On Adapting HTTP Protocol to Content Centric Networking

Sen Wang
Tsinghua University
Beijing, China

Jun Bi
Tsinghua University
Beijing, China

Jianping Wu
Tsinghua University
Beijing China

Xu Yang
Tsinghua University
Beijing China

Lingyuan Fan
Huawei Inc.
Shenzhen China

{wangsen, yangxu}@netarch.tsinghua.edu.cn; {junbi, jianping}@cernet.edu.cn; fanlingyuan@huawei.com

ABSTRACT

Designed around host-reachability, today's Internet architecture faces many limitations while serving content-oriented applications which generate most traffic load to the Internet. CCN (Content Centric Networking) [1] is one of the most important proposals for future Internet architecture, which aims to build a content/data oriented network to solve these limitations. On the other hand, HTTP is the most important protocol to deploy new services and applications on current TCP/IP-based Internet. In this paper, we attempt to run HTTP protocol on CCN and combine the two by stitching them semantically on their content-oriented features, such as content caching. We expect that this combination can be leveraged to build CCN testbed with real HTTP traffic which is vital to validation and redesigning of specific mechanisms of CCN and to finding a transition way of CCN in which great incentive is provided for service providers in the economic ecosystem of content distribution. We designed and implemented a HTTP-CCN gateway to transform HTTP request and HTTP response into CCN Interest and Data respectively. We illustrate how to semantically map HTTP caching to CCN caching, which is one of the most attractive properties of CCN. We also discuss how to achieve transparent caching with CCN and find out that it is nontrivial to achieve complete transparency of caching with CCN given no cooperation with CDNs and content providers.

Keywords

Content Centric Networking, HTTP, Caching

1. INTRODUCTION

Recently, the networking research community has been recognizing the increasing importance of content and show tremendous interest on content oriented network which provide content-relevant features directly in the network layer. Content Centric Networking (CCN) proposed by Van Jacobson et al. in [1] is among many proposals of content/information/data oriented network architectures. In CCN, the sender and the receiver are decoupled in a way which is much similar to the Publish and Subscribe service model. Content is named directly by length-unlimited hierarchical URI-like names, which are used by routing system to transmit the information to the interested parties instead of network addresses. Every CCN node is enabled with caching capability. Obviously, CCN has a totally different using primitive for data transmission from traditional Socket-based primitive of TCP/IP network. It is obvious that enormous efforts are needed to rewrite current applications running over IP to run over CCN, which hinders the utilization of CCN.

Currently, an Open Source Platform named CCNx is provided by PARC, implementing the design described in [1]. Although many functions are still at an early stage, it could already support some relatively complicated applications as such VoCCN [3], ACT [4], etc. However, these prototypes are fully designed for CCN architecture. The capability of leveraging existing IP infrastructure and applications into CCN's research activities is helpful to further understand new features of CCN as well as to identify its missing pieces.

HTTP has become the predominant protocol for deploying new services and applications. In particular, the HTTP traffic has been growing explosive as we have observed [6], which is driven by the prevalence of the existing HTTP infrastructure. As an application-level protocol, HTTP can be viewed as a content centric protocol to some extent, as each HTTP method specifies the name of resource (content) it copes with [5]. Proxies along a request's path are allowed to cache the response and to redirect the request to the closest or least loaded server storing a copy of the content. These features are much similar with DONA [2] and CCN etc., which intend to build a content oriented network at the network layer instead of application layer.

So far, most research works on CCN are based on simulation or small-scale testbed. In-depth research on CCN needs relatively large-scale test and real traffic to prove its feasibility and performance. Furthermore, as a clean slate future Internet architecture, how to combine the new design with the traditional Internet and make an easy transition of CCN is still an open issue. Finally, although the content centric network provides many content-oriented features such as in-network caching, it still need massive effort to figure out whether or how this feature meet application needs and how the new architectural design interact with pre-existing content delivery infrastructure (e.g. CDN) and content identification mechanism(e.g. HTTP content description). To address the issues above, in this paper we try to run HTTP protocol on CCN and combine the two by stitching their semantically on their content-oriented features, such as content caching. We designed and implemented a

CFI'12, September 11–12, 2012, Seoul, Korea.

