CONIA: CONTENT (PROVIDER)-ORIENTED, NAMESPACE-INDEPENDENT ARCHITECTURE FOR MULTIMEDIA INFORMATION DELIVERY

Eman Ramadan, Arvind Narayanan, Zhi-Li Zhang

Department of Computer Science and Engineering,
University of Minnesota, Minneapolis, USA
{eman, arvind, zhzhang}@cs.umn.edu

ABSTRACT

We propose and present *CONIA*, a novel *content (provider)-oriented*, *namespace-independent* architecture for multimedia information delivery. *CONIA* is designed specifically to account for the diversity and complexity of multimedia content, and to recognize the prominent roles of content providers (CPs) in the network economics of content delivery. In this paper, we provide an overview of the content delivery architecture of *CONIA* and outline the basic functions of its key components. Using several use cases, we illustrate the flexibility of *CONIA* in allowing for CPs to employ various control policies to dynamically handle user demands and meet users' *quality-of-experience* expectations.

1. INTRODUCTION

With increasing popularity of multimedia content (especially video), recent years have seen the emergence and rapid expansion of large-scale content delivery systems such as YouTube, Netflix, and Hulu. It is reported [1] that Netflix alone consumed nearly a third of the peak downstream traffic in North America in 2014. Due to the enormous burden placed on the network substrate, online multimedia (in particular, video) streaming services have become a major driver that shapes the evolution of the Internet, with ever expanding roles of content distribution networks (CDNs) – whether operated by content providers themselves (e.g., Google/YouTube), ISPs (so-called Telco CDNs) or third-party CDN providers (e.g., Akamai and Limelight) – and a "flatter" Internet interconnection structure.

The complexity involved in building and operating a large-scale content distribution system in today's Internet (see, e.g., [2, 3, 4]) to handle user demands and meet the user desired *quality-of-experience* (QoE) also highlights some of the key limitations of today's Internet architecture. To partly address these limitations, alternative architectures for a *content-centric* or *information-centric* networks (ICNs) have been proposed. Arguably the best-known example is the NDN (*Named Data Network*) architecture [5, 6] which adopts a hierarchical namespace for direct "in-network" content query and delivery. There are a number of other architectural designs such as DONA [7] which adopts a flat namespace; see [8] for a survey of representative information-centric network (ICN) architectures. There are two *basic tenets* underlying all ICN designs: (i) *content is the first-class object that can be directly named and routed*; and (ii) *content storage should be part of the network substrate* (either in part or all of network switching or routing elements). Various existing ICN proposals differ in how namespaces are designed (e.g., flat vs. hierarchical) and how content is queried and delivered.

Deviating from existing ICN proposals, in this paper, we propose and advocate a *content (provider)-oriented*, *namespace-independent* network architecture (*CONIA* in short) which is designed in particular to account for the complexity involved in multimedia content delivery. While upholding the two basic tenets in ICN designs stated above, we also put forth the following maxim: *to enable a scalable, robust, economically viable, and evolvable ICN architecture, one must not dictate how the namespace for content is designed.* Our design is motivated by several key considerations: 1) We recognize that there is a vast array of *diverse* content types; no single namespace (whether flat or hierarchical) or schema can fit it all. 2) Unlike a physical host (or rather a network interface), a piece of content – in particular, *multimedia* content – is in general a complex, composite logical object with many constituent objects (composite or otherwise), some of which may be dynamically composed on demand. For example, a movie that a user is interested in is comprised of both video and audio components which are often segmented in various segments encoded in different bitrates; it may also include closed-captions in a language that the user selects. 3) We also recognize the prominent roles of *content providers* (CPs in short) in provisioning and distributing content in any ICN, these include their needs for managing digital rights, controlling user accesses, and handling other content delivery issues. 4) Closely related to 3), ICNs must take into account the *network economics* of content delivery so as to be economically viable. The confluence of these and other considerations leads us to argue that it is imperative to afford CPs the flexibility in determining their

978-1-4799-7082-7/15/$31.00 ©2015 IEEE

own naming schemas for diverse types of content and in employing appropriate content management policies and control logic (e.g., caching & load balancing policies) to dynamically handle user demands so as to best meet user QoE expectations.

