Peer-assisted Information-Centric Network (PICN): A Backward Compatible Solution

Zeinab Zali*,Ehsan Aslanian*, Mohammad Hossein Manshaei*, Massoud Reza Hashemi*, and Thierry Turletti†
*Department of Electrical and Computer Engineering, Isfahan University of Technology, Isfahan 84156-83111, Iran
Email: z.zali@ec.iut.ac.ir, ehsan.aslanian@gmail.com, manshaei@cc.iut.ac.ir, hashemim@cc.iut.ac.ir
†Université Côte d'Azur, Inria, France
Email: thierry.turletti@inria.fr

***Abstract*—Information-Centric Networking (ICN) is a promising solution for most of Internet applications where the content represents the core of the application. However, the proposed solutions for the ICN architecture are associated with many complexities including pervasive caching in the Internet and incompatibility with legacy IP networks, so the deployment of ICN in real networks is still an open problem. In this paper, we propose a backward compatible ICN architecture to address the caching issue in particular. The key idea is implementing edge caching in ICN, using a coalition of end clients and edge servers. Our solution can be deployed in IP networks with HTTP requests. We performed a trace-driven simulation for analyzing PICN benefits using IRCache and Berkeley trace files. The results show that in average, PICN decreases the latency for 78% and increases the content retrieval speed for 69% compared to a direct download from the original web servers. When comparing PICN with a solution based on central proxy servers, we show that the hit ratio obtained using a small cache size in each PICN client is almost 14% higher than the hit ratio obtained with a central proxy server using an unlimited cache storage.**

***Index Terms*—Information Centric Network, Caching, Peer-to-Peer.**

I. Introduction

As a result of a significant increase in content oriented services, currently content delivery constitutes a big portion of the Internet traffic. While the Internet is initially based on end-to-end communication among hosts, which does not necessarily correspond to the best solution for content delivery, ICN proposes a content-centric paradigm in which the host location is no more a key parameter. In this model, the original server publishes the content into the network, and the network is responsible for delivering it to every user who seeks that content.

The main objective in ICN is replacing the content location address with the content's name. In addition to providing easier and quicker access to content as a main part of the existing network traffic, this feature will enable many capabilities in future Internet including IoT. Therefore, enabling ICN will make a big difference for the network services and applications.

Prior to ICN, providing content based services to the users led to proposing Peer-to-Peer (P2P) [1] and Content Distribution Networks (CDNs) [2]. P2P was originally used to share contents among users, by creating an overlay on top of existing IP networks. A content requested by a user is provided by the clients that have stored that content. CDN was initially designed to decrease the load on central servers by caching the content in replica servers [2]. In addition, with the help of Content Distribution Network, the content is moved closer to the end-user, ensuring a lower latency for the end-users and a lower traffic load in the network. Both of these solutions are implemented on the existing network augmenting it with additional capabilities. However, the aim in ICN is that these capabilities are integrated into the network. Following this principle, in ICN, contents are cached in network nodes, and users request them using their names, while the network provides each content from a close cache that contains it [3].

The main interests of ICN are lower response latency, simplified traffic engineering, security, mobility and ability to use in ad-hoc networking. To achieve these goals, ICN architectures adhere to some principles including location-independent naming, pervasive caching, nearest replica routing, and content-oriented security [4], [5]. Several ICN architectures have been proposed based on these principles. Despite the valuable benefits of these new designs, these benefits are achieved with significant costs [6], [7]. First, most of the ICN proposals are clean-slate designs, which makes their deployment in real networks almost impossible in reality. Indeed, it is important to propose a practical solution with a migration path from existing networks to the new ICN technology [8]. Second, developing a general infrastructure that provides all the requirements of ICN (such as in-network caching and distribution of content copies by name routing) at the core of the network leads to significant hardware and software complexities.

Several studies have analyzed the performance and benefits of ICN accounting for the complexities inherent in ICN architectures [6], [7]. Authors in [6] show that most of the performance benefits of pervasive caching and nearest-replica in Content-Centric Networking (CCN) can be achieved through edge caching. So they have proposed edge caching instead of pervasive caching to prevent complexity. Moreover, the transition to a new network with ICN primitives has to be smooth. So incremental deployment of ICN is recommended instead of a clean-slate design that leads to invent new hardware and software technologies with ICN primitives. In [6], the authors suggest using HTTP to incrementally deploy ICN. They also show that in-network caching can be limited to edge

networks to avoid major changes in network core.

Considering the above-mentioned challenges, in this paper, we propose a backward compatible ICN architecture that can be deployed incrementally in existing IP networks using HTTP. We call this architecture Peer-assisted Information-Centric Network (PICN), standing for Peer-assisted ICN.

The key idea in PICN is to implement edge caching in ICN with the help of a coalition of end clients and network servers. Furthermore, by joining to this coalition, each customer network permits other customer networks to use its cache storage for their web content requests. Also, PICN provides content-centric security as one of the main ICN objectives.

The idea of sharing cached content by end clients in PICN is similar to P2P solutions and uses some of the ideas developed in P2P applications, but there is a major difference between PICN and P2P. PICN is deployed within the local network and is embedded transparently under the application layer while P2P is deployed totally in the application layer as an overlay over the Internet.

Our motivations to deploy PICN are based on three observations. First, as the request popularity obeys a Zipf distribution [6], [9], so, popular contents should exist in local regions with a good probability. Second, people in the same region usually have similar tastes and needs, because they share cultural, linguistic and social characteristics. Third, edge networks such as enterprises, campus networks or Internet Service Providers (ISP) are eager to decrease the external web traffic. Indeed, they prefer using their local networks, because external traffic for an ISP increases their operating cost, while intranet communication results in low cost [10]. On the other hand, it is valuable to leverage for caching the considerable amount of unused resources (internal network links and storage capacity) that are likely to exist locally [11]. An intuitive solution for increasing efficiency and decreasing the cost is to share the local cache of client machines via the local network. With this approach and by applying ICN ideas, we introduce a simple deployable ICN architecture in this paper. Finally, we have developed a simple prototype of PICN and made several experiments and simulations to evaluate its performance. The code for prototype and the scripts of simulation are available at GitHub[1].

The main contributions of this article are: (1) deploying an ICN at the edge of the legacy network that is compatible with all the network and application layer protocols including HTTP. (2) Leveraging clients resources for caching instead of pervasive caching to ease the network cost to deploy ICN while benefiting from lower latency and lower uplink traffic. (3) Proposing a content-based security as one of the main ICN primitives; the solution uses self-certifying content instead of self-certifying name in order to be compatible with HTTP URIs (Uniform Resource Identifier). (4) Suggesting some privacy and incentive algorithms; since we use the collaboration of the clients in providing the cache storage, we need to deal with privacy and incentive issues.

[1]PICN prototype is available at https://github.com/zzali/PICN-framework, and the simulation script is available at https://github.com/zzali/PICN-simulation.

We have implemented a simple framework for PICN as a proof of concept, and our simulations show that the cached content by end clients can actually reduce the access time as well as the uplink load in the network. Our simulation results show that in average, PICN decreases latency for 78% and speeds up the content retrieval for 69% compared to a direct download from the original web servers.

The rest of the paper is organized as follows: Section II describes the PICN architecture which is analyzed and evaluated in Section III. Section IV summarizes related works and compares PICN with other proposals. Finally, Section V concludes the paper.

II. PICN Architecture

PICN aims to realize the ICN paradigm in the legacy Internet simply by relying on content caching by the clients besides some content tracking servers at the network edge to provide the basic ICN features: location independent information access, lower access latency, and inherent information security. This will also result in a lower network uplink load by reducing the load on the original publisher's servers.

An important feature of PICN is that it is possible to deploy it simply in legacy Internet via HTTP. Publishers can be legacy HTTP servers who publish their contents as legacy web servers only by adding a PICN meta-data in the HTTP header. More precisely, PICN is deployable in each local network, using a software framework that consists of a $client\ software$ and a $seeker\ service$.

The $client\ software$ is a proxy program that is installed on every client host who desires to benefit from PICN. The simplest way to deploy a seeker is using a dedicated server connected to the local network's gateway more or less similar to the way a proxy cache server is deployed in similar networks. The seeker service is installed on this server by the network administrator. All HTTP requests of the end-user are received by the client software and redirected to a close cache (if it exists) in the network through the seeker service.

Note that the function of handling HTTP requests at the client could be later embedded into the network protocol stack of the operating system as a built-in ICN support in case this solution becomes widely deployed in the network.

A. PICN Framework

The PICN framework consists of two planes: a $cache\ storage\ plane$ and a $management\ plane$. Contents are cached on the client machines, while the management plane is implemented in seekers in the network. In this way, cache storage is indeed close to clients that results in a lower response time. Also, this is achieved with no extra hardware costs for caching. Figure 1 shows a high-level view of the proposed system.

