

# CONIC: Content-Oriented Network with Indexed Caching

Yuncheng Zhu\*, Maoke Chen\* and Akihiro Nakao†

\* National Institute of Information and Communications Technology, Tokyo, Japan

† The University of Tokyo, Tokyo, Japan

**Abstract**—In this paper, we present Content-Oriented Network with Indexed Caching (CONIC), a deployable and self-scaling architecture to exploit spare storage and bandwidth from end-systems to eliminate redundant traffic and enable efficient and fast access to content. Our trace-driven simulation indicates that CONIC can reduce 25% to 50% traffic volume and can cumulatively halve the latency of content access in the real-world network environment. Also, our prototype implementation verifies the deployability of the CONIC architecture.

## I. INTRODUCTION

While the communication pattern in the Internet has been evolving since its early days, from client-server data transmission to peer-to-peer networking, and to accessing services hosted in cloud platforms, we observe two noticeable trends in the current usage of the Internet: the modern usage of the Internet has become largely *content-oriented*, i.e., we tend not to care *where* (from which host) and *how* (via which protocol) to obtain a piece of content [1], [2]. Instead, we make much of how fast and reliably we can access the content, as indicated by more and more data being served through content delivery networks (CDNs) and cloud platforms. Also, the current Internet usage has become *cache-oriented*, where it makes sense to place cache capability inside the network since we observe larger and larger content being exchanged between edges nearly at the same time and causing redundant traffic in the network core [3].

In response to these trends, many research efforts have attempted to adjust and redesign our communication model defined several decades ago. 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. For example, the existing work proposes building a brand-new network architecture to deliver content from the place nearest possible to end-systems rather than from where it is published. While the concepts such as name-based routing [4] and hash-based addressing [5] have been proposed earlier, several content-oriented network architectures have been recently proposed [1], [2], [6], [7]. 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 [8], object caches [9], packet caches [10], [11], etc. and wide-area content delivery networks [12], [13]. 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.

In this paper, we propose Content-Oriented Network with Indexed Caching (CONIC), a *deployable* and *self-scaling* architecture to exploit spare storage and bandwidth from end-systems *to eliminate redundant traffic and to enable efficient and fast access* of content. In contrast to the existing approach lacking the perspective of deployability, CONIC is designed to be incrementally implemented and deployed, for example (but not limited to), from the edge of the Internet. Also, in CONIC, while the content itself is cached on individual clients, the functionalities of redirection and corresponding indexing are integrated into routers. This design eliminates the cost for storage space and the drawback of a 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 CONIC 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 CONIC routers. Since the redirection decision is made within the CONIC 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, CONIC can eliminate 25% to 50% of the redundant traffic and to enable twice faster access to the cached content localized around the users.

Section II introduces the design of CONIC. Section III presents quantitative evaluation on the effectiveness of CONIC in eliminating redundant traffic through trace-based simulation. Section IV describes our prototype implementation of CONIC and verifies its feasibility and deployability in the current Internet. Section V discusses open problems before Section VI briefly concludes the paper.

## II. BASIC DESIGN

One of the most important goals in CONIC is to enable clients to retrieve content from the nearest possible location.

