2011 Seventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems

COPSS: An Efficient Content Oriented Publish/Subscribe System

Jiachen Chen†, Mayutan Arumaithurai†, Lei Jiao†, Xiaoming Fu†, K.K.Ramakrishnan‡
† Institute of Computer Science, University of Goettingen, Germany.
‡ AT&T Labs Research, Florham Park, NJ, U.S.A.
email: jchen3,arumaithurai,jiao,fu@cs.uni-goettingen.de,
kkrama@research.att.com

ABSTRACT

Content-Centric Networks (CCN) provide substantial flexibility for users to obtain information without regard to the source of the information or its current location. Publish/subscribe (pub/sub) systems have gained popularity in society to provide the convenience of removing the temporal dependency of the user having to indicate an interest each time he or she wants to receive a particular piece of related information. Currently, on the Internet, such pub/sub systems have been built on top of an IP-based network with the additional responsibility placed on the end-systems and servers to do the work of getting a piece of information to interested recipients. We propose Content-Oriented Pub/Sub System (COPSS) to achieve an efficient pub/sub capability for CCN. COPSS enhances the heretofore inherently pull-based CCN architectures proposed by integrating a push based multicast capability at the content-centric layer.

We emulate an application that is particularly emblematic of a pub/sub environment—Twitter—but one where subscribers are interested in content (e.g., identified by keywords), rather than tweets from a particular individual. Using trace-driven simulation, we demonstrate that our architecture can achieve a scalable and efficient content centric pub/sub network. The simulator is parameterized using the results of careful microbenchmarking of the open source CCN implementation and of standard IP based forwarding. Our evaluations show that COPSS provides considerable performance improvements in terms of aggregate network load, publisher load and subscriber experience compared to that of a traditional IP infrastructure.

1. INTRODUCTION

Users increasingly desire access to information, ranging from news, financial markets, healthcare, to disaster relief and beyond, independent of who published it, where it is located, and often, when it was published. Content centric networks (CCN) are intended to achieve this functionality with greater ease for users, greater scalability in terms of the amount of information disseminated as well as number of producers and consumers of information, and greater efficiency in terms of network and server resource utilization. Publish/subscribe (pub/sub) systems are particularly suited for large scale information dissemination, and provide the flexibility for users to subscribe to information of interest, without being intimately tied to when that information is made available by publishers. With the use of an appropriate interface, users can select and filter the information desired so that they receive only what they are interested in, often irrespective of the publisher.

A consumer may not wish (or it may even be infeasible) to subscribe to all of the 'channels' belonging to a myriad of information providers that disseminate items of interest, either on demand (such as web, twitter, blogs and social networks), or tune to a broadcast channel (e.g., television, radio, newspaper). In these cases, the consumer would rather prefer obtaining the data based on **Content Descriptors (CD)** such as a keyword, a tag, or a property of the content, such as the publisher identity, published date *etc.*

Intelligent end-systems and information aggregators (e.g., Google News and Yahoo! News, cable and satellite providers) have increasingly adapted their interfaces to provide a content-oriented pub/sub-based delivery method. However, these mechanisms are built on top of a centralized server-based framework and can also result in a waste of network resources as shown in [1, 2], since the Internet protocol suite is focused on end-to-end delivery of data. Furthermore, issues of "coverage" and "timeliness" still exist in such forms of dissemination, where the aggregator may be selective in what information is made available. Having a network that is capable of delivering the information from any of the producers to all subscribers may overcome such limitations. However, unlike using multicast at the IP layer which can result in a substantial amount of duplicate information being delivered to the receiving end-system (which will have to be filtered out), it is desirable for the network to assist in delivering unique information to the subscriber.

There have been several recent proposals for CCN [3, 4, 5, 6]. One such effort is that of Named Data Networking (NDN) [3, 7]. NDN provides a substantial degree of flexibility for users and end-systems to obtain information without regard to their location or source. Exploiting caching, NDN improves the efficiency of content delivery. Subscribers can obtain the data from the closest node/cache serving it. Moreover, multiple requests for the same data arriving at an NDN router can be served simultaneously by the router, oblivious to the data source. However, this makes the content centric routers somewhat more heavy-weight as we will observe in our micro-benchmarking of such functionality. Moreover, due to its intrinsic design, we observe that enhancements are needed to efficiently support pub/sub applications using the NDN design. In the rest of the paper we use the term CCN to refer to the general content centric

978-0-7695-4521-9/11 $26.00 © 2011 IEEE
DOI 10.1109/ANCS.2011.27

based networking paradigm and use the term NDN to refer to the specific proposal named NDN [7].

A couple of key requirements for a pub/sub system are efficiency and scalability. We observe that the ability to exploit multicast delivery is key to achieving efficiency, and to avoid wasting server and network resources. Scalability requirements come in multiple forms: the ability to accommodate a large number of publishers; the ability to accommodate a large number of subscribers; enable a nearly unlimited amount of information being generated by publishers; allow for delivery of information related to subscriptions independent of the frequency at which that information is generated by publishers; allow for subscribers to not have to be connected to the network at all times, so that information production and reception by consumers can be asynchronous.

In this paper, we develop COPSS, an efficient content-centric pub/sub system leveraging the advantages provided by CCN. We evaluate the performance of COPSS by using a decentralized Twitter-like application and show performance gains in terms of aggregate network load, publisher load and subscriber experience.

The key novelties of COPSS to provide a full fledged and efficient content delivery platform for pub-sub applications include:

- COPSS supports the notion of Content Descriptor (CD) [8, 9] based publishing and subscription. A CD goes beyond name-based [3] and topic-based [10] content identification and allows for contextual identification of information and supports ontologies and hierarchies in specifying interests.
- COPSS provides support for a CD based subscription maintenance in a decentralized fashion, relieving the publishers and subscribers from having a detailed list of one another. This facilitates a highly dynamic and large scale pub-sub environment (in which the focus is on the content published) and facilitates the creation of new publishers and subscribers. This is analogous to recent events in Twitter wherein people belonging to the affected region were able to behave as publishers.
- COPSS provides a push based multicast capability to be able to deliver the content in a timely manner in addition to leveraging the NDN's inherent pull-based information delivery model. COPSS does that in a scalable and reliable manner.
- COPSS is designed to provide additional features for subscribers that are offline and a 2-step delivery model that allow information publishers to exercise policy control, access control (i.e., which subscribers are allowed to access which information) and a snippet based dissemination of large pieces of content in a scalable manner.
- COPSS also addresses the need to evolve from our current IP-centered network infrastructure to a content-centric network.

We review related work in §2. In §3, we identify the requirements of an efficient pub/sub system, provide a short background of NDN, results of a microbenchmark test performed and discuss its shortcomings as an efficient pub/sub system. We present the COPSS design in §4 and evaluation results are given in §5. We conclude our work and outline further work in §6.

2. RELATED WORK

Existing work on pub/sub systems can be broadly classified into two approaches depending on how subscribers obtain data: pull-based and push-based. In a pull-based model, subscribers poll the publisher (or a proxy) for any content/information update. This tends to create unnecessary overheads in server computation and network bandwidth when the update frequency is low compared to the polling frequency. Furthermore, pull-based mechanisms require the knowledge of the identity (DNS/IP address) of publishers (or servers acting as the proxy).