Each client can retrieve the contents from her own cache or other clients' cache in her local network. Similarly, each user can satisfy the requests of other users using her cache. The caches and the requests are all handled by the seekers in the management plane. Every customer network (LAN/ISP) in this architecture is expected to host at least one seeker which

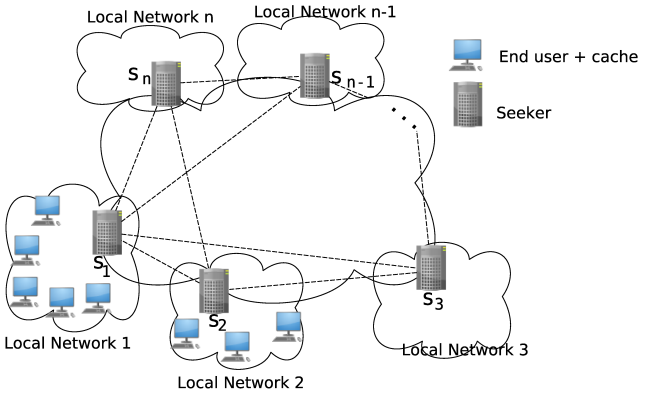


Figure 1: High-level view of PICN architecture

is known by the client. The seeker's address can be setup in the client software.

B. PICN Network Construction

ICN functionalities are deployed on traditional networks by constructing an overlay of PICN servers at the network edge comprising the customer networks (LANs/ISPs) who are interested to join.

In this way, seekers from different local networks in PICN establish an ICN overlay, constructed on the legacy IP network layer. Note that the overlay can also be constructed in transport or application layers [12]. However it is worth mentioning that this overlay is quite different from the application layer overlay built by the users of P2P networks. In PICN, when the HTTP request for a content is not matched with the cached contents in a local network, the request is forwarded by the seeker to some of other seekers on the overlay of seekers. The request is eventually forwarded to the original server if it is not found by the seekers in the neighborhood. Generally, the overlay of seekers provides the ICN service including the resolution of the HTTP requests.

In the PICN overlay, seekers with low latency from each other are assumed neighbors and virtual connections are established in between. Here we employ the binning method proposed in [13], where a set of nodes independently cluster themselves such that the nodes within a class are relatively close to one another regarding network latency. This mechanism is simple, scalable and distributed. For applying this scheme, we require a set of well-known web servers as landmark machines spread across the Internet. Each seeker measures its round-trip time to these machines and independently selects a particular class based on these measurement results. In this way, all the seekers that are finally in the same class are nearby seekers and therefore are considered as neighbors in the PICN overlay.

C. PICN Operation

To demonstrate the operation of the proposed architecture, we present herein a practical scenario of PICN for a single local network. Let us assume that Bob and Alice are studying in the same university and are in the same subnet of the campus network. They both have installed the PICN client software on their machines.

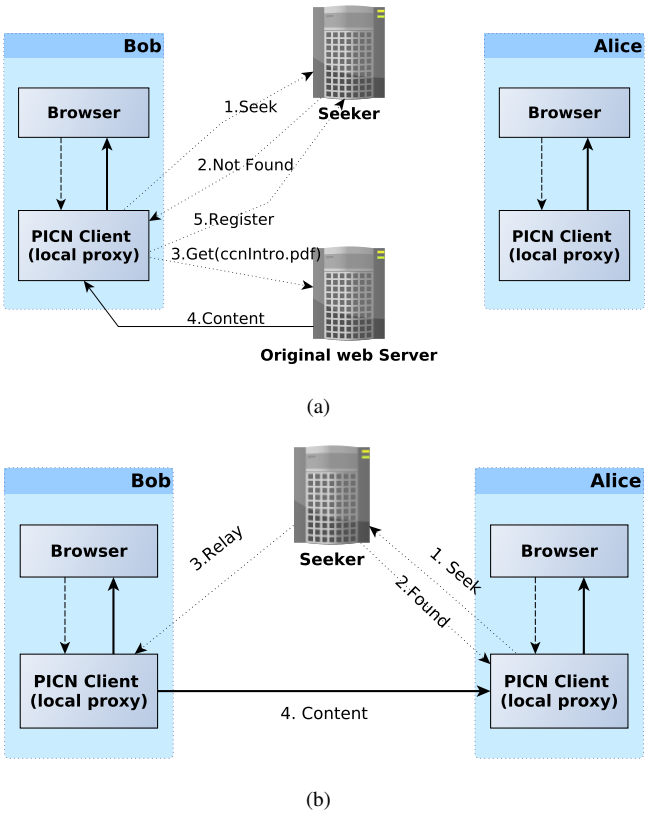


Figure 2: A simple scenario with PICN, (a) Bob requests a content for the first time, so the 5 steps in the figure is performed, (b) Then Alice requests the same content as Bob already received. So she fetches the content from Bob's cache storage in 4 steps.

Bob is looking for a presentation about content-centric networking. He finds a good one and clicks on the following link: http://netlab.iut.ac.ir/ccnintro.pdf. As Bob is running the PICN client, the HTTP request is firstly received by the client software (Figure 2-(a)). The client sends a *"Seek"* request for the desired URI to the seeker server. The seeker returns *"Not Found"* because the requested file does not exist in Bob's local network. So Bob's client software retrieves the URI using an ordinary HTTP request from the original server. After completing the download, Bob's client software registers this content on the seeker using the *"Register"* command.

After a while, Alice requests the same content (ccnIntro.pdf). As in the previous scenario, her PICN client software sends a seek request to the seeker. This time the seeker relays the request to Bob's client software which delivers the content to Alice (Figure 2-(b)). The seeker maintains a lookup table of mapping contents to clients. Details of this table will be described later. Alice's PICN client can check the integrity and provenance of the data by the meta-data embedded in the content (using the method described in the next section).

D. Cache Registration and Lookup

In this section, we describe registration and lookup protocols in PICN. First it should be mentioned that the $client\ software$ periodically sends $Hello\ message$s to the seeker. Therefore, the seeker is aware of the users' presence in PICN.

Registration: Each seeker manages a table of registered contents. This table is called Registered Information Table (RIT) that includes an entry for each cached content including the IP address of the client or storage that caches the content. The table size might be a matter of concern like any other ICN solution as it would be expected that the number of contents would be large. However considering that this solution is implemented in edge networks, the content pool size would be limited. Obviously the size of pointers to contents cached in a local network would be much less than the size of the contents themselves that are stored in proxy cache servers serving edge networks. Nevertheless the seeker can limit the caching based on the limits it put on its RIT table size.

When a client receives a content, it registers the required information about its cache in the seeker. Then the seeker decides whether or not to register the content in that client based on the PICN caching strategy. Various strategies can be employed for the decision about redundant copies of the same content on the local network. For example with $full\ redundancy$, each client always caches the retrieved content in its local storage and shares it with others; and with $no\ redundancy$, at most one copy of each content is cached in the local network. But we suggest $popularity\text{-}based\ redundancy$; this approach chooses the admissible number of redundant copies of a single content in the seeker's local network based on its popularity rank using the following sigmoid function

$$\frac{M-1}{1+\exp(\frac{p-0.5}{0.1})}+1, \tag{1}$$

where M is the maximum number of redundant copies, and $p \in (0,1]$ is the ratio of the content popularity to the number of existent popularity classes (e.g. having P popularity classes, for the qth popular content, p is computed by q/P). This function is defined such that each content is cached at least once and at most M times in a local network. Also, it permits almost 50% of the most popular contents to be cached up to M times.

For content retrieval in the case of redundant caching, the seeker can select a copy among the redundant copies using a cache selection policy such as choosing the least recently used cache. With this simple method, the requests for the same content can be distributed uniformly among the caches holding that content. Another policy is to select the cache on a client that has the lowest number of uploads to other clients at the time of the request. Since PICN permits existence of more than one providers for each content in a network, the requests for popular contents are distributed among different providers. So the upload traffic for each client is limited.

If the cache storage of the selected client machine for caching is full, the client chooses one or more registered objects to be replaced and then sends an unregister command to the seeker for those objects. Items selection for replacement is performed according to a cache replacement policy such as Least Recently Used (LRU).

Cache registration, redundancy strategy, cache selection for retrieval, cache replacement policy, and dynamic content cache policy are not fundamental parts of our proposal and any conventional method can be used. The frequency of cache registration may seem to impose an overhead in the network.

Yet given the size of the messages exchanged for the registration and given that this is all done within the boundary of the local network, the overhead is not significant compared to other local control messaging in the networks. However, as a precautionary measure, the content registration process can be done after aggregation to decrease the overhead. It means that a set of contents can be registered on the seeker in one registration request, instead of registering each single content immediately after caching it locally. The waiting time and the number of contents in one registration request can be custom setup for every network.

Lookup: Assume h_i^j is the jth host machine in the ith customer network, and s_i is its relevant seeker. When client h_i^j requests content C from seeker s_i, the lookup process for finding that content is as follows:

1. If some clients exist that have registered C in RIT, the seeker selects one of them, called h_i^k, with its cache selection policy.

2. If there is not an entry in RIT for C, s_i sends a request to a limited number of nearby seekers, sequentially or even in parallel, and asks about the availability of C. Each nearby seeker, s_n, that has C registered in its RIT, will reply to s_i.

3. If cache lookup fails in all the previous steps, h_i^j retrieves C from the original server.

When a seeker does not receive a $Hello\ message$ for a while, it recognizes that the user machine is disconnected from the correspondent seeker or has been turned off; so it removes all of its registered items from the RIT. Whenever this client connects again, the seeker registers all the items in its cache storage. It is also possible to apply a partial download mechanism when uplink and downlink bandwidth difference is considerably high. So, one request can be served partially by several responses from other clients and the original server.