We show that the *namespace independence* maxim we promulgate need *not* be in conflict with the two basic tenets of ICNs by outlining the basic design of *CONIA*. In our design, we explicitly separate two major functionalities (or *dimensions*) of any ICN: *content discovery* vs. *content delivery* (see Section 2 for more discussion). Due to space limitation, this short paper focuses primarily on the *content delivery dimension* which is concerned with delivery of a specific piece of content upon a user request. In *CONIA*, a CP is afforded with the autonomy of specifying its own content name schemas as well as the capabilities of dynamically installing appropriate *(content) control logic* in *content store and routing* (CSR) elements in the network substrate and supplying a *Content Map* to the content player at the client/user side for content request and delivery. In a sense, CONIA provides a *software-defined* paradigm for content distribution with *control* residing at each CP. In Section 2, we provide an overview of *CONIA*'s content delivery architecture, and in Section 3, we highlight the key functions of (*generic* and *namespace-independent*) CSR network elements and outline the basics of an *open* control framework (with *standardized* APIs) needed for CP controllers to install content control logic and Content Maps and for the communications between various entities involved. In Section 4, we illustrate the flexibility of *CONIA* via a few use cases in dynamically handling cache management and content adaptation, and coping with changes and surges in user demands. Related work is briefly discussed in Section 5 and Section 6 concludes the paper.

2. OVERVIEW OF AND CASE FOR *CONIA*

In *CONIA*, we explicitly separate the *content discovery* dimension from the *content delivery* dimension. Different sets of content namespaces or schemas are likely needed to realize the functionalities of these two dimensions. During the *content discovery* process, a user often does not know the exact content[1] (or its name) that she is interested in; instead she will issue a search query with a set of keywords and attributes. This results in a list of content returned to the user in the form of, say, `{content-provider-id, content-name, price, credentials, ...}`. From this list, she selects a content provider (CP) based on her credentials (e.g., whether a subscriber or not), price, and other considerations. To obtain the content she is interested in, the user submits a request using the content name learned from content discovery to the selected CP (or rather, a content controller of the CP, see below); thereafter the *content delivery* process commences. Due to the space limitation, the remainder of the paper will focus on the functionalities needed for realizing the (*content-provider-specific*) content delivery dimension. We remark here that the content discovery dimension may be realized using either a "centralized" search engine framework (*a la* Google), a collection of "hierarchically distributed" *publish-subscribe* systems organized in a similar fashion as today's DNS, or a purely "peer-to-peer" service using DHT (distributed hash table). A more in-depth exploration of these topics will be left to a longer version of the paper.

[1]In this paper, we use the term (a piece of) *content* to denote what a user is *actually* interested in, such as a movie or a TV show episode, while we use the term *objects* to denote units of data (of varying sizes, formats or types) which constitute the content and are to be delivered to the user, such as video or audio segments encoded in different rates or languages. An object can be composed of other (smaller) objects and each object has its own name (*object id*). Each CP has the freedom to decide how a piece of content is decomposed into various objects, and how they are encoded and named by specifying a name schema for the content.

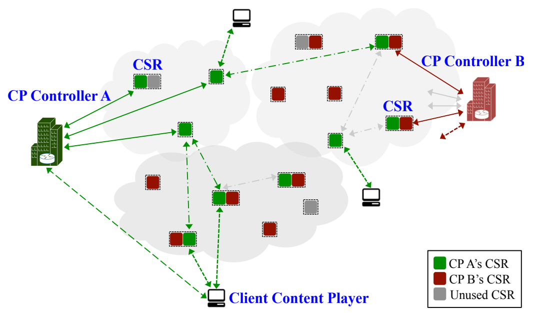

Fig. 1. A Schematic Illustration of the *CONIA* Content Delivery Architecture

Figure 1 provides a schematic illustration of the *CP-oriented content delivery* architecture in *CONIA*, which consists of three key components – *CP controllers*, *content store and routing* network elements (*CSRs* in short), and *content players* at the client side – as well as an open *control* framework with *standardized* APIs for managing the interactions among the components. One premise of our design is that as part of the (global) network substrate, CSRs are *generic* (i.e., independent of CPs) and *shared* content delivery resources that may be owned by ISPs and enterprise networks, third-party entities (e.g., CSR providers similar to today's CDN providers, communities or users), or CPs themselves, and can be procured by any CP (either *a priori* or on-demand) for its content delivery.

