DNS++: A Manifest Architecture for Enhanced Content-Based Traffic Engineering

Emrecan Demirors
Northeastern University
Boston, MA
Email: edemirors@ece.neu.edu

Cedric Westphal
Huawei Technologies & University of California, Santa Cruz
Santa Clara, CA
Email: cedric.westphal@huawei.com

***Abstract*—As network utilization keeps increasing, it is essential to have mechanisms to better schedule traffic in the network. Using content-based resource allocation allows to deliver a fine-grained scheduling and to monitor not only the instantaneous link utilization but also the backlog that is already scheduled onto a link in the future.**

We propose to use a manifest prior to a session to inform the session manager of the requested bandwidth. We propose to make that information available to the scheduler in the data center context. We have implemented our scheme in a testbed and demonstrate a significant benefit for a data center use case.

I. Introduction

Information-Centric Networks (ICN) [2] expose content information to the network layer. Content Centric Networking (CCN) [12], Named Data Networking (NDN) [1] and others [4], [5] want to organize the network architecture around the content. This would allow the network to cache content anywhere in the network, and to operate with a finer granularity in making routing decisions, selecting a distinct path for each piece of content.

Another benefit of ICN is that content objects being uniquely identified allows the network to learn meta-data about them, and to make use of this information for resource allocation [6], [17], [22]. For instance, a popular file download may be known to be an elephant flow, and therefore be allocated differently upon request for a file known to be a mouse flow. This is unlike the current Internet where these flow properties are opaque at the network layer.

However, ICNs are new architectures which aim to replace IP. NDN and CCN are supposed to be the new narrow waist of the Internet. While clean slate architectures allow to think of new features with fewer constraints, it places their eventual deployment in a far future. Already, people have expressed skepticism about ICN [10] and proposed some incremental solutions which bring most of the caching benefits [7].

Similarly, iDNS [15] attempts to leverage the benefits of ICN to the current Internet by leveraging the DNS system. iDNS resolves content names instead of URL, and returns a result which takes into account local caches and distributed copies of the content.

In a previous work [18], we proposed to extend iDNS to return not only the location of the content, but also some meta-data the network could use for better resource allocation. Namely, we suggested that a name resolution request returns a manifest that contains the potential location (or locations, if several are available) of the content as well as the size of the content. The manifest could also be expanded to include other type of information, such as QoS requirements.

This allows an IP network to realize most of the properties of an ICN network, while significantly reducing the deployment hurdles. In this paper, we describe the details of the DNS++, a manifest architecture which leverages DNS to expose content metadata at the network layer. We show how the data center can use the manifest to define a flow scheduling policy that significantly reduces the flow completion time. Upon inspecting the manifest and extracting its flow metadata, the network is able to select a path (or multiple paths) for the piece of content to traverse the network.

We have chosen the context of the data center, as it easily allows for islands of incremental deployment. We have implemented the proposed network architecture, and we have evaluated the flow scheduling policy against some common benchmarks to demonstrate a significant improvement.

Manifests are already widely used in application contexts, for instance in Dynamic Adaptive HTTP Streaming (DASH) [11]. The media is presented in a manifest denoted MPD. However, the MPD is not a network object, making its use by the underlying network complex, especially in the common case when the MPD is encrypted.

The notion of alerting the network at the start of a flow transmission is implicit in many current network architectures. For instance, one can observe the first packet in SDN/OpenFlow networks has a dual role: it is initiating the data transfer for a connection (in the data plane), but it also acts as an implied control message to set-up the path for this connection (in the control plane). From the early versions of OpenFlow, the first message could be forwarded to the controller for inspection and creating ad hoc rules for the session following this initial packet. A manifest makes this explicit, while at the same time, providing additional flow metadata to assist the controller.

The paper is organized as follows. In the next section, we discuss related work. In Section III, we present the architecture and the protocols of our method. In Section IV, we describe the implementation. In Section V, we validate our design by testing our implementation to demonstrate the benefit of

our scheduling method. Finally, Section VI offers concluding remarks.

II. Related Work

We describe here prior works which used ICN to enable better resource allocation and traffic engineering.

