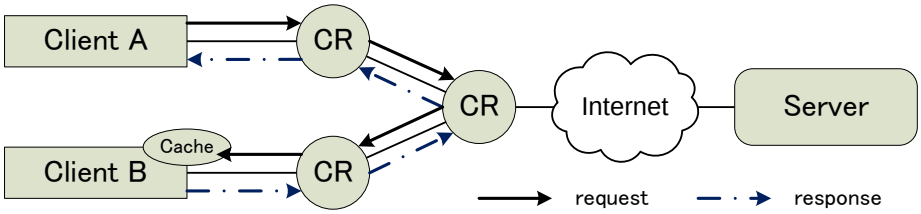
In COCINI, a piece of content is addressed by its globally unique *name*. An end host is either a *client* end-point or a *server* end-point according to its role in the content delivery. A server end-point is the host where the content is published, i.e., a principal of the content or a public forwarder of the content. A client end-point is the host that retrieves the content. A client end-point may, and must if no other host has, retrieve some content directly from the server end-point as in the current Internet. After the content is downloaded, its local copy is cached so that the client end-point may serve it to other client end-points later. Thus, contents are delivered not only from a server to a client, but also among clients.

When a local copy of some content exists, retrieval of the content is processed as depicted in Figure 4.1. A client, without knowing there exists the local copy, sends a *request* message towards the original server that holds the content. As the request is being forwarded via the network, it traverses through several *Content-Oriented Redirectors* (CRs). A CR looks up its local index table for the copies of content according to the name carried in the request message. If it finds a local copy, as in Figure 4.1(a), a *redirect* message is sent towards the client issuing the request, telling it the location of the local copy. Meanwhile, the original request message is suppressed at the CR. The client then follows the instruction of the CR, sending another request message towards the identified local copy. Upon receiving this request, a *response* message is generated and returned with the content. As described in Figure 4.1(b), the entire process completes without the involvement of the server.

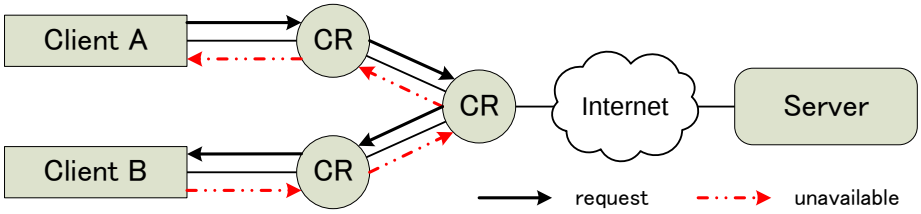
Since client may be disconnected from the network and cache entries may be get evicted, the local copies of content may become unavailable. However, until the CR gets notified, it still instructs clients to redirect their requests to the unavailable local copies. In this case, the client that receives such requests sends *unavailable* messages to both the requesting clients and CR, as shown in Figure 4.1(c). Upon receiving an unavailable message or when none of the traversed CRs can redirect the message, the request message sent by the client reaches the original server. As Figure 4.1(d) depicts, the server processes the request and responds to the client with the requested content. The reliability of content delivery in COCINI is no worse than that of the current Internet, since CRs in COCINI fall back on the original server to look for the content.



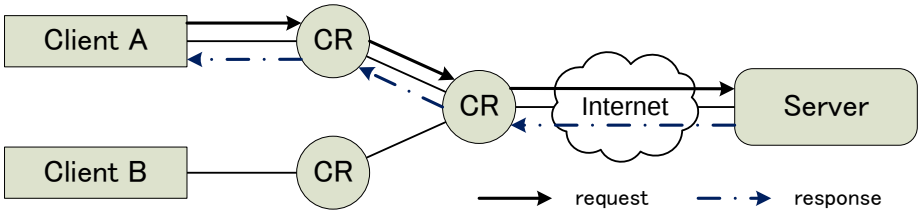
(a) Client A sends a request to the server; CR instructs A to redirect this request to client B



(b) Client A retrieves the content from client B



(c) Cache on client B is announced as "unavailable" to CRs and clients



(d) Client A retrieves the content from the server as the last resort

Figure 4.1: COCINI messages and their delivery

To summarize, there are four basic primitives in COCINI, as shown in Table 4.1. All the COCINI messages must carry names. The primitive *request* is the only one that takes optional arguments to indicate whether this request is redirect-able and who advises the redirection in case of a redirected request.

4.3.2 Content-Oriented Redirection

As described above, COCINI aims to work with the current Internet. As a result, CRs are almost the same as the current IP routers except that they have a cache index and are capable of processing COCINI messages. The goal of CRs is to provide

Table 4.1: Basic primitives in COCINI

message	arguments
request	name, (redirect flag), (redirect advisor)
redirect	name, location
response	name, content data
unavailable	name

clients cache information in local network as precise as possible with limited cache index capacity and computation overhead. The task of a CR may be divided into three parts: *index manipulation*, *cache redirection* and *implicit cooperation*.

Index Creation and Removal

While the content cache itself is stored in clients, a CR needs to populate the index of caches before advising clients to redirect. Storing only the index on CR saves storage space, but even so the storage space may become scarce resources in routers. For this reason, CRs should store only index entries useful to themselves.

When a response message traverses multiple CRs, those CRs infer that both the source S and the destination D have cached the named content. For creation of a cache index, a CR looks up routing information of S and D to calculate distances to them, l_S and l_D. Usually, "AS PATH" length is applied as the distance metric. But if both S and D are located in the same Autonomous System (AS) with CR, intra-domain routing metric is used. Indices for the content on S and D are created with the probability p_S and p_D, respectively,

$$\begin{cases} p_S = \dfrac{l_D}{l_S + l_D} \\ p_D = \dfrac{l_S}{l_S + l_D} \end{cases} \tag{4.1}$$

If a new index entry is created while the index storage is full, some index entries must be evicted. Replacement algorithms like Least Recently Used (LRU) [34] can be adopted to achieve this. When an unavailable message is received, a CR also needs to remove the corresponding index entry.

Cache Redirection

A CR's decision on redirection to the cached content is necessary when a request message traverses the CR and the CR holds some index entries to the requested content.

The target of redirection is selected from the set $\mathbf{C} = \{C_0 = D, C_1, \ldots, C_n\}$, where D is the original destination and $C_1, \ldots, C_n$ are hosts that have cached the requested content according to the index of the CR. The selection algorithm is as follows,

1. Calculate AS PATH length, l_i, from C_i to the requesting client S.
2. If S is located in the same AS as CR, and $\min(\{l_i\}) = 0$, calculate hop count h_i for all C_i where $l_i = 0$, and define $\mathbf{C}' = \{C_i : h_i = \min(\{h_j : l_j = 0\})\}$; otherwise, define $\mathbf{C}' = \{C_i : l_i = \min(\{l_j\})\}$.
3. If $D \in \mathbf{C}'$, pick up D as the target; otherwise, randomly pick one from $\mathbf{C}'$.

If the target is the original destination D, the CR simply forwards the request; otherwise, the CR takes the request and sends a redirect message.

Implicit Cooperation

CRs work independently of each other. To be more precise, CRs do not explicitly coordinate with each other except exchanging routing information just as conventional routers do. This means CRs incur neither network nor computational overhead.

However, CRs implicitly cooperate in COCINI. A client does not specify which CR to serve it but sends the request message towards the server through multiple CR. When a CR receives the request, it can tell that none of CRs by far had a corresponding index entry. Thus, different CRs never deal with the same request.

A response message may traverse multiple CRs as well. Different CRs populate different indices according to different weight parameters set in their algorithms. Also, the indices to the content may be distributed over different CRs according to the access pattern to the content. For example, if a small group of clients, e.g., within the same network send a request for some content and it is fulfilled by the CR closest to the group, the CR keeps the index to the content but the other CRs beyond it do not keep the index to the content. However, if a large group of clients distributed across multiple networks requested for the content, the number of CRs that store the index to the content may increase and their locations may vary.

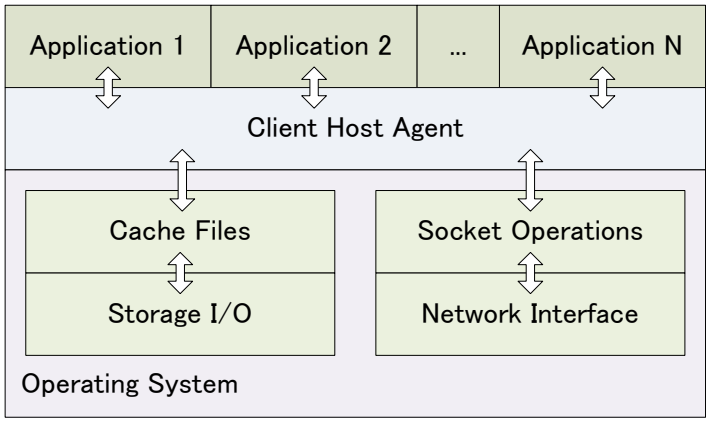


Figure 4.2: Layout of a COCINI client host agent

4.3.3 Client Host Agent

A COCINI client host agent is installed in each client host to manage local cache files and network communications. As described in Figure 4.2, the agent lies between applications and operating systems.

As illustrated in Figure 4.2, all the requests to retrieve content, from whatever application and via whichever protocol they come, e.g., HTTP, FTP, BitTorrent, eMule, etc., go through the client host agent instead of directly invoking socket operations. The client host agent first looks up the local cache files to see if the content data with the same name is available. If a local copy is found, it is provided to the application, finishing the transaction.

If the client host agent finds another pending request for the same content, it provides the content as soon as the previous request is finished. If neither a local copy nor a pending request exists, the client host agent encapsulates the request in a COCINI-compatible format and sends it to the network.

When a response with content data arrives, the client host agent creates a new cache file containing the payload of this response, while providing this response to the corresponding application.

A request for content may also be issued by another COCINI client and be arriving from the network. In this case, the client host agent only looks up its local cache files for the corresponding content. If the requested content is available, a response message is generated with its payload filled with the content data and is sent to the host that has issued the quest; otherwise, an unavailable message is sent back.

A COCINI client host agent may be either implemented as a middle-ware like proxy software for rapid deployment, or may be integrated as system-calls in operating systems. As opposed to the middle-ware design that takes over the control of network protocols, the system-call approach benefits from not only the performance boost and bandwidth saving, but also from simplifying network programming and removing manipulation of the proxy protocols.

4.4 Evaluation

We apply workload data from the real network to evaluate COCINI's effectiveness, measuring to what extent COCINI can reduce inter-network traffic and can improve content delivery efficiency with reasonable deploy cost. We implement a session-level simulator in C++ to achieve this and focus on Web contents although the COCINI scheme is generic enough to extend to the other protocols.

4.4.1 Experiment Setup

For our simulation, we replay the real traffic trace captured at a campus network. Besides the traffic behavior itself, the evaluation is also sensitive to the network topology. We apply "traceroute" from a series of hosts outside of the campus to all recorded clients and infer the network topology, finding 133 routers and inter-router links, as is illustrated in Figure 4.3.

The capacity of index impacts the overall performance of CRs, directly affecting the cache hit ratio. In our experiment, we choose index capacity from 10 entries to 10^9 at each CR to evaluate the performance of the COCINI scheme. We also compare the case where all the 133 routers are COCINI-enabled (denoted as Multi-CR below) with the case where only the upstream edge router is so (denoted as One CR below).

The simulator replays the sessions in the traffic trace by issuing requests for specific content from the corresponding client nodes in exactly the same timing as recorded. For each session, we examine whether its content is retrieved from a cache on a client or from the server, how much inter-network traffic is generated, and also the time elapsed to complete.

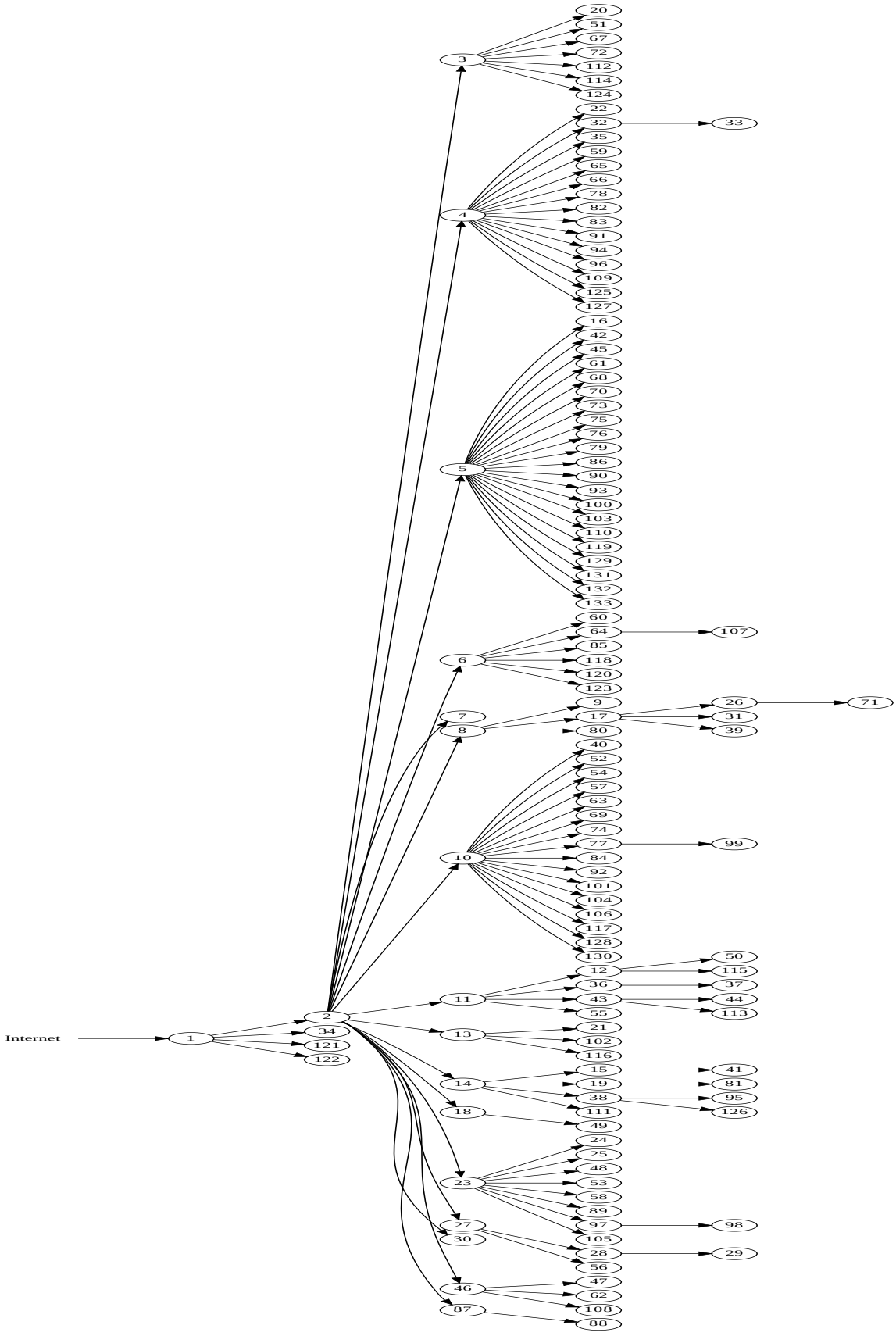


Figure 4.3: Network topology used in experiment simulation

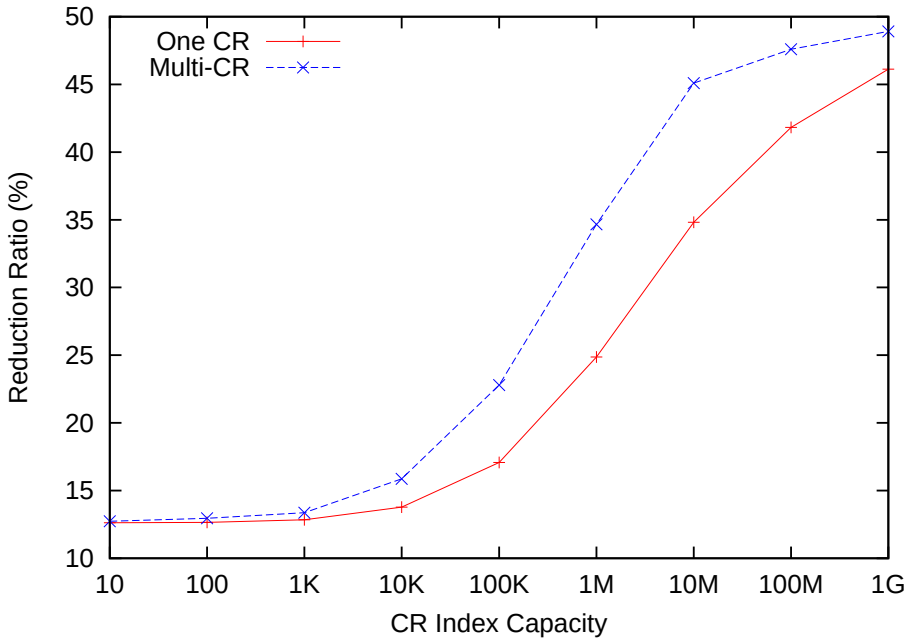


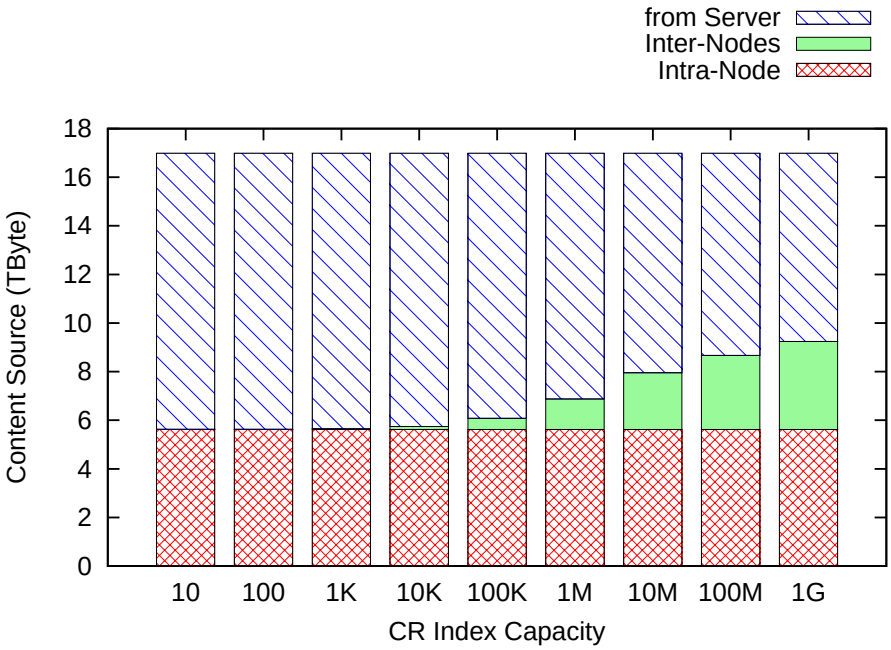
Figure 4.4: Traffic reduction ratio in two scenarios