Each CP is responsible to define and specify its own namespace and content-specific name schemas, and deploy one or multiple controllers for managing the content delivery by installing appropriate *control logic* in a selected set of CSRs that are used for the delivery of the constituent objects for each piece of content that users are interested in. Upon receiving a *content request* (or an *interest* message) from a user, a CP controller responds by supplying a *Content Map* (similar to *Media Presentation Description* (MPD) defined by MPEG-DASH [9]) to the content player at the client side, which is used to direct *object requests* to appropriate CSR resources (say, those closer to the user), parse and render the objects received. Each CSR maintains a table of content control logic installed by each CP, which is expressed in a *declarative* language (see, e.g., [10]), looks up and applies the corresponding control logic for each object request it receives. CP controllers, CSRs, and client content players communicate and interact with each other using a common *open* control framework with *standardized* APIs. In Section 3, we outline some of the basic functions of these key architectural components.

In summary, *CONIA* is designed specifically to account for the diversity and complexity of multimedia content, and to recognize the prominent roles of CPs in the *network economics* of content delivery. It affords each CP the autonomy and flexibility in specifying its own content schemas and installing control logic in CSRs to effectively exert appropriate cache management, load balancing, and other control strategies so as to dynamically adapt to user demands and optimize how content is delivered to meet users' QoE expectations (see Section 4). By separating the content discovery dimension from the content delivery dimension, *CONIA* allows for different content discovery platforms and providers to co-exist and compete with each other, while at the same time also creates a marketplace for multiple CPs to compete for content delivery. It also enables new business models between CPs and network substrate providers, and offer incentives for ISPs and other entities to deploy and operate CSRs to meet growing user demands for content. However, addressing the network economics issues is beyond the scope of this paper.

3. DESIGN OF KEY *CONIA* CONTENT DELIVERY COMPONENTS

In this section, we outline the design of key *CONIA* content delivery components and the open control framework, focusing in particular on the basic functions of the generic *content store and routing* (CSR) network elements.

3.1. CP Content Delivery Controller

In *CONIA*, a CP is in charge of managing the content delivery process via its content delivery controllers (in short, CP controllers). A CP controller defines namespace for objects, assigns name for each individual object used in content delivery process, specifies packet header format, determines how header fields are matched (i.e., prefix, exact or ternary matching), and decides the control logic used to handle and forward requests and data.

A CP controller maintains a global *view* of the network, controls and manages CSRs, and makes decisions driving caching and forwarding behavior. The global view of the network consists of: where each object is stored (i.e., which CSRs), client request patterns and topological view of the network (i.e., locations of CSRs and how they are connected to each other and to clients). A CP controller also keeps information about CSRs and their available resources such as: storage capacity, bandwidth, location, etc.

Based on the global view, a CP controller *proactively* caches objects at CSRs and directs client request to the *closest* copy of the requested object. For each object, *CONIA* introduces the concept of a *Content Delivery Swarm (CDS)*. CDS is an *object-specific view* of the overall topology revealing all CSRs that cache this object and understand its namespace.

A CP controller uses the CDS of an object to dynamically generate a custom *Content Map* (CM) file for each client requesting this object. A CM file of an object has two components: 1) a namespace which describes the *segment-level composition* of the object, 2) a mapping between the segments and CSRs caching them.

A CM file generation is based on several parameters like: client's location, bandwidth and location of CSRs caching the segments of the requested object, etc. Dynamic generation of CM files leads to different object-segment-client-CSR mappings which allows dynamic adaptation to various situations. For example, in case a CSR fails serving a client, the client may report the failure to a CP controller and in response the CP controller can generate a new CM file. The client can now resume downloading the object from the newly specified CSRs.

A CP controller analyzes statistics reported by different components of the system to make key strategic decisions like: *"what to cache"*, *"where to cache"*, and *"how to map clients to CSRs"*. A CP controller forces these decisions by configuring/modifying the caching and forwarding behavior of CSRs. This allows CSRs to dynamically adapt to the changing network conditions and client request patterns. Moreover, this helps achieve CP specific objectives such as: minimizing latency for clients, maintaining load of each CSR/link to be below a certain threshold, etc.

A CP controller uses two basic control framework APIs to interact with CSRs and Clients: 1) pushes objects and control logic to CSRs, and 2) sends Content Map files to clients.