In contrast, traditional push-based approaches maintain long-lived TCP connections (Elvin [11]) or notify subscribers via other means such as instant messaging (Corona [2]) or Rendezvous nodes (PSIRP [12]). Both approaches have scalability issues since it requires the maintenance of too many connections and states; and sometimes require that every publisher and subscriber are known to each other. The wide existence of Network Address Translators (NATs) makes it impractical for every subscriber to have global visibility, thereby complicating push based mechanisms. Overlay based pub/sub approaches like Astrolabe [13] and SpiderCast [10] are agnostic of the underlying topology and therefore cause a lot of extra overhead.

To overcome the limitation of these approaches where a subscription requires the knowledge of every content source, approaches such as ONYX [14], TERA [15], SpiderCast [10], and Sub-2-Sub [16] have been proposed as *topic/content-based* systems. In such systems, users express their interest in content rather than sources (*e.g.*, to a publisher in Twitter[1]). COPSS adopts a **Content Descriptor (CD)** based approach wherein a CD could refer to a keyword, tag, property of the content and *etc.*; similar to that adopted by XTreeNet [8] and SEMANDEX [9]. RSS feeds and XMPP pub/sub [17] are used to publish frequently updated content such as news headlines, blog entries and etc. and allows users to subscribe to topics/publishers. Though both are intended as push based applications, in reality they are essentially pull based mechanisms that frequently poll various RSS sources or XMPP servers.

To our knowledge, there is no prior work which aims to build the content delivery network for efficient pub/sub. NDN [7] and native IP multicast [18, 19, 20] also provides an efficient delivery mechanism, but are not able to serve as an efficient full-fledge content-based pub/sub system as shown in §3. This paper proposes COPSS to fill this gap.

3. PROBLEM STATEMENT

We first describe the requirements that an efficient pub/sub content delivery system has to address. Then, we examine why existing IP multicast, overlay multicast and the current NDN solutions may be inadequate.

[1]http://twitter.com/

3.1 Requirements

An efficient pub/sub information delivery system ("the target system") needs to support:

- **Push enabled dissemination:** To ensure that subscribers receive information in a timely manner, the target system must provide the ability for publishers to push information to online subscribers interested in it. Such timely dissemination is useful in many scenarios such as disaster (e.g., Tsunami) warnings, stock market information, news and gaming.

- **Decouple publishers and subscribers:** As the number of publishers and subscribers increases, it is important for the network to be content-centric (using content names rather than addresses for routing), while still providing the appropriate association between them (publishers need not know who the subscribers are, and vice versa). Furthermore, each subscriber may be a publisher as well (e.g., Twitter allows users to be both subscribers and publishers of data).

- **Scalability:** The target system must handle a large number of publishers and subscribers. Minimizing the amount of state maintained in the network, ensuring the load on the publisher grows slowly (sub-linearly) with the number subscribers, the load on subscribers also grows slowly with the number of publishers (e.g., dealing with the burden of duplicate elimination). Importantly, the load on the network should not grow significantly with the growth in the number of publishers and subscribers. We also recognize the need to accommodate a very large range in the amount of information that may be disseminated, and the need for all elements of the target system in a content centric environment to scale in a manageable way.

- **Efficiency:** The system must utilize network and server resources efficiently. It is desirable that content is not transmitted multiple times by a server or on a link. Furthermore, the overhead on publisher and subscriber end-points to query unnecessarily for information must be minimized.

- **Incremental deployment:** It is desirable that the system be incrementally deployable as we transition from an IP (packet-based) to a content-centric environment. The target system must ensure that its features are beneficial for early adopters, and provide a seamless transfer from an IP dominated environment to a content-centric environment.

Additionally, to **support a full-fledge pub-sub environment**, it is desirable that the target system support the following additional features:

- **Support hierarchies and context in naming content:** We believe it is desirable to be able to exploit both context and hierarchies in identifying content. Hierarchical naming has been recognized by NDN as well. Exploiting context enables a richer identification of content (in both subscriptions and published information), as noted in the database community (and adopted in [8]).

- **Supporting two-step dissemination for policy control and efficiency:** We recognize the need for pub/sub environments to support a two-step dissemination process both for reasons of policy and access control at the publisher as well as managing delivery of large volume content. In such a scenario, the target system would be designed to publish only a snippet of the data (containing a description of the content and the method how to obtain it) to subscribers.

- **Subscriber offline support:** Another typical characteristic of pub-sub environments is that subscribers could be offline at the time the data is published. There is clearly a need for asynchronous delivery of information in a pub/sub environment in an efficient, seamless and scalable manner. The system needs to allow users who were offline to retrieve the data that they have missed. It should also allow new subscribers to retrieve previously published content that they are interested in. We envisage a server that stores all the content published. While important, the storage capacity and policy for replacement is beyond the scope of what we are able to address here in this paper.

3.2 Why Does IP/Overlay Multicast Fall Short as an Efficient Pub/Sub Platform?

IP multicast [18] is another candidate solution for efficiently delivering content to multiple receivers. A sender sends data to a multicast group address that subscribers could join. Multicast routing protocols such as PIM-SM [19] construct and maintain a tree from each sender to all receivers of a multicast group. However, IP multicast isn't an efficient pub/sub delivery mechanism for several reasons: 1) IP multicast is designed for delivery of packets to connected end-points. Dealing with disconnected operation (when subscribers are offline) would have to be an application layer issue. Overlay multicast solutions such as [21, 22, 23] are agnostic of the underlying network topology, usually relying on multiple unicasts in the underlay path and are therefore also inefficient as a pub/sub delivery mechanism. 2) The somewhat limited multicast group address space makes it difficult to support a direct mapping of CDs to IP multicast addresses. 3) Current IP multicast is not able to exploit relationships between information elements, such as CDs. CDs may be hierarchical or may have a contextual relationship, which enables multiple CDs to be mapped to a group. For example, consider a publisher that sends a message to all the subscribers interested in $football$, and subscribers who are interested in receiving messages about all $sports$. The message from the publisher will have to be sent to two distinct IP multicast groups. If there happens to be a subscriber of messages on $sports$ and $football$, (s)he will receive the same message twice and will have to perform redundancy elimination in the application layer. The result is a waste in network traffic and processing at both ends.

3.3 State of the Art: Named Data Networks

NDN: Technical background

NDN [7] has been proposed as a content centric network architecture. Content sources register their availability of content by prefix (akin to a URL), and these prefixes are announced for global reachability (in a manner similar to BGP in inter-domain IP routing). There are two kinds of

packets: *Interest* and *Data* (i.e., content). An Interest packet is sent by a consumer to query for data. Any data provider who receives the Interest and has matching data responds with a Data packet. Both the Interest packet and a Data packet have a content name. For an Interest packet, this name is the name of the requested data; for a Data packet, the name identifies the data contained in this packet. The current design of NDN adopts a URL-like scheme for the content name, e.g., a multimedia item may be named as */uni-goettingen/introduction.mp3*. An NDN router has three data structures (see Fig. 2): the *Forwarding Information Base (FIB)* that associates content names to the next hops (termed *face*); the *Pending Interest Table (PIT)* that maps full content names with incoming face(s); and the *Content Store (CS)* that caches content from a provider upstream. The router forwards an Interest by doing a longest-match lookup in the FIB on the content name in the Interest. When forwarding an Interest, the router also records the name of this Interest and the (inter)face from which this Interest comes into PIT. NDN only routes Interest packets. Data packets follow the reverse path established by the corresponding Interest. When an Interest packet arrives at a router, first the CS is checked to see whether the requested data is present in the local cache. If so, then this Data packet is sent out on the face that the Interest was received and that Interest is discarded. Otherwise, an exact-match lookup is done in PIT on the content name of the Interest. If the same Interest is already pending, then the incoming face of this new Interest is added to the face list of the matched entry and this new Interest is discarded. Otherwise, a longest-match is done on the content name in FIB and the Interest is stored in the PIT and a copy of it is forwarded based on the FIB entry. If there is no matched entry, then the Interest is sent out on all the corresponding outgoing faces. When a Data packet arrives, the CS is checked first. A match implies that this data is a duplicate of what is cached and the packet is discarded. Otherwise, the PIT is checked and a match means the Data has been solicited by Interest(s) forwarded by this node. In such a case, the Data can be validated, added to the CS and sent out on each face from which the Interest arrived.