4.4.2 Result and Analysis

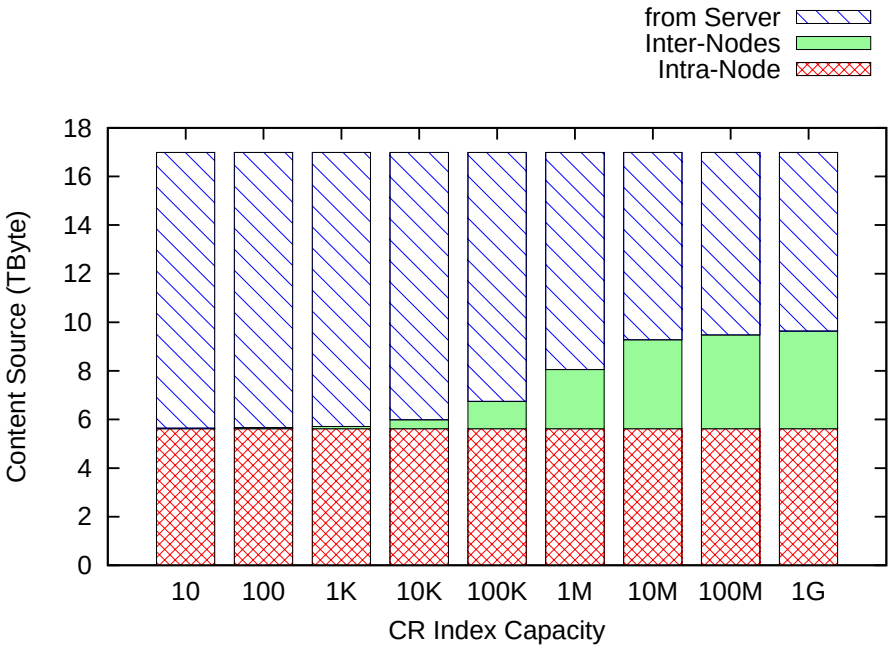
Figure 4.4 describes how much traffic for downloading content from outside of the network is reduced as the index capacity of CRs increases. This result gives us two observations.

First, index capacity is important to the traffic reduction ratio. Only less than 12% traffic is reduced when a CR hosts up to 10 index entries, and the reduction ratio is improved over 45% when the capacity increases to 10^9 entries. This explains how it is significant to store only the index instead of the whole content in CRs that would require much more storage capacity. An index can be accommodated within hundreds of bytes while the average size of an HTTP object is measured 88 kbytes on average in our trace. Therefore, a CR with 1 Gbyte memory can contain about 10^7 index entries, while a cache server with the same storage can accommodate only 10,000 Web contents. According to Figure 4.4, with the same 1 Gbyte storage, a cache server achieves a little higher than 13% traffic reduction rate, which is 21% lower than a single CR case can achieve.

Second, even though CRs perform their parts independently, without coordinating with each other, Multi-CR can achieve obviously higher traffic reduction rate when the index capacity is larger than 10^4. For example, Multi-CR with 10^7 entries at each CR



(a) One CR scenario



(b) Multi-CR scenario

Figure 4.5: Distribution of content sources in two scenarios

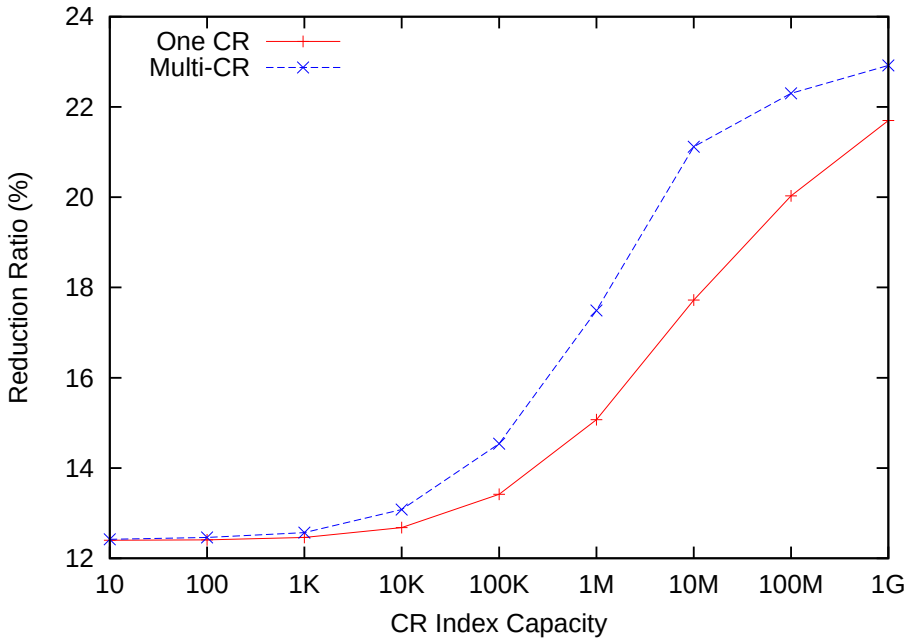


Figure 4.6: Reduction ratio of cumulated session time in two scenarios

works almost the same as one CR with 10^9 entries. Although complex coordination among CRs is possible, this result verifies the effectiveness of our simple design of COCINI without coordination in terms of reduction of redundant traffic.

We further explore the detail of the traffic reduction. As shown if Figure 4.5, besides retrieving the content directly from the original server, there are two alternative ways: a) retrieving the content from a client according to CRs' advice (inter-node retrieval), or b) using locally available cache (intra-node access).

Interestingly, Figure 4.5 shows that the contribution of the intra-node caching stays constant about 5.5 Tbytes on average, and accounts for significant portion of the total traffic reduction. This implies that we can eliminate this portion of traffic even from the local network, since COCINI enables a client to access the content from itself and there is no traffic generated outside. Also, if the index capacity at CRs increases, the contribution of inter-node content retrieval becomes significant, e.g., as much as 4 Tbytes with 10^9 index entry capacity.

This result verifies the effectiveness of our COCINI design that every client may delegate cache management and spare storage space. Furthermore, intra-node contribution frees CRs from maintaining index entries that only a single host may be interested in.

Not only network operators can benefit from COCINI through reducing redundant traffic, the efficiency of content retrieval is also improved. As shown in Figure 4.6, average content retrieval latency is reduced by from 12% to 23% with different index capacities in One CR and Multi-CR scenarios. This result verifies that COCINI achieves efficient and fast access to the content.

4.5 Summary

In this chapter, we have presented Content-Oriented Caching with In-Network Index (COCINI) that achieves a *deployable* and *self-scaling* solution for eliminating redundant traffic and enabling efficient and fast access to content. Unlike traditional caching mechanisms where both cache storage and index are located in the same node, COCINI is innovative in the sense that it aggressively caches the content in the network to reduce redundant traffic and to facilitate fast access to the content from the nearest possible end systems, such as a client with the content cached, or an origin server of the content. However, COCINI significantly differs from the caching policies in other ICN in its deployable and self-scalable design as follows. COCINI only requires a minimal change in a router, i.e., indexing the content accessed by the end-systems, thus, it can be incrementally deployed, e.g., from the edge of the Internet. Also, COCINI exploits spare storage on end-systems for caching the content and unused bandwidth for accessing it so that a COCINI-enabled router (CR) intercepts the access to the content and redirects it to the nearest end-system holding its cached copy. As a result, COCINI is not just scalable but *self-scaling* in its nature — the more end-systems enable COCINI, the more cache storage and network resources COCINI can utilize.

In order to quantify the effectiveness of the proposed scheme, we have conducted a simulation using real-world traffic traces. Our evaluation shows that through applying COCINI to the real-world campus network environment, 12% Web traffic can be reduced with index capacity of 10 index entries, and the reduction rate rises to 49% when with index capacities of 10^9 index entries. The cumulative latency in accessing content can also be shortened by about 20%. This means a significant reduction of cost for bandwidth enhancement and implies the improvement of user experiences when retrieving contents.

The result confirms the significance of the caching scheme introduced in COCINI

that separates the storage and index of in-network caching system. Efficient caching is achieved in COCINI by exploiting spare storage and bandwidth from end-systems and only storing cache index via minimal change in routers. That is to say, COCINI is a require caching policy for an ICN architecture, without which content delivery would not be achieved in a scalable manner.

Chapter 5

Information-Centric Transport Protocol

5.1 Introduction

In this chapter, we propose *Information-Centric Transport Protocol* (ICTP), a transport layer protocol to support most features of ICN over the current Internet infrastructure by embracing following features:

1. ICTP is designed to be compatible with the current Internet Protocol (IP), with the ability to be incrementally implemented and deployed.

2. ICTP is a protocol abandoning socket addresses. All operations in this protocol are content-oriented and connection-less.

3. ICTP includes congestion control to be fair with existing transport protocols and enable traffic engineering.

We posit that it is the current transport protocols that are based on socket addresses (a tuple of an IP address and a port), that constrain the realization of information-centric communication. For example, in COCINI scheme introduced in Chapter 4, CR needs to suppress the request sent to the original server and advise the client to send a new one (as depicted in Figure 4.1), because all the messages are expected to send with socket addresses and are difficult to be redirected transparently to the client.

As far as we know, ICTP is the first transport protocol operable today that supports a variety of ICN concepts.

5.2 Related Work

All transport protocols widely used nowadays, whether connection-oriented ones, such as Transmission Control Protocol (TCP) [35] and Stream Control Transmission Protocol (SCTP) [36], or connection-less ones, such as User Datagram Protocol (UDP) [37], are operated based on Berkeley sockets [38]. Socket addresses, which are usually presented as a tuple of an IP address and a port, are used as the identifiers of data communication in these protocols. Thus, these communications are constrained to local and remote socket addresses, which obviously leads to the current host-based communication model.

Besides the constraint brought from using socket addresses, these protocols are not in favor of supporting information-centric communication since they treat their payloads as arbitrary byte flows. This behavior in these protocols has two problems. First, the identity of information objects is excluded during transfer. Thus, information transfer is opaque to the whole network it traverses except negotiating endpoints and thus it is impossible for intelligent devices within network to advise a better location for information access than the original destination. Second, the security of transfer relies on untrustworthy connection information such as is done in Transport Layer Security (TLS) [39], which ought to be placed in the transferred information object itself.

In the light of this observation, we come up with the design decision that a transport protocol supporting information-centric transport should abandon Berkeley sockets and identify the information object it is carrying.

5.3 System Design

In this part, we describe major design decisions we have made in designing ICTP to realize information-centric transfer over current Internet infrastructure.

5.3.1 Naming of Information Objects

Just as socket addresses are the bases of today's host-based communication, information objects will be the foundation of information-centric transport. A precise definition of information object is required in designing the protocol.

A piece of information object is uniquely identified its *name*. ICTP adopts a human-readable, hierarchical naming system, where a name is composed of two parts, the

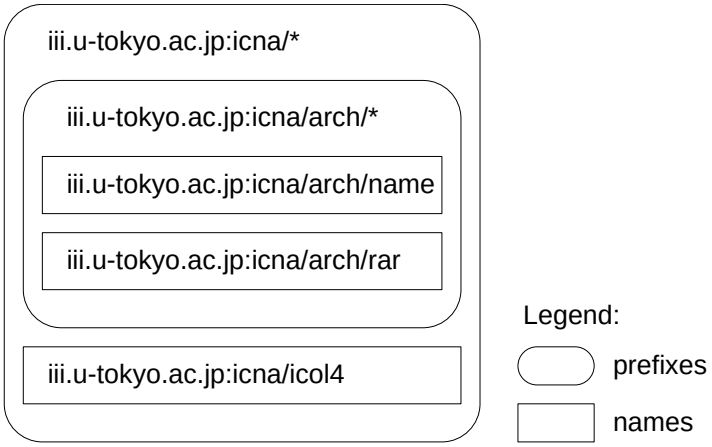


Figure 5.1: An example of names owned by the publisher "iii.u-tokyo.ac.jp"

publisher identifier and the publisher-dependent name. The former is a registered global-unique identifier that is similar to a domain name today and corresponds to a certificate issued by the registration organization so that content and services can be signed and verified, while the latter is a segmented name that is arbitrarily specified by the publisher to describe a specific piece of content or service provided by that publisher. The fact that publishers can arbitrarily specify the publisher-dependent name brings two advantages. First, although network devices will treat names as flat ones for faster processing, a publisher can still organize internal structure of naming (for example, including some version information to update its information with the same name) according to the actual requirement of the publisher or its applications independent from other publishers. Second, different publishers can publish information under the same name, which enables information access when the user has only the information about the name but does not know or care who the publisher of some piece of information object is. To represent a group of contents or services that share similar features, publishers can use a *prefix* that is composed by the publisher identifier and the prefix part of the publisher-dependent names. An example of names and prefixes is shown in Figure 5.1.

Sometimes, it is not necessary to acquire the whole piece of information, especially when it is very large, such as multimedia contents. ICTP supports the partial transfer of content by defining *partitions* of a piece of information object as the minimum unit in the content transfer. Except the last partition (or information objects with only one partition), all partitions are in size of 2^n bytes, which is assigned by the publisher

ranging from 1 kbyte to 1 Mbyte. In current prototype implementation, there may be at most 32767 partitions in a piece of information object, that is, the maximum length of any piece of information object is 32 Gbyte, sufficient for use because it is even larger than a typical Blu-ray disk (with 25 Gbyte capacity).

5.3.2 Enabling In-Network Processing

One of the most essential things for supporting ICN architectures is that in-network processing should be enabled by the transport protocol. That is to say, devices within network should be able to know which piece of information object is being transferred by some packet. Moreover, such information should be able to be acquired without deep packet inspection (DPI) as it will bring devices high computation overhead and thus harm the scalability of the architecture.

ICTP fulfills this requirement by carrying necessary information to identify a piece of information object in packets. As a transport protocol, it is intuitive that ICTP supports incremental deployment as it carries all information in layer-4 headers. Network devices that do not support ICTP (such as traditional IP routers) can simply forward packet according to IP headers and ignore ICTP headers as they typical do with today's transport protocols. On the other hand, intelligent network devices supporting ICTP can do shallow packet inspection (SPI) to acquire necessary information, and this would not bring much overhead to those devices thanks to the information located in ICTP headers with constant structure.

Despite that ICTP is a transport protocol that expects in-network processing to assist information delivery procedure, the protocol does not specify the forwarding strategy that should be adopted by network devices. Network operators, or network users themselves if they are using slice-based network facilities, can decide the forwarding strategy network by network as long as it would not tamper the reliability of the communication. That is, the forwarding strategies of ICTP must satisfy the following two rules. First, we enforce forwarding to be loop-less. Second, fall-back forwarding using IP header information ensures the reachability to a destination, not necessarily the optimal one.

5.3.3 Information Security

Content-based security is adopted in most ICN architectures. That is to say, the protection and trust of content is convinced by the transferred content itself, and not by the transferring method or its endpoints. Information security in ICTP also follows this concept, where all information objects are mandatorily authenticated with digital signatures, and private information is optionally protected with encryption.

In contrast to TLS (and some ICN architectures), ICTP does not perform per-packet signatures, but requires all partitions to be publicly authenticable instead. Each partition transferred in ICTP carries a small amount of auxiliary data authenticating the binding between the name of the piece of information object and the actual data of the partition. For example, publisher can use standard public key signatures to generate signed checksum for every partition and append it to the partitioned data. Anyone that retrieved the partition with its signed checksum can verify the identifier-data binding is signed by a specific key of the publisher. We will not elaborate how to establish trust in keys as this issue is already discussed in the other literature, and there are existing models, for example, Simple Public Key Infrastructure (SPKI) [40] that can fulfill the content security requirements in ICTP.

The situation is more complicated if the ICN architecture allows clients to send user-generated data with names registered by service providers. In this case, there is a mismatch between the owner of the name (service provider) and the owner of the data (end user). First, this may be considered a necessary trade-off for allowing user-generated content pushing in ICN. The security threats, such as denial-of-service attack, can be taken care of by other mechanisms. We also argue that such threats cannot be completely eliminated even without the content uploading functions. Second, although there is problem using SPKI in this case, adopting Certificate Authorities (CA) approach of public-key infrastructure [41] resolves the validation problem without introducing much additional complexity. In brief, the service providers play the role of CA, signing and publishing the public keys bound to users they provide services, so that ICTP routers can verify that an access message with data is sent by a valid user of the designated service name in the message header.

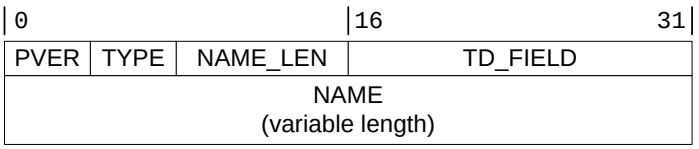


Figure 5.2: Message header fields of ICTP

5.4 Protocol Specifications

ICTP is designed to operate on top of the current network layer protocols such as IP and deliver information objects over the existing network infrastructures. This part describes message formats and the detailed transfer processes of ICTP.