3.2. Content Store and Routing (CSR) Elements

As part of the network substrate, CSRs are *generic* and *shared* content delivery resources used by CPs for their content delivery. Each CSR maintains a *Content Control Logic Table (CCLT)* installed by each CP, in which the control logic is expressed in a *declarative* language [10], looks up and applies the corresponding control logic for each object request it receives. Next, we describe one possible way to specify the control logic to illustrate what is possible.

An entry in a CCLT has three fields: i) object ID which identifies a certain object , ii) a set of declarative rules to be applied to this object, and iii) statistics which keep track of processed requests and data messages related to this object.

Each declarative rule consists of a *predicate* and a set of *corresponding actions*. A predicate is a set of condi-

Table 1. CCLT Example

obj_x :
$['NOT\_CACHED'] \Rightarrow forward(select([A1, A2], 'LOAD'))$
$['CACHED'] \Rightarrow \begin{cases} select\_rate('BW') \wedge r \in c : reply \\ select\_rate('BW') \wedge r \notin c : trcode(r), reply \end{cases}$
# BW: bandwidth, r: selected rate, c: cached, trcode: transcode

obj_y:
$[hit\_ctr > thresh1 \wedge 'NOT\_CACHED'] \Rightarrow cache$

all_objects:
$[CSR\_load > thresh2] \Rightarrow push([B1, B2, B3], popular\_objects)$

default:
$[] \Rightarrow forward\_to\_controller(message)$

tions which specify a context. There are two types of **contexts**: *content-related* and *system-related*. Content-related context describes the state of an object such as: 1) an object is cached locally or not, 2) a data object is forwarded to a client through the current CSR, and 3) an object becomes popular which means within a specified time interval it is requested more than N times, where N is a threshold specified by CPs. System-related context describes the state of CSRs and network conditions such as: 1) load on CSR, 2) link load between CSRs, and 3) available bandwidth between CSRs and clients. **Actions** include: 1) forward request to other CSRs or controller, 2) drop request, 3) push objects to other CSRs, and 4) forward data to clients. The execution of some actions may involve some local decisions to be made by CSRs according to system-related context determined by measurements and statistics received from other CSRs.

For example, the rule which allows a CSR to forward an object request not cached to other CSRs is the following:
$['NOT\_CACHED'] \Rightarrow forward(select([A1, A2], 'LOAD'))$
In this example, the content-related context is: object is not cached, the action is: forward the request to CSR $A1$ or $A2$, the logic in the forwarding action is: select one of the CSRs based on their load, where load is a system-related context.

Table 1 shows more examples of possible rules. 1) Before sending a video object to a client, a CSR may actively adapt it to a certain bitrate calculated according to the available bandwidth. If the calculated bitrate is not in the cache, it will transcode it. 2) If a CSR is overloaded, it can push some popular objects to other CSRs (an example of a rule defined for multiple objects). 3) All objects which do not otherwise match any rule will match a default rule (see the last rule).

In summary, for each request or data message received by a CSR, it does the following: 1) identifies the object ID, 2) determines the context, 3) finds the control logic (rule) defined for this object and context, 4) executes the actions of the matched rule, and 5) updates the statistics of the matched rule. A CSR uses four basic control framework APIs to interact with other CSRs, CP controllers, and clients: 1) sends health information to other CSRs; 2) pushes objects to other CSRs to cache them, 3) reports statistics to a CP controller, and 4) sends data to clients.

3.3. Client Content Player

A client content player is a piece of (media-specific) software similar to a web browser, music or video player (e.g., the Real player or Netflix app). It renders and displays content it receives. Given a piece of content that a user is interested in (learned via the content discovery process), the client content player sends a content request (i.e., an *interest* message) to a CP controller and receives a *Content Map* (CM) in return. The CM not only provides the name schema to help the client content player parse and render various constituent objects of the content it receives (similar to what an HTML file does to a web browser or what an MPD manifest file does to a Netflix video player), but also tells the client content player where (e.g., a sets of close-by CSRs) to fetch by sending individual *object requests* to these CSRs.

The client content player also collects and reports relevant statistics to the CP controller (e.g., performance it experiences in fetching various objects), and the CP controller may dynamically update or issue a new CM to the content player, e.g., with a new set of CSRs to bypass overloaded or failed CSRs. This dynamic adaptation allows the CP to adapt to changing network and system conditions, thereby enhancing the QoE perceived by clients. According to the control logic embedded in the CM, the content player can choose one of the suggested CSRs with the best performance, or fetch the same object from multiple CSRs concurrently.