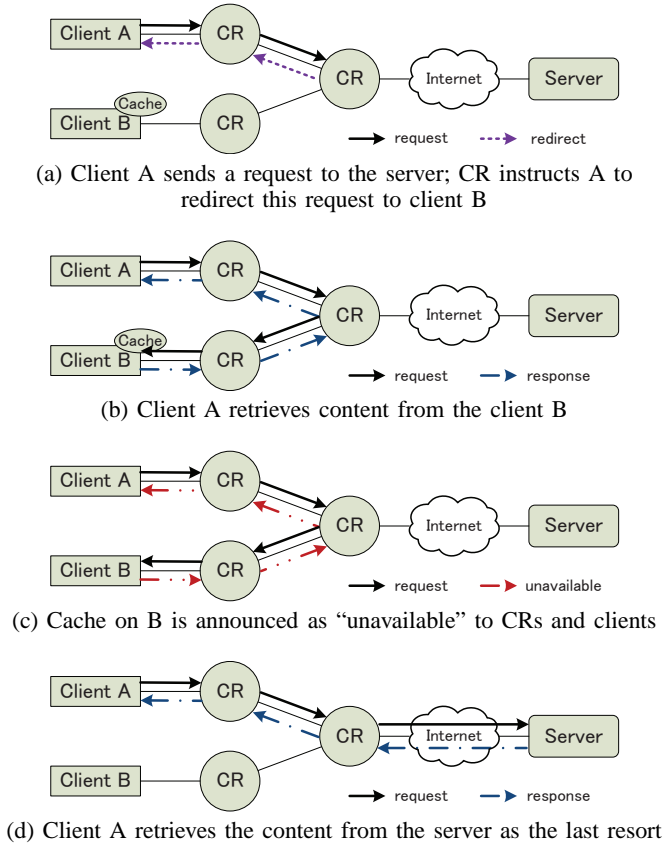


Fig. 1. CONIC messages and their delivery

We face two challenges in achieving this goal: first, to make the content and its copies discoverable and retrievable for clients, and second, to realize the content retrieval as a deployable mechanism over the current Internet infrastructure. This section first describes the architecture for the content discovery and retrieval and then introduces modifications required on both routers and clients for the CONIC design.

#### A. Architecture

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

When the local copy of the content exists, retrieval of the content is processed as depicted in Fig. 1. A client, without knowing there exists the local copy, sends a *request* message towards the origin server that holds the content. As this request message is forwarded via the network, it traverses through

TABLE I  
BASIC PRIMITIVES IN CONIC

message	arguments
<i>request</i>	label, (redirect flag), (redirect advisor)
<i>redirect</i>	label, location
<i>response</i>	label, content
<i>unavailable</i>	label

several *Content-Based Redirectors* (CRs). A CR looks up its local index for the copies of content according to the label (defined above) carried in the request message. If it finds the local copy, as in Fig. 1(a), a *redirect* message is sent towards the client issuing the request, telling it the location of the local copy. And 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 Fig. 1(b), the entire process completes without the involvement of the server.

Since clients may be disconnected from the network and cache entries may get evicted, the local copies of content may become unavailable. However, until the CR gets notified, it still instructs the client to redirect its request to the unavailable local copy. In this case, the client that receives the request sends *unavailable* messages to both the requesting client and the CRs, as shown in Fig. 1(c). Upon receiving an *unavailable* message or when none of traversed CRs can redirect the message, the request message sent by the client reaches the original server. As Fig. 1(d) depicts, the server should process the request and respond to the client with the requested content. The reliability of content access in CONIC is no worse than that of the current Internet, since CRs in CONIC fall back on the origin server to look for the content.

To summarize, there are 4 basic primitives in CONIC, as shown in Table I. All the CONIC messages must carry labels. The only primitive *request* takes optional arguments that indicate whether this request is redirect-able and who advises the redirection in case of a redirected request.

#### B. Content-Based Redirectors

As described earlier, CONIC 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 CONIC messages. The goal of CR is to provide 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*.

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

When a response message traverses multiple CRs, those CRs infer that both the source  $S$  and the destination  $D$  have cached the labeled 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 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:

$$\begin{cases} p_S = \frac{l_D}{l_S + l_D} \\ p_D = \frac{l_S}{l_S + l_D} \end{cases} \quad (1)$$

If a new index is created while the index storage is full, we must evict some index. Algorithms like Least Recently Used (LRU) [14] could be adopted to achieve this. When an *unavailable* message is received, a CR also needs to remove the corresponding index entry.

2) *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  $\mathbf{C} = \{C_0 = D, C_1, \dots, C_n\}$ , where  $D$  is the original destination and  $C_1, \dots, C_n$  are hosts that have cached the requested content according to the index in CR. The selection algorithm is as follows:

- Calculate AS PATH length,  $l_i$ , from  $C_i$  to requester  $S$ .
- If  $S$  is located in the same AS with CR, and  $\min(\{l_i\}) = 0$ , calculate hop count  $h_i$  for all  $C_i$  where  $l_i = 0$ , and define  $\mathbf{C}' = \{C_i : h_i = \min(\{h_j : l_j = 0\})\}$ ; else define  $\mathbf{C}' = \{C_i : l_i = \min(\{l_j\})\}$ .
- 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 the redirect message.

3) *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 CONIC. A client doesn't specify which CR to serve it but sends the request message towards the server through multiple CRs. When a CR receives the request, it could tell that none of CRs by far has 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

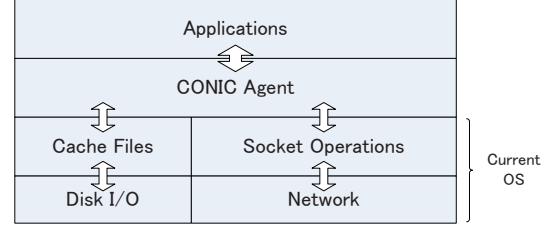


Fig. 2. Design of a CONIC agent

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.