Difficulties with NDN for pub/sub systems: Multicast

NDN has limited intrinsic support for pub/sub systems, a critical need in a content centric environment. The aggregation of pending Interests at routers achieves efficient dissemination of information from NDN nodes. But this aggregation is similar to a cache hit in a content distribution network (CDN) cache, which occurs only if subscribers send their Interests with some temporal locality. Thus it avoids multiple Interest queries having to be processed directly by the content provider. Note however that this is still a pull-based information delivery method and depends both on temporal locality of interests and a large enough cache to achieve effective caching in the (content centric) network. On the other hand, native multicast support allows for a much more scalable push-based pub/sub environment, since it is not sensitive to issues such as the cycling of the cache when a large amount of information is disseminated. In COPSS we strive to achieve a full fledged push-based multicast capability in addition to the efficient query based information dissemination available in NDN.

Table 1: Forwarding performance (μs)[std. dev]

	CCNx	UDP-K	UDP-U
200B/Interest	2295.6[1106.42]	6.4[0.52]	37.4[8.21]
4096B/Data	2135.4[876.04]	4.0[0.82]	75.2[16.31]

Difficulties with NDN: Obtaining information from unknown publishers

NDN is essentially built as a query-response platform where subscribers are required to know the publishers or the precise identity of the content (URL) to send an interest. Since we believe it is important for subscribers to not know who are the publishers of a particular information item (i.e., a certain CD), NDN poses difficulties in achieving a true pub/sub environment. Consider a 2-publisher, 1-subscriber scenario. Sub sends a query "/query/sports/football" because (s)he is interested in football. On receiving the query, if Pub_1 and Pub_2 happen to have different new items of information, they will return their items to R. But only the response that arrives first will be returned by R since it will consume the PIT entry in R. If Sub wants the second update, he/she would either have to know the exact content name of Pub_2's response, or find a means to exclude Pub_1's response. Obviously, the first option is infeasible, since it is undesirable to negotiate a global name space across all the (possibly unknown) publishers. For the second option, the subscriber will have to specify a large number of names in the exclude field, especially if there are a large number of publishers (one can imagine thousands of such publishers). Moreover, since Sub does not know of the number of available publishers, it needs to send the Interest repeatedly till it stops receiving data from the various publishers and will have to repeat this process periodically. This model thus becomes similar to polling, with its associated overhead (as we will show in §5).

3.4 NDN Performance: Micro Benchmarking

We performed measurements to study the processing overhead of CCN compared to a pure IP-based forwarding (albeit recognizing that the functionality offered by a CCN node is significantly different). We ran simple benchmarks on the open source CCNx implementation[2] and standard IP based forwarding. The measurements were performed on Linux 2.6.31.9 machine (3.0 GHz Intel® E8400, 4GB Memory). Note CCNx is currently implemented as a user-space overlay, using UDP or TCP encapsulation for exchanging CCNx protocol packets between different nodes. To have a reasonably fair comparison, we use two kinds of packets in our measurements: 200-byte UDP packet compared to an NDN Interest packet, and a 4096-byte UDP packet to compare with an NDN Data packet (both with the same UDP payload size). The time between the incoming and outgoing instants of each packet is measured using Wireshark[3]. Three forwarding behaviors are measured: NDN, kernel-space UDP (UDP-K), and user-space UDP (UDP-U).

From Table 1, we observe that to achieve the functionality of name-based routing, NDN routers are about 500 times slower than UDP-K and about 50 times slower than UDP-

[2]http://www.ccnx.org/
[3]http://www.wireshark.org

U. Though a hardware-level implementation would be able to achieve better performance, the requirements placed on an NDN router is higher than that placed on IP router. Since COPSS supports NDN functionality, we understand the need to minimize overhead unless required. Further, we also explore the possibility of a hybrid COPSS + IP approach where the COPSS aware nodes at the edge support the content-centric pub/sub functionality while the intermediate nodes are just IP routers seeking to preserve forwarding efficiency.

4. COPSS ARCHITECTURE

COPSS is designed to be a content centric pub/sub platform that meets the requirements listed earlier - efficient, scalable, exploiting a push-based delivery using multicast for timely but efficient delivery, allow publishers and subscribers to be unaware of each other's identity, and support hierarchies in categorizing information. COPSS leverages the benefits provided by NDN for efficient content delivery and enhances it to provide a full fledged pub-sub platform. Publishers focus on their core task of publishing while not having to maintain membership status, and subscribers receive content from a multitude of sources without having to worry about maintaining a list of publishers and frequently polling them for the availability of fresh data. COPSS naturally deals with substantial churn in subscription state, allowing a large number of users to join and leave and frequently change their subscriptions. The topics may change frequently as well (*e.g.*, in a Twitter-like publishing environment, where the popular topics change frequently).

4.1 COPSS Overview

COPSS introduces a push-based delivery mechanism using multicast in a content centric framework. At the content centric forwarding layer, COPSS uses a multiple-sender, multiple-receiver multicast capability, in much the same manner as PIM-SM, with the use of Rendezvous Points (RP). Users subscribe to content based on CDs. Subscriptions result in 'joins' to the different CDs that are part of the subscription. Each CD is associated with an RP, and the CCN may have a set of RPs to avoid traffic concentration. Although not necessary, the number of RPs may potentially be as large as the number of CCN nodes. Publishers package the CDs related to the information being published by them. As with PIM-SM, the published information is forwarded along the COPSS multicast tree towards the RPs, taking advantage of short-cuts towards subscribers where appropriate. COPSS makes use of hierarchical and context based CDs to aggregate content. COPSS aware routers are equipped with a *Subscription Table (ST)* that maintains CD-based subscription information downstream of them in a distributed, aggregated manner, as in IP multicast. An incoming publication is forwarded on an (inter)face if there are subscribers downstream for any one of the CDs in that publication. However, only one copy is forwarded on a given link (to gain the same advantage as multicast). This also ensures that subscribers subscribed to various groups do not receive duplicate content, to the extent possible. We note that our use of Bloom filters (described below) may result in false positives that would have to be filtered out at the first hop router next to the subscriber.