The client content player uses three basic control framework APIs to interact with CPs and CSRs: 1) sends an interest message with a specified content name to a CP controller; 2) reports statistics to a CP controller, and 3) sends an object request with a given object id to fetch the object from a CSR.

4. USE CASES

In this section, we provide a few use cases to illustrate how *CONIA* allows a CP to employ various control policies to dynamically adapt to user demands and optimize its content delivery to meet user QoE expectations. As discussed before, *CONIA* enables a CP to provision a set of CSRs to form a Content Delivery Swarm (CDS), see Figure 2, for delivery of a piece of content, say, a video v_1, to a group of users/clients. For simplicity, suppose v_1 consists of several segments $s_1, s_2, ...,$ and v_1's CDS initially consists of three level-1 CSRs B1, B2, B3, which are connected to two level-2 CSRs A1, A2. Clients $(c_1, c_2, ..., c_n)$ are interested in viewing v_1 initially. Note that CSRs, A3, B4, etc. in Figure 2 are *not* part of the initial CDS. We shall introduce them later in this section. We will refer to step numbers annotated in Figure 2 to help understand the use cases discussed below.
A Simple Use-Case: c_1 shows interest in viewing a video object named v_1 (step 1.1). A CP controller receives this interest message. Since CP has the overall global view (or CDS) for v_1, it responds with a dynamically generated Content Map

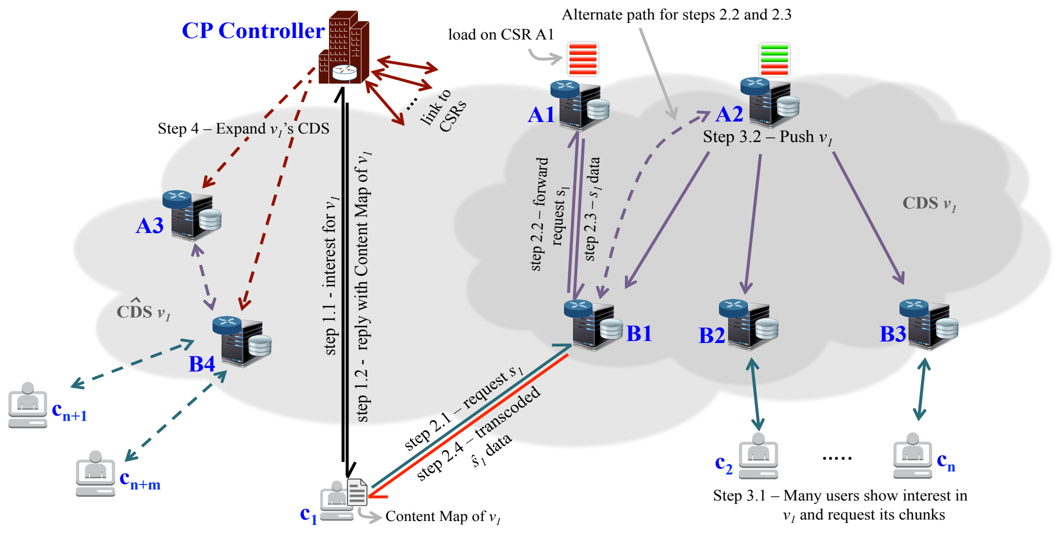
CP Controller
load on CSR A1
Alternate path for steps 2.2 and 2.3
link to CSRs
Step 4 – Expand v_1's CDS
A1
A2
Step 3.2 – Push v_1
A3
CDS v_1
$\hat{\text{CDS}}$ v_1
B4
B1
B2
B3
step 1.1 - interest for v_1
step 1.2 - reply with Content Map of v_1
step 2.2 – forward request s_1
step 2.3 – s_1 data
step 2.1 – request s_1
step 2.4 – transcoded $\hat{s}_1$ data
c_{n+1}
c_{n+m}
c_1
Content Map of v_1
c_2
c_n
Step 3.1 – Many users show interest in v_1 and request its chunks

Fig. 2. A content distribution swarm (CDS) for video "v_1"