5.4.1 Message Formats

ICTP has four types of messages, which are *registration*, *access*, *result* and *transfer control*. An end host sends registration messages to declare the availability information of some piece of information object to neighboring hosts in edge networks. The destinations of these messages are all the hosts on the same network (i.e., 224.0.0.1 in IPv4). Users of a host, or their agent applications, can configure which pieces of information objects are announced by the protocol. Access messages are sent in order to retrieve a specified piece of information object, and result messages are sent in response to request messages to provide a specified piece of information object. Transmission in ICTP is conducted in unit of partitions. Hence, the transmission procedure of a partition may span across several messages, as each message has to fit into a network layer packet. The detail of fragmentation and transfer control message will be introduced later.

ICTP message header follows lower-layer header (e.g., IP header in the current Internet) in network transport to facilitate information-centric in network processing. Although ICTP has four primitives, message header processing is simplified by introducing a common message header format for all types of messages. As shown in Fig. 5.2, the message header includes four fixed-length fields and a variable-length field.

The first 4-bit field `PVER` has a constant value of 2, indicating that the following message is an ICTP message. The second 4-bit field `TYPE` indicates the message type. The 1-byte field following these is `NAME_LEN`, which specifies length of the name carried in the message. And the last fixed-length field is a 2-byte `TD_FIELD`, a type-dependent field. In register messages, the field carries attributes of registration and registered

names; in access messages, the field carries the *access ID*; and in result messages and transfer control messages, the field is used for explicit transfer control flags. The last field in the message header is a variable-length field carrying the DSINA name, whose length is specified in `NAME_LEN`. This field finishes with the character "`#`" if it is a full name and finishes with the character "`*`" if it is a prefix.

In one word, only necessary information for in-network processing is carried in the message header. All the remaining information in the messages, for example, the authentication information of the payload data, is contained in message body in the format not specified by the transport protocol but the applications, to ensure the extendibility of the architecture.

5.4.2 Fragmentation

Many ICN architectures support content partitioning to enhance content retrieval flexibility and to refine the transfer unit. However, many of them fail to take the necessity of content data spanning various messages, and some architectures suppose the size of a content partition fit into a single message. Considering packet size of the current Internet, such design will result in significant overhead brought by authentication information and so on. Therefore, it is necessary for ICTP to support fragmentation in transferring bulk data.

The challenge of supporting fragmentation in ICTP is that routers have to guarantee that fragmented messages for the same access request are forwarded to the same content or service end-point, while at the same time to retain the information-centric advantage that the forwarding should be independent of addresses and automatically load-balanced. ICTP solves this problem using the *access ID* carried in the `TD_FIELD` of access messages. A client always send fragment messages for the same access request with the same (*name*, *access ID*) tuple, and a router should always forward access messages with the same (*name*, *access ID*) tuple to the same destination. In this way, ICTP always transfer a series of fragmented access messages to the same content or service end-points.

5.4.3 Transfer Control

The purpose of the transfer control in ICTP is two-fold, reliability assurance and congestion avoidance. The former guarantees that recipient can retrieve content effi-

Table 5.1: Four transfer control instructions

instruction	usage
`feedback`	updates necessary information for congestion avoidance
`jump`	skips to a specified fragment for retransmission or partial retrieval
`stop`	terminates the current transmission
`new_addr`	resumes the current transmission to a new address

ciently while the latter prevents networks from congestion collapse. Both of the functions are carried out in ICTP by the sender of result messages, by reacting to transfer control messages sent by recipients of content data.

Recipients provide required information for transfer control with four instructions described in Table 5.1. The `feedback` instruction provides information for congestion avoidance, such as the timestamp of the last fragment received, the transmission rate estimated by the receiver, loss event rate, and so on. When the recipient wants to terminate the transmission procedure as it has already held the whole partition or the partition is no longer required, a control message with the `stop` instruction is sent to indicate the termination of current transmission. Sometimes, out-of-order transfer and packet loss happen in networks as ICTP operates on unreliable network layer protocols. Control messages with `jump` instruction signify the occurrence of such incidents and indicate the next expected fragment in the partition. Furthermore, the `new_addr` instruction is used to enable information delivery in mobile environments where the addresses of clients change frequently.

Generally speaking, ICTP adopts TCP-Friendly Rate Control (TFRC) [42] to retain the fairness of transmission between ICTP and other transport protocol. Thus, ICTP can operate efficiently even when there is no network device supporting this protocol. Moreover, borrowing ideas from explicit congestion notification (ECN) [43] and its latest research work [44], ICTP utilizes explicit congestion level indication to allow senders adaptively adjust sending rate according to different congestion levels on the bottleneck routers. If the congestion level of a router is higher than that indicated in the messages, the router will replace the congestion level value with that of local status. In this way, the recipient of a result message will be aware of the status of the most congested routers in the networks. The sender of result messages can accurately

estimate the congestion level with the existence of such indicator and will rapidly adjust to the optimal sending rate.

5.5 Summary

Research efforts on ICN architectures are often in favor of clean-slate designs, which consequently result in difficulty in the deployment in today's Internet infrastructure. In this chapter, we posit that it is the existing transport protocols that constrain the development of content-oriented networks with current practice. Accordingly, we propose the Information-Centric Transport Protocol (ICTP) to support most features of ICN architectures over the current Internet infrastructures. We have presented the design decisions and the protocol specifications, showing that the proposed protocol is compatible with the current IP and can be incrementally implemented and deployed. In-network processing of information-centric strategies can be benefited from the genuine connection-less feature of the protocol. Moreover, the protocol achieves a congestion control mechanism that is fairness with the existing transport protocol, TCP, which is a necessary feature to make the protocol co-exists with today's Internet.

ICTP is the first transport protocol operable today that supports a variety of ICN concepts. The protocol provides an indispensable "narrow-waist", namely, fundamental general transport mechanism for various kinds of ICN architectures that is deployable in the current Internet infrastructure and satisfies the requirement to be TCP-friendly.

Chapter 6

Distributed Resolution Service

6.1 Introduction

In this chapter, we take a novel approach to the challenges of feasible and efficient name resolution service. Because of the similarity between the naming scheme in ICN and Uniform Resource Identifier (URI) [45], we evaluate the time and space complexity of a typical data structure for lookup of ICN names using URIs collected from the traffic trace of a campus backbone network. We propose a simple and empirical *Distributed Resolution Service* (DRS) scheme according to the time and space complexity model. Finally, we optimize the proposed distributed resolution service scheme with regard to network performance.

Despite the differences in architectural designs of the ICN proposals, almost all of them share the features of adopting *names* as globally available identifiers of information and supporting the *route-by-name* scheme in the network, which means that network devices should forward data according to the names rather than the destination addresses as it is now.

However, such systems face feasibility and efficiency challenges, since the number of information objects (and as a result, names) is large already and growing rapidly. People propose that any of such systems should be prepared to handle *at least* 10^{12} entries [46], estimated by the current size of the Web. Moreover, the number may be still conservative because of rapid proliferation of mobile devices and wide deployment of Internet-of-Things (IoT).

6.2 Related Work

The existing research efforts dealing with this problem can be categorized into two. One reduces the number of name entries in the world by aggregating entries, as in CCN [6] and "deepest match" [9]. In such systems, names share similar characteristics are combined in the Forward Information Base (FIB) to reduce the total entry number. The other provides name resolution service in a distributed manner, especially, adopting an improved form of Distributed Hash Table (DHT) [10, 11].

Both of these two kinds of research efforts reveal the correct directions of implementing a scalable name resolution service, but it remains unclear how much cost is required for distributed resolution service, as well as its impact to network performance. Some preliminary assessments have been conducted [5, 10], but special equipment, e.g., data center and large-volume Solid State Disk (SSD), are assumed to be required.

6.3 Characterization of Name Resolution

To study the characteristics of name resolution without globally deployed ICN yet, We elect to use Uniform Resource Identifier (URI) [45] as a substitution of name in ICN not only because both URI and name in ICN specify information object, but also because the generic URI syntax consists of components such as authority and path, which is exactly the same as the structure of human readable names proposed for ICN (e.g., that defined in Chapter 5 and also in NDN [6]). URI can also be transformed into the self-certifying name form P:L (where P is a cryptographic hash of the principal's public key, and L is the label assigned by the principal) [5, 9], although it is out of scope of this article because security mechanisms are orthogonal to the part of name resolution and have no influence on it.

6.3.1 URI Properties

The popularity of recorded URIs, which represents that of Web content as has been studied in Web proxy workloads [4], follows the "Zipf-like" distribution as follows

$$f = cr^{\beta} \tag{6.1}$$

where f is the frequency of occurrence and r is the relative rank. As illustrated in Figure 6.1, the result of linear regression indicates $\beta = -0.8304$ for the recorded URIs.

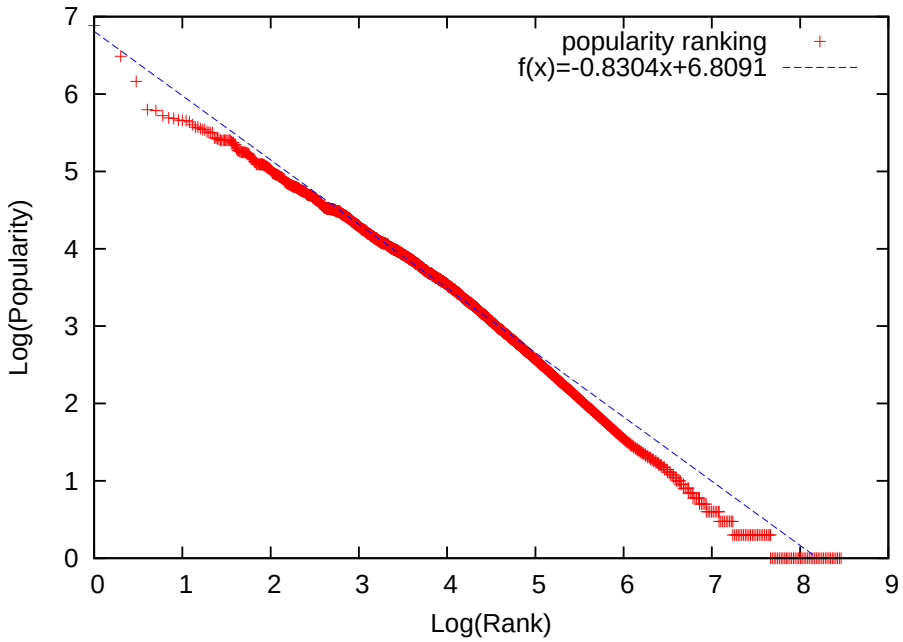


Figure 6.1: Popularity ranking for recorded URIs

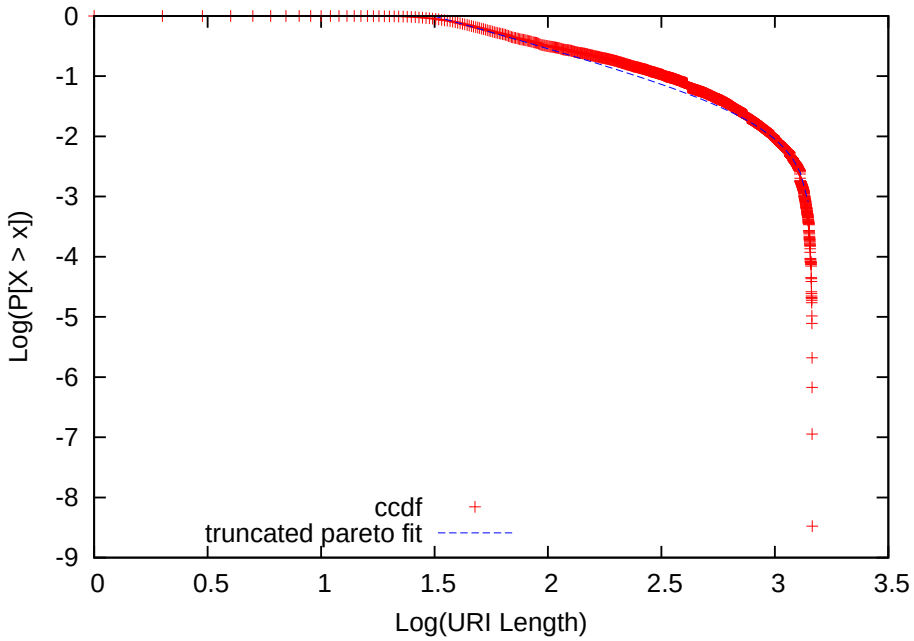


Figure 6.2: CCDF plot for recorded URI length

The length of URI has significant impact on storage and lookup speed of URIs. About 300 million URIs recorded in our measurement have an average length of 133.9 characters, and show that the length distribution is heavy-tailed with a nature upper bound. Truncated Pareto distribution [47] can be adopted to fit this distribution model. That is,

$$P(X > x) = \frac{k^{\alpha}(x^{-\alpha} - u^{-\alpha})}{1 - (k/u)^{\alpha}},\ \alpha > 0, 0 < k \leq x \leq u < \infty \tag{6.2}$$

The complementary cumulative distribution (CCDF) of recorded URI length is shown in Figure 6.2, which follows the truncated Pareto distribution with parameters, tail index $\alpha = 1.0433$, lower truncation limit $k = 32$, and upper truncation limit $u = 1460$.

6.3.2 Space Complexity

One of the critical reasons for developing distributed resolution service is the excessive memory usage caused by a large number of names. Radix trees[1] naturally aggregate common parts at the beginning and are also used for address lookup nowadays. In our experiment, they are utilized to store and lookup names. Each node in a tree contains a unique segment of a recorded name, a pointer to a specified forwarding information (i.e., the out-bound interface or a locator for the next hop), and pointers to parent and child nodes.

To reveal the relationship between the number of names and total space occupation, we randomly choose a specified number of names and measure the memory usage of generated radix tree. The result is shown in Figure 6.3, indicating the space complexity of name storage is linearly proportional to the number of names. Because each name costs about 204.4 bytes of memory, we extrapolate that at least 205 Tbytes of storage space is necessary to store all 10^{12} name entries.

6.3.3 Time Complexity

Another critical reason for the necessity of distributed resolution service is the time required for lookup with a large number of names. In the experiment, we measure the average time elapsed or looking up every single name in a generated radix tree with a specified number of names. [2]

[1] A kind of space-optimized data structure for string lookup. Nodes in a radix tree are labeled with subsequences so that a common prefix is stored for only once in a radix tree.

[2] The experiment is conducted on a server with Xeon X5570 processors and 192 Gbytes of memory to avoid memory swap during lookup.

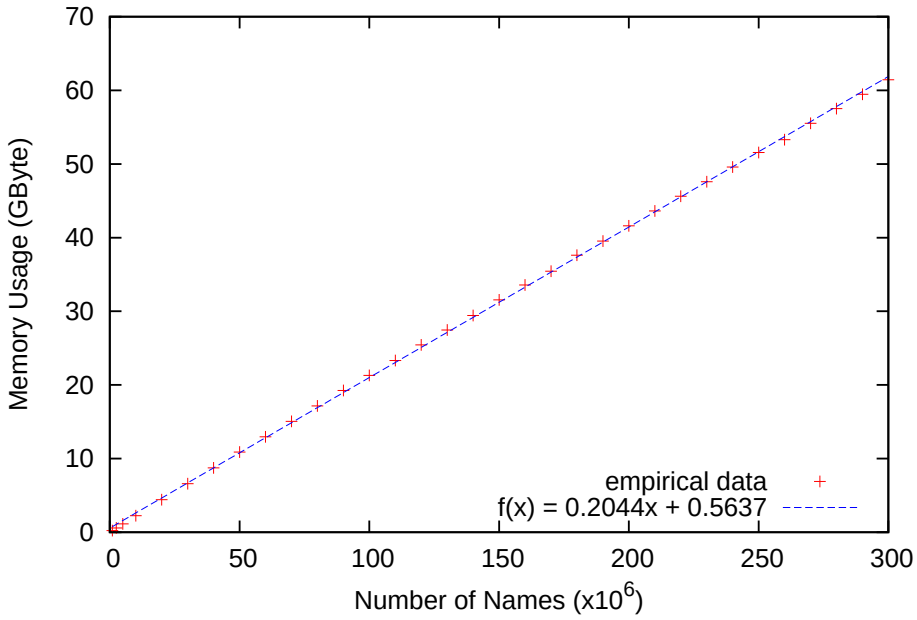


Figure 6.3: Memory usage with radix tree structure

The result is shown in Figure 6.4, demonstrating that the time complexity of name lookup follows the power function $t = cn^b$, where t is the average time and n is the number of name entries. The scaling factor c should vary with the computing capacity of the PC or router used, but the exponent b, which determines the rate of growth, is expected to be stable, i.e., $b \approx 0.35$. In the case of our experiment result, the average lookup time would be on the order of 0.7 msec if the amount of name entries were 10^{12}, which means a lookup capacity of roughly 1,500 requests per second. This lookup speed is much lower than the average request rate in our sample traffic, which is about 2,300 requests per second. Of course, this performance is far from satisfying the requirement of 20,000 requests per second that is expected to be generated by a fully-loaded Gbps link [5].