Additional capabilities in COPSS include the means to perform a two-step data dissemination capability to provide control of policy and access at publishers and to manage delivery of large volume data. Publishers send snippets of the content (which includes the CDs) and subscribers interested in the content query for it. Additionally, the COPSS architecture supports an efficient delivery mechanism for subscribers who were offline when the data was published. Agents/servers are responsible for also storing published content. Thus, subscribers who were offline could seamlessly query the network with the ID of the last received data and the COPSS aware network delivers content from such an agent/server. It also allows new subscribers to receive previously published information of interest.

In order to support pub/sub operation, COPSS introduces two additional types of packets, namely *Subscribe* and *Publish*, and are used in much the same manner as NDN's existing CCN packets *Interest* and *Data* being used for query/response interactions. By issuing a **Subscribe** for a CD, a COPSS subscriber will then receive updates when publishers **Publish** new content. We have attempted to make COPSS backward compatible with NDN as much as possible (e.g., taking advantage of caching in the CCN aware routers etc.). We now address each of the main aspects of COPSS in detail.

4.2 CDs: Hierarchical and Context Based Names

A CD can be any legal Content Name. For efficiency though, we exploit hierarchies in the structure of CDs. For example, a name $/CD/CD1/CD2$ includes CD, $CD1$ and $CD2$ as part of a hierarchy and could have multiple levels. It facilitates COPSS aware routers to aggregate subscription information and avoiding the forwarding of duplicate content. An example name may be $/sports/football/Germany$, where the CDs are $/sports$, $/sports/football$ and $/sports/football/Germany$. A subscription therefore can be at different granularities, taking advantage of this hierarchy.

4.3 Basic One-Step Communication

The basic one-step communication in COPSS is used for information dissemination via a push-based delivery using multicast. This is suitable for a 'pure' pub/sub environment, i.e., sending information where there is no need for policy control or for small volume content (e.g., Twitter-like short messages). The key operations of COPSS in this one-step communication model are *Publish* and *Subscribe*.

4.3.1 Publish using Rendezvous Nodes

COPSS supports sparse mode multicast at the content layer. This is done by introducing a rendezvous node (RN) as the root of a CD's subscription tree. As with an RP in PIM-SM, the RN receives content associated with a CD from a multitude of publishers and forwards it to all subscribers of the CD. The RN is a logical entity and handles information for more than one CD (possibly load-balanced across RNs using a policy for allocation of CDs to RNs) and resides on a physical COPSS aware router. a) An RN needs to be reachable from all publishers sending *Publish* packets with header prefix */rendezvous/* b) RNs receive packets from the publishers, strip the prefix */rendezvous/* and forward it to the interest subscribers.

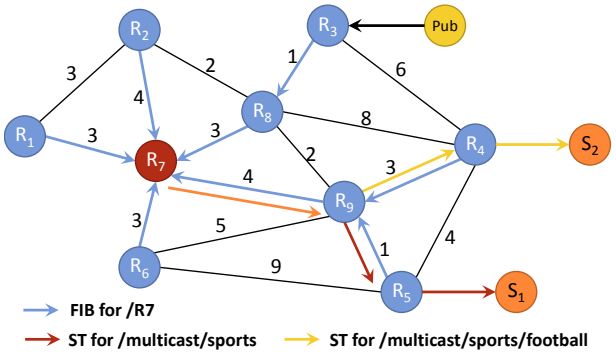


Figure 1: Multicast in COPSS aware network

Note that a publisher does not expect a reply to a *Publish* packet and sets the timeout value to 0. Therefore COPSS aware routers only forward it, rather than putting it into the Pending Interest Table (PIT). See §4.3.3 for a detailed example. The job of a RN is to receive and forward and can therefore be easily replaced whenever necessary.

4.3.2 Subscribe

A subscription received in a *Subscribe* packet (which may include one or more CDs) is treated just like a join in IP multicast. Subscription state is retained in a COPSS aware router, and the *Subscribe* is forwarded (if needed) towards the RN.

Forwarding (via Subscription table)
COPSS uses NDN's forwarding engine with an additional *Subscription Table (ST)* (see Fig. 2). The subscription table is used to maintain a list of downstream subscribers and the outgoing faces towards the subscribers. The *Subscribe* packet is forwarded towards the RN based on the forwarding table entry at a COPSS aware router. Along the path, the COPSS aware routers add this entry in their *ST*. If the COPSS aware router has an entry in the *ST* (the equivalent of being an on-tree node in IP multicast), the *Subscribe* information is aggregated and the incoming interfaces are included in the list of subscribers downstream. If not, the *Subscribe* is also forwarded towards the RN. *Data* messages with CDs matching the entries in the *ST* are forwarded on the list of interfaces that have subscribers downstream. The *ST* can be implemented as a <face:bloomfilter> [24] set. Every incoming data packet is tested against the bloomfilters and will be sent to the faces whose bloomfilter contains CD(s) the packet satisfies. This prevents multiple copies of the same data being sent out on the same interface. A COPSS user needs to subscribe to (i.e., declare an interest in) a CD. The subscription of a *user* to a certain *CD* is performed by creating an ST entry {Prefix="/*CD*"; Face=*user*} in COPSS. This allows a publisher's *Publish* packets with the corresponding CD to reach subscribers.

4.3.3 Overall Example

Rendezvous node set up: Assume that *R*7 in Fig. 1 is the rendezvous node assigned to handle the CD groups */sports/* and */sports/football*. *R*7 is assigned the name */rendezvous* and the COPSS aware network propagates this

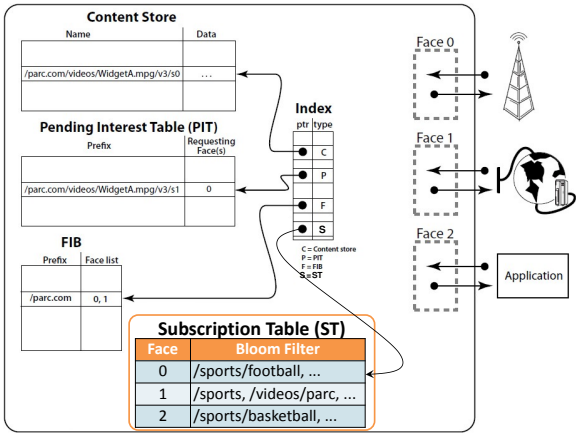


Figure 2: NDN's Forwarding engine adapted with subscription table

information and updates the forwarding tables (FIB) in the COPSS aware routers.

Subscription set up: Lets assume that *S*1 and *S*2, in Fig. 1, are subscribers interested in */sports/football* and */sports* respectively. Therefore *S*1 and *S*2 will forward a subscribe packet towards *R*7. COPSS aware router *R*9 along the path would store both the subscriptions in its *ST* (see Fig. 2) and forward the *Subscribe* packets towards the rendezvous node *R*7. *Subscribe* messages are similar to *Interest* packets in NDN, and can be considered 'standing queries' and uses the FIB for fowarding towards the RN and also updates the *ST*.

Content Delivery: A publisher just sends the content using *Publish* packets (the *Publish* packet is semantically similar to a *Data* packet, carrying data). However, it will be forwarded by looking up the forwarding table (FIB) unlike the PIT in NDN) with the header */rendezvous/sports* and/or */rendezvous/sports/football*. The COPSS aware network will first deliver it to *R*7, the RN. *R*7 on receiving this data will strip the header */rendezvous* and disseminate the *Publish* packet to subscribers downstream. Based on the *ST*, *R*7 realizes that there are subscribers for */sports* as well as */sports/football* and forwards this packet downstream. Intermediate router *R*9, then duplicates the packet and forwards it to *S*1 and *S*2. Hence, the reduction in bandwidth consumption and router processing overhead is achieved at *R*7.