If the clients of a customer network are behind a Network Address Translation (NAT) and have private IP addresses, NAT traversal methods such as ICE (Interactive Connectivity Establishment) [14] can be used.

E. Naming and Security Model

PICN uses standard HTTP URIs for content naming to achieve backward compatibility. But like other ICN proposals, it needs a self-certifying naming system. We use the term "self-certifying content" instead of "self-certifying name" and, in our terminology, the content is a composite of the *data* plus *meta-data*. Meta-data lets the content *consumer* check the *integrity* and *provenance* of the data, without communicating with the original content *publisher*.

Our proposed security model is similar to the data-centric security method in NDN [15] since in both cases the security

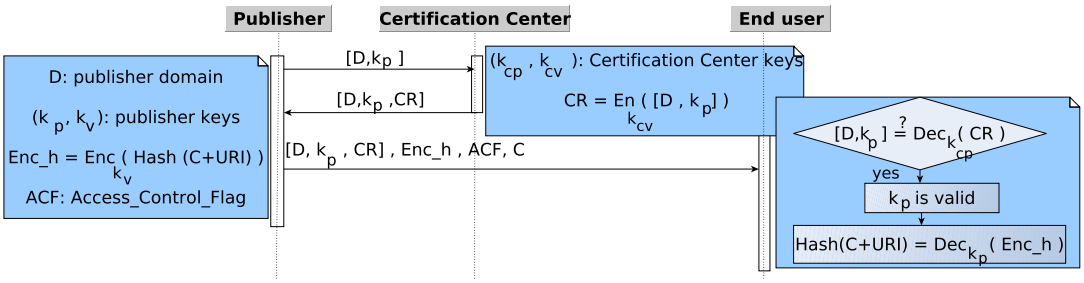


Figure 3: Sequence diagram of providing authenticity of original publisher and the data integrity

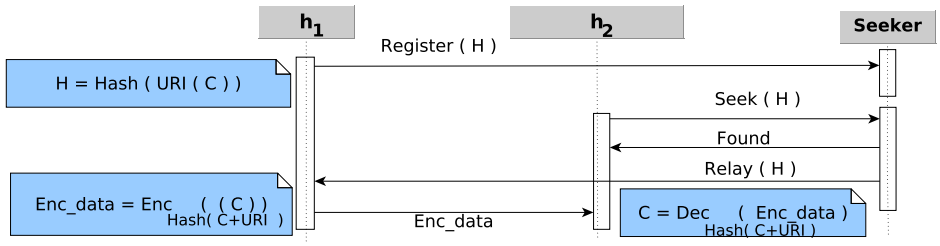


Figure 4: Sequence diagram of providing security during data request and transmission

is built into the data itself. The difference is that in NDN, cryptographic keys are verified against consumers' policies, and each piece of data is signed together with its name, securely binding them. But with PICN, verification is based on Public Key Infrastructure (PKI) [16], and security binds to the content instead of the content name. Consequently, PICN achieves backward compatibility of the names with the legacy HTTP URIs. Let's see how meta-data can provide this capability. The meta-data contains three fields: Publisher-Domain-Certification, Encrypted-Hash, and Access-Control-Flag. Here we describe the functionality of the two first fields.

Let us assume that we have a content C and its URI is U_c in the domain D. Figure 3 presents the process of data provenance and integrity. The Publisher-Domain-Certification field binds the content domain to the publisher's public key K_p and is verified using trusted third party's public key K_c. The Encrypted-Hash binds data to its hash and is verified using K_p. These two fields in the meta-data allow checking the provenance and integrity of a received content in off-line mode without additional delay. This leads to the following benefits of ICN: (1) short response time, (2) ad-hoc operation, (3) satisfaction of security issues for popular contents (i.e., provenance and integrity).

Through a simple change in the web server, it is possible to include the meta-data in the HTTP header field. The HTTP header contains valuable information about the content. According to RFC 2068 [17], an extension-header mechanism allows additional entity-header fields to be defined without changing the protocol.

Finally, we propose an access control method in case content access control and confidentiality are required. When the web server desires to limit the access to some special contents, it enables the Access-Control-Flag in meta-data. In this case, the scenario of requesting a content starts with two parallel requests: (1) The client requests an HTTP header from the original web server using the content URI, (2) the client sends a $seek$ message for the content to the seeker using the hash value of the content URI. The HTTP header can be transfered using HTTPS in order to avoid man-in-the-middle attacks.

As shown in Figure 4, when a client h_1 downloads the content C, h_1 advertises C by registering it in the seeker with name $Hash(URI(C))$. Note that it is encrypted with $Hash(C+URI)$. Now suppose that client h_2 requests content C from the seeker. This request is sent with the same name, sending $Hash(URI(C))$ to the seeker. The seeker matches the request with registered contents and finds client h_1 which has content C. Then the seeker sends a $find\ message$ to h_2 and also a command to h_1 to relay the content to h_2.

When client h_1 receives the relay command, it delivers the encrypted content C to client h_2. h_2 decrypts the encrypted content with $Hash(C+URI)$ (which is taken from the HTTP header request in initial steps) and checks the integrity of content C also using $Hash(C+URI)$. Since in PICN security is embedded in the content itself, therefore publishers do not require HTTPS anymore to provide channel-based security.

F. Privacy Preserving Model

Without caution, some unwanted information leak happens during PICN's registration and fetching processes. In particular, the requester can identify who has the content, the provider can identify who wants the content, and the seeker can detect both. This issue can violate PICN client users' privacy. We propose a mechanism based on the Tor anonymous network [18] to account for this problem. The basic principle of Tor is to redirect traffic over virtual tunnels through independent Tor nodes, to anonymize the communication between two users.

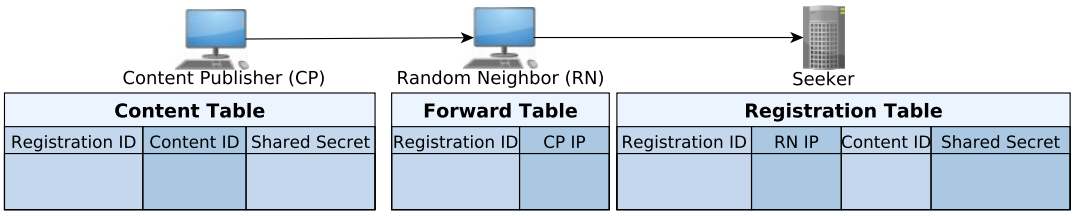


Figure 5: Private content registration

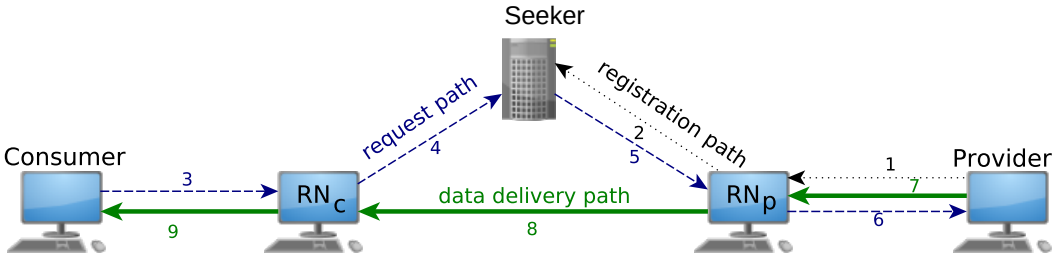


Figure 6: Private content fetching

Here we describe how PICN constructs a virtual secure tunnel in communications with the help of the seeker.

Let the seeker own a pair of (private, public) keys and every client have the seeker's public key. This assumption enables clients to asymmetrically encrypt data (using seeker's public key) and then send it to the seeker. Also, every client has a *neighborhood* list, containing the IP addresses of some other active clients (i.e. *neighbor*s) in the network.

The client that wants to register some items on the seeker (i.e. the content provider) chooses one of its neighbors randomly, encrypts the registration message by the seeker's public key, and sends it to this random neighbor. This message contains:

1. A *registration ID*: a unique identifier for keeping the record of this registration. This field is not encrypted and the random neighbor records it.

2. A *random shared secret*: a symmetric encryption key for confidential communication in reverse direction from the seeker to the content provider.

3. A set of *content ID*s (i.e. content hashes) that the client wishes to register on the seeker.

The client also keeps the records that map these content IDs to (registration ID, random shared secret) tuples in its *content table* (Figure 5). Then, this randomly chosen neighbor (RN_p) relays the message to the seeker. For keeping track of this registration, the random neighbor records this transmission by a pair of (registration ID, content provider's IP address) in a *forward table* (Figure 5). The seeker receives and decrypts the message. To store the back-trace route, the seeker records the random neighbor's IP address as well as the registration ID, random shared secret and content IDs (Figure 5). This way, the seeker will be unaware of the original provider of the content.

When the seeker needs to communicate with the provider, it encrypts the message using the shared secret embedded in its registration table and sends it back to the random neighbor. The random neighbor forwards it to the client using its forward table.