6.4 System Design

The time and space complexity models of name resolution revealed above indicate that it would be infeasible to deploy an ICN with uniform resolution services either spatially or temporally. State-of-the-art data centers might be able to handle the memory and processing requirements, but such solution has high cost and lacks scalability. In this part, we introduce a simple and empirical *Distributed Resolution Service* (DRS)

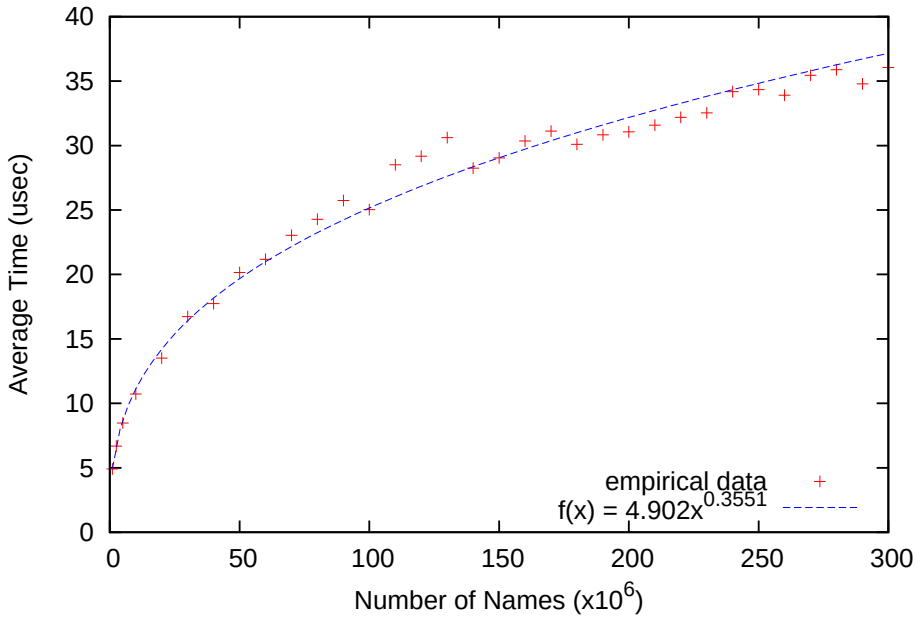


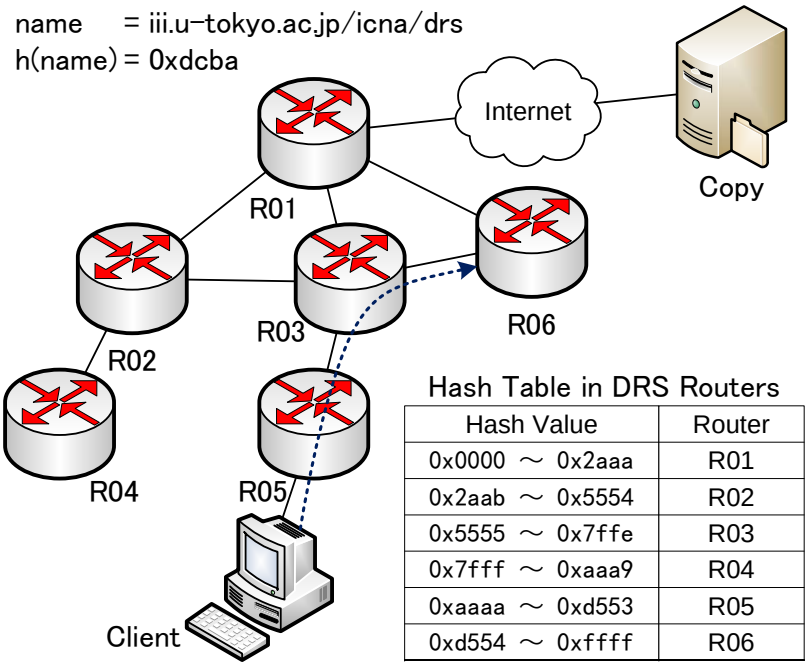
Figure 6.4: Average time usage for name lookup

scheme with regard to the established models.

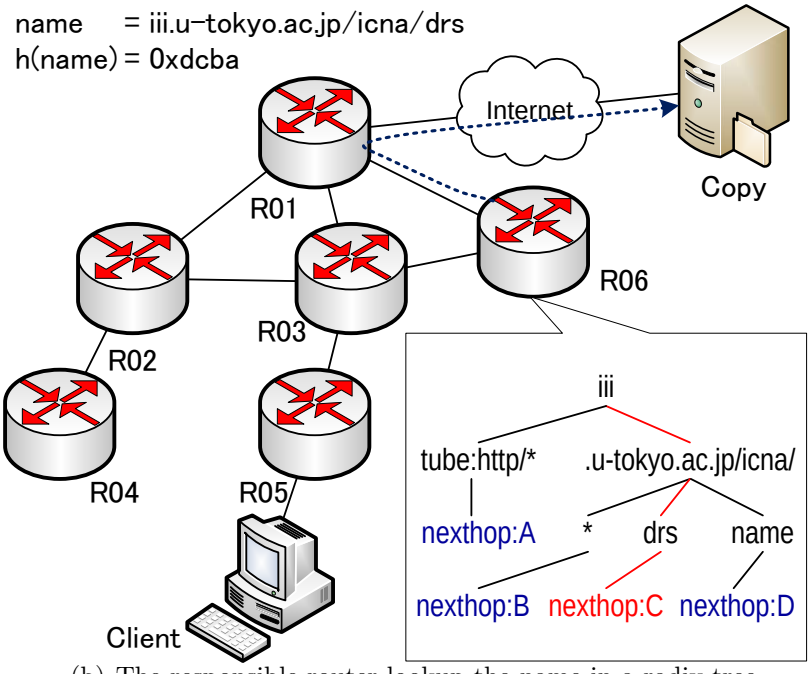
6.4.1 Scheme Overview

With the DRS scheme, the name resolution service is usually carried out by all DRS routers within a network service provider. That is to say, a DRS router handles a specific fragment of the whole name space, and for any specified fragment of the whole name space, there is always a router responsible for it within the local network. It is also possible for several small-scale networks to share a DRS domain.

Each name is allocated into a DRS router in the DRS domain according to its hash value. As shown in Figure 6.5(a), all routers in a DRS domain shares the same hash table, where one or more entries of the hash table point to the local lookup unit, and others provide locators of the responsible routers. When a DRS router identifies itself as the responsible router for the name, it looks up the name in its local lookup unit, which is usually a radix tree, as shown in Figure 6.5(b). The lookup unit provides the next-hop information, which is a locator if the destination is in the same domain, or is a network (domain) identifier otherwise.



(a) Names are distributed by a hash table shared in the domain



(b) The responsible router lookup the name in a radix tree

Figure 6.5: An example of name resolution in DRS

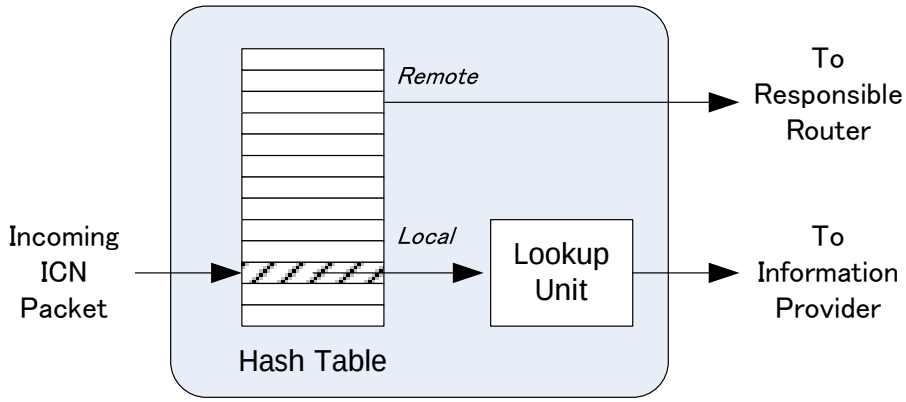


Figure 6.6: Key components of a DRS router

6.4.2 Key Components

The key components of a DRS router include a hash table and a lookup unit, as shown in Figure 6.6. The hash table maps names to routers responsible for them. There is no constraint on the hash function to be used and the array size of result slots can be determined by the numbers of routers in the system. For example, a DRS domain can adopt the FNV-1a hash algorithm [48], which accepts multi-byte data (a name in the case of DRS) as a hash key and generates a 32-bit or longer hash value. In that case, each DRS router holds a hash table with 65,536 result slots, which only takes up several kilo-bytes of memory and thus can be disregarded compared to the memory cost of the lookup unit. The time complexity of looking up a responsible router for a specific name is always $O(1)$, because there is no collision in this hash table.

The capacity of a lookup unit, on the other hand, is determined according to storage and computing capacity of the router using the established models. The lookup unit provides the locator of information provider if it is located in the local network, or the locator of the corresponding edge router that is connected to a network containing the specified information object. In this way, a request packet is redirected in a DRS domain twice at most, first time by the edge router connected to clients or other DRS domains, and second time by the responsible router of the requesting name if the responsible router is different from the edge router.

Assuming the same computing capacity with the machine used for modeling, deploying a DRS scheme capable to store 10^{12} name entries and deal with a request rate of 20,000 per second demands roughly 3,300 routers with 64 Gbytes memory each. This

is a reasonable requirement for a network service provider.

6.4.3 Comparison with Other Schemes

Compared to a centralized solution that requires state-of-the-art data centers, the DRS scheme can achieve similar or better resolution performance with commodity hardware. As described above, a DRS scheme demands roughly 3,300 routers with 64 Gbytes memory each, which is a reasonable requirement for a network service provider compared to the centralized solution. Besides, there is neither single-point-of-failure nor a bottleneck in the DRS scheme and overall capacity can be adjusted easily by adding or removing DRS routers in the system.

A common category of name resolution services is Distributed Hash Table (DHT). A DHT-based resolution service deployed per network shares most of the characteristics with the DRS scheme, except that such systems usually have high lookup latency, i.e., $O(\log n)$ where n is the number of routers in a network, because their designs try to keep only a small amount of node routing information on each node. However, we have shown that computing storage cost of such information is negligible compared to name routing information itself. There exists some efficient routing effort for DHT [49], but we argue that the mechanism for dealing with churns adopted in most DHT schemes is not necessary in our case, since routers rarely join or leave the network.

While DHT-based resolution services can be deployed locally, most such services are proposed to be deployed globally, such as in hierarchical DHT systems. However, there are several problems with such globally deployed DHT-based solutions. In the first place, such solutions usually depend on globally routable underlying locators available in the networks, which introduce the very problem that the current IP network faces into ICN. Moreover, globally deployed DHT often causes indirections, i.e., resolution requests may be forwarded over multiple long-latency hops. This not only results in degradation of network performance but also raises problems in interest and policies of network providers.

A brief summarization of the comparison can be found in Table 6.1, from which one can tell that DRS is a reliable scheme with good feasibility and low latency.

Table 6.1: Comparison of resolution schemes

Scheme	Feasibility	Latency	Other Problems
Centralized	unlikely	low	bottleneck, single-point-of-failure
DHT (per network)	good	high	–
DHT (global)	good	high	requires global routable locators
DRS	good	low	–

6.4.4 Fingerprint-Based Synchronization

As the scheme divides the whole network into DRS domains, it raises a problem of how to exchange name information among these domains. We posit that this problem can be resolved by using a fingerprint-based synchronization mechanism adopting the existing technologies. To simplify the discussion, we focus on the fingerprint-based synchronization between two DRS domains here.

Given two DRS domains, for example, domain #1 and #2 in Figure 6.7, there is a primary transit link to exchange name information between them, and the DRS routers at the two ends of the link are edge routers in charge of the fingerprint-based synchronization between the two domains. Every DRS router maintains a fingerprint (i.e., an MD5 hash value) of all the records in its lookup unit. Upon any new register or withdraw of name information, the DRS router updates its fingerprint and sends it to the edge router. The edge router collects all the fingerprints as a special information object and registers it at the other edge router on the same transit link. Edge routers retrieve fingerprint data from each other periodically to check whether synchronization is necessary.

If synchronization is required, edge routers decide which pair of DRS routers to exchange name information according to the differences of fingerprints. For example, RA and RB in the figure are elected to conduct the exchange of the name information. Such *remote data synchronization problem* has been studied well and one of the famous solutions is the *rsync* algorithm and protocol [50]. Once the DRS routers get the different entries, they resolve the conflict according to the register or withdraw timestamp and/or other configured priority factors, and finally they should become synchronized.

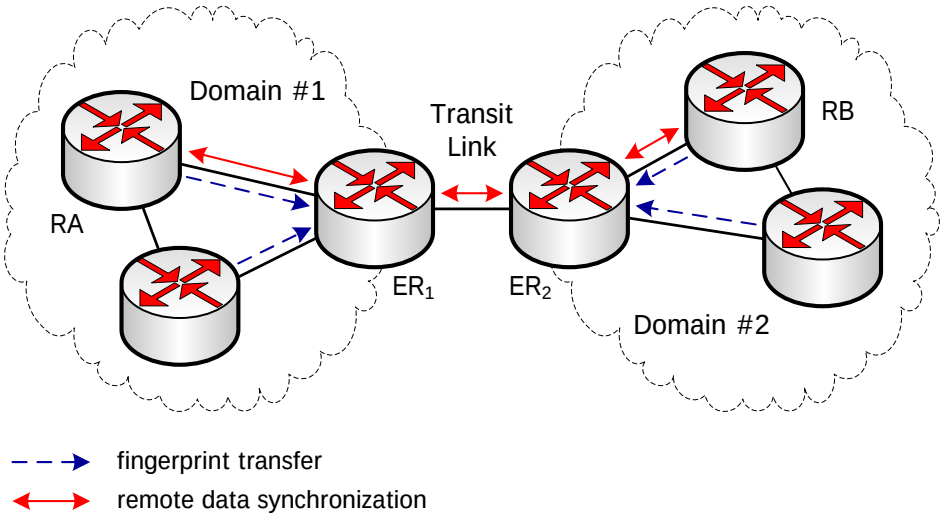


Figure 6.7: An example topology of two DRS domains

6.5 Performance Enhancement

DRS proposed above is a simple and empirical scheme that fulfills the requirements of name-based routing of ICN. In this section, we further examine the influence of the DRS scheme on network performance and suggest enhancements on it according to the analysis.

The lookup latency in the DRS scheme consists of three ingredients, time cost of hash function t_h, time cost of transmitting to responsible router t_p, and time cost of lookup in radix tree t_l. The average overall latency can be expressed by the equation

$$t = (2 - \alpha)t_h + (1 - \alpha)t_p + t_l \tag{6.3}$$

where α is the probability of the name of a request can be resolved by the local router. According to the equation, we can tell that the overall latency of the DRS scheme be smaller with a higher hit ratio α.

6.5.1 Local Entry Caching

One way to optimize hit ratio is to make use of the characteristics of spatial locality of the Internet access that has been revealed [51]. There are two kinds of information objects that are likely to be accessed, a) information objects registered by a client in the local network, and b) information objects recently accessed by a client in the local network.

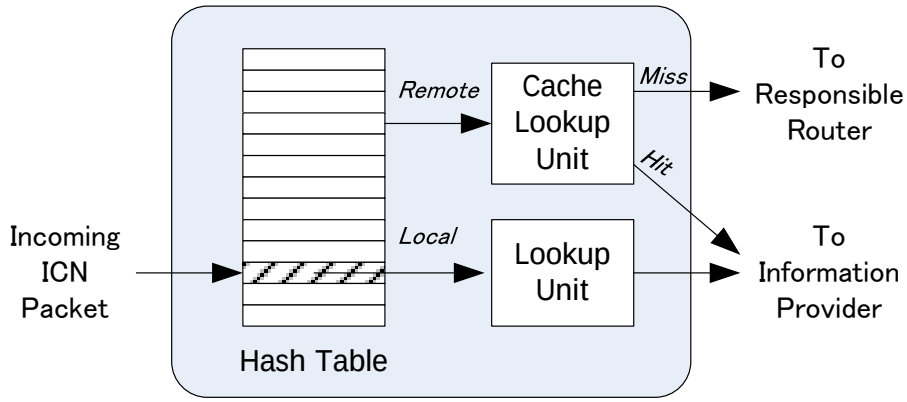


Figure 6.8: Local entry caching in the DRS scheme

Based on such observation, a cache lookup unit can be added for improving the hit ratio in the DRS scheme, as illustrated in Figure 6.8. Cache lookup unit adopts the same data structure with the main lookup unit, except that it has a fixed size that is controlled by the cache replacement algorithm (e.g., Least Recently Used). If cache hit ratio is β, the average lookup latency after optimization is

$$t' = (2 - \alpha' - \beta)t_h + (1 - \alpha' - \beta)t_p + (1 - \beta)t'_l + (1 - \alpha')t_c \tag{6.4}$$

where t_c is the lookup latency of the cache lookup unit, and α' and t'_l are the hit ratio and lookup latency of the main lookup unit. Assuming the cache lookup unit and the main lookup unit has the same size, and overall memory capacity remains the same, hit ratio becomes $\alpha' = 0.5\alpha$ and lookup latencies become $t'_l = t_c \approx 0.81t_l$. As a result, the reduction of lookup latency is

$$\begin{aligned} \Delta t_1 &= t - t' \\ &= (\beta - 0.5\alpha)(t_h + t_p) + (0.81\beta + 0.41\alpha - 0.62)t_l, \\ &\qquad 0 < \alpha \ll \beta < 1 \end{aligned} \tag{6.5}$$

As a result of the transmitting latency t_p usually dominates the total lookup latency, the Δt_1 is positive, which means that the latency gets shortened. For example, assuming $t_p = 20$ msec, $t_l = 0.05$ msec, and $\alpha = 0.03\%$, the total lookup latency would be 20.04 msec without cache lookup unit according to Equation (6.3), and it would be reduced to 18.07 msec, which means a 10% improvement, if the cache lookup unit achieves a cache hit ratio of 10%, according to Equation (6.4). Obviously, the improvement is sensitive to the cache hit ratio. The higher the cache hit ratio, the larger the

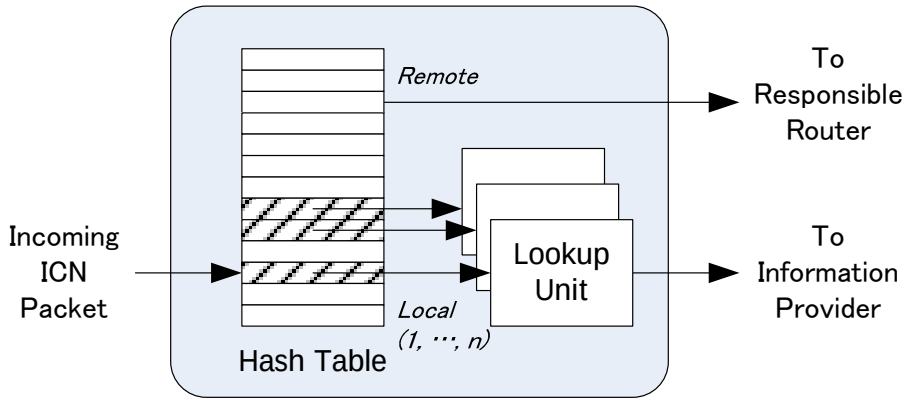


Figure 6.9: Multiple lookup units in the DRS scheme

lookup latency reduction.