HTTP-CCN gateway to convert http request and http response to CCN Interest and Data. We present considerations in our design course and the tradeoffs when accommodate CCN features to HTTP communication patterns.

The rest of this paper is organized as follows. In Section 2, the basic CCN communication model is present. Section 3 illustrates our prototype design and implementation of the HTTP-CCN gateway. In Section 4, we present how to enable the HTTP-CCN gateway to caching HTTP video traffic transparently and also present some obstacles we cannot overcome yet. Finally, we give our conclusion in Section 5.

2. CCN COMMUNICATION MODEL

CCN maintains three tables to guide the router for content distribution: *Forwarding Information Base* (FIB), P*ending Interest Table* (PIT) and *Content Store* (CS). Entries in FIB are name prefixes instead of IP prefixes, which are propagated by routing protocol supporting name prefix exchange. Communication in CCN is driven by the receiving end, normally the content consumer. An **Interest** packet, which carries a name that identifies the desired content, is initiated by the content consumer to get a piece of content. A CCN router first would look up its CS to find out if it has a cached copy of the desired content. If so, it will respond the Interest with the cached copy. Otherwise, it will record the interface from which the Interest comes in with PIT, and forwards the Interest packet by looking up the name in FIB. The router stores the Interest in its PIT along with the interface from which the Interest has been received, until the expected data is received. When more than one Interests for the same data arrive, only the first Interest received is forwarded upstream towards the data source, and the receiving interfaces of other Interests are recoded in the existing PIT entry. When the Interest packet eventually reaches a node with the desired data, a **Data** packet is sent back by tracing back the footprints in PIT left by the Interest packet. The Data packet carries the name, the requested data and a signature signed by the original data producer.

CCN routers along the forwarding path will forward the data packet to the interfaces recorded in the matching PIT entry. The PIT entry is then removed from the router's PIT and the data packet is cached in the CS. CS is basically an area of buffer memory in the router and is subjected to a cache replacement policy such as LRU, LFU etc. The cached data packet is identified only by its name and independent of its former requester and its producer and can be used to satisfy potential future matched requests.

3. HTTP-CCN GATEWAY

3.1 Overview

In this section, we start with an overview of the HTTP-CCN gateway design. As an adaptor of CCN and HTTP, the gateway must be both compatible with CCN protocol and HTTP protocol. The HTTP-CCN gateway consists of two parts, namely Ingress gateway and Egress gateway (IG and EG for short), depending on which side it communicates with in a normal HTTP session. The Overview of CCN network with HTTP-CCN gateways on the edge to interact with traditional Internet is shown in Figure 1.

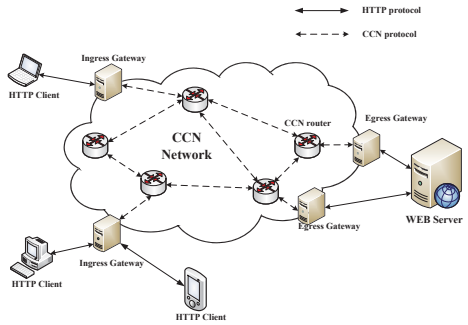


Figure 1. Overview of CCN network with Ingress and Egress Gateway.

To implemented the HTTP-CCN gateway (both IG and EG) we modified and extended an open source WAP and SMS gateway named kannel [7]. Figure 2 shows the module diagram of the HTTP-CCN gateway.