4.4 Two step communication

With the basic push based communication model using multicast, COPSS uses *Publish* packets to carry the published content. However, to provide the flexibility for publishers to exercise policy and access control, as well as to efficiently distribute large volume content, we propose a two-step data dissemination method. Publishers first distribute snippets or a portion of the content (e.g., preview) as part of the payload. Subscribers then explicitly query for the content (an NDN *Interest*). This two-step model avoids subscribers from being overwhelmed with large content that may not be of interest (and saves network bandwidth).

A **snippet** is a payload carried in the *Announcement* instead of the actual content. It contains the meta information (the CDs), message abstract(s), pricing information and anything else that helps a subscriber decide if he would like to have the whole content. Additionally the snippet consists of the *contentName* that would help the subscriber obtain the data from the publishers or other sources that are serving the same *Data*. The *contentName* can be realized in a similar manner as in NDN and must be a unique way of identifying the served content. Published *Data* can belong to multiple CDs, e.g., a news article about "An injury to an American football player" could belong to a CD for */news/america* and a CD for *sports/football*. In such a case, the snippets sent by the publisher to the different CDs could either be the same or even be different, pertaining to the taste of the subscribers. But the *contentName* carried in the snippet would be the same allowing for the possibility of PIT hits (if requests arrive at nearly the same time) and content store hits (if content is available in cache), when requests arrive from the subscribers of the two groups. Below, we detail our two-step(see Fig. 3) communication design.

Publisher: Announce
When a publisher has new content to publish, he multicasts an *Announcement* with the payload carrying a snippet of the data. The *Announcement* is implemented by sending a *Publish* packet, with the format for the name being "*/rendezvous/CD/*". Rendezvous nodes (RNs) do not differentiate between a normal *Publish* packet and an *Announcement* in their processing (i.e., eliminating the prefix "*/rendezvous*" and then forwarding the packet). When a subscriber receives multiple *Announcements* pointing to a same piece of data (identified by the same *contentName* portion), he would only need to query for that data once.

Publisher: Register
With two-step communication, a subscriber generates a query in response to a snippet, if he is interested in the data. For the query to reach the publisher(s) that send(s) the *Announcement*, publisher(s) must first propagate to the network the name prefixes (e.g., *contentName*) of all the content to be published (similar to the propagation of the content sources in NDN). This is called *Register* with the network. For example, a publisher who issues an *Announcement* of "*/rendezvous/CD/*" is expected to propagate the name prefix of "*/contentName*" (in essence the name of the content) in order to make the network aware of the availability of that content. Other subscribers that have already received the data could also serve the content by propagating the appropriate FIB entry to minimize the load on the publishers, especially in the case of large volume content (access control will have to be negotiated with the publisher).

Subscriber: Retrieve Data
On receiving an *Announcement*, the subscriber sends a query using an NDN *Interest* packet whose content name is the *contentName* portion in the snippet received in the payload. The publisher responds with the *Data* associated with the query. This mechanism benefits from NDN's communication paradigm and has the advantages of a potential cache hit on the way (which reduces the access latency of this content) and potential PIT hit (which reduces the query traffic) if

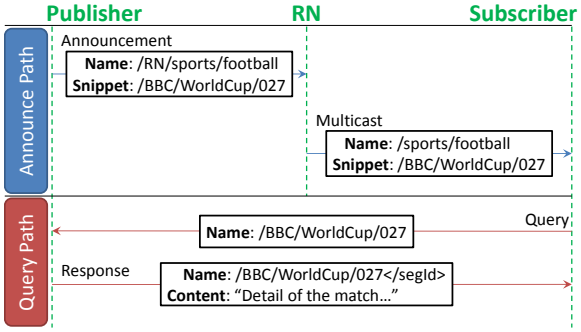


Figure 3: **Platform overview of the one-step and two-step communication.**

the same data has been requested by some other subscriber through the same router.

4.5 Supporting Asynchronous Data Dissemination: Subscribers Going Offline

A true pub/sub environment needs to be able to support 'asynchronous' data dissemination. By this we mean that when a subscriber goes offline (turns off the end-system or moves to a different location), the pub/sub environment will still enable the user to receive messages that were missed while being offline. Furthermore, the user should not have to know who the publishers were, or even whether they are still connected to the network. For instance, the original publisher may no longer be online (e.g., transmitted a warning before being disabled). COPSS supports this by having a dedicated *broker/server* that acts as a store for all COPSS multicast messages. It is likely that a very large amount of information would have to be stored at the broker, which poses scalability challenges. Our solution to this is the natural one: allow the logical broker to be a set of collaborating, distributed servers (i.e., a *broker cloud*). Load is shared among them and they provide some level of redundancy. But most importantly, such a design offers the desired scalability. During COPSS's bootstrapping, the FIB information with a pointer to the broker cloud is propagated to the whole network – just like the FIB pointing to rendezvous nodes is propagated – so that the broker cloud is reachable from everywhere in the network.

The broker subscribes to any newly created CD based on *Announcement* and *Publish* messages received from publishers. Thus, it obtains a copy whenever a publisher publishes new messages. While storage space is a concern, and issues such as the content replacement policy on the broker are relevant, their solutions are likely to be similar to what has been adopted in a non-content centric, server oriented information infrastructure.

Querying for missed messages
In order to query for missed messages, COPSS requires a subscriber to remember the content name of the final message he received when he was last online. We also require that the broker cloud order all received messages based on their arrival using its local time. When a subscriber goes offline and later rejoins the system, he queries the broker cloud

with two pieces of information: the group he subscribed to and the message he received last (by sending a *Subscribe* packet with the header */broker/CD*). The COPSS aware routers forward the *Subscribe* based on the header */broker/* to the broker cloud. The broker cloud has to look up in the log to find the match to the message from the subscriber. For each multicast message received after this, the broker cloud checks whether it belongs to any of the CDs provided by the *Subscribe*. The broker then sends all the matched messages to the subscriber. Below, we address scalability and reliability issues:

Scalability: Retrieving missed content
A subscriber may have subscribed to a very large number of CDs. Some of the CDs subscribed to could be related to infrequent events such as a "disaster warning", or popular, frequently updated information such as sports. When dealing with asynchrony, a subscriber coming back online would have to send a query for every CD that he has subscribed, since it is impossible to predict which groups have had new content. With the magnitude of subscribers and CDs envisioned, such a pull-based approach for information every time a subscriber comes back online (or moves to a different point in the network) could result in a lot of traffic. To reduce the query load that a subscriber generates and the corresponding processing overhead at the broker, COPSS seeks to aggregate processing by grouping content across multiple CDs. Instead of querying for content related to individual CDs, the subscriber queries for the group. The tradeoff is that the subscriber might receive false positives, which have to be eliminated at the receiving end-point. Aggregation functions include traditional hashing schemes (e.g., based on the CD string), but have the disadvantage that their decision does not take the semantics related to CDs into consideration. COPSS's use of the hierarchical structure of CDs allows subscribers and/or brokers to exploit it for aggregation. We believe that this will help to minimize the retrieval overhead and in reducing false positives. For example, a subscriber who is subscribed to a large number of disaster warning related CDs could aggregate them to a higher level CD such as */disaster/* and issue one query to the broker. It is particularly attractive when these updates are infrequent and the number of false positives are small.