### C. Client Hosts

A CONIC agent is installed in each client host to manage local cache files and network communication. As described in Fig. 2, the agent lies between applications and operating systems.

As illustrated in Fig. 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 CONIC agent instead of directly invoking socket operations. The agent first looks up the local cache files to see if the content with the same label is available. If it finds a local copy, it is provided directly to the application, finishing the transaction.

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

When a response (including the content) arrives, the 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 CONIC client and be arriving from the network. In this case, the 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 and is sent to the client that has issued the request; otherwise, an *unavailable* message is sent back.

A CONIC agent may be either implemented as a middleware like proxy software for rapid deployment or may be integrated as system-calls in operating systems. As opposed to the middleware design that takes over the control of network connections from applications through several different proxy 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.

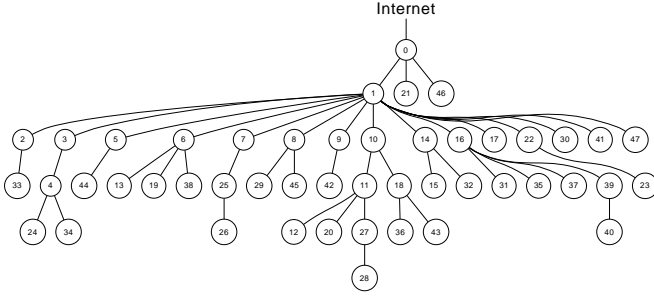


Fig. 3. Network topology used in experiment simulation

### III. EVALUATION

We apply workload data from the real network to evaluate CONIC’s effectiveness measuring to what extent CONIC can reduce cross-network traffic and can improve the content retrieval latency. We implement a packet-level simulator in C++ to this end. We focus on HTTP content for a preliminary evaluation, although our CONIC architecture is generic enough to extend to the other protocols.

#### A. Experiment Setup

For our simulation, we replay the real traffic trace captured at a campus network. We have collected traffic data at one of upstream edge routers during a 12-hour period<sup>1</sup>, which amounts to 6,904GB. Then we extract all the cacheable HTTP download sessions from these traffic data, whose total data volume is 700GB, obtaining 10,642,148 HTTP sessions initiated by 7,679 clients. At last, we apply traceroute from a host outside of the campus to all these clients and infer the network topology, finding 48 routers and inter-router links, as is depicted in Fig. 3.

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 and also the time elapsed to complete.

The capacity for indexing impacts the overall performance of CR, directly affecting the cache hit ratio. In our experiment, we choose index capacity from 100 entries to  $10^6$  at each CR to evaluate the performance of CONIC. In addition, we assume the round-trip delay between a pair of adjacent nodes is 1 ms and link bandwidth is 100 Mbps, while the delay and end-to-end bandwidth between the upstream router and servers are determined from the traced data.

#### B. Results and Analysis

Figure 4 shows how much traffic for downloading content from outside of the network is reduced as the index capacity of CR increases. We also compare the case where all the 48 routers are CONIC-enabled (denoted as Multi-CR) with the

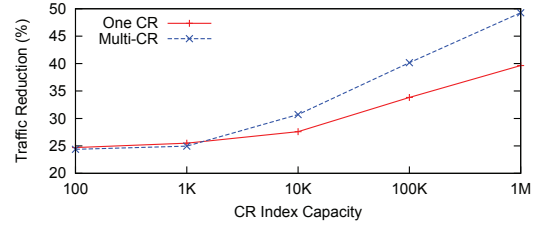


Fig. 4. Traffic reduced in two scenarios

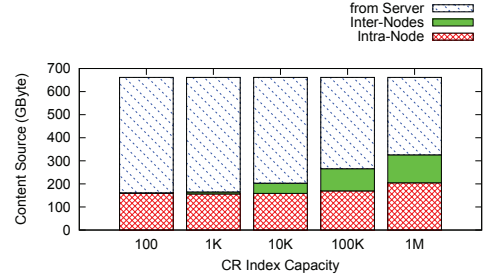


Fig. 5. Distribution of content sources in Multi-CR scenario

case where only the upstream edge router is so (marked as One-CR).