On the other hand, the client that requires a content (i.e. consumer), sends her $query\ message$ to the seeker through one of its random neighbors(RN_c), as shown in Figure 6. This message is also encrypted using the seeker's public key and contains two fields:

1. A *transfer shared secret*: a symmetric encryption key for confidential data transfer from provider to consumer.

2. A set of *content ID*s (i.e. content hashes) that the client wishes to acquire.

The seeker receives the $query\ message$, decrypts it using its private key, looks up the content IDs in its registration table and then sends $deliver\ message$(s) to the provider(s). A $deliver\ message$ contains the transfer shared secret and content ID(s) fields and random neighbor's (RN_c) IP address. This message is passed to the provider in encrypted form using the mechanism mentioned before in Figure 6. On the other side, the content provider after receiving the request message encrypts the requested content using the transfer shared secret, then it sends the content through backward route (to RN_p then RN_c) for the consumer. Finally, RN_c delivers the content to the consumer. During this process, none of the agents acquire additional information, and privacy cannot be violated unless by a cooperation of a large number of agents.

Note that some users would prefer not to waste their bandwidth and performance to prevent possible privacy concerns. For this purpose, each client registers its content either directly or through a random neighbor, with an adjustable probability β_{client}. One important outcome of this mechanism is that the overall privacy increases significantly. For instance, assume that in the case a client has chosen to register its content directly, this would not mean that there is no privacy for this client. Indeed, the server does not know that the client has registered the content directly or via a random neighbor. Therefore the seeker cannot conclude that all contents and

queries are related to that client.

G. Incentive Design

As mentioned in Section II-F, each client in PICN has a set of neighbors that cooperates with them. PICN is a reputation-based system. It means that every client desires to cooperate with clients that previously cooperated with them and has a history of good cooperation. So for each client, a trust score must be assigned that presents the amount of his recent cooperation record in PICN. The value of local trust score that each client assigns to every acquaintance can be calculated by the number of satisfactory $request$ or $registration$ queries carried between them.

One of the most well-known mechanisms for measuring the trust in distributed networks is proposed in [19]. This method assigns a global trust value to each client based on his history of cooperation in the network. This global value is computed for each client using the vector of local scores that is received from the neighbor clients. In PICN, it is possible to compute global trust values centrally in the seeker using the trust scores of all the clients on the local network.

Trust value is an appropriate parameter in selecting the neighbors for creating virtual tunnels in preserving privacy. In data transfer, especially uploading a large data file, we propose to use a token-based method [20] to incite the clients to collaborate. As described in Section II-F, when a consumer requests a content through its neighbor, data is transferred back from the provider through this neighbor and the neighbor of the provider. The consumer can give some tokens to her neighbor. The consumer's neighbor passes these tokens to the neighbor of the provider and the provider itself. The number of tokens owned by the clients can be taken into account when computing the trust value of the clients. Besides, each consumer must have sufficient tokens to request a content in PICN. The details of trust computation and token distribution are out of the scope of this paper and is part of future works.

III. PICN Analysis and Evaluation

This section provides a comprehensive evaluation of performance of PICN. We first analyze the signaling overhead in PICN. Then, we present experimental results obtained with PICN, and finally, in Section III-C, we present our simulation and numerical results.

A. Signaling Overhead

Implementing PICN and its security mechanism involves some signaling messages exchanged between the nodes and some processing in the nodes that result in an overhead in time and an extra latency in retrieving the content. The most significant process in the seeker is a lookup in the dictionary data structure of the cache. The lookup time is never more than few milliseconds using indexing and hash table; therefore, we can neglect the process time of the seeker. Here we determine constraints on the delay values between two neighbor local networks and the size of the requested content for the retrieval latency of a content from the cache to be significantly less than its retrieval latency from the original web server, in spite of the signaling overhead.

First, let us assume that the average bandwidth and the delay between the clients and the original web server are respectively equal to $r\ Mb/s$ and $d\ ms$. The authors in [21] compute an average delay of $2\ ms$ for internal links of domains and $34\ ms$ for external links. Accordingly, we can assume that the communication delay within a local network is less than the communication delay of external links. Also, the bandwidth is usually more abundant between machines on a local network than between two neighbor local networks. Based on these observations, we determine the bandwidth and delay values on links in terms of r and d with the help of coefficients R_l, R_{nn}, D_l, D_{c_s}, D_{s_s}, and D_{nn} as follows:

$$\begin{aligned} bw_{local\_net} &= rR_l, \quad R_l > 1, \\ bw_{neighbor\_net} &= rR_{nn}, \quad R_{nn} < R_l, \\ delay_{local\_net} &= d/D_l, \quad D_l > 1, \\ delay_{client\_seeker} &= d/D_{c\_s}, \quad D_{c\_s} < D_l, \\ delay_{seeker\_seeker} &= d/D_{s\_s}, \quad D_{s\_s} < D_{c\_s}, \\ delay_{neighbor\_net} &= d/D_{nn}, \quad D_{nn} < D_{s\_s}. \end{aligned} \tag{2}$$

where bw_{local_net} and $bw_{neighbor\_net}$ are respectively the bandwidth between two clients in a local network and the bandwidth between two clients belong to two neighbor networks. Furthermore, $delay_{local\_net}$, $delay_{client\_seeker}$, $delay_{seeker\_seeker}$, and $delay_{neighbor\_net}$ are respectively the delay in local network, between a client and the seeker in its local network, between two neighbor seekers, and between two clients belonging to two neighbor networks. Now the transfer time of a content is computed by

$$\begin{aligned} T_{local\_client} &= S/(rR_l) + 2d/D_l + O_l, \\ T_{remote\_client} &= S/(rR_{nn}) + 2d/D_{nn} + O_r, \\ T_{web\_server} &= S/r + 2d + O_w, \end{aligned} \tag{3}$$

where S is the content size; the transfer time of retrieving a content from a client in the local network is T_{local_client}, from a client in the neighbor network is T_{remote_client} and from the original web server is T_{web_server}. O_l, O_r, and O_w are overheads added to compensate for representing the total overhead in transferring the content including server load, network congestion and in particular the TCP protocol overhead. Obviously, this overhead is smaller for transferring the content in the local network. Note that, it is negligible for contents with small size compared to large size. Therefore we can ignore the overheads in the following comparisons and constraints calculation in this section.

In the lookup function, two UDP packets are required for two signaling messages including the $seek$ request to the seeker and its reply. Maximum size of each one is $85\ B$ corresponding to the $minimum\ size\ of\ UDP\ packet(52\ B) + ContentID(32\ B) + Message\ Type(1\ B)$ (content ID is the hash value of content URI generated by SHA-256). Since the size of $seek$ messages is negligible, we can estimate the overhead of sending a $seek$ message or receiving its reply by the delay between a client and the seeker. According to

Equation (2), when a client requests a content from the seeker, the signaling overhead is estimated by

$$\begin{aligned} T_{local\_seek} &= 2d/D_{c\_s} + O_l, \\ T_{remote\_seek} &= 2d/D_{c\_s} + 2d/D_{s\_s} + O_r, \end{aligned} \tag{4}$$

where T_{local_seek} is the signaling overhead if the content exists in its local network; otherwise if the seeker asks its neighbor networks, the signaling overhead is T_{remote_seek}. Again because of the small size of the $seek\ request$s, the value of O_l and O_r are negligible here. As we described in Section II-E, to provide security for each content an HTTP header request is used which is requested concurrently with the content retrieval from the original server or a cache. Hence, the signaling overhead of $HTTP\ head\ request$ and its reply can be computed by

$$T_{HTTP\_head} = (h + md)/r + 2d + O_h, \tag{5}$$

where h is the HTTP header size without meta-data, md is the size of PICN meta-data, and O_h is the extra overhead in transferring HTTP response. Now if the inequalities

$$\begin{aligned} T_{local\_client} + T_{local\_seek} &< T_{HTTP\_head}, \\ T_{remote\_client} + T_{remote\_seek} &< T_{HTTP\_head}, \end{aligned} \tag{6}$$

are true, the HTTP header reply is received after retrieving the whole content, so the content retrieval latency is equal to the time of requesting and receiving an HTTP header; otherwise, it is equal to $T_{local\_client} + T_{local\_seek}$ or $T_{remote\_client} + T_{remote\_seek}$. In the first case, to make sure that the time for requesting and receiving an HTTP header from the original web server is smaller than the time of retrieving the whole content from it, we must solve the inequality

$$T_{HTTP\_head} < T_{web\_server}. \tag{7}$$

Here if we assume that the whole content size is the aggregate of its header size, h, and the content itself, s, then by substituting Equation (5) into Equation (7), we obtain

$$\frac{h + md}{r} + 2d < \frac{h + s}{r} + 2d. \tag{8}$$

This inequality is true when

$$s > md, \tag{9}$$

so in this case, when the content size is greater than the meta-data size, the PICN latency is lower than the original web server latency. In the second case, corresponding to a worst case scenario, the content is retrieved from the remote client, so the retrieval time is equal to $T_{remote\_client} + T_{remote\_seek}$. Therefore in order to retrieve the content from PICN faster than the original web server, the inequality

$$T_{remote\_client} + T_{remote\_seek} < T_{web\_server}, \tag{10}$$

must be true. By substituting Equation (3) and (4) into Equation (10) and integrating, we get

$$\frac{S}{rd} > \big(\frac{R_{nn}}{R_{nn} - 1}\big)\big(\frac{2}{D_{c\_s}} + \frac{2}{D_{s\_s}} + \frac{2}{D_{nn}} - 2\big). \tag{11}$$

Table I: Experiment information

Total duration	6 $hours$
Number of users	4
Total requests	11790
Total PICN supported requests	8065
Total content size	3.24 GB
Hit ratio in clients	22.30%
Reduced external traffic ratio	27.36%

Now since $D_{nn} < D_{s\_s} < D_{c\_s}$, by substituting D_{c_s} and D_{s_s} into D_{nn}, we obtain

$$\frac{S}{rd} > \big(\frac{R_{nn}}{R_{nn} - 1}\big)\big(\frac{6}{D_{nn}} - 2\big). \tag{12}$$

The equality in Equation (12) is true if

$$D_{nn} \geq 3. \tag{13}$$

So despite the signaling overhead, the total time for retrieving a content with PICN is less than its retrieval time from the original web server. Therefore, the two constraints for a lower latency of PICN compared to the retrieval from the original web server are: (1) the content size must be greater than the PICN meta-data size (Equation 9), and (2) the delay between the client and the original web server must be at least 3 times greater than the delay between the two clients in the two neighbor local networks (Equation 13).