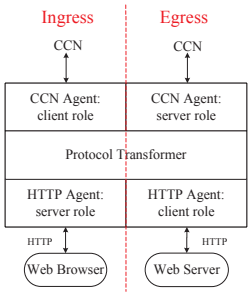


Figure 2 The module diagram of HTTP-CCN gateway

Via setting the IG as a HTTP proxy, HTTP requests from browsers will be sent to the IG. It is also possible to use a DNS hijacking [8] technology to direct requests to our gateway. After receiving a HTTP request, IG parses and then translates it to a CCN Interest. Since HTTP protocol has nine methods (e.g. HEAD, GET, POST, PUT etc.), currently the IG only copes with the three methods relevant to getting a identified resource from server, namely HEAD, GET and POST. Requests with other methods are sent directly to server through Internet. The resulting Interests will be routed by CCN router to one or multiple EG simultaneously. When an EG receives an Interest from CCN network, it will reconstruct the original HTTP request from the Interest and send it to the server that the URL in the request specified via IP network. After receiving the corresponding response from http server, the EG will send it back to the corresponding IG via CCN Data packet. It would also store it in its content store for subsequent identical requests, which make it a content producer to the CCN network. The intermediate CCN nodes could cache the Data packet according to their cache policies. After receiving the Data packet, the IG will extract the content and construct a HTTP response and send to the requester. The Figure 3 shows the process of message delivery described above.

3.2 Protocol Transformation

3.2.1 Naming convention

Commonly, a HTTP request consists of a request line and several headers. The request line is composed of a HTTP method and URL which identifies the resource requested. In CCN, the name of Interest is used by routing system to deliver the Interest to the corresponding content producer. We combine the URL with a prefix which indicates the routing information needed by CCN router to form the new name of the corresponding Interest. For example, the following is the URL of a famous searching engine in China.

http://www.baidu.com/s?wd=yi&rsv_bp=0&rsv_spt=3&inputT=1488

We prefix the URL with the string "ccnx:/default", which means that the corresponding Interest will be routed to the nearest Egress Gateway, provided that all the EGs have announced the prefix ccnx:/default/ to the routing system. The resulting Interest name is like the following.

Ccnx:/default/http/Get/www.baidu.com/s?wd=yi&rsv_bp=0&rsv_spt=3&inputT=1488

As the above name shows, the protocol component "http" and HTTP method component "Get" also appear in the Interest name. We could also prefix the URL with a string indicating specifically an EG as the following example shows.

Ccnx:/Egress-1/http/Get/www.baidu.com/s?wd=yi&rsv_bp=0&rsv_spt=3&inputT=1488

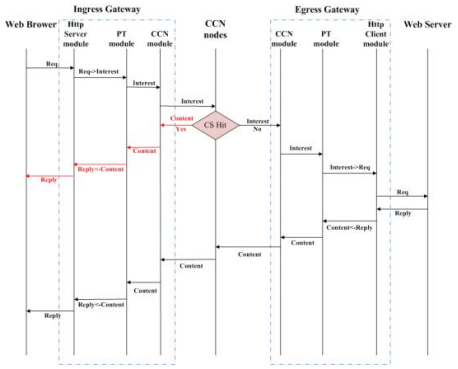


Figure 3 Message transmission among entities

3.2.2 Transmission of HTTP headers in Interest

Since a HTTP request needs to be reconstructed by the EG, all the headers should also be sent to the EG. The problem here is a typical issue for CCN. The data transmission of CCN is actually a pull mode. When it comes to a typical push application, the pull mode of CCN would act inefficiently. Typical solutions of this problem include the following two:

1) Since the name is length-unlimited, we can append the extra information into the name as the way used in [3], in which the name of requester is appended in the Interest name.

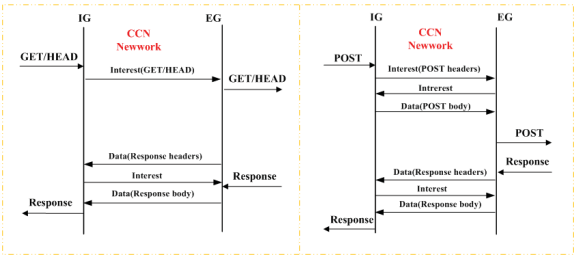
2) Send an Interest with some session information to elicit a resulting Interest from the EG. Then the extra information will be sent in the responding Data packet.

Obviously, the later solution would result in an extra delay of a period of RTT, which would degrade the user experience of performance significantly. The former solution would make the Interest name unexpected long. On the other hand, adding the headers into name may disable the cacheability of the requested resource, since the name is the identity of a cached piece of content in CCN. For example, two requests with the same URL could not share the same HTTP response due to different kind of user agent (e.g. IE, Mozilla). Based on the analysis above, we solve this problem by expending the Interest of CCN with a new attribute named metadata. The headers of a HTTP request are encapsulated in this attribute and are extracted to reconstruct the request by EG.