Scalability: Message delivery
The scale of the subscribers envisioned is likely to lead to many offline users becoming online in a burst at peak periods (e.g., in the morning), resulting in a large burst of query traffic. But this also provides opportunities to optimize network traffic. We propose using markers in the message sequence to allow for batching responses. E.g., assume a subscriber of a CD requests for content that he missed since 9 pm and another subscriber of the same CD requests content that he missed since 11 pm. Using a marker to delineate the message sequence into batches allows the broker to multicast the overlapping message sequence between the subscribers. This reduces both network and broker load.

Reliability: Possible loss of sequence
COPSS multicast messages may arrive at the subscribers and the broker cloud in different order. This can be caused e.g., by varying latencies from different publishers. Thus, if the broker simply provides the subscribers with the messages received after the matched message in the log, some published information may be missed. We solve this problem by requiring the broker cloud to group all the messages in its log into different *windows*. Let the size of this window be n, indicating a set of consecutively received messages at the broker. Upon recept of a query, the broker finds the *target window* that contains the matched message. Then messages related to the queried CD belonging to the same window are sent to the subscriber. By sending these extra messages within the *target window*, messages that may have been received out-of-order can be included and delivered to the subscriber. The subscriber would be responsible for duplicate elimination. The subscriber would also maintain a local *window* that stores the messages that the subscriber received as soon as he came online. This ensures that once the broker starts sending messages that the subscriber has already received, the subscriber could stop sending the query. Note the parameter n has to be tuned to make a good tradeoff of delivering unnecessary information to subscribers and network load versus the success probability of recovering all the messages when the subscriber is offline, depending on the message frequency and user online/offline pattern.

4.6 A Hybrid Model for Incremental Deployment of CCN Capability

We have so far considered an ideal case where all routers are COPSS aware. However, our micro benchmarking of the forwarding performance of CCN routers (see §3.4) indicated that the processing a CCN packet in a CCN router can be substantially more expensive than forwarding a packet at a normal IP router. Given that forwarding a *Data* packet involves examination of the CCN headers (to parse the CDs, examine if it is in the cache etc.), it is desirable that these functions be performed on when essential.

Therefore, we develop a practical yet effective solution for a hybrid environment (COPSS + IP): COPSS aware routers are present only at the edge of the network and connected to the native IP network. We provide the necessary COPSS functionality at the edge while still achieving efficiency of forwarding packets in the core. Parsing of content centric names and forwarding decisions are made by the COPSS aware (edge) routers, while leveraging the high performance of IP forwarding in the core IP network. Such an architecture naturally allows subscribers to subscribe to CDs and allows publishers to publish data based on CDs. Furthermore, the deployment cost may be reduced by only replacing edge routers with COPSS aware routers.

To facilitate the push-based multicast in COPSS, we make use of native IP multicast capabilities in the core of the network and perform the mapping from CDs to IP multicast group addresses. This allows the network to maintain the advantages provided by the COPSS for both publishers and subscribers such as maintenance of the subscription list, CD-based subscription and publishing, one-step and two-step dissemination and support of users going offline but still allowing asynchronous communication between publishers and subscribers. The downside is given that the limited availability of IP multicast addresses compared to the envisioned scale of the number of CDs in a COPSS aware edge router, multiple CDs may be mapped to a IP multicast address.

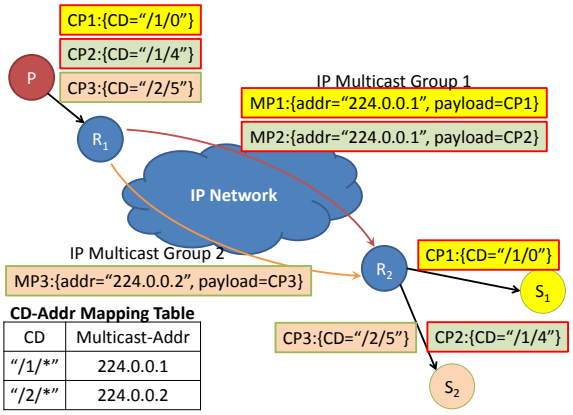


Figure 4: **Multicast in COPSS + IP.**

Such a mapping may resulting in messages being unnecessarily delivered to subscriber end-nodes that then have to discard irrelevant messages.

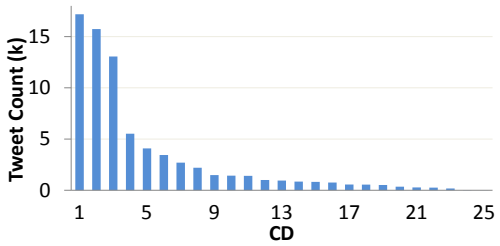
An example: In Fig. 4, we give a brief example about how hybrid-COPSS works. S_1 (subscribed to "/1/0") and S_2 (subscribed to "/1/4" & "/2/5") are connected to R_2, which is a COPSS-aware edge router. On receiving the subscription request, R_2 translates these CDs to IP multicast groups according to the mapping table. As a result, it joins groups 224.0.0.1 and 224.0.0.2 in the IP network. When a publisher P (connected to R_1) multicasts a COPSS packet, R_1 will encapsulate the packet according to the CDs it contains. In this example, $CP1$ is encapsulated as a UDP/IP packet using the IP multicast address 224.0.0.1 based on the mapping table. When R_2 receives the packets, it disseminates the packets to the end hosts according to the subscription table, ST as described above. This way, S_1 then receives $CP1$ and S_2 receives $CP2$ and $CP3$. However, according to the mapping table, R_2 can also receive packets with CD "/2/3". Since there is no subscriber downstream, R_2 will discard this packet. This results in some wasted network traffic being transmitted on some links. This is the tradeoff because of the aggregation of CDs.

5. EXPERIMENTAL EVALUATION

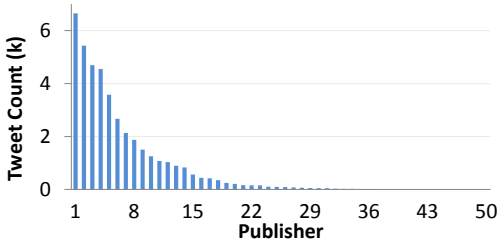
We use simulations to evaluate the benefit of COPSS. We show how pub/sub capability of COPSS achieves improved performance compared to a pull-based NDN implementation as well as pure IP multicast. We use the results of our micro-benchmarking measurements to parameterize the simulations. We built an event-driven simulator in $C\#$. To drive our simulator, we use a set of traces collected from Twitter. We evaluate the performance of COPSS in multiple dimensions: forwarding performance with both one-step and two-step (delivering a snippet and allowing subscribers to query for content) communications, subscribers going offline and then retrieving messages missed, as well as the benefits of a hybrid model.