The result gives us two observations. First, index capacity is important to the traffic reduction ratio. 25% traffic is reduced when a CR hosts up to 100 index entries, and the reduction is improved over 40% when the capacity increases to  $10^6$  entries. This explains how it is significant to store only index instead of whole content in CR that would require much more storage space. An index can be accommodated within tens of bytes while the average size of an HTTP object is measured 60 KB on average in our trace. Therefore, a CR with 128 MB memory can contain  $10^6$  index entries, while a proxy server with the same storage can accommodate only 2000 objects. According to Fig. 4, with the same 128MB storage, a proxy achieves a little higher than 25% traffic reduction rate, which is 15% 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  $10^3$ . For example, Multi-CR with  $10^5$  entries at each CR works better than One-CR with  $10^6$  entries. Although complex coordination among CRs is possible, this result verifies the effectiveness of our simple design of CONIC without coordination in terms of reduction of redundant traffic.

We further explore the detail of the traffic reduction. As shown in Fig. 5, besides retrieving the content directly from the original server, there are two alternative ways: a) retrieving the content from a client according to CR’s advice (inter-node retrieval), or b) using locally available cache (intra-node retrieval).

Interestingly, Fig. 5 shows that the contribution of the intra-node caching stays almost constant at about 180GB on

<sup>1</sup>from 2009-07-15 18:53:45 JST to 2009-07-16 06:44:53 JST

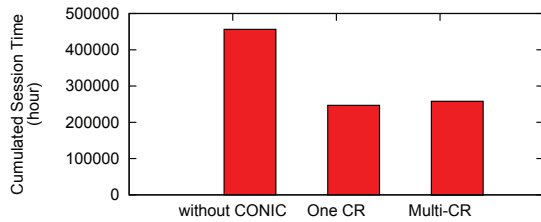


Fig. 6. Comparison of cumulative time to finish all sessions

average, but 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 CONIC will enable a client to retrieve the content from itself so there would be 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 120GB with 1M index capacity.

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

Not only network operators can benefit from CONIC through reducing redundant traffic, the efficiency of content retrieval is also improved. As shown in Fig. 6, average content retrieval latency is reduced by 46% and 43% in One-CR and Multi-CR scenarios, respectively. This result verifies that CONIC achieves efficient and fast access to the content.

#### IV. PROTOTYPE IMPLEMENTATION

CONIC implementation is required to be deployable as it is one of our goals. The architecture is designed over today’s Internet routing infrastructure with slight modification to end systems and edge routers<sup>2</sup>.

A generic identification for content is also required. Currently URL is a good candidate for this purpose and therefore CONIC can be easily applicable to any URL-driven applications, like HTTP, FTP, BitTorrent, eDonkey etc. Our preliminary prototype is implemented for HTTP, as a quick example presenting the feasibility of the system, but not limited to these protocols.

The prototype system consists of two building blocks: a local agent in client hosts and a CR module for routers. In applications using HTTP protocol, i.e., browsers, the local agent is specified and all HTTP requests are redirected towards this agent. The agent is in turn equipped with a CONIC cache management component and is able to communicate with CONIC gateways and peers. The CR module is capable of processing CONIC messages and instructing clients according to the CONIC index. We modify the non-caching proxy called Privoxy<sup>3</sup> and the open-source wireless router firmware

<sup>2</sup>Our design does not preclude the modification to non-edge routers, but introducing CONIC from the edge is a viable scenario.

<sup>3</sup>cf. <http://www.privoxy.org/>

OpenWrt<sup>4</sup> into the local agent and the CR, respectively.

Most CONIC signaling messages are transmitted over UDP, while the *response* message and successive content is delivered over the connection-oriented TCP. All CONIC client agents should agree to use a specific port number, e.g., 8118 in our prototype, to send and receive CONIC messages. The CR module in a router identifies CONIC message with the existence of this port number in packets.

For the backward compatibility with the current Internet, our prototype is made transparent to HTTP servers. When the local agent accepts a request from an application, it sends a CONIC request followed by an HTTP request, both towards an HTTP server. For any router that is not CONIC-enabled, the CONIC request is transparent. If both requests arrive at the destination web server, the CONIC request is dropped and only the HTTP request will be processed. However, when both requests arrive at any CR with the appropriate cache index entry, the CONIC request is responded and the HTTP request is dropped. In this way, CONIC-enabled hosts can benefit from CONIC wherever available, but still work with non-CONIC components without any performance degradation.

The implemented system is deployed and tested on a CR-patched wireless router and several hosts equipped the local agent over a variety of operating systems. It provides smooth access to most popular web sites.

#### V. DISCUSSION

This section discusses open problems we have encountered in designing CONIC.

##### A. Content Labeling

The traditional Internet applications typically use a host-name to identify a specific piece of content. For example, a URL combines a host locator and a host-wise file location. In a content-oriented network, a *label*—that addresses the content—does not necessarily represent its location.