Chanda et al. [5] proposed to use SDN to manage content in an ICN network, adding content management functions such as caching and content-based routing to an SDN controller. This was generalized and extended to other content management functions [6] such as Traffic Engineering and management of content meta-data. Gao et al. [9] proposed a hierarchical decomposition for distributing the controllers in a software definied ICN.

Avci et al. [3] proposed some content-based traffic engineering policies that make use of machine learning to improve on the performance of the TE algorithms.

Another early work on traffic engineering for ICN, Reed [13] considers intra-domain TE and shows the benefit of splitting traffic on a per content base rather than on a per flow base. In particular, allowing splittable flows permits the use of a fully polynomial-time approximation scheme which performs significantly better than the current schemes for non-splittable flows.

Sourlas et al. [16] showed that cache-aware content-based routing brought significant benefit. The architecture, based upon PURSUIT, encompasses a traffic manager (TM) in charge of traffic engineering and routing decisions.

Xie et al. [20] proposes a joint TE and caching mechanism. It is accomplished by formulating a joint optimization problem, and using primal/dual decomposition to identify both a caching policy and a traffic engineering policy.

Feng et al. [8] argues that the integration of content replication and inter-domain traffic engineering should be jointly designed, in order to achieve significant cost reduction for the ISP and reduced link utilization.

Salsana et al. [14] used SDN concepts to support ICN and tested an implementation on the OFELIA project testbed. This is not specific to resource allocation and traffic engineering.

Better resource allocation is required, as the traffic demand keep increasing. One specific application would be video delivery, where the QoE for the users is depending on the network performance. The application of ICN to video distribution is the object of RFC7933 [19], which mentions traffic engineering as part of the necessary tools to optimize.

III. Architecture

A. Overview

In [15], it is argued that most of the benefits of ICN can be achieved in IP through modification of DNS, and by adding a new *content record (CR)* type to the records returned by a DNS server. This CR type allows the DNS server to respond to request for Named-Data Objects (NDO) or name prefixes. The CR type is a list of IP addresses associated with the NDO.

We argue that this does not go far enough, and that the DNS could return more information about the object, namely that

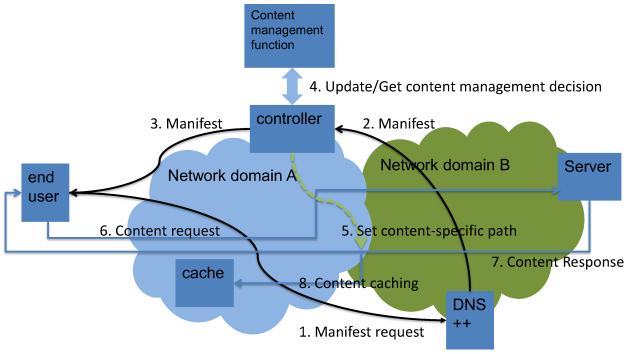

Fig. 1: Overview of the System Architecture

the DNS should return a manifest that describes the object properties. The manifest is an object (say, in xml) describing the properties of the content object that are relevant at the network layer. In particular, the manifest includes the location of the object, some security properties and some information regarding the transport of the object, including for instance its size.

This allows the network to observe the manifest as it is being requested, and to operate on this manifest. The typical envisioned usage is at the edge network: the client attaches at the edge network, and request a specific file to the DNS server, requesting a manifest. The DNS server (denoted DNS++) returns the manifest; the edge network is therefore informed of the content being requested by the client, and of the properties of the content.

Figure 1 presents the envisioned architecture, where an end-user (or client) will request the manifest from a DNS++ server, receive the manifest in return; the manifest is observed by the edge network on the way back before being returned to the client. The edge network can take some action based upon the manifest.