6.5.2 Multiple Lookup Entries

As analyzed above, the size limit of a lookup unit is determined by both memory and computing capacity. While computing capacity of a router is not upgradable, the memory capacity is usually expandable. Consequently, we can consider optimizing lookup latency using multiple lookup units when large memory is available.

In this solution, a router has several independent lookup units, which correspond to the same number of slots in the hash table, as illustrated in Figure 6.9. Given number of total name entries N, total memory size M and lookup time limit T, this turns a router design into an optimization problem

$$\text{minimize} \quad t'' = (2 - m\alpha)t_h + (1 - m\alpha)t_p + t_l \tag{6.6}$$

$$\text{subject to} \quad \alpha = n/N \tag{6.7}$$

$$m(an + d) \leq M \tag{6.8}$$

$$t_l = cn^b \leq T \tag{6.9}$$

where equations (6.8) and (6.9) come directly from the modeled spatial and temporal constraints in the case of m lookup units.

With a constant α, which means a constant lookup unit size, the overall latency of DRS becomes significantly reduced with larger number of lookup units, m. However, a network provider should then consider the trade-off between lookup performance and the cost of adding memory to routers.

6.6 Summary

In this chapter, we propose a simple and empirical *Distributed Resolution Service* (DRS) for ICN that does neither require fancy hardware nor demand complex node management as in DHT-based systems. This proposal is based on the analysis of URIs as substitution for ICN names. Our analysis of URIs captured from the real traffic trace shows the heavy-tail distribution of popularity and the length of URIs, which gives us the good grounds for using URIs as substitution for ICN names. Further characterization using URIs shows linear spatial complexity and power temporal complexity of a typical lookup unit for name resolution services for ICN.

DRS distributes the name resolution tasks to all routers in a DRS domain by sharing the same hash table which specifies a responsible router for a part of the name space. When a DRS router identifies itself as the responsible router for the name, it looks up the name in its local lookup unit, which is usually a radix tree and provides the next-hop information to the destination. The proposed scheme can handle at least 10^{12} name entries and satisfy a high resolution request rate of 20,000 per second using about 3,300 nodes with commodity hardware, proving a quantitative guideline for its deployment. Two ways of improving the scheme, using a cache lookup unit and multiple lookup units, are also discussed to further enhance the resolution performance.

In one word, we provides quantitative modeling of the complexity, and a practical solution, DRS, that meets the challenge of the scalability of name resolution. DRS makes it possible to deploy an ICN architecture with increasing information object numbers.

Chapter 7

A Deployable and Scalable Information-Centric Network Architecture

7.1 Introduction

We have presented four technologies that provide information-centric networking (ICN) features into the current Internet infrastructure in Chapter 3, 4, 5 and 6. All these technologies discovered provides necessary building blocks for constructing an integral architecture to migrate from today's Internet to ICN. In this chapter, we integrate all the technologies and propose a *Deployable and Scalable Information-Centric Network Architecture* (DSINA), which incorporates novel route-by-name system into the current Internet infrastructure. In the light of the host-to-host model in the traditional Internet protocol suite, we posit that the most appropriate abstraction for a new architecture should be host-to-content or host-to-service. Therefore, not only content and service accessibility knowledge but also traditional host location information is the critical information handled by network devices in DSINA. Integrating all the technologies that have been developed in this article, we propose *register-access-result* model to achieve content and client access, where data can be carried by both access and result messages.

Compared to the existing proposals for future network architectures, DSINA can handle not only content distribution as is done in existing ICN architectures, but also other applications, such as user-generated content uploading, notification pushing and information synchronization. Moreover, persistent data storage and transient caches are separately used in DSINA and different strategies are employed to handle these

two sorts of cache storage.

Besides providing unique information-centric functionalities, our efforts have also been put to enhance the deployability and scalability of the architecture. Considering the deployability, DSINA is designed to operate over the current Internet infrastructure and allow incremental deployment of new functionalities. Regarding the scalability of inter-domain name propagation and resolution, DSINA adopts a hierarchical naming system, and uses distributed service to enable quick name resolution. A prototype of DSINA is implemented in C++ with Click programming model, and deployed on the network testbed Emulab. Our evaluation experiments demonstrating a content distribution application verify that the prototype system has correct and stable functionality as well as valid deployability, and can outperform the traditional applications in the efficiency of content delivery.

7.2 Design Decisions

According to the analysis in Chapter 1 of the existing ICN architectures, it is obvious that there are three aspects to be improved in order to achieve efficient content and service access. In this part, we propose design decisions in the three aspects, functionality, deployability and scalability, that DSINA has to achieve.

7.2.1 Functionality

DSINA is designed to support a variety of content and service access, including but not limited to efficient content distribution. There are two unique novel functionalities employed in DSINA, user-generated content support and hierarchical caching scheme.

Upload of user-generated content, an emerging application of today's Internet, is one of the features that must be supported by DSINA. As analyzed in the previous section, the interest-data model that is widely used in information-centric networks is not optimal for supporting content pushing. Inheriting the idea developed in Chapter 3, DSINA supports content pushing by enabling packets to carry payload data not only as responses but also as requests. Caching and authentication mechanisms for data uploading are also available as options provided by the architecture.

Content caching is one of the most important features of information-centric networks. In reality, caching facilities can be classified into two categories, persistent data

storages, such as the ones used in Content Delivery Networks (CDNs), and transient caches, such as in-network caching and caching at client hosts. In DSINA, these two sorts of caching are distinguished in registration and propagation of content information, and different strategies are used correspondingly.

7.2.2 Deployability

Deployability is one of the distinctive features of DSINA compared to the existing information-centric networks. Different from clean-slate designs, the design of DSINA puts a lot of effort into incremental deployment from the edge networks. Operating on the self-scaling transport protocol introduced in Chapter 5, the more servers and routers are enabled with DSINA, the better the efficiency of content and service access becomes. The architecture does not require routers in the core network to be upgraded to make the whole network benefit from the new architecture.

Moreover, the difficulty in supporting new functionalities is reduced by separation of cache storage and forwarding. Adopting the idea from Chapter 4, exploiting spare storage and bandwidth from end-system, DSINA removes the limitation on locating content storage only in forwarding nodes. This enables the existing routers to be able to support the new architecture only by upgrading its software, without the cost to attach storage devices to all the network devices.

7.2.3 Scalability

Network routing scalability issues have been driving new network architecture designs, recently. Route-by-name, the new system adopted by information-centric networks, faces a scalability challenge that is much more critical than it is in the current Internet.

DSINA employs a hierarchical and distributed route-by-name system, where globally available registration information is handled by the overlay system DRS introduced in Chapter 6. An individual DSINA router only stores all locally registered names and a fragment of foreign registered names, providing efficient resolution of frequently used names. If a DSINA message carries a name that is not resolvable in some DSINA router, it gets forwarded to the DRS. The names themselves are also hierarchical and therefore aggregatable, so that high hit rate for resolution can be achieved with a limited number of entries in DSINA routers.

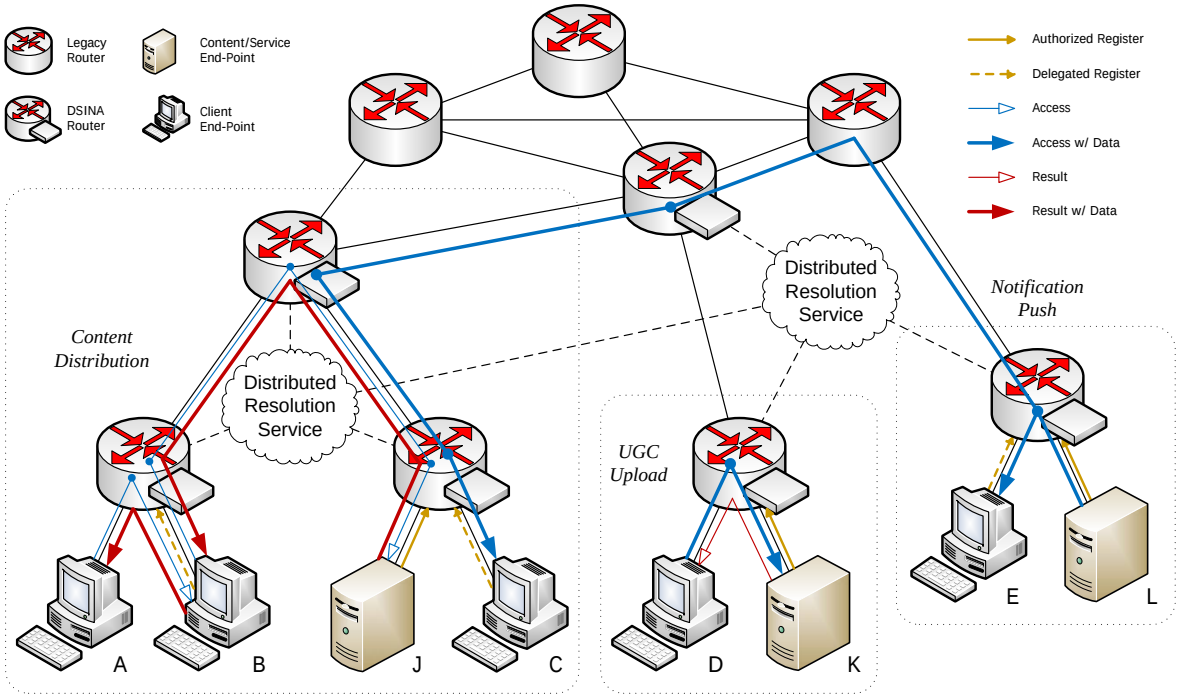


Figure 7.1: The overview of DSINA

7.3 Architecture

The purpose of DSINA is to achieve efficient content and service access by incorporating novel route-by-name system into the current Internet infrastructure. According to this objective and the design decisions stated above, this section first introduces the overview of the architecture, followed by its naming system design, and then presents the register-access-result model.

7.3.1 Overview

As illustrated in Figure 7.1, DSINA consists of three kinds of entities, client end-points, content/service end-points and DSINA routers. Communications in DSINA are abstracted as client-content/client-service asymmetric conversations. That is to say, in DSINA, a client end-point is located by its address as it is done in the current Internet, while a content end-point or a service end-point is located by a novel naming system. Running above ICTP and the existing Internet infrastructure, DSINA does not require all the routers to be migrated to benefit from the new architecture, especially the ones in the core networks (those not in the dotted frames in the figure) that can remain

legacy ones. When a packet with DSINA name passes through a DSINA router, it gets forwarded by the name and has its network layer locator (i.e. IP address in the current Internet) modified correspondingly. While it passes through a legacy router, it is simply forwarded according to its network layer locator.

7.3.2 Name System

DSINA adopts a human-readable, hierarchical naming system, as is defined in Chapter 5. For each name or prefix, there are two essential attributes, *delegation* and *forwarding*. The first attribute decides whether the content or services can be registered by a third-party without the certificate of the publisher. Names that allow delegation usually stand for the content that can be duplicated by anyone, while those that do not allow delegation usually designate services end-points that can only be deployed by the publisher and authorized partners. The second attribute, forwarding, specifies whether an access request to it should be forwarded to any end-point satisfying the request or all such end-points. An access message forwarded to all the end-points can be duplicated only at necessary network devices and saves bandwidth. The detailed usage of the attributes is described in the following parts.

7.3.3 Register-Access-Result Model

One of the major differences between DSINA and the existing ICN architectures is that DSINA extends the Interest-Data model to a flexible register-access-result model. The model corresponds to the three primitives of ICTP (and the remaining primitive, transfer control, is taken care of by the transport protocol itself), and realize the features proposed in the architecture design.

Register

Any content or service end-point must be registered first to be accessed in DSINA. There are two kinds of registration, authoritative registration and delegated registration. The registration information and its propagation formulate the control plane of DSINA.

Authoritative registration is done by a publisher or its authorized partners, such as CDNs. The registrant provides *Authoritative Registration Information* (ARI), which is composed by names, attributes and their locators with specific certificate of the

publisher, to its upstream router. The DSINA router that handles ARI also submits it to the DRS of the local network. An individual DSINA router stores three kinds of ARI records: all the locally registered ARI, a fragment of remote registered ARI that it is in charge of in the local DRS, and an ARI cache of frequently accessed remote registered ARI. The propagation of ARI among DRS is similar to that of today's inter-domain routing, except that the object is not an address prefix but ARI. A DRS receives the ARI from its neighboring network, selects and aggregates appropriate candidates according to its policy for forwarding, and sends them to other neighboring networks.

If a name is declared to be delegable in its ARI, any third-party can carry out delegated registration of that name. The registrant provides *Delegated Registration Information* (DRI), which is a tuple of a delegable name and its locators, to its upstream router for the delegated registration. Different from authoritative registration, DRI is *usually* not submitted to DRS and thus is not disseminated globally. Its propagation depends on the policy of service provider. That is, the accessible scope of delegated registration is limited, which not only preserves the scalability of global registration table but also prevents malicious DRI polluting globally. This unique strategy with ARI and DRI in DSINA is analogous to the caching scheme with stable storage (CDN) and temporary caching (proxy) used in the current Internet.

Access

In DSINA, client end-points issue access messages to access desired content or service. An access message always includes a *message header* that carries a DSINA name specifying the content or service to be accessed as well as other information necessary for the message being forwarded in the network. It can also optionally contain *access data* that provides information to be processed at the content or service end-point. For example, to access private data on remote file server, the client should provide the authentication information of its account on the server in the access data.

One of the most distinctive differences between access messages in DSINA and requests in other network architectures is that access data can also be used to upload user-generated content to online services. For instance, a client can send an access message to a video publishing service, embracing the whole video clip as the access data in the message. This design avoids the additional round-trip communication to trigger a request from the service end-point to the client end-point in data uploading

that is necessary in many other information-centric networks.

Result

Result messages are responses from content or service end-points to the access messages in DSINA. Similar to access messages, a result message includes a *message header* describing which access request it responds to, and optionally *result data* that provides additional information for the client end-points. Generally, if an access message requests for some piece of content, the result data provide the piece of content, and if the access message requests to execute some service operation, the result data provide the status of the executed operation.

Because result messages are delivered to client end-points, rather than names, locators of client end-points, i.e., IP addresses, are used in forwarding the messages. This saves the cost of route-by-name system from forwarding both access and result messages, and is expected to enhance the scalability of the architecture.

7.4 Applications

In this part, we present several typical applications of information-centric networks. DSINA is able to implement all these applications with its naming system and register-access-result model.

7.4.1 Content Distribution

Content distribution is now provided by a variety of mechanisms, including HTTP, FTP, P2P (e.g., BitTorrent), and so on. It becomes one of the most important and popular functions provided by the network infrastructure in ICN. DSINA achieves efficient content distribution through allowing delegated registration from clients that have finished downloading a piece of content.

As shown in Figure 7.2, a piece of content is authoritatively registered by a content end-point of its publisher, J. When the DSINA router receives an access message from client end-point B, it forwards the message to content end-point J according to the corresponding ARI. J serves the requesting client end-point with content data using result messages. After B has received the content, it can register itself to its upstream router if ARI of the content allows delegation. Once the delegated registration is carried

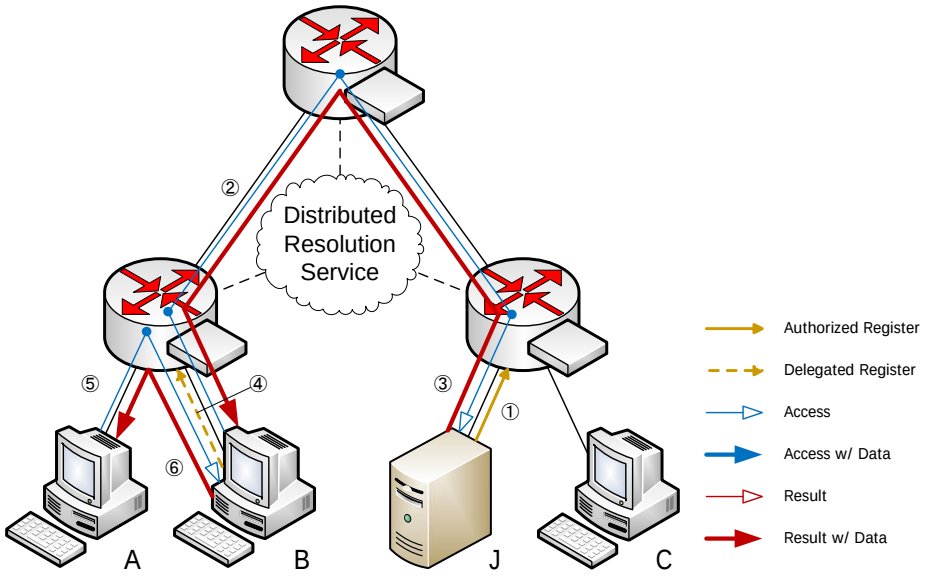


Figure 7.2: A typical scenario of content distribution in DSINA

out successfully, B becomes another content end-point for the specific piece of content and can serve other requesting client end-point in the same edge network, i.e., A, in turn.

Distributing contents from the same edge network reduces redundant traffic and improves efficiency, as proved in Chapter 4. Moreover, DSINA is better in content distribution than COCINI in that it does not require redirection messages and re-sending requests because it is deployed on connection-less ICTP.

7.4.2 User-Generated Content Publishing

Publishing user-generated photos and videos has become a widely provided service such as Flickr and YouTube, and it is expected to be more popular as content service providers begin to adopt cloud storage as an infrastructure for mobile devices (e.g., as done by Google+ and SkyDrive). Generally speaking, cloud services provide computation, data and other resources without requiring end users to know the location of the infrastructure. A typical deployment of cloud service is usually composed by a series of server clusters located at different places on the Internet. Existing ICN architectures usually implements this by clients sending requests to trigger servers initiate a session towards themselves. However, as illustrated in Figure 7.3, DSINA supports publishing UGC to cloud service without additional round-trips.