If the requested content misses the available cache storage in PICN, there is an overhead equal to T_{remote_seek} that is added to the latency of receiving the content from the original web server. We analyze this overhead by simulations, in Section III-C.

B. Experimental Results

In this section, a local implementation of PICN is examined. For this purpose, we developed the seeker and client softwares in the QT cross-platform development framework.

The seeker framework is executed on an arbitrary machine in a LAN. Any other machine on the LAN runs the client framework (given the seeker machine IP address) to share its own cache and use other clients' cache storage. The client framework plays the role of a proxy server for the browser, so a proxy setting configuration is required for the browser to use it. Accordingly, every HTTP request is forwarded to the client framework and is replied through it. We determine a set of content types for PICN that supports all the images, videos and flash files. The contents of this set are cached and retrieved by PICN. For all of the content types not covered by this set, the client framework requests the content directly from the original web server.

We performed a simple experiment (Table I). Four students of the department were asked to install the client framework on their machines. The seeker framework was installed on a separate machine. The experiment was done in about 6 hours, monitoring the clients' web surfing. We asked clients to browse their desired pages contained images and video downloads. In this period, a total number of 11790 HTTP requests with a total traffic size of 3.244 GB were issued by the users. 68% of the requests are in the supported set

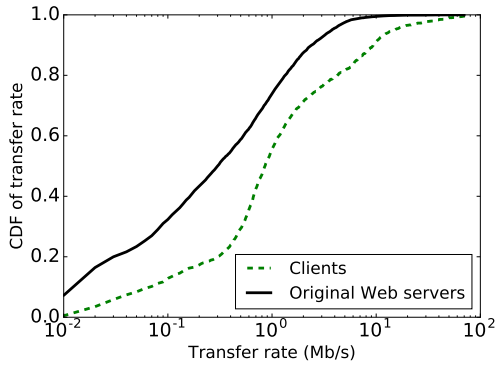


Figure 7: Average transfer rate in the experiment from different providers. (The contents in the supported set which hit the clients are retrieved from the clients; Content types that are not supported and those which miss the clients are retrieved from the original servers.)

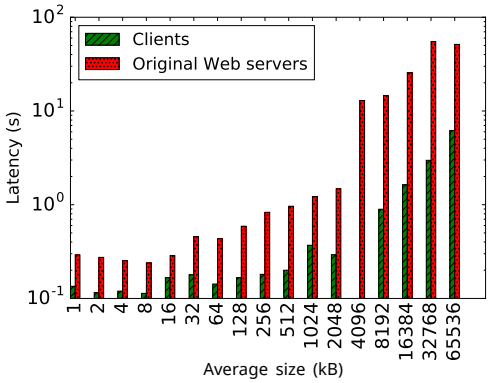


Figure 8: Average latency for different average sizes in the experiment

in this experiment. For 22.3% of this supported requests, PICN succeeded to supply them from the clients and 16.3% from the local cache of the clients, while 61.4% missed the clients' cache storage; therefore they were requested from the original web servers. In this experiment, 27.4% of the external bandwidth is saved by PICN (in addition to 6.9% which is saved because of browser's local cache).

Figure 7 presents the Cumulative Distribution Function (CDF) of the transfer rate of HTTP contents when they are retrieved from the original web servers or the clients. In the diagram, the downloads from original web servers include both missed contents in the clients and unsupported contents which are directly retrieved from the web servers. As shown in this figure, PICN downloads have significantly higher transfer rate compared to downloads from the original web servers. As a result, the latency of content retrieval is decreased when using cache storage of clients on the local network, which is shown in Figure 8.

Table II: Bandwidth and delay values for various client to client links of simulation topology

	layer1	**layer2**	**remote**
Bandwidth	80 Mb/s	70 Mb/s	60 Mb/s
Delay	0.25 ms	1 ms	10 ms

C. Simulation Results

We made a trace-driven simulation to investigate PICN with more than one seeker, using real HTTP requests patterns[2]. In this evaluation, UC Berkeley Home IP Web Traces [22] and Web cache access logs from the IRCache proxies[3] [23] were used to generate the HTTP request patterns. The details are as follows.

Each simulation is performed for some local networks. Every local network is a two-tier topology of access switches, as shown in Figure 9. A switch is connected to at most 24 clients and several layer 2 switches are connected to each layer 1 switch. It is also assumed that each local network is in the neighborhood of another network with the probability of 0.33. Also, if the content is not cached locally in the network, the request is resolved by a neighbor seeker (with one hop from the local seeker). If none of the seekers in the neighborhood of the local seeker has a registered entry for the requested content, it is requested from the original web server.

The client link is assumed to be 100 Mb/s from which depending on the aggregation level some part will be available for the PICN traffic. Table II represents the upload bandwidth and delay values for various links, between two clients connected to the same layer 1 switch ($layer1$ $client$ to $client$), between two clients connected to different layer 1 switches in the same local network ($layer2$ $client$ to $client$), between two clients in two neighbor local networks ($remote$ $client$ to $client$). In addition, the delay between a client and the seeker and between two neighbor seekers are assumed 0.5 ms and 10 ms respectively. Based on these delay values we estimate the signaling time for a lookup based on the RTT between a client and seeker or between two seekers. Also, the dedicated bandwidth value on the external link for PICN is considered 800 Mb/s (from total bandwidth of 1 Gb/s). This value is divided between all the transfers from original web servers. So, the transfer rate from the original web server depends on the web server speed and the available bandwidth on the external link.

We estimate delay and bandwidth values for accessing the web servers in trace files using a list of websites from different categories of Alexa [24] top sites. For this reason, the delay is computed by sending ICMP packets, and the average bandwidth is estimated using the retrieval time of the first page of each website. These values are assigned to the links between the edge of the simulation network and each web server in the trace files.

[2]The simulation script has been written in python using networkx and FNSS (Fast Network Simulation Setup) packages. The simulation package is available at code ocean with DOI: 10.24433/CO.d553b1ad-81dd-4152-8bd6-0165b78edad7

[3]Unfortunately IRCache web site is not available any more, however the authors can be contacted to provide the trace files.

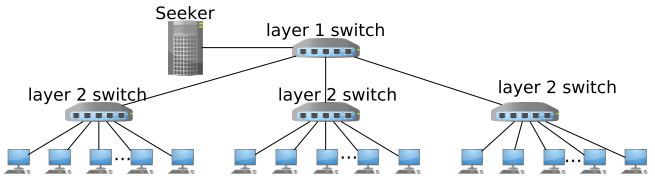


Figure 9: Topology of local networks in the simulations

Table III: Berkeley trace files information

Date	1996
Total duration	4 days
Number of users	8374
Total number of requests	8559556
Total number of PICN supported requests	5897643
Total number of contents	2230517
Maximum size of supported contents	4.3 GB
Average size of supported contents	9.6 KB
Total size of supported contents	57 GB

The security metadata length is assumed to be 1281 B[4]. Hit ratio and the average latency for retrieving the contents are computed by simulation. Latency is the time spent between requesting an HTTP content and receiving the last byte of its reply. For calculating the latency, the bandwidth on each link is shared between all the simultaneous content flows on that link. Transferring content between two clients on the same local network is called local client retrieval and between two clients of neighbor local networks is called remote client retrieval.

1) Berkeley trace files: This dataset includes 18 days of HTTP traces gathered from the Home IP service offered by UC Berkeley to students, faculty, and staff. There are 8559556 requests from 8374 clients for 2230517 different URLs in this trace file. Table III describes trace file information. Given that the number of different URLs is only about a quarter of the number of requests, there are many redundant requests for the same contents. So the hit ratio in the caches is high with this dataset.

We also assume 10 local networks and 10 seekers for the underlying overlay topology. We map 8374 clients uniformly to the machines of all local networks. So for each local network, there are about 837 clients that are connected to one seeker. A cache storage of 100 MB is assigned to each client. So the total cache storage of all the clients is about 83 GB which is comparable to the total size of the supported contents for caching in PICN which is about 57 GB. Simulation results for the trace file show that the hit ratio is 37.4% in the local machine caches, 24.2% in local clients and 10% in remote clients; so total hit ratio is 71.6%. According to this simulation, 2.9% of external traffic is reduced using PICN with 100 MB of cache size for each client.