The steps are as follows:

1) the client issues a request for content name resolution on port 53 to its edge network;
2) the DNS++ responds with a content record that is the manifest described below; the network has a rule set up (say, filtering on port 53) to forward manifests to the controller;
3) the controller returns the manifest to the client, potentially inserting local copies;
4) the controller updates its view of the network from a content management function;
5) the controller sets up a path for the content based upon the source, destination, and properties information in the manifest;
6) the client requests the content from the server;
7) the server responds to the client; the content follows the path in the domain set by the controller;

```
<?xml version="1.0" encoding="UTF-8"?>
<MDNSR xmlns="urn:MDNSR" xmlns:xsi=<a href="http://www.w3.org/2001/XMLSchema-
instance">http://www.w3.org/2001/XMLSchema-instance</a>
Filename="www.example.com/file_rep/files/movie12.mp4"
profiles="urn:MDNSR"
type="dynamic"
publishTime="2015-03-01T16:38:00Z"
<loc_list>
        <loc id="1" serverIP = "205.14.14.102">
        <loc id="2" serverIP = "103.1.2.3">
      ... the multiple locations may be further described with other
parameters (for instance, who inserted the location in the list; or some type
of preference to select a specific location)
</loc_list>
file_size = "1.3GB"
file_type = "video:dash:2011"
...add other files parameters...
<attributes>
        ...insert manifest object
</attributes>
<services>
       ... specific services may be available in the manifest, say transcoding
for video or enhanced delivery using network assisted streaming
</services>
<signatures>
        ...insert signatures/crypto data
</signatures>
```

Fig. 2: Example of Manifest

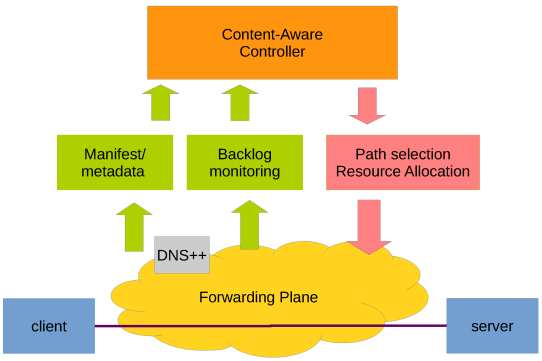

Fig. 3: Feedback Loop for Content Management and Resource Allocation

8) if allowed and beneficial, the content may be forked towards a local cache.

B. Manifest

The manifest is part of the content name resolution. It is the content record returned for a name resolution request for a specific content name.

Figure 2 presents an instance of a manifest in Extensible Markup Language (XML) format. Many fields should be extensible using type-length-value (TLV) to allow for greater expressivity.

This information can be either used and/or modified. Using the information in the manifest allows the network to accommodate certain properties of the data object. For instance, the manifest includes the size of the data, and therefore the edge network can make a different path selection decision depending on the content being an elephant flow or a mice flow. There is prior work on making resource allocation and traffic engineering decision based upon content properties [6], [17], [22].

Modifying the manifest allows the network to include known copies of the content that reside in a local cache for instance; this of course implies that the manifest can be updated in a trusted manner by the network. Our goal is not to describe such trust mechanism, but rather to highlight the potential benefits of a manifest.

Because the manifest is part of a DNS transaction, it is exposed to the network without the need of Deep Packet Inspection (DPI); because the DNS manifest is a network-layer object, it embeds only information that is exposed to the network layer, and therefore is not encrypted, or is encrypted using material that is shared with network operators.

C. Content Management Function and Algorithm

The network controller implements a content management function as in [5] or [17]. Figure 3 shows the feedback loop enabled by the manifest at the controller.

We denote by z_i the ith content request, as well as its size in bytes in its manifest. To each point-to-point content transfer from the source $s \in S$ to the destination $d \in D$, we keep track of a backlog function $B_{z_i}(t)$ at time t. We compute $B_{z_i}(t)$ at each link by subtracting the volume of content that has traversed the link from the original content size. $B_{z_i}(t)$ correspond to the still outstanding amount for the flow z_i. Note that this backlog is not inside the network, but rather corresponds to the amount of data which has not transited through the network yet for a specific object.