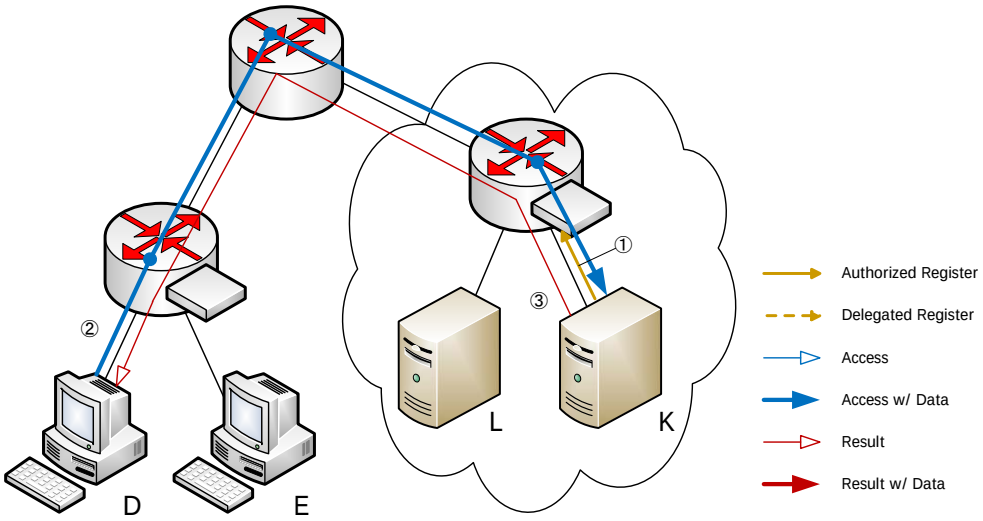


Figure 7.3: An example publishing UGC to cloud service in DSINA

In Figure 7.3, service end-points K and L stands for a server cluster of a specific cloud service, e.g., a video publishing service. Service end-points are usually not allowed to be delegated registered, while different service end-points deployed by the publisher or its partners can carry out authoritative registration to their neighboring routers, as is done by K in the figure. Such ARI is propagated among routers by DRS, and each router decides a (or a few, for load-balancing) nearest service end-point for the edge network connected to it. Client end-points do not need to care which service end-point they should send their access request to. As described in the figure, client end-point D sends its data to the name registered by service end-point K using access messages directly, carrying user-generated content data in the message payload. Such access messages are forwarded according to corresponding ARI, and are processed by service end-point K, producing an upload success reply using result message.

7.4.3 Notification Push

Notification push is a typical application of publish-subscribe (pub/sub) system, where clients subscribe to interested information and publishers send messages without the knowledge of clients. DSINA implements notification push using names that have attributes of allowing delegated registration and specifying forwarding to all registrants. This is an inverted usage of access messages, where service end-points send access messages to client end-points, as illustrated in Figure 7.4.

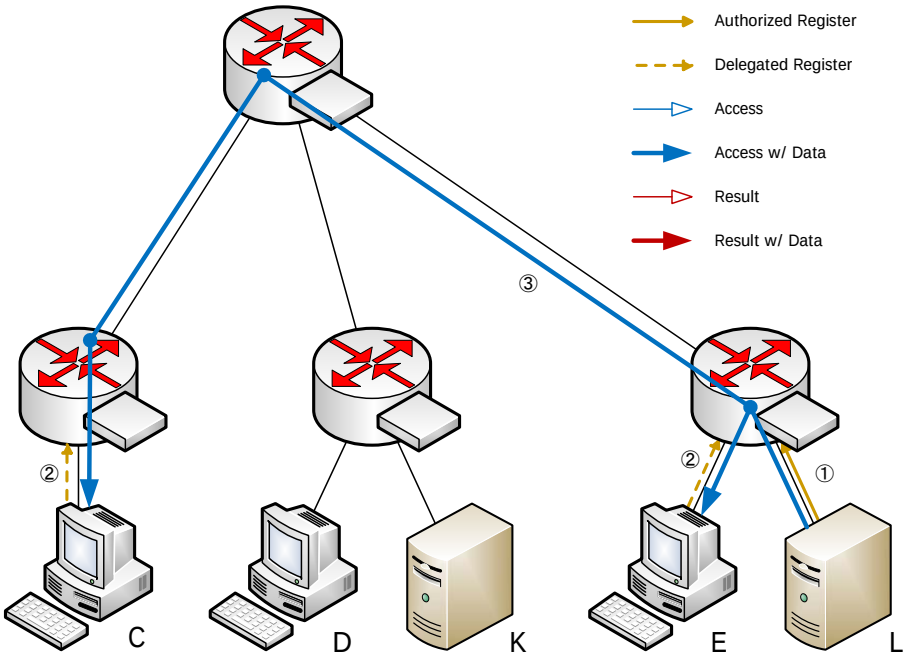


Figure 7.4: Notification push service in DSINA

In order to realize notification push, the publisher first performs an authoritative registration on a specific name from any of its service end-points (i.e., L in the figure). The name is specified as allowing delegated registration and forwarding to all in its ARI. Client end-points who have interest in such information (i.e., C and E in the figure) then perform delegated registration to their upstream routers to become subscribers. When there is a notification message to send, the service end-point sends it in an access message. Routers receiving such a message duplicate and forward it to the corresponding ports according to the DRI it has received. Bandwidth is saved as the message is duplicated on routers when it is necessary.

7.4.4 Information synchronization

Server clusters of a cloud service need to synchronize information among each other from time to time, and this application is also well supported by DSINA. The implementation of information synchronization in DSINA is almost the same as that of notification push, except that the names used are not allowed to be delegated registered.

Similar to the cloud service case, different service end-points deployed by the publisher or its partners execute authoritative registration to their neighboring routers,

using a special name that only allows authoritative registration and designates forwarding to all end-points. Thus, information that need to be synchronized, encapsulated by access message, sent from any service end-point will be forwarded to all other service end-points, achieving information synchronization among server clusters.

7.5 Implementation

In this part, we introduce the prototype implementation of DSINA. The prototype system is implemented with Click modular router [52], and deployed on a network testbed, Emulab [53]. Evaluations are also performed to verify the deployability of DSINA.

7.5.1 Prototyping with Click Modular Router

Click [52] is a software programming model for building flexible and configurable routers programmed in C++. In Click programming model, network functions are divided into modules called "elements." Click makes it possible to add a new network protocol without recompiling the kernel or even rebooting the operating system. Borrowing its feature of modular and easy to extend, the prototype system of DSINA is implemented with Click for not only DSINA routers but also content/service and client end-points. That is to say, similar to that in NDN, all nodes share the same modular structure, which simplifies the implementation and deployment.

As illustrated in Figure 7.5, the prototype implementation includes 6 DSINA elements and several example applications, which works over the current protocol stack including ordinary MAC and IP processing. [1]

The *MessageDispatcher* element handles packets from network layer (i.e., IPv4 processing elements in Click), examines the ICTP message headers (as illustrated in Figure 5.2), and dispatches them to corresponding handling logic elements. If the DSINA host works as a router, all messages are dispatched to the *RouteAgent* element regardless of their types; otherwise, the `TYPE` fields in message headers are examined and message are dispatched to the *ServerLogic* or *ClientLogic* elements, correspondingly.

Content and service end-points handle access messages using the *ServerLogic* element. All content and service applications register themselves as *Server* elements to

[1] Several built-in Click elements are omitted from the figure for simplicity and clarity.

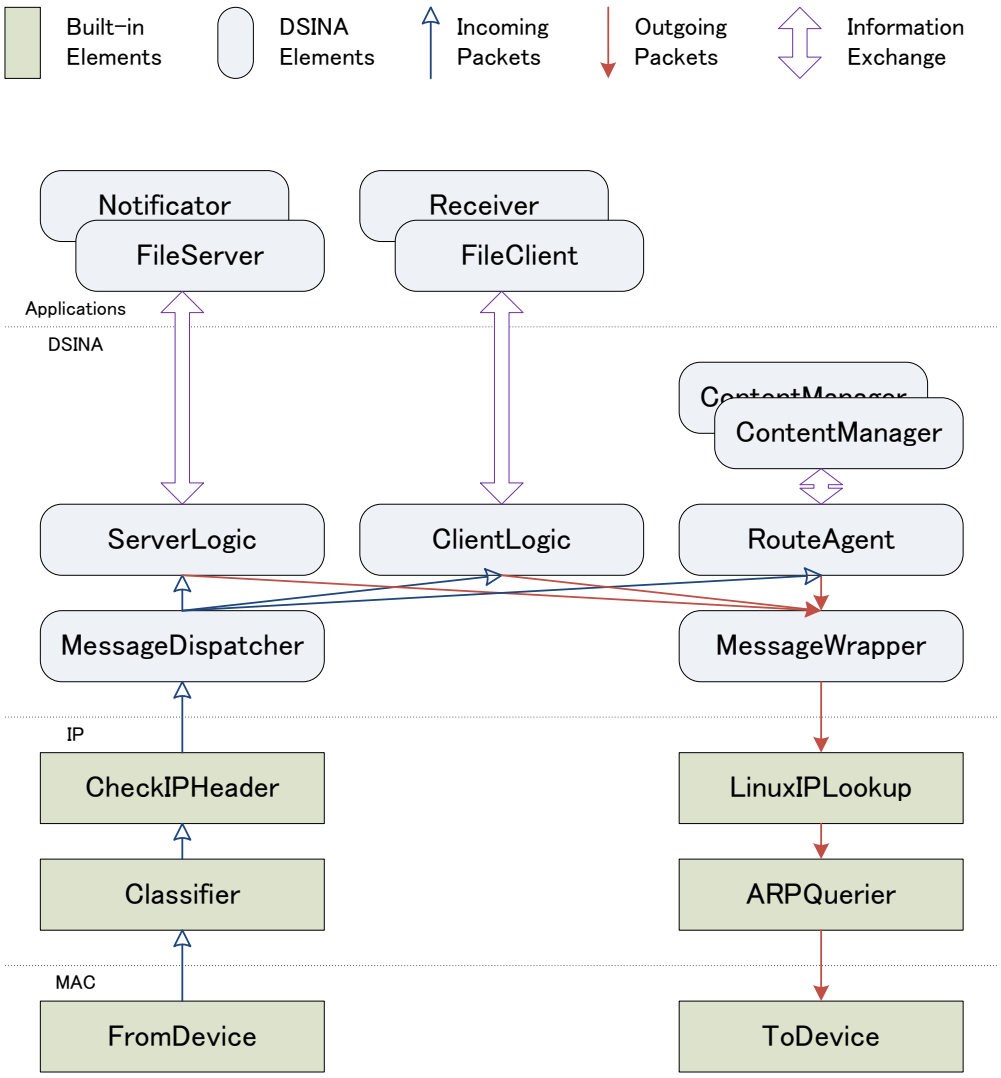


Figure 7.5: Overview of DSINA prototype implementation in Click

the local *ServerLogic* element. The *ServerLogic* element maintains a list of registered *Server* elements, examines the `NAME` field of access message headers, and sends the access request as well as access data, if there are any, to the *Server* element registered with the same information object name. The content/service applications process access requests and provide the process result to *ServerLogic* element. If it is a request to some piece of content, the result contains the requested content data; if it is a request to a specific service, the result contains the status information of the request being processed. The *ServerLogic* also includes a *Sender* sub-element to process transfer control messages and transmit the result data smoothly.

Client end-points have a similar structure formulated by the *ClientLogic* element

and *Client* application elements. The *Client* application elements register themselves (optionally, with access data) to issue access requests, and retrieve result data from the *ClientLogic* element. Just like that is done in TCP, fragment re-ordering and retransmission for packet loss are handled by a *Buffer* sub-element in the *ClientLogic* element.

Message forwarding in DSINA router is handled by the *RouteAgent* element, which communicates with several *ContentManager* elements. Comparing to Figure 6.6, the *RouteAgent* primarily implements the hash table function in DRS, and the *ContentManager* realizes a lookup unit including a radix tree. The separation of *RouteAgent* and *ContentManager* elements makes it possible to compose a DSINA router with different performance requirements (e.g., by adding caching lookup unit or multiple lookup units, or by changing the internal data structure of lookup units from radix trees) without much difficulty.

All messages sent by *ServerLogic*, *ClientLogic* and *RouteAgent* elements are wrapped by the *MessageWrapper* element to be sent to the underlying network layer.

7.5.2 Deployment on Emulab

Emulab [53] is a network testbed that provides a time- and space-shared platform for distributed systems and networks. It provides an experimentation facility that integrates network emulators, network simulators and live networks, allowing researchers to configure and access networks composed of emulated, simulated, and wide-area nodes and links. Researchers can access to hundreds of geographically-distributed nodes by specifying a virtual topology graphically or via an *ns* script, causing Emulab to automatically deploy a physical topology. The bandwidth, latency, loss and queuing behavior of virtual links are regulated, and any run-time dynamics can be executed on virtual nodes that are able to run any specified operating system.

As shown in Figure 7.6, the DSINA experiment on Emulab is formed by 17 virtual hosts, including 5 client end-points (client-1 to client-5), 3 content/service end-points (server-11 to server-13), 6 DSINA routers (droute-21 to droute-26) and 3 legacy (IP) routers (lroute-31 to lroute-33). This experiment topology reproduces the topology shown in Figure 7.1, which is chosen because it can demonstrate major functionalities in one compact topology.

Three kinds of links are defined in the experiment,

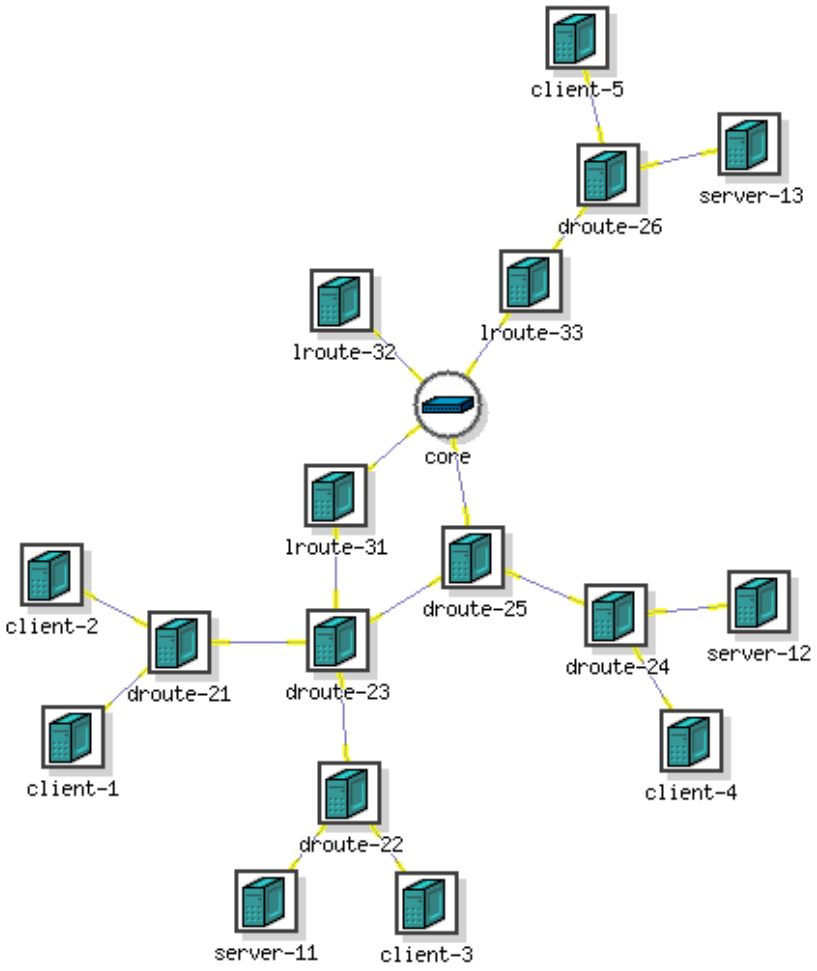


Figure 7.6: DSINA experiment deployed on Emulab

1. access links are duplex links connecting a client or content/server end-point to an edge DSINA router with 100 Mbps bandwidth and 0 msec transmission delay,

2. transit links are duplex links connecting an edge DSINA router to its uplink DSINA or legacy router with 10 Mbps bandwidth and 2 msec transmission delay, and

3. core links are duplex links connecting the 4 core routers (droute-25, lroute-31 to lroute-33) together with 10 Mbps bandwidth and 10 msec transmission delay.

On start-up (called "swap in" in Emulab), legacy routers automatically load corresponding IP routing tables from an external controller, and DSINA routers automat-

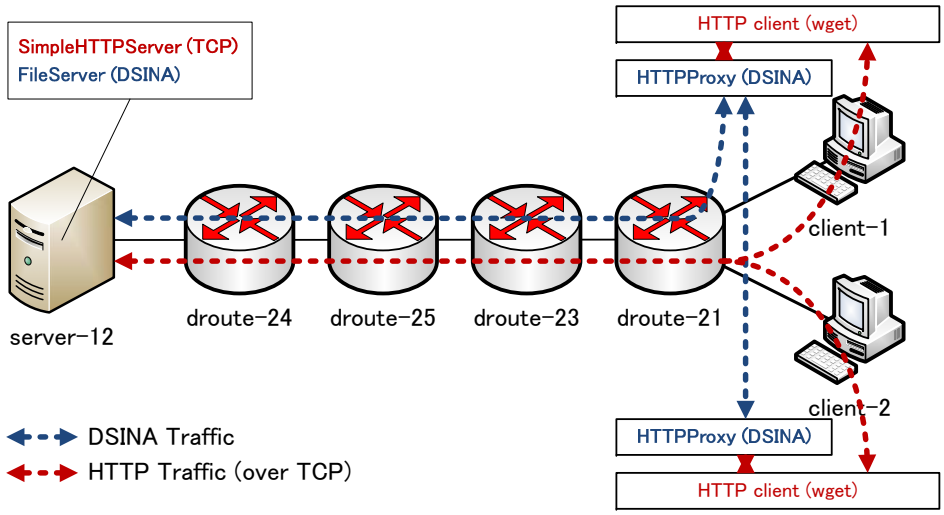


Figure 7.7: Overview of performance evaluation

ically execute pre-installed Click with proper router configuration. Experiments such as file transfer and message broadcasting are conducted on Emulab, which verifies that the prototype system of DSINA deployed has the correct functionality. In the next step, we evaluate the performance benefit of the deployed system.