5.1 Experimental Setup

Dataset: We use a Twitter data trace of tweets on technical topics obtained from the public Internet during a one-week



(a) # of tweets per CD



(b) # of tweets per pub

Figure 5: **Dataset information**

period in 2010, which totaled 68,695 tweets sent by 38,481 users. We identified and selected 25 hot keywords such as *iphone*, *ipad*, *blackberry*, *smartphone* as CDs. We filter this data trace and retrieve a subset in which every tweet contains at least one of the 25 keywords. This subset amounts to 41,613 tweets from 22,987 users (4.13 tweets/minute, up to 48 tweets posted at the same time) and is used as our simulation input. We then filtered additional hot keywords from these 41,613 messages in order to obtain sub-CDs for the selected 25 CDs. The sub-CDs range from 1 to 25 for the selected 25 keywords. By this method, we have a total of 407 distinct CDs that can be hierarchically grouped into 25 CDs. Fig. 5a shows the number of tweets associated with each CD respectively. To make the publishers of our system tweet more frequently, we assign the 22,987 users to 50 publishers by hashing them based on a power-law function. Fig. 5b shows the number of tweets per publisher. Without the means to inferring CD-Subscriber relationship from the data trace, we assume the more popular the CD is, the more subscribers there will be (based on [25]). Thus, we distributed the number of subscribers per-CD based on the CD-Tweet relationship. Subscribers can subscribe to multiple CDs, but a subscriber who is subscribed to a top level CD will not be subscribed to a lower level CD belonging to it. To simplify the simulation, subscribers will query the data as soon as they get announcements.

Network topology: We use the Rocketfuel [26] backbone topology (id=3967) for the core routers. Besides, we put 200 edge routers on the 79 core routers, with each core router having 1-3 edge router(s). We randomly distributed 50 publishers, and uniformly distributed the subscribers (varying from 200 to 600) on the edge routers. The link weights between the core routers were obtained from the topology and interpreted as delays (ms). The delay between each edge router and its associated core router is set to 5 ms; the delay between each the host and its associated edge router is

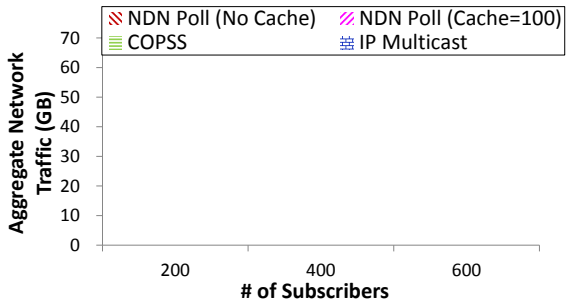


Figure 6: Aggregate Network load (NDN (cache=0, cache=100) vs. COPSS one-step and IP Multicast)

set to 10 ms.

Aggregate network load: We study the impact of COPSS on the network by measuring the *aggregate network load* calculated by:

$$\sum_{i=1}^{packetCount} packetSize_i \times hopCount_i, \quad (1)$$

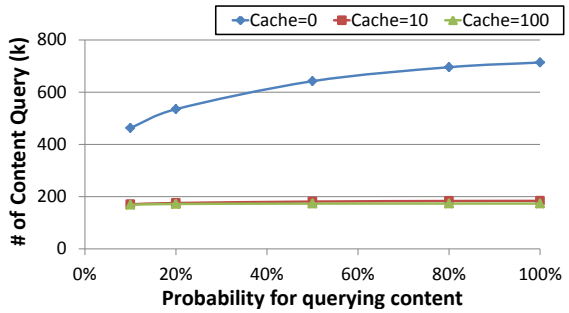
i.e. when a packet with size 1kB is sent from host A through router R to host B, 2kB is added to the aggregate network load. For a fair comparison, COPSS packets are encapsulated into UDP packets when transmitted over an IP underlay. The encapsulation overhead is therefore the same as in a CCNx implementation.

5.2 Performance of COPSS's Basic Communication Model, NDN and IP Multicast

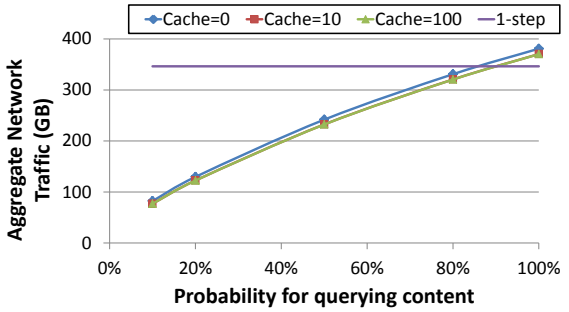
Fig. 6 illustrates the aggregate network load driven by our emulated Twitter-like application over the standard pull-based NDN (both without a cache as well as a cache of 100 packets), using COPSS's one-step communication model as well as native IP multicast. We observe that the COPSS has a lot less load in this case, because of its design of a content centric push mechanism that improves upon the NDN proposal by adding a multicast capability. In addition: 1) Polling is not used by COPSS unlike NDN. With NDN, subscribers need to poll periodically (every 30 minutes in the default setting) introducing additonal overhead. 2) Network traffic is further reduced because of multicast with COPSS even compared to the use of caches in NDNs. Moreover, COPSS performs no worse than native IP multicast, because of the use of the hierarchy based grouping of CDs. This results in fewer messages being transmitted to reach the subscribers of different CDs. Note that the aggregate network load due to NDN and IP multicast increases linearly with the number of subscribers. With COPSS, the increase in subscribers results in only a marginal increase in the aggregate network load since the data is only duplicated very close to the subscriber (in the optimal case at the subscriber's first hop router).

5.3 Performance of COPSS (Two-Step)

Here, we evaluate the performance of the COPSS to transfer large volume content using its two-step communication model. Large files that are 128 times larger than the original Twitter messages are created from the dataset. We



(a) Publisher Load



(b) Aggregate Network Traffic

Figure 7: **COPSS in two-step mode**

Table 2: **COPSS + IP performance vs COPSS-everywhere and IP-multicast**

	IP-Multicast	COPSS-everywhere	COPSS + IP-Multicast
Content Dissemination Latency (ms)	78.76	82.73	77.62
Aggregate Network Traffic (GB)	19.50	12.32	13.15

study the load on the publisher (see Fig.7a) by calculating the number of queries for data that reach the publisher for varying cache sizes (0, 10, 100). COPSS is able to leverage the benefits of NDN, by first pushing snippets to subscribers who then immediately query the publisher if they are interested in the content.We observe that a cache size of 10 packets is sufficient to reduce the load on the publisher significantly.

Fig. 7b illustrates the aggregate network traffic when a varying number of subscribers request for the full content on receiving the snippet. For reference, we have also shown the case of COPSS in one-step mode delivering the full content instead of sending a snippet. When a small percentage of users request for the content, substantial network resources are saved by adopting the two-step mode. However, when the number of subscribers requesting for the content reaches more than 85%, the two-step mode is more expensive because the sending of snippets in the first step is just additional overhead compared to the one-step delivery mode.