Algorithm 1 Minimum Backlog Policy

Require: $\mathcal{P}_{s,d}$ for each (s,d) traffic demand pair AND $B(z_i)(t)$ for every content z_i being transferred, $i = 1,\ldots,n-1$ with $B(z_i)(t) > 0$

1) Select one path $P \in P_{s,d}$, $1 \leq i \leq K_{s,d}$ from the candidate paths set and insert it to the allocation set by $V \rightarrow V + z_n \Rightarrow P$.
2) Given the remaining backlogs $B(z_i)$, at each link l on the path P, calculate
$$B_l = \sum_{i \in l} B(z_i) + z_n.$$
3) Normalize B_l by the capacity c_l of the link l and keep the maximum normalized backlog for path P.
4) Iteratively go back so step 1 and select the next candidate path until all candidate paths are selected in series.
5) Given the computed normalized backlog of each candidate path, select the one which will give the minimum backlog. Add this path to the existing set of paths.

For each arrival z_i, there is a subset $\mathcal{P}_{s,d}$ of all the paths between source s and destination d that we can assign z_i to. Denote by $K_{s,d}$ the cardinality of the candidate path subset, and by $V_{P_i}, i = 1,\ldots,K_{s,d}$ the allocation set that describes the current allocation plus the potential allocation of z_n to

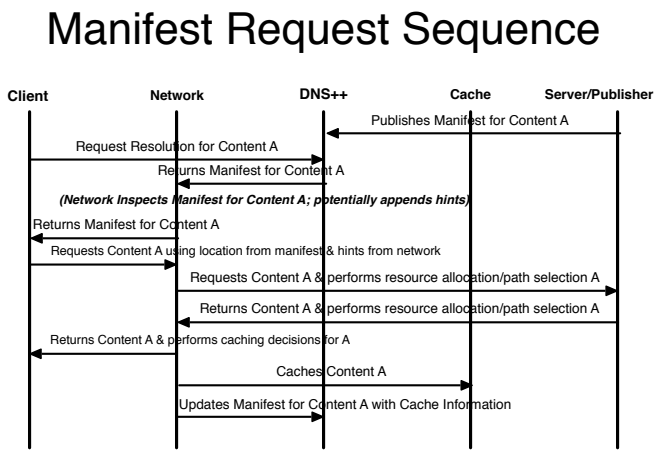

Fig. 4: The messaging flow

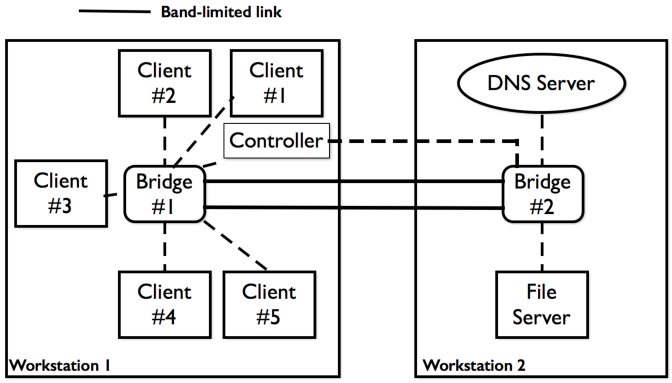

Fig. 5: Evaluation Setup

the i-th path $P_i \in \mathcal{P}_{s,d}$. For instance, V_{P_1} is the allocation of $z_1, \ldots, z_{n-1}$ to their current path with backlog $B(z_i)$ and of z_n to the first path in $\mathcal{P}_{s,d}$ with backlog (or in this case, object size) $B(z_n)$.

We attempt to find the path $P \in \mathcal{P}_{s,d}$ such that:

$$\underset{P \in \mathcal{P}_{s,d}}{\text{minimize}} \max_{l \in P} B_l / c_l \tag{1}$$

that is, to find the path with the minimal max backlog for all objects in the system.

In our simulations, we consider $\mathcal{P}_{s,d} = \{P^k_{s,d}, k = 1, \ldots, K_{s,d}\}$ as the set of $K_{s,d}$ shortest paths given by the output of Yen's k-shortest path algorithm [21]. The scheduling algorithm is summarized in Algorithm 1.

IV. Implementation Set-Up

We have implemented a DNS++ server which returns a manifest. This involves modifying the DNS client of the end-user and the DNS server. The DNS++ client needs to format the requests differently to include the whole name of the object, or the name prefix. DNS++ requests have to be sent over TCP, as the response may be large enough to be split into multiple packets. The DNS++ request also includes the source address, as the network needs to know which client is making the request for path optimization. The DNS++ server is modified to return a manifest if it holds one for the request of a manifest record type. If it does not hold a manifest, it returns the A record corresponding to the domain name, as in a typical name resolution.