7.5.3 Performance Evaluation

We focus on content distribution functionality in performance evaluation so that we can best demonstrate the benefit of DSINA. Briefly speaking, the evaluation experiment reproduces the scenario illustrated in Figure 7.2 with Emulab nodes client-1, client-2 and server-12. According to the topology shown in Figure 7.6, DSINA routers droute-21, droute-23, droute-24 and droute-25 are also involved in the experiment. In order to show that DSINA benefits applications in the real world, we choose to retrieve a video file in a HTTP client, which emulates the popular video playback web service, as the application to be compared in the experiment. As shown in Figure 7.7, we run two server daemons at the server concurrently at the server providing the same 44 Mbyte video file, the SimpleHTTPServer which provides HTTP service over TCP, and the FileServer which provides it over DSINA. At the client side, we use a console HTTP client wget [54], which can provide throughput statistics information. We compare the performance of the retrieval directly from the HTTP server over TCP, and that through a ProxyServer running at the same client node, which converts the HTTP

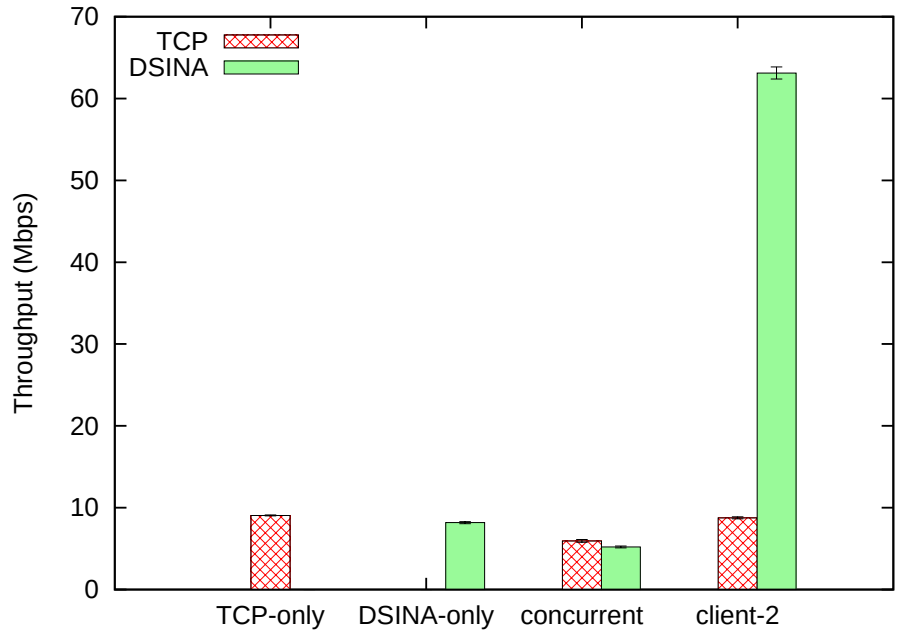


Figure 7.8: Results of performance evaluation experiments

request into a DSINA one and vise versa, thus underlying communication is performed in DSINA.

The evaluation experiment is divided into 4 steps. In step 1 and step 2, client-1 retrieves the video file from server-12 using HTTP over TCP and DSINA, respectively. In step 3, client-1 retrieves the video file from server-12 using the two methods concurrently. And after both retrieval procedures are finished, client-2 tries to retrieve the same video file using both method concurrently in step 4. Each step in the experiment is repeated for 10 times. The throughputs in each step are calculated and plotted in Figure 7.8. When running separately, the throughput of HTTP over TCP and DSINA are 9.06 Mbps and 8.18 Mbps, respectively. When both applications retrieve content from server-12 to client-1, the performances are 5.94 Mbps and 5.21 Mbps. Finally, when client-2 tries to retrieve the contents, DSINA router redirects the DSINA access message to client-1, while HTTP over TCP still goes to server-12 for the content. Thus, a dramatic difference appears in the throughput, which for TCP is 8.76 Mbps and for DSINA is 63.1 Mbps.

The evaluation result clarifies two characters of DSINA. First, DSINA shows good fairness with TCP in all circumstances, which means the traffic generated by DSINA end hosts shares the bandwidth close to equally with TCP flows. The differences in

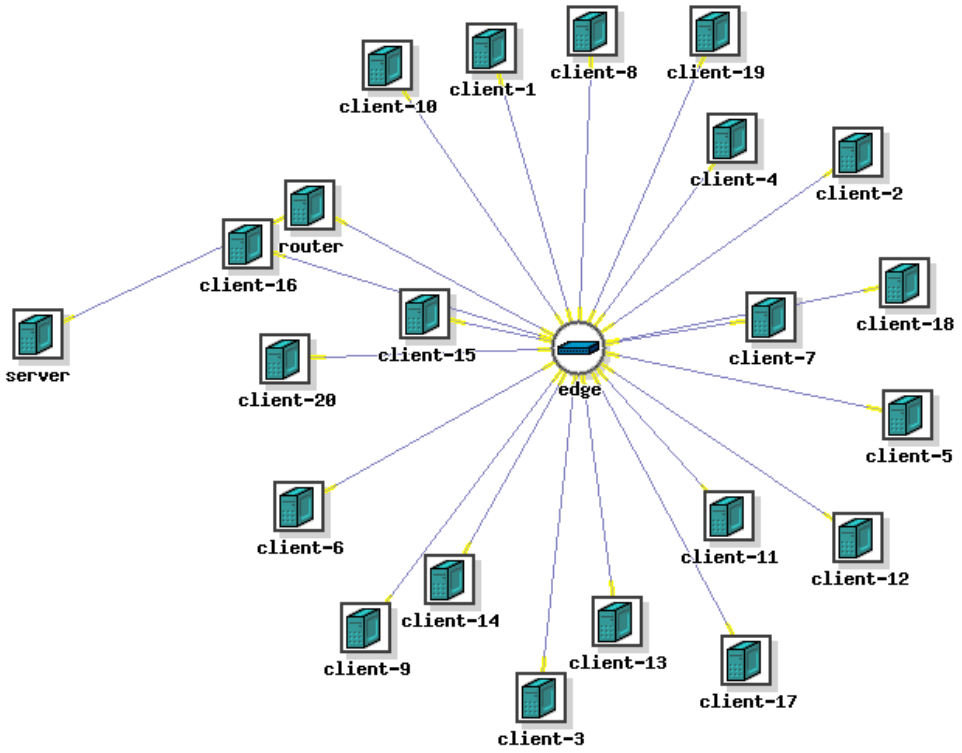


Figure 7.9: DSINA scalability verification experiment

independent and concurrent cases are 9.7% and 12.3%, and the tiny deviation also shows good stability of ICTP used by DSINA, which is acceptable as a TCP-friendly transport protocol. This ensures that DSINA can be deployed into today's Internet without any problem because it is compatible with the current Internet infrastructure and the existing transport protocols. Second, step 4 shows excellent efficiency of DSINA when there are multiple requests for the same piece of content from the same edge network. Not only the throughput of DSINA is increased to roughly 7 times because client-2 retrieves the content from client-1 in the same edge network, but also that of TCP is also increased by almost 50% because the TCP session can occupy the bottleneck transit link exclusively.

7.5.4 Scalability Verification

In Chapter 4, we have proved that COCINI is both deployable and self-scaling. To verify that the self-scaling property is valid in DSINA that adopts COCINI as its

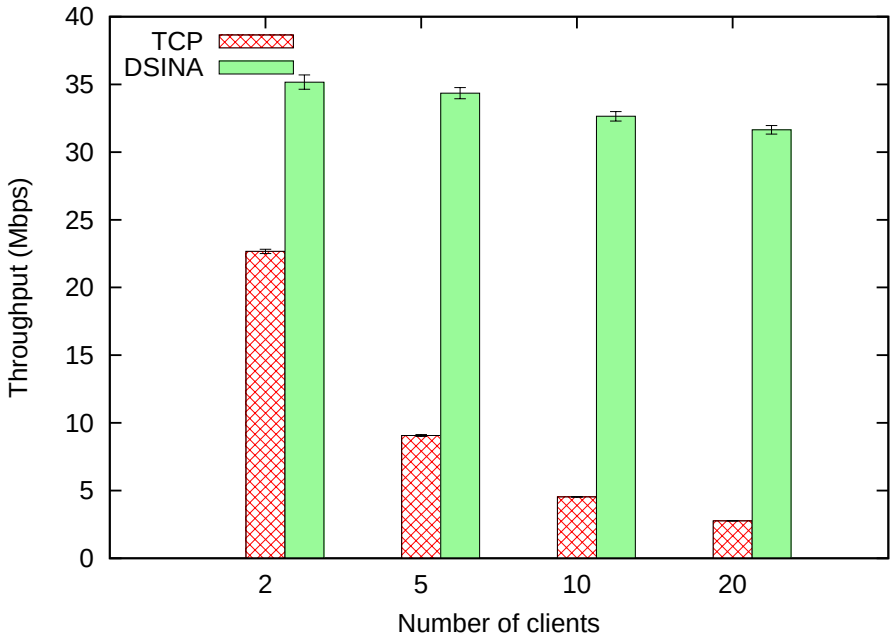


Figure 7.10: Results of scalability verification experiments

caching policy for content delivery, a scalability verification experiment is conducted. As shown in Figure 7.9, the edge network under the edge router droute-21 is extended to a network attaching 20 clients, and routers and hosts that are not concerned with content delivery are omitted from the topology. In the experiment, client-1 to client-n retrieve the same content that is published by the server in succession with a specified interval distribution that is sampled from the actual Web access trace, and the throughputs of content retrieval of client-2 to client-n is recorded. Each experiment with the same value of n is repeated for 10 times.

Figure 7.10 shows the average throughput of all clients retrieving a 100 Mbyte content except client-1 in the cases of $n = 2$, 5, 10 and 20. It is obvious that the throughput does not fall much as the number of clients increases, as the difference between the throughput with 20 clients and the one with 2 clients is only 10%. The reduction is logarithmic so that it can be calculated that the reduction is less than 30% even if there are 1000 clients in the edge network requesting for the same content. Compared with the case over TCP as in the traditional networks that the average throughput will reduce by the number of requesting clients, the result illustrates that DSINA is self-scaling with regarding to content delivery, because the content is stored not in the router but in end-hosts and the requests for it are distributed in the same

way.

7.6 Summary

In this chapter, we have proposed a Deployable and Scalable Information-Centric Network Architecture (DSINA), a novel architecture incorporating route-by-name system into the current Internet infrastructure by integrating all technologies developed in previous chapters, such as Chapter 3, 4, 5 and 6. DSINA not only provides the original information-centric functionalities, but also enhances the deployability and scalability of ICN. In DSINA, both *name* information and the traditional host location are handled by network devices.

With the register-access-result model, DSINA allows client end-point to send data with access requests, and allows access requests to be sent to multiple end-points. As a result, DSINA can not only handle content distribution and user-generated content (UGC) publishing, which are two of the popular applications of today's Internet that generate most traffic, but also other information-centric applications, such as notification pushing and information synchronization.

In order to demonstrate the utility of DSINA, we choose one application, content distribution, using our prototype DSINA implementation. The prototype of DSINA is implemented in C++ with Click programming model. Our prototype implementation deployed on the network testbed, Emulab, verifies the feasibility and the correctness of functionality of DSINA in the first place. Second, the performance evaluation shows DSINA is TCP-friendly (guaranteeing fairness with TCP) with no more than 15% difference in throughput, and can perform as many as roughly 7 times in the efficiency of content delivery comparing to the traditional applications. Finally, the scalability verification experiment proves that DSINA is self-scaling, because contents are provided by all clients having cached copies, so that the average throughput drops by only 10% when 20 clients in an edge network request for the same piece of content, and it can be calculated that the reduction is less than 30% even if there are 1000 clients.

Chapter 8

Conclusion

8.1 Summary

This thesis identifies four new technologies in three key aspects, namely, caching policy, transport protocol and name resolution, to solve the three significant open problems in the existing ICN architectures, i.e., inadequate functionality, deployability and scalability that hinder the implementation of ICN, for successful migration to an ICN architecture in the current Internet. Combining all these pieces of technologies together, this thesis propose a new architecture for deployable and scalable ICN.

First, caching policy is one of the most important research topics in ICN. In order to solve the problem of lack of functionality in efficient service access, such as not supporting publishing user-generated content (UGC), *Upload Caching in Edge Networks* (UCEN) is proposed. Taking advantage of gateway located close to end users in the edge network, UCEN allows end users to upload their contents to the gateway with token retrieved from destination servers and schedules the gateway to upload them to the destination servers without the involvement of the end users. The gateway also delegates the destination server for serving the uploaded content in the edge network. Trace-based simulation based on the HTTP extension implementation of UCEN shows that deploying UCEN brings benefit for both end users and service providers. For end users, it significantly shortens the holding time for 41% of uploading by half, so that people can leave a network quickly after the uploading. For service providers, it flattens the traffic peak for the edge network by 49% via scheduling upload time, so that capital expense for improving bandwidth can be saved.

To address deployability problem, *Content-Oriented Caching with In-Network Index* (COCINI), a deployable and self-scaling caching scheme exploiting spare storage

and bandwidth from end-systems, is proposed to be incrementally implemented and deployed. Unlike the existing caching schemes such as caching proxies and NDN, the content itself is cached on individual clients in COCINI, while the functionalities of redirection and corresponding indexing are integrated into routers. This design eliminates the cost for storage space and the drawback of single-point-of-failure. The trace-driven evaluation quantifying the effectiveness of COCINI shows that COCINI benefits both network providers and end users. For network providers, 12% to 49% Web traffic can be reduced, which is up to 21% more than the existing schemes and can reduce a significant amount of cost for bandwidth enhancement; as a side effect for end users, the cumulative latency in accessing content can be shortened by about 20%. The result confirms the significance of the index capacity in a router and storage cache on end-systems, in order for COCINI to be deployable and scalable.

Second, *Information-Centric Transport Protocol* (ICTP) is proposed also to resolve the deployability problem. ICTP is designed to provide a fundamental common transport mechanism for ICN architectures, by supporting names of information objects and information security over the current Internet infrastructures. The design decision and the protocol specifications show that the protocol is compatible with the current IP and can be incrementally implemented and deployed. In-network processing of information-centric strategies can be benefited by the genuine connection-less feature of the protocol. Moreover, congestion control mechanism that achieves fairness with the existing transport protocols is included in the protocol so that ICN architectures can focus on the strategy design. Our implementation of ICTP also shows no more than 20% difference of throughput with TCP and tiny deviation, which proves that ICTP is acceptable as a TCP-friendly transport protocol.

Finally, to deal with the scalability problem of route-by-name scheme, a simple and empirical *Distributed Resolution Service* (DRS) scheme is proposed. By using URIs as substitution for ICN names, characterization of a typical lookup unit for name resolution implementing radix tree shows linear spatial complexity and power temporal complexity. According to the analysis, DRS propose to distribute the name resolution service to be carried out by all DRS routers within a network service provider, by including a hash table and a lookup unit as key components of a DRS router. Assuming the same computing capacity used for modeling, deploying a DRS scheme capable to store 10^{12} name entries and deal with a request rate of 20,000 per second demands

roughly 3,300 routers with 64 Gbytes memory each, which is a deployable requirement. Two ways to optimize the average lookup latency, a cache lookup unit and multiple lookup units, are also discussed.

Combining the proposed technologies to migrate to ICN from the existing Internet, a *Deployable and Scalable Information-Centric Network Architecture* (DSINA) is proposed. Integrating all the technologies that have been developed, DSINA is an innovative ICN architecture that incorporates the novel route-by-name system, UCEN and COCINI caching policies into the current Internet infrastructure. ICTP is used as the transport protocol and DRS is embraced as the primary name resolution service. DSINA not only provides abundant information-centric functionalities, but also enhances the deployability and scalability of ICN. In DSINA, both DSINA name information and traditional host location are handled by network devices. With the register-access-result model, DSINA can handle content distribution as well as UGC publishing, notification pushing and other information-centric applications. A prototype of DSINA is implemented in C++ with Click programming model, and deployed on the network testbed Emulab. Our evaluation experiments verifies the prototype system works as expected, thus, shows it can be deployed in the real networks and all the functions work correctly there, for example, can perform roughly 7 times more efficiently for content delivery than it has been done in the existing Internet architecture, and is self-scaling with less than 10% average throughput drop when multiple clients request for the same piece of content.

8.2 Comparison with Other ICN Architectures

We compare our proposed DSINA with three major existing ICN architectures, Data-Oriented Network Architecture (DONA), Named Data Network (NDN) and the Pursuing a Pub/Sub Internet (PURSUIT) in Table 8.1.

Regarding information-centric functionalities, all the ICN architectures support content distribution, but UGC publishing is not supported in DONA and NDN because both of them adopt only pull method in communications. PURSUIT allows UGC publishing by making use of rendezvous nodes, while notification push is not enabled because it also asks recipients to initiate the communications. All of the three major functionalities are supported by DSINA with its register-access-result model that

Table 8.1: Comparison with other architectures

	DONA [5]	NDN [12]	PURSUIT [13]	DSINA
content distribution	OK	OK	OK	OK
UGC publishing	NG	NG	OK	OK
notification push	NG	NG	NG	OK
deployability	OK	NG	NG	OK
scalability	NG	NG	OK	OK

enables access messages to carry content data.

On the subject of deployability, neither NDN nor PURSUIT is likely to be deployable over the current Internet infrastructure because the forwarding scheme is completely incompatible with the current one and additional network elements (e.g., content storage in forwarding nodes and label switching) are necessary all over the global network. On the other hand, DONA and DSINA can expect a smooth migration from the current Internet architecture to the ICN architecture. DONA requires every network provider to own a resolution handle to handle name resolution, while DSINA allows DSINA routers to be deployed incrementally, which can be done without hardware upgrade necessity.