Table 3: **Broker load vs. router cache size**

Cache Size	0	10	100
Broker load (# of query)	895.953	892.714	778.270

5.4 Performance of Hybrid-COPSS (COPSS at Edge + IP at Core)

In §3.4, we observed that content oriented processing is more expensive than IP. The aim of hybrid-COPSS (COPSS at edge + IP in the core) is to achieve the best of both worlds by obtaining the functionality of content centricity while retaining the forwarding efficiency of IP. To validate this, we evaluate the performance of hybrid-COPSS in one-step mode (COPSS aware routers at edges and IP routers at core) to that of COPSS-everywhere in one-step mode (all routers are COPSS enabled) as well as native IP-multicast. Table 2 shows the content dissemination latency which is dominated by the processing overhead incurred at every router (the processing overhead was obtained through measurements 1). The latency incurred is high in the case of COPSS-everywhere since every COPSS aware router will have to process the *Publish* and *Subscribe* packet. Comparatively, routers enabled with IP multicast perform much better in terms of latency. Hybrid COPSS is able to achieve lower latency than both COPSS-everywhere and IP-multicast. This is due to the fact that even though the edge COPSS aware routers incur high processing overhead, they are able to send a single copy to multiple groups and are thus able to send much lower traffic into the network. IP-multicast on the other hand needs to send the same message multiple times to be able to reach subscribers belonging to different CDs. This can be clearly seen in the row titled "Aggregate Network Traffic" where we observe that IP-multicast generates higher network traffic. The aggregate network traffic of hybrid-COPSS is slightly higher than COPSS-everywhere due to the fact that CDs are mapped to IP multicast groups and therefore results in a small amount of false positive content being delivered at the last hop COPSS enabled router close to the subscriber.

5.5 Performance of COPSS for Offline Users

Based on our preliminary analysis on the twitter data trace available, we synthesize the users' offline/online pattern as follows: everyday about 75% of all subscribers randomly go offline between 2am and 3am, and then go back online randomly between 10am and 11am. For each of the remaining users, we randomly choose two time points as the start and end points of their offline period with an average offline duration of 20 minutes. Though this does not match a real world scenario, we created such a behavior to study the impact of a large portion of the subscribers coming online at nearly the same time. In Table 3, we observe that as the cache size increases, the number of queries reaching the broker reduces. The cache hit rate is boosted by the division of content based on the markup message and the grouping of subscribers into higher level CDs when appropriate.

5.6 Additional Observations

Table 4 shows that in the case of NDN-pull and COPSS (2-step), the publishers need to be visible and therefore have to propagate their entry throughout the network. In the case of COPSS (1-step), since it is an RN based multicast, the publishers need not propagate their entry and only the RN propagates the entry to all the (278) COPSS enabled routers. This shows that COPSS (2-step) behaves in a manner similar to NDN-pull with regards to FIB propagation whereas in the case of COPSS (1-step), the size of the FIB in the network is considerably lower.

Table 4: **The FIB entry created due to Server/RN and publishers of the specific content**

	NDN-Pull		COPSS (1-step)		COPSS (2-step)	
Node type	Pub	Server	Pub	RN	Pub	RN
FIB entries	13,950	279	0	278	13,950	278

6. SUMMARY

In this paper we presented and evaluated COPSS, an efficient pub/sub-based content delivery system exploiting the fundamental capability of CCN for efficient information dissemination. COPSS builds on existing proposals for CCN/NDN to provide pub/sub functionality. COPSS is designed to be scalable to a large number of publishers, subscribers and CDs. We recognize that a pub/sub platform needs to be able to accommodate users going offline, and allow them to retrieve content that has been published during the time the subscriber was offline. We also explicitly address the need for incremental deployment of CCN capability in traditional IP networks. We use trace-driven simulations to evaluate the COPSS architecture. COPSS reduces the aggregate network load and the publisher load significantly. The additional CCN layer processing with COPSS compared to IP multicast is relatively small, achieved by considerable efficiency in avoiding duplicate and unnecessary delivery of content to subscribers. COPSS is substantially more efficient than existing pull-based CCN proposals (such as NDN) because of its inherent pub/sub capability.

7. REFERENCES

[1] K.V. Katsaros, G. Xylomenos, and G.C. Polyzos. Multicache: an overlay architecture for information-centric networking. *Elsevier, Computer Networks*, 55(4):936–947, March 2011.

[2] Venugopalan Ramasubramanian, Ryan Peterson, and Emin Gün Sirer. Corona: a high performance publish-subscribe system for the world wide web. In *NSDI*, 2006.

[3] L. Zhang, D. Estrin, J. Burke, V. Jacobson, and J.D. Thornton. Named data networking (ndn) project. Tech. report ndn-0001, PARC, 2010.

[4] 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.

[5] S. Arianfar, P. Nikander, and J. Ott. On content-centric router design and implications. In *ReArch*, 2010.

[6] B. Ahlgren, M. D'Ambrosio, C. Dannewitz, M. Marchisio, I. Marsh, B. Ohlman, K. Pentikousis, R. Rembarz, O. Strandberg, and V. Vercellone. Design considerations for a network of information. In *ReArch*, 2008.

[7] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F.

Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *CoNEXT*, 2009.

[8] W. Fenner, D. Srivastava, K. K. Ramakrishnan, D. Srivastava, and Y. Zhang. Xtreenet: Scalable overlay networks for xml content dissemination and querying. In *WCW*, 2005.

[9] M. Ott, L. French, R. Mago, and D. Makwana. Xml-based semantic multicast routing: an overlay network architecture for future information services. In *GLOBECOM*, 2004.

[10] Gregory Chockler, Roie Melamed, Yoav Tock, and Roman Vitenberg. Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication. In *DEBS*, 2007.

[11] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content based routing with elvin. In *AUUG2K*, 2000.

[12] C. Esteve, F. Verdi, and M. Magalhaes. Towards a new generation of information-oriented internetworking architectures. In *ReArch*, 2008.

[13] R. V. Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM TOCS*, 21:66–85, 2001.

[14] Yanlei Diao, Shariq Rizvi, and Michael J. Franklin. Towards an internet-scale xml dissemination service. In *VLDB*, 2004.

[15] Roberto Baldoni, Roberto Beraldi, Vivien Quema, Leonardo Querzoni, and Sara Tucci-Piergiovanni. Tera: topic-based event routing for peer-to-peer architectures. In *DEBS*, 2007.

[16] Spyros Voulgaris, Etienne Riviére, Anne-Marie Kermarrec, and Maarten Van Steen. Sub-2-sub: Self-organizing content-based publish and subscribe for dynamic and large scale collaborative networks. Research report, INRIA, December 2005.

[17] Xep-0060: Publish-subscribe. Xep-0060 (standard track, v 1.13, XMPP Standards Foundation, http://xmpp.org/extensions/xep-0060.html, 2010.

[18] S. Deering. Host extensions for ip multicasting. RFC 1112, August 1989.

[19] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification. RFC 2362 (Experimental), June 1998.

[20] T. Pusateri. Protocol Independent Multicast - Sparse Mode (PIM-SM) IETF Proposed Standard Requirements Analysis. RFC 4602 (Informational), August 2006.

[21] Y.-H. Chu, S.-G. Rao, and H. Zhang. A case for end system multicast. In *SIGMETRICS*, 2000.

[22] S. Banerjee, B. Bhattacharjee, and C. Kommareddyand. Scalable application layer multicast. In *SIGCOMM*, 2002.

[23] J. Jannotti, D.-K. Gifford, and K.-L. Johnsonand. Overcast: Reliable multicasting with an overlay network. In *USENIX OSDI*, 2000.

[24] Bernard Chazelle, Joe Kilian, Ronitt Rubinfeld, and Ayellet Tal. The bloomier filter: an efficient data structure for static support lookup tables. In *the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2004.

[25] H. Kwak, C.Lee, H.Park, and S.Moon. What is twitter, a social network or a news media? In *WWW*, 2010.

[26] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. Inferring link weights using end-to-end measurements. In *IMW*, 2002.