The average latency for different sizes is computed and presented in Figure 10 to clarify the latency improvement. Figure 11 has some more details; we show latency in these

[4](300 B for maximum length of an RSA-2048 public key structure) + (245 B for an Encrypted SHA-512 hash value that is a maximum block size for encrypting with RSA-2048) + (735 B for certification, since 3 blocks are required for encrypting pair of publisher domain name and public key) + (1B for access-control flag)

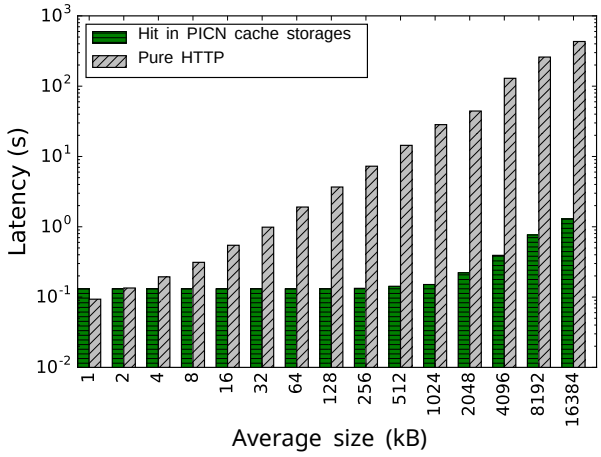
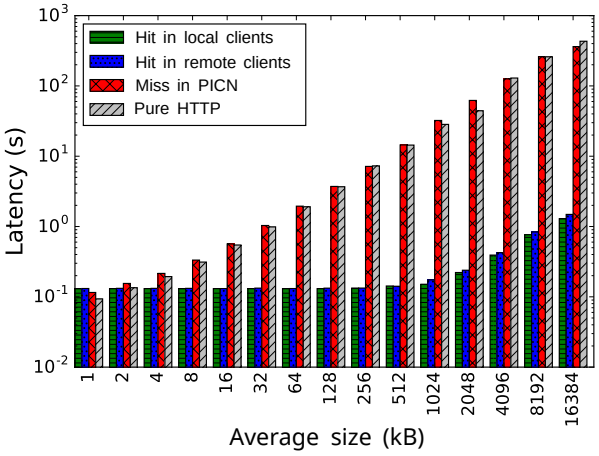
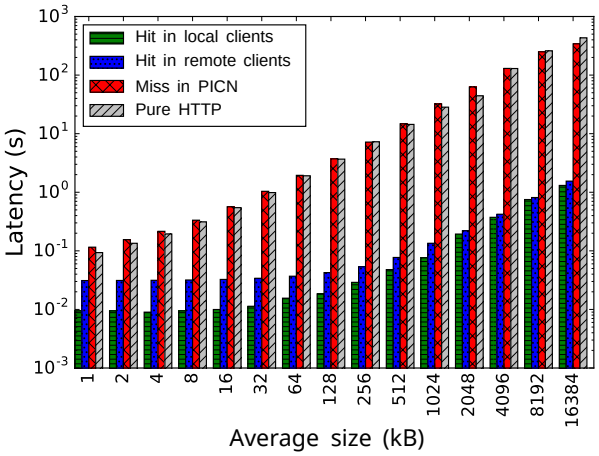


Figure 10: Content latency for Berkeley trace files simulation



(a) With security signaling overhead



(b) Without security signaling overhead

Figure 11: Content latency for different providers when using PICN or without PICN for Berkeley trace files

Table IV: IRCache trace files information

Date	$9th\ Jan$ 2007
Total duration	24 hours
Number of users	816
Total number of requests	3803030
Total number of PICN supported requests	3561233
Total number of contents	3290208
Maximum size of supported contents	730 MB
Average size of supported contents	46 KB
Total size of supported contents	163 GB

diagrams when the content is retrieved from different providers in PICN. $Pure\ HTTP$ indicates retrieving all the requests using HTTP without asking PICN or using the local cache. The diagrams show that for each average size, latency is slightly lower for retrieving the content from PICN caches compared to the original web server, except for the first content size class. The different behavior of the diagram for the size below $2KB$ is because of signaling overhead that is discussed in Section III-A.

Since the meta-data is assumed to be 1281 B in the simulations, for the content sizes below this value, PICN latency is greater than the original web server. This is because of the overhead of access control in PICN. The diagram in Figure 11-(b) represents the case for PICN without applying security signaling. The delay between two neighbor local networks is significantly higher than the delay between clients in a local network. So remote client retrieval has a higher latency compared to local client retrieval, even for small size contents. Totally, latency is decreased using PICN with average ratio of 78% and up to 99.7%. Consequently, content transfer rate is improved with average ratio of 69.2% and up to 96.3%.

2) IRCache trace files: This dataset includes cache access logs from the IRCache proxies during day 9th of January 2007 for 8 root cache servers located in cities throughout the US (Table IV). We performed two simulations for each day because the clients IP addresses have been anonymized by changing the IP addresses and the mapping of real-to-anonymous IP addresses are different from day to day. Some IP addresses in the trace files are NAT addresses, so there are many requests from these addresses for the same contents. Therefore we consider 10 most repeated client addresses in each trace file as NAT and assign uniformly one of the other client addresses to the correspondent requests.

In this simulation, there are 816 clients, each one has a cache storage of only 100 MB. With the described simulation conditions for IRcache trace files, hit ratio is 11.8% in local clients, 0.15% in remote clients and 5.2% in local machines. Therefore totally 17.1% of the requests hit in PICN and consequently 4.4% of external traffic is reduced. Figure 12 presents the latency diagram for content retrieval. It is apparent that the latency is decreased when using PICN for replaying IRCache trace files. Further, content latency without signaling overhead is shown in Figure 13-(b). According to these diagrams, for IRCache trace files, latency is decreased using PICN with an average ratio of 77.8% and up to 99.8%. Also, content transfer rate is improved with an average ratio of 71.6% and up to 97.8%.

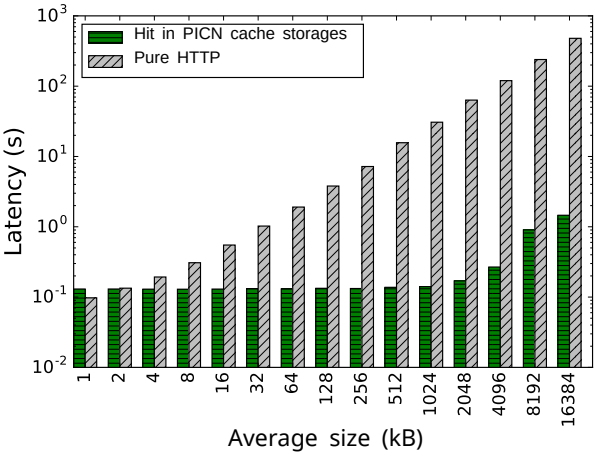
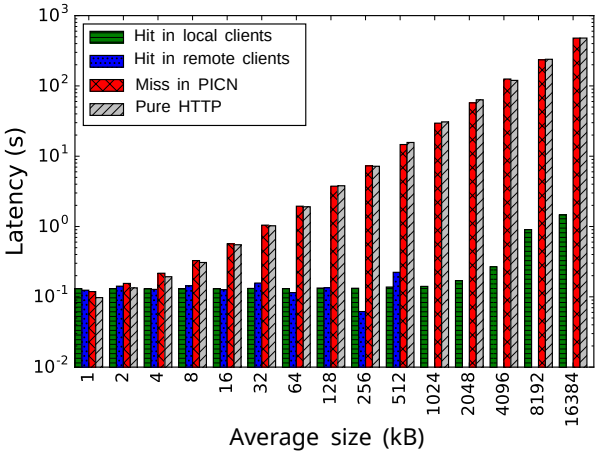
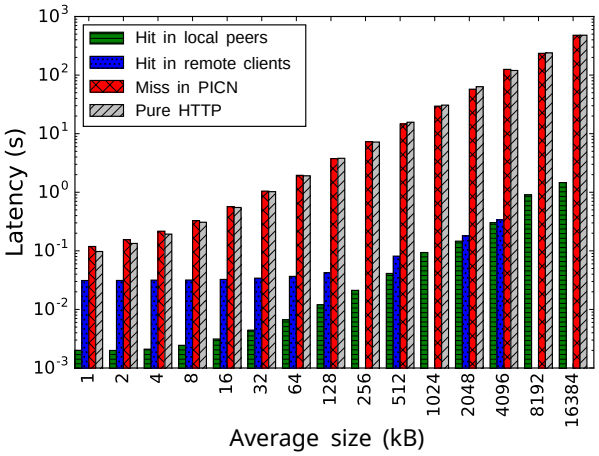


Figure 12: Content latency for IRCache trace files simulation



(a) With security signaling overhead



(b) Without security signaling overhead

Figure 13: Content latency for different providers when using PICN or without PICN for IRCache trace files

Table V: Comparing hit ratio value in PICN with proxy servers

(a) Simulated proxy servers with Berkeley trace files

	Client cache size	Local clients	Remote clients	Local cache	Proxy servers	Total hit	Reduced external traffic
PICN	10 MB	24.1%	10.5%	36.1%	-	70.7%	2.6%
PICN	50 MB	24.2%	10%	37.4%	-	71.6%	2.8%
PICN	100 MB	24.2%	10%	37.4%	-	71.6%	2.9%
PICN	1 GB	24.2%	11.8%	37.4%	-	73.4%	4.8%
PICN	2 GB	24.2%	11.1%	37.4%	-	72.7%	5%
PICN	5 GB	24.2%	12%	37.3%	-	73.5%	92.4%
Proxy servers	-	-		24.7%	34.4%	59.1%	80%