3.2.3 Deal with different HTTP methods

HTTP protocol has defined many methods for different purpose. The most commonly used two are GET and POST, which are also required to be implemented in a basic HTTP server. Therefore, our gateway needs at least support for these two methods.

The GET method is used to retrieve resource from a server. In contrast, the POST method is to submit data to a server and then get a response generated according to its submission. Semantically, GET is similar to CCN interest and POST is partly similar to CCN data as a poster is also a data producer. On the other hand, a client initiating a HTTP request with POST method is more than a content produce, actually it is also a data consumer. A POST may result in the creation of a new resource or the updates of existing resource or both, and the result finally forms a response the poster consumes. To keep this semanteme, we transform requests with the GET or POST method into CCN representation differently. Figure 4 shows the differences in handling a GET method and a POST method.



(a) GET/HEAD (b) POST

Figure 4 different treatments for different methods

Assume both the request and response can be fitted into a single CCN packet (we will discuss the CCN packet size in the following subsection), a GET request is first transformed into an interest packet in IG and recovered in EG, then it is sent to the web server. After the response arrives, it is transformed into a Data packet and travels back to IG, where the Data packet is decoded into a response and sent to the original requestor. Unlike GET, POST needs one more round trip to complete. Firstly, request line and headers of POST are transformed into an interest and sent to CCN network by IG. When EG receiving the first interest from IG, it sends an Interest to retrieve the body part of the POST from IG. IG would

then response with the body part and waits for incoming Data packet containing the finally response from web server. Once EG gets the body part, it sends the reconstructed POST request to the original server.

3.2.4 Packet size of CCN Content

Running on TCP connection, a HTTP response can be as large as the order of magnitude of megabits (e.g. a VOD chunk). Although CCN router (CCNx) runs on TCP or UDP currently, which make it possible to transmit unrestricted large content, large Data packet would be inefficient due to extremely large propagation delay incurred by the multi-hop relay-based transmission paradigm of CCN. On the other hand, every chunk of CCN to be sent is signed with a signature and processed by some security algorithm. In each router along the forwarding path, the signature may need be extracted and verified. If the packet size is too small, too much overhead would be introduced. Here we choose 4096 bytes as the maximum CCN chunk size. A HTTP response received by EG would be split into chunks and sent back to the IG, which sends a series of Interest to get the whole HTTP response. Each chuck has a unique sequence and is sent in a pipeline way as described in [1], where Interest is send before the corresponding Data is generated and the Data is sent back immediately after being generated.

As the HTTP response is received chunk by chunk progressively, the IG will first get the headers of response and then the body. After obtaining the headers, it starts to send the response to the requester immediately, instead of after having received the whole response body. This is especially important to the on-demand video transmission, where most response is extremely larger than normal response and the requester commonly support HTTP progressive download, which make it possible to playback the media before the whole file have been received.

3.3 Adapt HTTP Caching into CCN

Caching is considered as one of the most important features of CCN. We also think that it may be the main incentive for ISP to deploy CCN. The IG and EG are designed to leverage this feature to support the HTTP application running over CCN.

Here we list the CCN features relevant to caching. In CCN, a requester can indicate whether this Interest can be satisfied by a still-fresh copy cached in intermediate node with the attribute *AnswerOriginKind* [9]. A stale copy can also be thought eligible if the requester set the stale bit of *AnswerOriginKind*. CCN provides an age-based way to estimate the freshness of a cached copy. The content producer could assign the fresh time by setting the attribute *FreshnessSeconds* in the Data packet. Every piece of content is associated with a timestamp denoting the time of its generation.