(CM) file to c_1 (step 1.2). This file contains the *structure* and *composition* of v_1, segment-level *location* information, and other meta-data defined by the CP controller. c_1's content player, a software provided by the CP, can interpret the Content Map file. Using this file, c_1 sends request for s_1 to CSR B1 (step 2.1). On receiving the request, B1 performs a lookup operation in its Content Control Logic Table (CCLT) for s_1. Recall, this look up operation is part of the control logic, which is part of every CSR. B1 understands that s_1 is not cached and should forward the request to a level-2 CSR (A1 or A2). In other words, for s_1 and content-related context as "NOT_CACHED", CCTL specifies an action to forward the request to CSR A1 or CSR A2 (similar to rules shown in Table 1 for *obj_x*). Now, B1 needs to choose one of the two CSRs. Considering a simple approach for this use case, B1 *randomly* selects a CSR, say A1, forwards the request to it (step 2.2), and in response gets the required data (step 2.3).

Load-aware Forwarding: In the previous approach, B1 did not make a conscious effort in deciding whether to forward c_1's request to CSR A1 or CSR A2. A random approach may not always be desirable. For example, if A1 is overloaded or the link conditions between B1 and A1 deteriorate, B1 could have forwarded it to A2, which at that point of time was the desirable choice. Hence, we use our notion of *system-related context*, where using the control framework API, different components communicate with each other to help individual components (like CSR) make better decisions. We do not explicitly show this communication in Figure 2, however, we expect all CSRs (in this case A1 and A2) to periodically broadcast their health information. This information includes, but not limited to, current load on CSR, and free cache space. When B1 receives this information, it understands A1 is busy, and therefore selects an alternate path, i.e., selects A2 as the forwarding directive. We envision the existence of a piggybacking scheme or protocol used to exchange such health information, thereby enabling other components to be cognizant of their decisions.

Dynamic Adaptation: Continuing the discussion, A2 receives the content request from B1 and responds back with s_1 data. Recall s_1 is part of a *video* object v_1. For this use case, we assume the return path from A2 to c_1 is the same as the request path, i.e., from c_1 to A2. Prior to receiving the response message from A2, B1 realizes the network conditions between itself and c_1 have deteriorated (see the red link in Figure 2). When B1 receives the response message from A2, it attempts to *mitigate* the aforementioned issue by *actively transcoding* s_1 to a lower bitrate segment $\hat{s}_1$ before delivering it to c_1 (step 2.4). Thus, with the help of system-related context information, *CONIA* is able to make decisions and dynamically adapt to changing network conditions resulting in better content delivery.

Handling Flash Crowds: Next, we describe how our architecture envisions to handle scenarios when there is an *abrupt* increase in the number of requests made for a specific object within a short span of time. For some reason, consider v_1 becomes very popular, and clients from several locations show interest in fetching the same object, causing a "flash crowd" scenario over v_1 (step 3.1). Many such clients' requests reach A2, which has the object cached and is responsible to serve the content. A2 realizes the "hit_ctr" value i (part of the *Stats*) for s_1 (i.e., a segment of v_1) associated with the content-related context "CACHED" increases abruptly. A2 therefore decides to push segments associated with v_1 into B1, B2, and B3 (step 3.2). Note, the CP Controller had proactively informed A2 to push a *popular* object into B1, B2, and B3 as a flash crowd mitigation strategy. In other words, the CP controller had installed the content-related context directive "POPULAR" into A2, directing it to push v_1 into B1, B2, and B3 (see rule for *all_objects* in Table 1). This causes a new rule to be added in the CCLT of B1, B2, and B3. This rule

enables them to directly serve segment requests related to v_1.
Load Management: Intermediate network objects like CSRs may get overloaded by handling client requests. To further investigate, let us continue from where we left in the previous use case. B1, B2, and B3, all may get overloaded serving the flash crowd requests (not annotated in Figure 2), and may not have enough information on how to mitigate this load. In such a situation, B1, B2, and B3 notify the CP controller with a 'BUSY' message as defined by the control framework API. The controller realizes the overloaded situation, and resolves this by, first, expanding the CDS of v_1 (shown in Figure 2 as $\hat{CDS}\ v_1$), by pushing v_1 into additional CSRs (step 4), and second, reply to additional clients' $(c_{n+1}, ..., c_{n+m})$ interest with a modified Content Map directing them to the newly added CSRs for fetching content.