(b) IRCache proxy servers

	Client cache size	Local clients	Remote clients	Local cache	Proxy servers	Total hit	Reduced external traffic
PICN	10 MB	11.9%	0.2%	3.5%	-	16.2%	3.3%
PICN	50 MB	11.8%	0.1%	5.1%	-	17%	4.2%
PICN	100 MB	11.8%	0.1%	5.1%	-	17%	4.4%
PICN	1 GB	11.8%	0.1%	5.2%	-	17.1%	4.4%
PICN	5 GB	11.8%	0.2%	5.2%	-	17.2%	4.4%
Proxy servers	-	-		-	18.8%	18.8%	5.4%

3) Comparing with Central Cache Servers: We also compare PICN with a solution based on a central proxy server. For Berkeley trace files, we replace each seeker with a central cache server and perform a simulation to compute the hit ratio in proxy servers. The local cache is still enabled for each client, but it does not share its cache with others. The cache storage size of each central server is chosen to be sufficiently large to be able of caching all the contents of the trace files. Also, the bandwidth on upload link of proxy server is considered 800 Mb/s. For IRCache trace files, 8 local networks with 8 seekers are assumed instead of 8 root proxy servers. So we compare hit ratio in PICN with the hit ratio in 8 root proxy servers according to hit ratio in the trace files.

Hit ratio and reduced external traffic load are presented in Table V for PICN with different sizes of client cache storages and central cache servers with a sufficiently large cache. Comparing total hit ratio in PICN with proxy servers, we conclude that the same value of hit ratio is achieved in PICN as in proxy servers, even with a small size of cache for each client. But to achieve the same value of reduced external traffic load for PICN as in proxy servers, it is required to choose a suitable cache size for clients. It is more obvious in the case of Berkeley trace files because the maximum size of content that is repeated in the requested contents is larger than that in IRCache trace file. The average size of contents is only about $9.6KB$ in these trace files while the maximum size of contents is more than 4 GB. As a result the external traffic reduction ratio increases from 5% to 92.4% when the client cache storage exceeds from 2 GB to 5 GB, because the content with maximum size is repeated many times in the trace file.

Nowadays, a storage of 10 GB is not considered a large size, so each client can easily dedicate such an amount of memory for PICN and share with others. Average and largest size of contents in IRCache files which are more recent trace files compared to Berkeley, are respectively 46 KB and 730 MB. So, for IRCache trace files, even with a cache storage of 50 MB for each client, the external traffic ratio reduction is close to its value in proxy servers.

Table VI: Comparing hit ratio value in PICN for different availability probability of clients

(a) Berkeley trace files

Availability probability	Local clients	Remote clients	Local cache	Total hit
1	24.2%	10%	37.4%	71.6%
0.8	22.8%	9.5%	37.3%	69.6%
0.6	20.9%	11.2%	37%	69.1%

(b) IRCache trace files

Availability probability	Local clients	Remote clients	Local cache	Total hit
1	11.8%	0.15%	5.1%	17.0%
0.8	11.1%	0.15%	5.1%	16.3%
0.6	10.2%	0.25%	5.1%	15.5%

Some extensions can be considered for PICN, such as caching large files in more than one client and enabling partial download mechanisms. Another solution is to use a static central cache server in each local network for large files.

4) Clients Availability: In the previous simulation, all clients were assumed to be available during the simulation time. Next, we consider that each client is available with a given probability. The results are presented in Table VI for cache storage of 100 MB. As it is expected, the hit ratio decreases when the clients are not permanently available. However, the results from both trace files show that even for the availability probability of 60%, hit ratio only decreases from 71.6 to 69.1 for Berkeley trace files and from 16.8 to

15.2 for IRCache.

IV. Related work

There are many proposals in ICN such as *DONA* [25], *PURSUIT* [26], *CCN* [15], *SAIL* [27], and *COMET* [28]. Despite the valuable benefits of these ICN proposals, the gains are achieved with significant costs [6], [7]. In fact, there are two main drawbacks with theses proposed ICN architectures. First, they apply a self-certifying name system to the current Internet. This means that the traditional URIs must change which requires application-layer modifications. There exist millions of content providers (websites) running different kinds of application platforms. It is unlikely one can convince the content providers to make such a big modification in their applications. Second, the proposed ICNs require some fundamental changes in the network infrastructure. It is extensively discussed in previous work [6]. Such a significant modification in the network layer looks unrealistic.

To overcome these problems, PICN makes minimal changes in the HTTP layer. There is no need to change the application or the network layer. We only add two fields to the HTTP header to achieve content-centric security. These fields can be ignored by non-PICN traffic [17]. So the system is backward compatible and can be incrementally deployed at the network edge.

With a similar approach to PICN, [29] proposes ICN-enabled wireless access points which deploy ICN nodes at the edge of the network. This solution results in avoiding numerous hops to content files as well as localizing bursty traffic caused by increasing number of mobile devices. The main idea is based on Nano Data Centers (NaDa) [30] that provide facilities for ISPs to use some gateways that act as nano servers for computing and storage services. NaDa has been proposed to overcome the problems of centralized design principles such as high energy consumption. According to [30], energy consumption is reduced because of distributed nature of NaDa compared to central data centers with enormous energy consumption. Traffic localization resulting from the co-location between NaDa and end users and utilizing the gateways that are already powered up and active also contribute to the reduction of the energy consumption. As a similar edge based solution, [31] proposes to benefit from the storage available at wireless access points in MobilityFirst architecture.

As a more traditional solution in Internet, proxy servers have been largely deployed as a popular caching system in customer networks [32]. Proxy caching solution suffers from the very same centralization problems that NaDa tries to solve. Generally speaking, this solution has some important drawbacks: (1) It has high energy and management cost, (2) a proxy cache fails to work with secure protocols like HTTPS, (3) it may become a bottleneck, (4) it has limited storage capacity for caching, and (5) user privacy is easily violated.

In [33], the authors proposed Home Router Sharing based on Trust (HORST) which uses information from social networks to enable the users sharing their retrieved contents via WiFi access points. HORST establishes a NaDa on the home router for content caching. In PICN, for WiFi networks, the seeker service can be implemented within WiFi access points. In [33], it is suggested that the cache storage is also placed in the WiFi access points, which is less practical idea, while seeker in PICN is only responsible for the cache lookup and management.

Unlike in NaDa-based methods and proxy servers, PICN replaces the NaDa gateway or cache proxy server with a content seeker and provides caching by clients systems at the network edge. In this way, PICN achieves large cache storage at almost no cost. For instance, let us assume 1000 clients and each one can dedicate $10\ GB$ of their storage devices to PICN, with a dedicated bandwidth of $1\ Mbps$ on its connected links. Hence, a $10\ TB$ cache storage would be obtained. Achieving this large cache with high available upload bandwidth is important as authors in [6] show the cache size plays an essential role in different performance metrics in ICN. Also, they conclude that putting all the caches at the edge and enabling local-scoped cooperation can improve the performance metrics.

Another noticeable improvement in PICN compared to solutions based on proxy cache servers is that in PICN the seeker acts as a mediator that communicates only some short messages with the clients. Consequently, there is no data communication between the seeker and the clients. So unlike a proxy cache server or NaDa gateways, PICN avoids computational and bandwidth bottlenecks. The next important benefit is that the seeker cannot violate privacy easily in comparison to a simple cache server. Because the seeker works with content hashes and encrypted data and the users can use a privacy preserving mechanism. Last, but not the least, an important advantage of PICN compared to solutions based on proxy cache servers is its support of content-centric security. The signaling overhead when a request misses the caches in PICN causes a small amount of increase in latency. Although this might be considered as a drawback for PICN. But high request rate and high load on links could impose a high load on a central proxy server that consequently results in a higher latency and lower transfer rate compared to PICN. Obviously the amount of storage for RIT tables in PICN seeker is much less than the amount of storage needed to store the contents themselves in cache proxies.

The idea of enabling web browsers on end users to share local caches has been proposed in [11], with *squirrel*. The main purpose of squirrel is to form an efficient and scalable web cache, without the need for a dedicated cache storage. In squirrel, each request routes through other nodes to find a target node which is responsible for introducing a new node containing the requested content. In contrast, PICN sends each request directly to a known seeker in the local network to find the content. This results in decreasing the lookup function overhead. Furthermore, PICN adopts a different paradigm from squirrel in proposing a comprehensive architecture including some protocols for providing ICN primitives such as content-based lookup, content-based security, and privacy.

Also, the authors in [34] survey the different proposals on the combination of CDN and P2P. One particular instance of such a system is Akamai's Netsession system [35]. PICN is

fundamentally different in terms of business model, in addition to differences in its architecture. In short, PICN is an extension of ICN with P2P while Netsession is an extension of a CDN with P2P.