Once the controller received the DNS manifest and gets the information about the content size and IPs (server/destination), it starts expecting a new flow from a new source port with corresponding IPs. Once it gets the first packet, it maps the content name to the source port in its dictionary as in [5] and perform the routing to the packets that matches that specific port number, which also means that specific content. We also introduce a periodic polling to get the statistics from the switches to forecast the traffic/backlog on each link and backlog on each content. Therefore, the controller can understand when a specific content is fully delivered, so that this specific port and content pair can be removed from the dictionary. This step is necessary, so that the source port can be assigned again at the client for a new content.

We have also implemented an edge network that listens for manifests on the DNS port 53 and is able to take a corresponding action based upon the upcoming data transfer. Figure 4 shows the data flow. All these elements have been implemented and are used in the evaluation.

In our evaluation, the client requests a file manifest from the DNS and in return, receives the manifest. The network controller observes the manifest and allocates the flow. The flow is mapped to a unique port. We have implemented the following functions:

- a flow-based monitoring function polls the switches periodically to get how much data has been transferred per content flow; therefore the controller knows the outstanding amount of data for each flow at given interval;
- a flow assignment function which computes the total outstanding backlog on a link and assigns the next flow so that the total backlog is minimized.

The set-up comprises a server holding virtual machines (VMs) for the server, the DNS and a virtual switch and another server with VMs for the cache, the client and the network controller. Both servers will be connected over two distinct links on different ports on the physical switch connecting both servers.

V. Evaluation

To demonstrate the benefit of a manifest, we compare the minmax backlog policy with common scheduling policies that are commonly deployed in data centers. We compare with a weighted random policy that loads balance traffic based upon the link capacity (such that the number of flows on each link

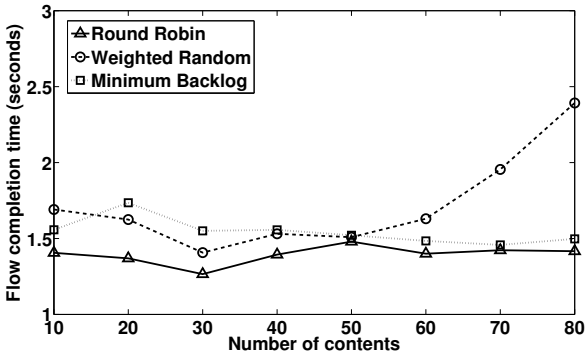

Fig. 6: Completion time of Min-Backlog versus other policies with light traffic load (i.e., poisson arrival rate of 0.12 per client).

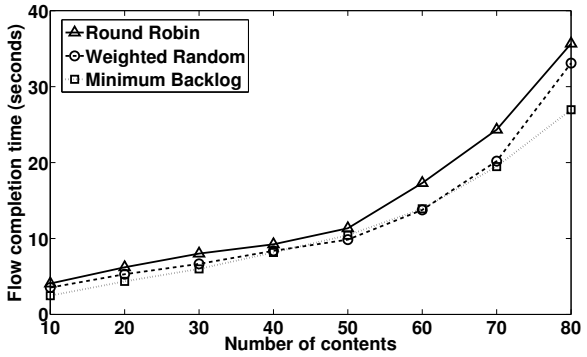

Fig. 7: Completion time of Min-Backlog versus other policies with heavy traffic load (i.e., poisson arrival rate of 0.32 per client).

is proportional to the link capacity) and a similarly weighted round robin.

Note that the key point of the evaluation is not to provide a numerical comparison, but to demonstrate the benefits of the manifest architecture. Namely, by provided meta-data at the network layer, the network is indeed able to improve the network performance, or the QoE of the user. We make no claim that our scheduling policy is near optimal. We chose it for its intuitive simplicity. Better policies, and better learning algorithms can be deployed to improve the network performance.