Last but not least, neither the flat naming scheme proposed by DONA nor the aggregatable one proposed by NDN is scalable considering the large amount of possible names. PURSUIT handles the scalability problem by dividing the name space into multiple scopes and distributing the name resolution task to different rendezvous nodes, although that requires all the clients to know which rendezvous node to resolve a specific name. DSINA develops simple and empirical Distributed Resolution Service to handle the scalability problem of name resolution with commodity hardware.

8.3 Future Work

There are still open problems and some room for further improvement that can be carried out throughout the consequent research and develop process.

First, off-path caching on end systems, i.e., making use of caching storages on end systems which are not on the path of the request sent to the original server, is conducted

in COCINI. Although the caching policy has been proved to be effective, it is possible to improve COCINI with a systematic study on in-network content caching considering the combination of on-path and off-path caching, and caching storage on routers and end systems. Moreover, adopting selective caching [55, 56, 57], which computes the benefit of caching an object, is also a possibility for enhancing the cache performance, especially when requests consist of frequent but isolated references to a set of objects.

Second, information security, which has been briefly discussed in ICTP, needs to be studied thoroughly. Although the security mechanism is satisfactory theoretically, the details of content-based security, including the transactions of SPKI and CA approaches, can be further examined. Thus, we plan to implement signature authentication modules in the DSINA implementation and to evaluate the overhead of the verification.

Third, we also recognize difficulties to be solved before bringing the DRS into the reality. First, the churn in information object registration may cause flooding traffic due to frequent inter-domain synchronization or delayed reachability if the synchronization interval is large. Second, the reliability and redundancy of DRS can also be further enhanced by distributing the name information over several DRS routers, while the states to be stored and synchronized would be increased. The degree of trade-off in both cases should be determined carefully through more research.

Finally, DSINA has been designed to be adopting a wide variety of environments and applications. It is still possible to extend this architecture in different cases. For example, we plan to study the potential of the architecture specifically applied in wireless and mobile systems. We also intend to embrace more emerging applications, such as content streaming, into the prototype implementation of DSINA.

8.4 Future Directions

To the best of our knowledge, this is the first research attempt to propose an ICN architecture that takes the serious approach towards "deployability" over the current Internet. In this thesis, we initiate a new research area that aims at enabling "deployable" ICN architectures, by identifying four technologies and proposing an architecture for deployable and scalable ICN. Unlike existing ICN studies that make "clean-slate" ICN designs from scratch, the thesis investigates necessary information-centric improve-

ments and integrates them into an integral architecture to migrate the current Internet into ICN. Our research should change people's mindset and put forth the study of deployable ICN further, and also opens more opportunities for creating applications and services adopting information-centric features.

The future directions of deployable and scalable ICN should be the large-scale deployment and verification of the migrating to ICN. First, as the functionality of DSINA has been validated within testbed environment, it can be deployed in some edge networks locally to enable fresh ICN features. After that, the architecture can also be deployed in some datacenter networks so that existing applications and services can migrate to ICN gradually without loosing the availability in the current Internet architecture, since incremental deployment feature is provided in the architecture. Finally, ICN will become globally available as more and more client end-points and content/service end-points has migrated into this architecture.

Appendix A

Data Preparation for Simulations

A.1 Introduction

Trace-driven simulations based on real-world traffic are used for evaluation experiments in Chapter 3, 4 and 6 of this thesis. All these simulations use a common data set prepared from the same traffic source. In this appendix, we introduce the source of the traffic used, the method of data preparation and the basic profile of the prepared data.

A.2 Data Source

The source traffic is captured from one of the edge link of the campus network of the University of Tokyo during the period of `May 16 06:01:51 2011` to `May 19 21:41:57 2011` in local time. The about-88-hour captured data are filtered to be Web traffic only, including 26.7 billion packets and 21.5 Tbytes. Because the tendency of Web access usually repeats with a period of 24 hours, this data source is representative for Web access analysis.

A.3 Data Preparation

The object for data preparation is to extract useful information from the 13,693 libpcap [58] files that store the traffic data packet by packet. Because the evaluation experiments use the session-level information rather than the packet-level one, HTTP sessions are restored by the data processor to obtain the 21 properties for each session as shown in Table A.1.

To deal with the large amount of traffic data, the data processor is programmed

Table A.1: Properties to be extracted for each HTTP session

Item	Description
time_syn	Timestamp of the TCP SYN packet sent by HTTP client
time_synack	Timestamp of the TCP SYN-ACK packet sent by HTTP server
time_cfirst	Timestamp of the first packet including HTTP request
time_clast	Timestamp of the last packet including HTTP request
time_sfirst	Timestamp of the first packet including HTTP response
time_slast	Timestamp of the last packet including HTTP response
sockaddr_c	Socket address (IP address and TCP port) of HTTP client
sockaddr_s	Socket address of HTTP server
packet_c	Total number of packets sent by HTTP client
packet_s	Total number of packets sent by HTTP server
traffic_c	Total number of bytes sent by HTTP client
traffic_s	Total number of bytes sent by HTTP server
method	Method of HTTP request
host	Host of HTTP request
uri	URI of HTTP request
length_c	Length of HTTP request body
code	Code of HTTP response
range	Range of HTTP response
cache	HTTP cache control instructions
auth	HTTP authentication instructions
length_s	Length of HTTP response body

in C++ to support multi-threading. The processor can be started with n ($n > 2$) threads as specified by the user, where 2 threads reads the libpcap input files and write text output files respectively, and $n-2$ worker threads process packet trace data. The input thread reads packet information from specified libpcap input files, and sends the packet into the buffer queue of a worker thread according to the hash value of the source and destination IP addresses. A worker thread distinguishes a session under processing into 5 stages: *TCP handshaking*, *sending request header*, *sending request body*, *sending response header* and *sending response body*. Different operation is done on the incoming packet for the session according to the different stage, for example,

Table A.2: Statistics of sessions with different request methods

Request method	Number of sessions
GET	613,037,426
POST	42,704,685
HEAD	2,064,421
PUT	183,477
OPTIONS	48,244
DELETE	10,777
CONNECT	4,162
TRACE	953

if a session is in *sending request header* stage, the packet content will be examined to extract useful header field, while if a session is in *sending response body* stage, only packet length will be counted. When a TCP session is closed or a new HTTP request is sent in the same session, the current information of this session will be sent to the output thread.

A.4 Data Profile

The prepared data include 658,054,147 HTTP sessions, Table A.2 shows the distribution of request methods. "POST" sessions are used for evaluation in Chapter 3 and "GET" sessions are used in Chapter 4. In Chapter 6, all sessions are used regardless of the method.

The sessions include 300 million unique HTTP documents, among which are 245 million HTTP documents that are accessed only once. That is to say, one-timers take up 81% of all unique documents. The smallest file size is 0 byte while the largest one is 59 Gbytes, and the mean and median are 25 kbytes and 1.0 kbytes respectively.

Comparing the profile of our data set and that of other research efforts (e.g., in [4]), we can tell that (1) our data set embrace the desired statistical characteristics of Web access, including the long tail distribution, and (2) although the our data set shares the same distribution with previous ones, the parameters, such as the maximum file size, change as the usage model and contents are changed over time. Both of these conclusion show that it is necessary and effective to use our data set.

References

[1] R. Braden, editor, "Requirements for Internet Hosts – Communication Layers," IETF RFC 1122, Oct 1989.

[2] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, "Internet Inter-Domain Traffic," in *Proceedings of ACM SIGCOMM 2010*, New Delhi, India, Aug 2010.

[3] L. Wang, K. Park, R. Pang, V. Pai, and L. Peterson, "Reliability and Security in the CoDeeN Content Distribution Network," in *Proceedings of USENIX 2004 Annual Technical Conference*, Boston, MA, USA, Jun 2004.

[4] M. Busari and C. Williamson, "ProWGen: A Synthetic Workload Generation Tool for Simulation Evaluation of Web Proxy Caches," *Computer Networks*, vol. 38, no. 6, pp. 779–794, 2002.

[5] T. Koponen, N. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A Data-Oriented (and Beyond) Network Architecture," in *Proceedings of ACM SIGCOMM 2007*, Kyoto, Japan, Aug 2007.

[6] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard, "Networking Named Content," in *Proceedings of ACM CoNEXT 2009*, Rome, Italy, Dec 2009.

[7] N. Fotiou, D. Trossen, and G. C. Polyzos, "Illustrating a Publish-Subscribe Internet Architecture," *Telecommunication Systems, Special Issue on "Future Internet Services and Architectures: Trends and Visions"*, 2011.

[8] D. Joseph, N. Shetty, J. Chuang, and I. Stoica, "Modeling the Adoption of new Network Architectures," in *Proceedings of ACM CoNEXT 2007*, New York, NY, USA, Dec 2007.

[9] A. Ghodsi, T. Koponen, J. Rajahalme, P. Sarolahti, and S. Shenker, "Naming in Content-Oriented Architectures," in *Proceedings of the ACM SIGCOMM Workshop on Information-Centric Networking (ICN'11)*, Toronto, ON, Canada, Aug 2011.

[10] M. D'Ambrosio, C. Dannewitz, H. Karl, and V. Vercellone, "MDHT: A Hierarchical Name Resolution Service for Information-Centric Networks," in *Proceedings of the ACM SIGCOMM Workshop on Information-Centric Networking (ICN '11)*, Toronto, Canada, Aug 2011.

[11] H. Liu, X. D. Foy, and D. Zhang, "A Multi-Level DHT Routing Framework with Aggregation," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Information-Centric Networking (ICN '12)*, Helsinki, Finland, Aug 2012.

[12] Named Data Networking (NDN) Project Team, "Named Data Networking (NDN)," http://www.named-data.net/.

[13] PURSUIT FP7 Project, "Pursuing a Pub/Sub Internet," http://www.fp7-pursuit.eu/.

[14] National Science Foundation, "NSF Future Internet Architecture Project," http://www.nets-fia.net/.

[15] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System," in *Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC '07)*, San Diego, CA, USA, Oct 2007.

[16] B. Ager, F. Schneider, J. Kim, and A. Feldmann, "Revisiting Cacheability in Times of User Generated Content," in *Proceedings of 13th IEEE Global Internet Symposium 2010*, San Diego, CA, USA, Mar 2010.

[17] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," IETF RFC 3031, Jan 2001.

[18] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "Delay-Tolerant Networking Architecture," IETF RFC 4838, Apr 2007.

[19] M. F. Arlitt and C. L. Williamson, "Internet Web Servers: Workload Characterization and Performance Implications," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 631–645, 1997.

[20] R. Droma, "Dynamic Host Configuration Protocol," IETF RFC 2131, Mar 1997.

[21] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1 ," IETF RFC 2616, Jun 1999.

[22] G. Maier, A. Feldmann, V. Paxson, and M. Allman, "On Dominant Characteristics of Residential Broadband Internet Traffic," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference (IMC '09)*, Chicago, IL, USA, Nov 2009.

[23] L. Popa, A. Ghodsi, and I. Stoica, "HTTP as the Narrow Waist of the Future Internet," in *Proceedings of 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets '10)*, Monterey, CA, USA, Oct 2010.

[24] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," in *Proceedings of ACM SIGCOMM 1998*, Vancouver, Canada, Sept 1998.

[25] M. Gritter and D. Cheriton, "An Architecture for Content Routing Support in the Internet," in *Proceedings of 3rd USENIX Symposium on Internet Technologies and Systems (USITS '01)*, San Francisco, CA, USA, Mar 2001.

[26] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," in *Proceedings of ACM SIGCOMM 2001*, San Diego, CA, USA, Aug 2001.

[27] A. Carzaniga and A. Wolf, "Forwarding in a Content-Based Network," in *Proceedings of ACM SIGCOMM 2003*, Karlsruhe, Germany, Aug 2003.

[28] M. Demmer, K. Fall, T. Koponen, and S. Shenker, "Towards a Modern Communications API," in *Proceedings of 6th Workshop on Hot Topics in Networks (HotNets-VI)*, Atlanta, GA, USA, Nov 2007.

[29] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. Levy, "On the Scale and Performance of Cooperative Web Proxy Caching," in *Proceedings of 17th ACM Symposium on Operating Systems Principles (SOSP '99)*, Charleston, SC, USA, Dec 1999.

[30] M. Hefeeda, C.-H. Hsu, and K. Mokhtarian, "pCache: A Proxy Cache for Peer-to-Peer Traffic," in *Proceedings of ACM SIGCOMM 2008*, Seattle, WA, USA, Aug 2008.

[31] A. Anand, V. Sekar, and A. Akella, "SmartRE: An Architecture for Coordinated Network-wide Redundancy Elimination," in *Proceedings of ACM SIGCOMM 2009*, Barcelona, Spain, Aug 2009.

[32] N. T. Spring and D. Wetherall, "A Protocol-Independent Technique for Eliminating Redundant Network Traffic," in *Proceedings of ACM SIGCOMM 2000*, Stockholm, Sweden, Aug 2000.

[33] Akamai Technologies, "Akamai CDN," http://www.akamai.com/.

[34] S. Dar, M. Franklin, B. Jónsson, D. Srivastava, and M. Tan, "Semantic Data Caching and Replacement," in *Proceedings of 22nd Int'l Conf. on Very Large Data Bases*, Mumbai, India, Sept 1996.

[35] J. Postel, editor, "Transmission Control Protocol," IETF RFC 793, Sept 1981.

[36] R. Stewart, editor, "Stream Control Transmission Protocol," IETF RFC 4960, Sept 2007.

[37] J. Postel, "User Datagram Protocol," IETF RFC 768, Aug 1980.

[38] J. M. Winett, "The Definition of a Socket," IETF RFC 147, May 1971.

[39] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," IETF RFC 5246, Aug 2008.

[40] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen, "SPKI Certificate Theory," IETF RFC 2693, Sept 1999.

[41] M. Cooper, Y. Dzambasow, P. Hesse, S. Joseph, and R. Nicholas, "Internet X.509 Public Key Infrastructure: Certification Path Building," IETF RFC 4158, Sept 2005.

[42] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," IETF RFC 5348, Sept 2008.

[43] S. Floyd, "TCP and Explicit Congestion Notification," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 5, pp. 8–23, 1994.

[44] R. Diana and E. Lochin, "ECN Verbose Mode: a Statistical Method for Network Path Congestion Estimation," in *Proceedings of IEEE INFOCOM 2010*, San Diego, CA, USA, Mar 2010.

[45] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," IETF RFC 3986, Jan 2005.

[46] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Rajahalme, "Information-Centric Networking: Seeing the Forest for the Trees," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks (HotNets-X)*, Cambrige, MA, USA, Nov 2011.

[47] I. B. Aban, M. M. Meerschaert, and A. K. Panorska, "Parameter Estimation for the Truncated Pareto Distribution," *Journal of the American Statistical Association*, vol. 101, no. 473, pp. 270–277, 2006.

[48] Landon Curt Noll, "FNV Hash," http://www.isthe.com/chongo/tech/comp/fnv/index.html.

[49] A. Gupta, B. Liskov, and R. Rodrigues, "Efficient Routing for Peer-to-Peer Overlays," in *Proceedings of 1st Symposium on Networked Systems Design and Implementation (NSDI '04)*, San Francisco, CA, USA, Mar 2004.

[50] A. Tridgell, "Efficient Algorithms for Sorting and Synchronization," Ph.D. dissertation, Australian National University, 2000.

[51] B. N. Padmanabhan and L. Qiu, "The Content and Access Dynamics of a Busy Web Site: Findings and Implications," in *Proceedings of ACM SIGCOMM 2000*, Stockholm, Sweden, August 2000.

[52] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click Modular Router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, 2000.

[53] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An Integrated Experimental Environment for Distributed Systems and Networks," in *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI '02)*, Boston, MA, USA, Dec 2002.

[54] Free Software Foundation, Inc., "GNU Wget," http://www.gnu.org/software/wget/.

[55] S. Hosseini-Khayat, "Improving Object Cache Performance Through Selective Placement," in *Proceedings of the 24th IASTED Int'l Conf. on Parallel and Distributed Computing and Networks (PDCN '06)*, Innsbruck, Austria, Feb 2006.

[56] Z. Miao and A. Ortega, "Scalable Proxy Caching of Video Under Storage Constraints," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 7, pp. 1315–1327, 2002.

[57] H. R. Oh and H. Song, "Scalable Proxy Caching Algorithm Minimizing Client's Buffer Size and Channel Bandwidth," *Journal of Visual Communication and Image Representation*, vol. 17, no. 1, pp. 57–71, 2006.

[58] Lawrence Berkeley Laboratory, "TCPDUMP/LIBPCAP Public Repository," http://www.tcpdump.org/.

Publications

International Conference Papers (peer-reviewed)

[1] Y. Zhu, and A. Nakao, "Content-Oriented Transport Protocol." In *Proceedings of the 7th Asian Internet Engineering Conference (AINTEC '11, ACM In-Cooperation Conference)*, Bangkok, Thailand, Nov 2011.

[2] Y. Zhu, and A. Nakao, "Upload Cache in Edge Networks." In *Proceedings of 26th IEEE International Conference on Advanced Information Networking and Applications (AINA '12)*, Fukuoka, Japan, Mar 2012.

[3] Y. Zhu, and A. Nakao, "A Deployable and Scalable Information-Centric Network Architecture." In *Proceedings of IEEE International Conference on Communications 2013 (ICC '13)*, Budapest, Hungary, Jun 2013.

[4] Y. Zhu, and A. Nakao, "A Practical Study on Distributed Resolution Service for ICN." In *Proceedings of 1st IEEE International Workshop on Future Internet Technologies (IWFIT '13)*, Kyoto, Japan, Jul 2013.

International Conference Posters

[5] Y. Zhu, and A. Nakao, "A Deployable and Scalable Information-Centric Network Architecture." In *Poster Session on AsiaFI 2012 Summer School*, Kyoto, Japan, Aug 2012.