On the other hand, the HTTP specification for regulating caching behavior is pretty complicated. From a perspective of HTTP protocol, the whole CCN network can be viewed a large distributed caching entity. So the ideal design is to follow completely the HTTP caching specification. Constrained by the caching functionality CCN provide, a subset of caching capacity of HTTP is implemented by HTTP-CCN gateway in this paper. Actually, we found that providing content oriented functionality in network layer as CCN also raises a question about whether this functionality could fit for upper-level application needs. Actually, we found that it is difficult to completely map the HTTP caching requirements into CCN caching functionality naturally and effectively. In HTTP, server can indicate the expiration times or the fresh period of time of a response which is much similar with the age-based way in CCN. Besides expiration time, HTTP also provides a "validation" mechanism to reduce network bandwidth requirements, in which a stale entry in a cache can be used as a response after check its usability with the origin server. This mechanism is not included in the CCN design currently. Actually in contrast to the object-level feature of HTTP, the intrinsic chunk-level feature of CCN make this kind of optimization mechanism seems less attractive. So we just ignore the headers relevant to this mechanism and simply send the request to the original server.

A HTTP request could specify the maximum age it is willing to accept of an unvalidated response, while specifying a value of zero forces the cache(s) to revalidate all responses. A client could also specify the minimum time remaining before a response expires. A client could also specify that it will accept stale responses, up to some maximum amount of staleness. We can see that for requester HTTP protocol is more flexible to express its directive to cache-enabled entities than CCN does.

From the description above, we can derive the design as follows. Only part of HTTP directives from client is mapped into CCN functionality. That is when a client indicates no cached copy would be responded or it could accept a stale content. This could be implemented by setting the attribute *AnswerOriginKind*. From the perspective of HTTP server, only the part of fresh time is mapped into CCN. We calculate lifetime of a response and assign it to content chunks derived from it. Lifetime of a response is determined by examining its headers and status code, as well as some headers of its corresponding request. We referred to the implementation of Squid [10] and implemented a simplified subset of HTTP caching regulation to determine both whether a request can be satisfied by a cached response and how long a response would be still valid for responding subsequent request. The criteria of are listed below:

1) The maximum caching time for CCN content is 4296 seconds, according to its implementation.

2) If value of "Cache-Control" field of the request is "no-cache", the attribute *AnswerOriginKind* of corresponding Interest will be set to ignore any cached copy and the request would be force to be sent to the EG.

3) Only responses with special status code (e.g. *NON_AUTHORITATIVE_INFORMATION*(203) etc.). are considered as cacheable. Otherwise, lifetime of others is set to zero.

4) Response with special "Content-Type" is not cached. For example, response with "multipart/x-mixed-replace" is viewed as uncacheable.

5) Lifetime is calculated by headers “Cache-Control”, “Expires” and “Pragma” of a response. These directives in response headers are treated sequentially according to their priorities. More detailed information can be found in Table 1.

Table 1. Calculation rule of content lifetime

□□□□□□□	□□□□□□□□	□□□□□□□□□□ □□□
0	Pragma:no-cache	Lifetime=0
1	Cache-Control:s-maxage	Lifetime=s-maxage
2	Cache-Control:max-age	Lifetime=max-age
3	Cache-Control:no-store	Lifetime=0
4	Cache-Control:no-cache	Lifetime=0
5	Cache-Control:private	Lifetime=0
6	Cache-Control:public	Lifetime=default
7	Expires	Lifetiem=Expiration_Time – Current_Time

4. DISCUSSION ABOUT TRANSPARENCY CACHING

In a recent report, cisco forecasts that Internet traffic will grow 5 times between 2009 and 2013, and video will constitute 90% of overall traffic [11]. More and more providers of video on demand (e.g. youtube.com, yuku.com) use HTTP to delivery their content due to its extensive availability, enormous infrastructure already existing in Internet (e.g. Web cache) and its ability to penetrate corporate firewalls. On the other hand, one of the trends that have emerged in the streaming media industry has been a steady shift away from classic streaming protocols (RTSP, MMS, RTMP, etc.) back to plain HTTP download. In 2008 Microsoft used a kind of HTTP chunking named Smooth Streaming [12] to stream the Beijing Olympics for NBC.com.