We considered the evaluation setup that is illustrated in Fig. 5. The evaluation setup consists of five client nodes, a DNS++ server node, and a file server node. We implemented each node in the setup by using VirtualBox VMs and Open vSwitch virtual switches. Moreover, for implementing the content management function running on the controller, we used POX OpenFlow controller. There were two physical links that connects workstation houses clients and workstation houses servers. To emulate the behavior of resource limitation, we reduced the capacity of these links to $5\ \mathrm{Mbit/s}$ and $10\mathrm{Mbit/s}$. We assumed the arrivals of content requests follow a Poisson process.

In the first set of experiments, we aimed to observe efficiency of the Minimum Backlog policy compared to other scheduling policies under different traffic loads. To that end, we used a fixed content size of $10\ \mathrm{Mbits}$ and a poisson arrival rate for request of 0.12 for each of the five clients. We measured the flow completion time, measured as the time difference between the first packet of a content entering the network until the last packet reaches the destination. Fig. 6 shows the flow completion times of different policies for different number of contents. As it can be observed, all the policies were performing similarly as the resource limitations were not reached with the generated traffic load.

To better understand the performance of different policies, we increased the generated traffic load by using a poisson arrival rate for request of 0.32 for each of the five clients. Similar to previous experiments, we measured the flow completion time. Fig. 7 shows that Minimum Backlog policy can reduce the completion time compared to other policies with the advantage of having per content meta-data. While the reduction is close to 3% for smaller number of contents, it increased up to 20% for larger number of contents compared to other policies with the advantage of having per content meta-data.

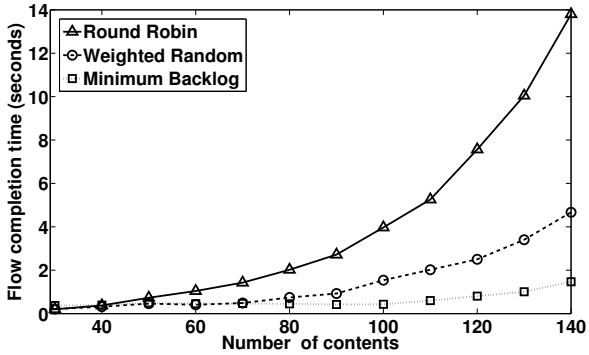

Fig. 8: Completion time of Min-Backlog versus other policies with different content sizes.

For evaluating the performance of the Minimum Backlog policy under more realistic traffic loads, we used contents with different sizes. Specifically, we defined content sizes range from $10\ \mathrm{kbits}$ to $10\ \mathrm{Mbits}$, that are uniformly distributed. Similar to previous experiments, we used a poisson arrival rate for request of 0.32 for each of the five clients. From Fig. 8, we can observe that the benefit of using Minimum Backlog policy increased as the content sizes are diversified. Specifically, the Minimum Backlog policy showed a 70% reduction in terms of flow completion time compared to closest Weighted Random policy. Therefore, we could claim the benefit of the manifest architecture that provides per content meta-data at the network layer, especially for traffic loads that contains diverse content sizes similar to real world.

VI. Discussion & Conclusions

Defining the proper manifest and what properties to include is an on-going task. There is a tension in making the manifest expressive, but at the same time, in keeping it simple. How to scale the manifest is another issue: small objects do not need a manifest, and the advantage of a native ICN architecture is that they can be fetched directly from any intermediate cache as there is no binding of the session to a destination. There is also a concern of getting a per-object manifest, as certain content may contain many objects. For instance, a web page, or a facebook or twitter-like application, will include many pictures and referral to other objects such as advertisement embeds. While it takes only one DNS resolution step currently, per object resolution would dramatically increase the number of such steps. It could even be a new DDoS avenue to generate objects referring to many more objects.

On the other hand, the manifest of the parent web page could include information regarding the other objects. Furthermore, it could also include information on which manifests to get, and which not to (say, if all embeds are small, none of them would require a manifest resolution). This is also a topic of further study.

Our evaluation shows that having a manifest and a simple scheduling policy brings significant benefit, with 70% faster completion time by using the simple min max backlog policy compared to the random weighted algorithm policy.

References

[1] NDN. http://named-data.net/.

[2] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman. A survey of information-centric networking. *Com Mag, IEEE*, 50(7):26–36, July 2012.