5. RELATED WORK

In [2, 3, 4], the authors study today's large video streaming services such as YouTube, Netflix, and Hulu, and show that these services have to resort to various complex mechanisms/tricks (e.g., HTTP redirection, DNS-based IP geomapping) to circumvent the limitations of the current host-centric Internet architecture. As two representative examples of information-centric architectures, NDN [5] and DONA [7] support content as the first-class object, routing by name and in-network caching. There are a number of other ICN proposals (see, e.g., [8]) and a flurry of recent activities on ICNs. Due to space limitation, we will not discuss them here.

CONIA differs from existing ICN designs in that it explicitly accounts for the complexity and diversity of multimedia content and recognizing the prominent roles of CPs in content delivery. By designing generic CSRs that allow CPs to install customized control logic and defining a common control framework, *CONIA* enables CPs to specify their own name schemas and directly control how content should be delivered to users. Similar to software-defined networking (SDN), *CONIA* enables each CP to employ controllers to effect "centralized" control over content delivery. But unlike SDN, *CONIA* supports installation of (higher-level) *declarative content control logic* in CSRs instead of simple forwarding rules based on Ethernet/IP/TCP headers. In a sense, CONIA provides a *content-based software-defined networking* paradigm with CSRs as the key network elements and the *control* residing at each content provider.

6. CONCLUDING REMARKS

In this paper, we have proposed *CONIA*, a *content (provider)-oriented*, *namespace-independent* network architecture for multimedia information delivery. We provided an overview of *CONIA*'s content delivery architecture and described the basic functions of the key components. Through several use cases, we illustrated how *CONIA* enables a content provider to dynamically adapt to changing network conditions, manage server loads and handle flash crowds.

We emphasize that *CONIA* is merely a *straw-man proposal* aimed to argue for the need for *namespace independence* for complex information delivery and the importance of taking the network economics of content delivery into account so as to make ICN architectures more scalable, robust, economically viable, and evolvable. Due to space limitation, in this paper we only briefly touched on or completely left out several important aspects of *CONIA* (e.g., the content discovery dimension, object request routing, CSR resource naming), while many other aspects (e.g., control framework API primitives, declarative control logic language specification) are yet to fully defined and implemented. Addressing each of these issues is a potential challenge. Issues such as security and network economics warrant new lines of research. We are currently working on the specification of CSRs with declarative control logic and API primitives for the control framework, with the goal to develop a *proof-of-concept* prototype using the Linux container framework. All in all, we strive to make *CONIA* a general content delivery framework, thereby affording greater overall flexibility to various stakeholders/entities involved in the content distribution ecosystem.

Acknowledgement. This research was supported in part by NSF grants CNS-1017092, CNS-1117536, CRI-1305237, CNS-1411636 and DTRA grants HDTRA1-09-1-0050, HDTRA1-14-1-0040 and DoD ARO MURI Award W911NF-12-1-0385.

7. REFERENCES

[1] Sandvine, "Global Internet Phenomena Report - 2H2014" .

[2] Vijay K. Adhikari, Yang Guo, Fang Hao, Matteo Varvello, Volker Hilt, Moritz Steiner, and Zhi-Li Zhang, "Unreeling netflix: Understanding and improving multi-cdn movie delivery," in *INFOCOM, 2012*.

[3] Vijay K. Adhikari, Yang Guo, Fang Hao, Volker Hilt, and Zhi-Li Zhang, "A tale of three CDNs: An active measurement study of Hulu and its CDNs," in IEEE Conference on *Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2012, pp. 7–12.

[4] Vijay K. Adhikari, Sourabh Jain, Yingying Chen, and Zhi-Li Zhang, "Vivisecting youtube: An active measurement study," in *INFOCOM, 2012*.

[5] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 1–12.

[6] "Named data networking," http://named-data.net/.

[7] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica, "A data-oriented (and beyond) network architecture," in *ACM SIGCOMM Computer Communication Review*. ACM, 2007, vol. 37, pp. 181–192.

[8] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher, and Börje Ohlman, "A survey of information-centric networking," *Comm. Magazine, IEEE*, vol. 50, no. 7, pp. 26–36, 2012.

[9] Thorsten Lohmar, Torbjorn Einarsson, Per Frojdh, Frederic Gabin, and Markus Kampmann, "Dynamic adaptive http streaming of live content," in IEEE International Symposium on a *World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE, 2011, pp. 1–8.

[10] Andreas Voellmy, Ashish Agarwal, and Paul Hudak, "Nettle: Functional reactive programming for openflow networks," Tech. Rep., DTIC Document, 2010.