As a result of persisting growing of video traffic, transparent caching in carrier network is becoming more and more attractive. Transparent caching tries to intelligently and dynamically identify content and adapt to shifting content access patterns, which can enlarge the cacheable volume and raise the caching efficiency significantly. On the other hand, the transparent caching is much attractive to service providers. Although the video traffic grows rapidly, the revenue of service providers does not increase since their customers pay a fixed amount per month. In contrary, they have to invest in their networks to scale to support this traffic. with no end of the grow of content distribution in sight, network operators seems willing to realizing and deploying transparent caching inside their networks to address a broader range of Internet content.

In-network caching is designed as one of CCN foundation. We think that CCN can be used to provide an ideal foundation of transparent caching for carrier network. With the great incentive of transparent caching, it can be a promising way for CCN to roll out. We argue that the HTTP-CCN gateway described in this paper can be viewed as the first step to this direction.

Here we discuss how to enable the HTTP-CCN gateway to caching HTTP video traffic transparently and also present some obstacles we cannot overcome yet. To achieve transparent caching of VOD traffic in CCN, we reexamine our name-based transformation scheme where a HTTP request is transformed into CCN Interest identified by URL-based name. After analyzing some cases from some famous VOD websites, we found that:

1) Application of HTML rewriting-based CDN service makes it inefficient to cache transparently in CCN.

We analyze HTTP-based VOD delivery of the most famous VOD website in China, namely *youku.com*. We find that a video file is split into several large chunks. Each Chunk is of the order of magnitude of Megabits and has a URL like the following.

http://118.228.18.32/youku/6975350854E3482E8FCDBE6E0E/03000201004F759257B7F300946C19B8D94FF0-563F-4D8D-A687-35BEBCA497B4.flv

The first component is the IP of a specific CDN host which hold one copy of the requested content. Here dynamic HTML rewriting is used to direct HTTP requests to different CDN hosts. It makes several same copies of a video chunk have different URLs. Under our design of transforming HTTP request to Interest (as described in Section 3.2), the requests for them cannot share the responses. What is worse is that the third components of the above URL is a random string which make no request for the same content can share response in caches. We argue that it is because the content provider has significant privacy concerns and don’t want its content to be cached and reused. Actually, the video chunk is associated with neither age nor expiration information, which means the provider doesn’t want this content to be cached. We also find that the last component of the above URL, a long random-like string, is unchanged between different requests. This component seems like a digest of the video chunk. We use it as an identifier of the video chunk in implementation of HTTP-CCN gateway. But we also know that it is not one hundred-percent right since it can only be confirmed by the content provider.

2) Customized content make transparent caching difficult

Another issue is that content provider usually provides customized content to a specific client. The parameters indicating the client features can be encoded in the URL of a HTTP request or appears in the cookie header. For example, we get the following URL for *youtobe.com*.

http://tc.v3.cache7.c.youtube.com/videoplayback?algorithm=throttle-factor&sparams=algorithm%2Cburst%2Ccp%2Cfactor%2Cid%2Cip%2Cipbits%2Citag%2Csource%2Cexpire&key=yt1&sver=3&expire=1332882801&signature=9453CF40D01B5DB92E27CC810DF7846F27498572.1B350162E6B7853438641E68D2BF807FFA0B51F7&source=youtube&id=5ec151f036d24501&cp=U0hSR1hTUF9FS0NOMl9QTVRJOkJZTFM2UktNdjM3&itag=34&ipbits=48&burst=40&ip=2402%3Af000%3A1%3A%3A&factor=1.25&fexp=913700%2C905024&ptchn=RussiaToday&ptk=russiatoday&playretry=1&cm2=1

Unlike the previous example, the first component indicating the host name is unchanged. In the second component are some parameters from requesting client, which is separated by character '&'. We found that the order of these parameters is changing between different requests from the same client, but the set of these parameters and their values are the same. So we sort these parameters to form a unique name for this video chunk. Using this name in the CCN, we can make the video chunk cacheable. On the other hand, we also found that for different clients the parameters cannot be confirmed to be the same. We cannot fully understand the meanings of these parameters without help from content providers, letting alone the cookies in the request. Actually, the content provider has tagged this video chunk as "private" (in the header of *cache-control*), which means that this content can be only used by a private cache of the requesting client. We can see that without cooperation with content provider, it is difficult to cache customized content transparently.