[3] Serhat Nazim Avci and Cedric Westphal. A Content-Based Traffic Engineering Policy for Information-Centric Networks. In *Proc. of IEEE CCNC*, January 2016.

[4] Bita Azimdoost, Cedric Westphal, and Hamid R Sadjadpour. On the throughput capacity of Information-Centric Networks. In *Proc. Int'l. Teletraffic Congress 25*, September 2013.

[5] Abhishek Chanda and Cedric Westphal. ContentFlow: Mapping content to flows in Software Defined Networks. In *Proc. of IEEE Globecom*, December 2013.

[6] Abhishek Chanda, Cedric Westphal, and Dipankar Raychaudhuri. Content based traffic engineering in Software Defined Information Centric Networks. In *Proc. IEEE INFOCOM NOMEN Wksp.*, April 2013.

[7] Seyed Kaveh Fayazbakhsh, Yin Lin, Amin Tootoonchian, Ali Ghodsi, Teemu Koponen, Bruce Maggs, K.C. Ng, Vyas Sekar, and Scott Shenker. Less pain, most of the gain: Incrementally deployable ICN. In *Proc. of ACM SIGCOMM*, 2013.

[8] Zhen Feng, Mingwei Xu, Yuan Yang, Qi Li, Yu Wang, Qing Li, Borje Ohlman, and Meng Chen. Joint optimization of content replication and traffic engineering in ICN. In *Proc. of IEEE Conf. on Local Computer Networks (LCN)*, 2015.

[9] S. Gao, Y. Zeng, H. Luo, and H. Zhang. Scalable area-based hierarchical control plane for software defined information centric networking. In *Proc. of Int'l. Conf. on Computer Communication and Networks (ICCCN)*, Aug 2014.

[10] Ali Ghodsi, Scott Shenker, Teemu Koponen, Ankit Singla, Barath Raghavan, and James Wilcox. Information-centric networking: Seeing the forest for the trees. In *Proc. of ACM Wksp. on Hot Topics in Networks*, HotNets-X, 2011.

[11] Reinhard Grandl, Kai Su, and Cedric Westphal. On the interaction of adaptive video streaming with Content-Centric Networking. In *Proc. of IEEE Packet Video Wksp*, December 2013.

[12] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In *Proc. of Int'l. Conf. on Emerging Networking Experiments and Technologies (CoNEXT)*, pages 1–12, 2009.

[13] M. J. Reed. Traffic engineering for information-centric networks. In *Proc. of IEEE Int'l. Conf. on Communications (ICC)*, June 2012.

[14] S. Salsano, N. Blefari-Melazzi, A. Detti, G. Morabito, and L. Veltri. Information centric networking over SDN and openflow: Architectural aspects and experiments on the OFELIA testbed. *Computer Networks*, 57(16):3207 – 3221, 2013.

[15] S. Sevilla, P. Mahadevan, and J.J. Garcia-Luna-Aceves. idns: Enabling information centric networking through the dns. In *Proc. of Computer Communications Wksp. (INFOCOM WKSHPS)*, April 2014.

[16] V. Sourlas, P. Flegkas, P. Georgatsos, and L. Tassiulas. Cache-aware traffic engineering in information-centric networks. In *Proc. of IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, Dec 2014.

[17] Kai Su and Cedric Westphal. On the benefit of Information Centric Networks for Traffic Engineering. In *Proc. of IEEE ICC Conf.*, June 2014.

[18] Cedric Westphal and Emrecan Demirors. An IP-Based Manifest Architecture for ICN. In *ACM ICN Demo*, September 2015.

[19] Cedric Westphal (Ed.) and et al. RFC7933: adaptive video streaming over information-centric networking (ICN). In *IRTF ICN Research Group*, August 2016.

[20] Haiyong Xie, Guangyu Shi, and Pengwei Wang. TECC: Towards collaborative in-network caching guided by traffic engineering. In *Proc of IEEE INFOCOM*, March 2012.

[21] Jin Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

[22] Jin Yichao, Yonggang Wen, and Cedric Westphal. Towards joint resource allocation and routing to optimize video distribution over Future Internet. In *Proc. of IEEE/IFIP Networking Conf.*, May 2015.