Generally speaking, there are two types of content labeling methods for a content-oriented network. One is to create a new naming system, like in CCN [1]. A CCN name is usually composed of several human readable elements. It also allows us to map a name to different content when necessary. Another method is a semantic-free identification system, usually using a hash value like MD5, which is broadly adopted in today’s P2P applications. Systems using hash-based labels benefit from avoiding the same content to be named differently.

CONIC (re)use the label recognizable for today’s Internet applications, i.e., URL, deployable and compatible with the current Internet. CONIC does not interpret a URL as a “locator” but only as an “identifier” for content. In future, hash-based labels may also be adopted into CONIC’s labeling system, so that the same content represented with different URLs could be identified and shared, but the question as to incremental deployability must be addressed.

<sup>4</sup>cf. <http://openwrt.org/>

## B. Push vs. Pull

Different content-oriented network designs adopt different content delivery methodologies. For example, push-based and pull-based delivery are used in [4] and [15], respectively. A push-based content-oriented network works like today's IP multicast, which could maximize network utilization for content delivery, but requires clients to register their requirements beforehand. On the other hand, a pull-based content-oriented network allows users to retrieve content after it has been published. Meanwhile, it usually cannot achieve the maximum optimization of network utilization.

To preserve the flexibility that any clients can retrieve any content at any time, CONIC adopts pull-based content delivery. We must optimize network utilization through indexing and redirection at CRs, so that every client may retrieve content in the closest proximity.

## C. Selective Caching

It is well known that cache replacement algorithm is important for a practical cache system with limited storage. Selective caching [16], [17], [18] not only weighs the cost of removing an entry, but also computes the benefit of caching an object. This enhances the cache performance, especially when requests consist of frequent but isolated references to a set of objects.

In CONIC, CR adopts the concept of selective caching for index creation, as described in Sect. II. While, only the distance of an entry is considered in CONIC, object size and type can also be added in order to optimize the decision of index creation.

## VI. CONCLUSION

In this paper, we have presented Content-Oriented Network with Index Caching (CONIC) that achieves a *deployable* and *self-scaling* solution for *eliminating redundant traffic and enabling efficient and fast access* to content. As a content-oriented network architecture, CONIC aggressively caches the content in the network to reduce redundant traffic and facilitate fast access to the content from the nearest possible place. However, CONIC significantly differs from the other content-oriented network architectures in its deployable and self-scalable design. CONIC 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, CONIC exploits spare storage on end-systems for caching the content and unused bandwidth for accessing it so that a CONIC-enabled router (CR) intercepts the access to the content and redirects it to the nearest end-system holding its cached copy. The more end-systems enables CONIC, the more cache space CONIC can utilize, thus, CONIC is not just scalable but self-scaling in its nature.

In order to quantify the effectiveness of the proposed architecture, we have conducted a trace-driven simulation. Our evaluation shows that 25% to 50% HTTP traffic could be reduced and the cumulative latency in accessing content could be halved through applying CONIC to the real-world network

environment. The result confirms the significance of the index capacity in a router and storage cache on end-systems, in order for CONIC to scale. Also, our prototype implementation verifies the deployability of the CONIC architecture.

As immediate future work, we plan to improve CONIC in the following three aspects. First, we address the open questions we have encountered and discussed in this paper. Second, we conduct more evaluation using traces from different networks and with different application protocols. Finally, we extend our prototype implementation and deploy it in the real network to benchmark the implementation.

## REFERENCES

- [1] 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.
- [2] 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.
- [3] H. Xie, Y. Yang, A. Krishnamurthy, Y. Liu, and A. Silberschatz, "P4P: Provider Portal for Applications," in *Proceedings of ACM SIGCOMM 2008*, Seattle, WA, USA, Aug 2008.
- [4] 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.
- [5] 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.
- [6] A. Carzaniga, M. Rutherford, and A. Wolf, "A Routing Scheme for Content-Based Networking," in *Proceedings of IEEE INFOCOM 2004*, Hong Kong, China, Mar 2004.
- [7] 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.
- [8] 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.
- [9] 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.
- [10] 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.
- [11] 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.
- [12] Akamai Technologies, "Akamai CDN," <http://www.akamai.com/>.
- [13] 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.
- [14] 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.
- [15] A. Carzaniga and A. Wolf, "Forwarding in a Content-Based Network," in *Proceedings of ACM SIGCOMM 2003*, Karlsruhe, Germany, Aug 2003.
- [16] 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.
- [17] 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.
- [18] 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.