In terms of business model, in a CDN such as Akamai, the content providers pay the CDN for their caching service [8]. Moreover, cache storages are located within the access operator's network. This is done without any charges and only based on a mutual agreement because both the CDN and network operators have their own incentives to deploy the storages in this way. On the contrary, in almost all ICN proposals the solution is built into the network and becomes the inherent operation of the network. Our solution allows to speed up the response time of the customer network for its clients and to decrease outbound traffic load. It is possible to incite the end-users to contribute in this system as detailed in Section II-G. Content providers may have interest in PICN as well because it can improve the Quality of Experience (QoE) for their customers by bringing down the content retrieval time. It also reduces their costs by reducing the load on their servers.

In terms of architecture, Netsession relies on a control plane operated by a collection of Akamai servers for providing security, NAT and firewall functions, and registrations and lookup for contents on peers. Also, some edge servers of Akamai play some roles in Netsession. While it is sufficient for a network that wants to join PICN to dedicate a server as the seeker, and install the seeker service on it.

Finally, unlike Netsession, PICN can be used in ad-hoc mode for home networks. In this simplified case, PICN can work without any seekers. It eliminates the seeker's role in the original PICN. For this reason, the $client\ software$ broadcasts each request in the home network. Since almost all users' devices run content retrieval applications such as social networks and there exists a significant amount of common interests among home users, PICN is likely to achieve good performance in this case. It decreases the home external bandwidth consumption and content retrieval latency by only installing a simple application.

V. Conclusion

In this paper we propose a practical ICN framework that requires no changes neither in application layer nor in network layer. PICN leverages unused resources at the network edge including communication capability of local networks and storage capacity of client machines to provide ICN capabilities at low cost and equipment overhead. Also with the content-centric security method of PICN, one of the most important qualitative benefits of ICN goals is achieved. Using edge caching with the help of end users, in average 68% decrease in HTTP response time is met while the external traffic is also reduced significantly. Our simulations show that PICN results in almost the same or even higher hit ratio in small cache storages of clients compared to a solution based on central proxy servers with large and costly cache storage and relatively much higher management and maintenance cost.

Future work includes the improvement of incentive design, cache policies, and complementary solutions for privacy preserving. We plan to evaluate a comprehensive implementation of PICN in real practical conditions and also customization for various application scenarios such as home networks, enterprise solutions and WAN.

Acknowledgements

The authors would like to thank Mathieu Goessens and Davide Frey for their help and support in using IRCache Data.

References

[1] R. Schollmeier, A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications, in: Proc. 1th International Conference on Peer-to-Peer Computing, August 2001, pp. 101–102.

[2] S. Paul, J. Pan, R. Jain, Architectures for the future networks and the next generation internet: A survey, Computer Communications 34 (1) (2011) 2–42.

[3] G. Carofiglio, G. Morabito, L. Muscariello, I. Solis, M. Varvello, From content delivery today to information centric networking, Computer Networks 57 (16) (2013) 3116–3127.

[4] G. Xylomenos, C. Ververidis, V. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. Katsaros, G. Polyzos, A survey of information-centric networking research, IEEE Communications Surveys Tutorials 16 (2) (2014) 1024–1049.

[5] G. Zhang, Y. Li, T. Lin, Caching in information centric networking: A survey, Computer Networks 57 (16) (2013) 3128–3141.

[6] S. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, S. Shenker, Less pain, most of the gain: Incrementally deployable icn, in: Proc. the ACM SIGCOMM Conference, August 2013, pp. 147–158.

[7] M. Mangili, F. Martignon, A. Capone, A comparative study of content-centric and content-distribution networks: Performance and bounds, in: Proc. IEEE Global Communications Conference (GLOBECOM), June 2013, pp. 1403–1409.

[8] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, B. Ohlman, A survey of information-centric networking, IEEE Communications Magazine 50 (7) (2012) 26–36.

[9] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, Web caching and zipf-like distributions: evidence and implications, in: Proc. INFOCOM, March 1999, pp. 126–134.

[10] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, A. Zhang, Improving traffic locality in bittorrent via biased neighbor selection, in: Proc. 26th IEEE International Conference on Distributed Computing Systems, July 2006, pp. 66–.

[11] S. Iyer, A. Rowstron, P. Druschel, Squirrel: A decentralized peer-to-peer web cache, in: Proc. 21th Annual Symposium on Principles of Distributed Computing, July 2002, pp. 213–222.

[12] A. Chanda, C. Westphal, Contentflow: Adding content primitives to software defined networks, in: IEEE Global Communications Conference (GLOBECOM), 2013, pp. 2132–2138.

[13] S. Ratnasamy, M. Handley, R. Karp, S. Shenker, Topologically-aware overlay construction and server selection, in: Proc. 21th Annual Joint Conference of the IEEE Computer and Communications Societies, Vol. 3, June 2002, pp. 1190–1199.

[14] J. Rosenberg, Rfc5245: Interactive connectivity establishment (ice): A protocol for network address translator (nat) traversal for offer/answer protocols, https://tools.ietf.org/html/rfc5245 (April 2010).

[15] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, R. L. Braynard, Networking named content, in: Proc. 5th International Conference on Emerging Networking Experiments and Technologies, December 2009, pp. 1–12.

[16] A. Arsenault, S. Turner, Internet x.509 public key infrastructure: roadmap internet draft, http://www.ietf.org/internet-drafts/draft-ietf-pkix-roadmap-09.txt. (2002).

[17] IETF Network Working Group, Hypertext Transfer Protocol – HTTP/1.1, http://tools.ietf.org/html/rfc2068 (January 1997).

[18] R. Dingledine, N. Mathewson, P. Syverson, Tor: The second-generation onion router, in: Proc. 13th Conference on USENIX Security Symposium, Vol. 13, August 2004, pp. 21–21.

[19] S. D. Kamvar, M. T. Schlosser, H. Garcia-Molina, The eigentrust algorithm for reputation management in p2p networks, in: Proc. 12th International Conference on World Wide Web, May 2003, pp. 640–651.

[20] J. Xu, W. Zame, M. van der Schaar, Token-based incentive protocol design for online exchange systems, in: Proc. 3th International ICST Conference on game Theory for Networks: Revised Selected Papers, 2012, pp. 248–258.
[21] J. Rajahalme, M. Särelä, K. Visala, J. Riihijärvi, On name-based inter-domain routing, Computer Networks 55 (4) (2011) 975–986.
[22] Berkeley, UC Berkeley Home IP Web Traces, http://ita.ee.lbl.gov/html/contrib/UCB.home-IP-HTTP.html (November 1996).
[23] Web cache access logs from the ircache proxies during the "day in the life of the internet" in january 2007, http://imdc.datcat.org/collection/1-01J0-5=IRCache+traces+for+DITL+January,+2007.
[24] The top sites on the web, http://www.alexa.com/topsites (March 2017).
[25] T. Koponen, M. Chawla, B. Chun, A. Ermolinskiy, K. Kim, S. Shenker, I. Stoica, A data-oriented (and beyond) network architecture, SIGCOMM Computer Communication Review 37 (4) (2007) 181–192.
[26] N. Fotiou, P. Nikander, D. Trossen, G. Polyzos, Developing information networking further: From psirp to pursuit, in: Proc. 7th International ICST Conference on Broadband Communications, Networks, and Systems, Vol. 66 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, October 2010, pp. 1–13.
[27] SAIL, Scalable and Adaptive Internet Solutions (SAIL), http://www.sail-project.eu/wp-content/uploads/2013/01/SAIL-DB3-v1.1-final-public.pdf (January 2013).
[28] G. Garcia, A. Beben, F. Ramon, A. Maeso, I. Psaras, G. Pavlou, N. Wang, J. Sliwinski, S. Spirou, S. Soursos, E. Hadjioannou, Comet: Content mediator architecture for content-aware networks, in: Proc. Future Network Mobile Summit (FutureNetw), December 2011, pp. 1–8.
[29] S. Eum, Y. Shoji, M. Murata, N. Nishinaga, Design and implementation of icn-enabled ieee 802.11 access points as nano data centers, J. Netw. Comput. Appl. 50 (C) (2015) 159–167.
[30] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, P. Rodriguez, Greening the internet with nano data centers, in: Proc. 5th International Conference on Emerging Networking Experiments and Technologies, December 2009, pp. 37–48.
[31] F. Zhang, C. Xu, Y. Zhang, K. K. Ramakrishnan, S. Mukherjee, R. Yates, T. Nguyen, Edgebuffer: Caching and prefetching content at the edge in the mobilityfirst future internet architecture, in: Proc. IEEE 16th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), June 2015, pp. 1–9.
[32] A. Luotonen, K. Altis, World-wide web proxies, Computer Networks and ISDN Systems 27 (2) (1994) 147–154.
[33] M. Seufert, V. Burger, T. Hossfeld, Horst - home router sharing based on trust, in: Proc. 9th International Conference on Network and Service Management, October 2013, pp. 402–405.
[34] N. Anjum, D. Karamshuk, M. Shikh-Bahaei, N. Sastry, Survey on peer-assisted content delivery networks, Computer Networks 116 (C) (2017) 79–95.
[35] M. Zhao, P. Aditya, A. Chen, Y. Lin, A. Haeberlen, P. Druschel, B. Maggs, B. Wishon, M. Ponec, Peer-assisted content distribution in akamai netsession, in: Proc. Conference on Internet Measurement, October 2013, pp. 31–42.