Although transparent caching is a promising technology that simultaneously benefits a content provider, a network operator, and most importantly content consumers. From the analysis above, we can see that caching in this manner within CCN is a very difficult challenge to solve technically under the current content distribution environment. In CCN, a piece of content is identified only by its name and several pieces of content with same name would be regarded as exchangeable to some extent. In practice, naming content and determining whether two piece of content is the same functionally can be very complex as HTTP examples above shown.

On the other hand, as previously described, the service providers are disadvantaged within the economic environment of content distribution. Many service providers start to try to form federated content delivery networks to have an unprecedented opportunity to compete directly with market leaders like Akamai and Limelight. A main block of federated content delivery networks is to form a common data model for all the service providers. With this trend, we expect that achieving transparent caching with CCN is still very promising in the future.

5. CONCLUSION

In this paper, we strive to run HTTP protocol on CCN and combine the two by stitching them semantically on their content-oriented features, such as content caching. We design and implement a HTTP-CCN gateway to transform HTTP request and HTTP response into CCN Interest and Data respectively. We illustrate how to semantically map HTTP caching to CCN caching, which is one of the most attractive properties of CCN. We find that constrained by the caching functionality CCN provided, it is difficult to completely map the HTTP caching requirements into CCN caching functionality naturally and effectively, which also raises the question about whether or how the content-oriented functionality provided in network layer could meet the needs of upper-level application. That will be part of our future work.

In this paper, we also try to enable the HTTP-CCN gateway to cache HTTP video traffic transparently within the CCN network behind. We found that: 1) application of HTML rewriting-based CDN service makes it inefficient to cache transparently with CCN, 2) customized content make transparent caching difficult. So it is nontrivial to achieve complete transparency of caching with CCN given no cooperation with CDNs and content providers. Although the difficulties founded in transparent caching with CCN, given the great incentives of transparent caching for service providers and the trend to federated content delivery networks formed by service providers, we expect that achieving transparent caching with CCN is still very promising in the future and it is also a promising way for CCN to roll out. As a future work, we will continue exploring the solution space of transparent caching with CCN and working towards a mature and attractive product for service provider in cooperation with vendors like Huawei.

6. ACKNOWLEDGMENTS

Our thanks to the valuable comments and discussions from Doctor Van Jacoboson and Professor Lixia Zhang while presenting at AsiaFI NDN Hands-on Workshop. This work is supported by National Science Foundation of China under Grant 61073172, Program for New Century Excellent Talents in University, National Basic Research Program ("973" Program) of China under Grant 2009CB320501, and Huawei University Research Grant YBWL2009119.

7. REFERENCES

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. Plass, N. Briggs, and R. Braynard. Networking Named Content. In CoNext, 2009.

[2] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A Data-Oriented (and Beyond) Network Architecture. In SIGCOMM, 2007

[3] V. Jacobson, D. K. Smetters, N. Briggers, M. Plass, P. Stewart, J. D. Thornton and R. Braynard, "VoCCN: Voice Over Content-Centric Networks", ACM ReArch'09, December 2009.

[4] Zhenkai Zhu, Sen Wang, Xu Yang, Van Jacobson and Lixia Zhang, "ACT: Audio Conference Tool Over Named Data Networking", ICN'11, August 19, Toronto.

[5] Lucian Popa, Ali Ghodsi, Ion Stoica, "HTTP as the Narrow Waist of the Future Interest", Hotnets'10, October 20-21.

[6] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian. Internet Inter-domain Traffic. In ACM SIGCOMM, 2010.

[7] http://www.kannel.org/

[8] http://en.wikipedia.org/wiki/DNS_hijacking.

[9] http://www.ccnx.org

[10] http://www.squid-cache.org/

[11] Cisco Visual Networking Index: Forecast and Methodology, 2009-2014, 2010.

[12] A. Zambelli, "IIS smooth streaming technical overview", Microsoft Corporation, 2009.