Experience with Collaborative Conferencing Applications in Named-Data Networks

Jun Wei, Duy Nguyen, J.J. Garcia-Luna-Aceves, Kathleen Nichols

1 Huawei Technologies US R&D, Santa Clara, CA 95050, jun.wei@huawei.com
2 University of California, Santa Cruz, CA 95064
3 PARC, Palo Alto CA 94304, {duy, jj}@soe.ucsc.edu
4 Pollere Inc., Menlo Park, CA 94025, nichols@pollere.com

Abstract—We propose a hybrid approach for conference applications for Named-Data Networks (NDN). This hybrid approach combines a participant-driven, distributed server-based approach for conference control and management with a server-less approach for media forwarding. The participant-driven, distributed server-based approach provides robust and efficient conference control and management that is resilient to single point of failure and can scale dynamically with any conference size. The server-less media forwarding eliminates traffic concentration and congestion in the network. We have carried out experiments using XMPP-based conference tools over CCNx and have successfully built the media forwarding for two conference tools (audio and whiteboard) on Android platform. This hybrid approach leverages NDN to support robust and efficient conference applications.

Index Terms—Collaborative Conferencing Applications, Named-Data Networking, Content-Centric Networking, Future Internet Architecture.

I. INTRODUCTION

The current Internet has probably reached its prime time with the commercial movement of converging legacy networks into all-IP networks. However, IP was designed when connecting host computers was the primary goal of networking. Many evidences have indicated that IP's connection-oriented communication does not fit well for today's and future communication needs. Content-Centric Networking (CCN) [5] has been proposed to change the communication model from connection-oriented to content-oriented. An open source community Project CCNx[10] has been built to stimulate research activities around CCN. Furthermore, Named-Data Networking (NDN) [8] extends such research effort within the NSF funded Future Internet Architecture programs.

NDN uses a pull-based communication model with two primitives, namely interest and data [5], and one interest packet pulls one data packet. Each packet has a name. An example of the name prefix can be URL-like format, such as /ccn/myinterest. A NDN router includes Content Store (CS) and Pending Interest Table (PIT) in addition to the FIB in an IP router [5]. A data packet can be cached in content store or CS for later retrieval. The pending interest table or PIT aggregates multiple interest packets with the same name prefix and forwards only one interest packet for each name prefix. PIT also remembers the interfaces (called faces in NDN [5]) those interest packets are from for later use. An interest packet is routed by its name, based on the FIB entries. Once a matched data packet is founded, either from original data source or from a cached copy in a NDN router, it follows back through the same route of the interest packet. To forward a data packet, a NDN router checks with PIT entries against the name prefix of the data packet, and forwards one copy towards each face that has a pending interest. The CS allows caching inside NDN networks and the PIT enables native multicast distribution. Both CS and PIT provide better support for conference applications natively in NDN than in IP networks.

In this paper, we present an example of leveraging NDN to better support conference applications. We propose a hybrid approach for conference applications in NDN networks which combines a participant-driven, distributed server-based approach for conference control and management with a server-less media forwarding among participants. Leveraging the caching and multicast support from NDN, the server-less media forwarding eliminates traffic concentration and congestion. At the same time, the participant-driven, distributed server-based approach enables efficient and robust conference control and management resilient to single point of failure with elastic scalability.

We have carried out experiments using XMPP MUC [9] for conference control server and have built two XMPP conferencing tools over CCNx. Our experience in design and implementation provide valuable insights on application development for NDN networks. Section II reviews related work on conference applications. Section III describes our hybrid design and implementation experiences. Section IV shares our experiments and insights on application development for NDN networks. Section V concludes this paper.

II. RELATED WORK

One of the early online collaborative conference experience can be traced back to a group of distributed conferencing tools, vic (video), vat (audio), and wb (whiteboard) [7], over the multicast test-bed Mbone [6]. However, they have very limited usage outside research community due to the difficulty of commercializing of multicast services. On the other hand, many of the today's web conference applications rely on centralized servers to process data from participants before redistributing back to participants. The media forwarding usually creates traffic concentration and possible congestion around centralized servers. Sometimes, servers are also involved in mixing individual media streams. To improve the
scalability, end system mixing is often used instead of server-based mixing [3].

ACT [12] has proposed a distributed design of audio conference tool for NDN. However, it has not yet considered many conference control features other than conference and speaker discovery.

III. DESIGN

We propose a hybrid approach for conference applications in NDN. We combine a server-based conference control and management or control plane with a server-less media forwarding or media plane shown in Figure 1.

We employ a server-based approach for control plane so we can leverage existing technologies, e.g. XMPP with MUC extensions. However, server-based approaches inherit problems such as traffic concentration and congestion, single point of failure, etc. To address those problems, we expand single control server to a distributed server cluster. Participants take full control when communicating with servers. Leveraging with NDN, our participant-driven, distributed server-based approach enables robust and efficient conference control and management that is resilient to single point of failure with elastic scalability.

The server-less media forwarding eliminates traffic concentration and congestion in network. NDN's multicast support and caching inside network makes media forwarding more efficient than in IP.

A. Control Plane

We use control server to facilitate conference control and management functions through message exchanges with participants. One example of such server is XMPP with MUC extensions. In our design, we put participant at the driver seat that will play active roles in all control operations. The control server reacts passively.

In this paper, we focus on the message exchanges between participants and the conference control servers. We use a single control server to describe the conference control and control server to describe the conference control and management functions in subsection 1), message exchange in NDN in subsection 2), and then we extend from single control server to a distributed server cluster in subsection 3).

1) Conference Control and Management

The conference control and management functions consist of conference admission (e.g. registration), floor control (e.g. moderation) and updates. We start with a conference created by an organizer. The organizer creates a conference agenda and invites moderators and speakers offline. Each moderator hosts a session with a series of talks and a short question and answer (Q&A) session after each talk. To ask questions at a Q&A session, a participant needs to get permission from the moderator. Throughout the conference, the organizer may update agenda as needed.

A typical series of operations for organizer may look like:
1. request registration to create conference
2. submit conference agenda
3. submit conference update

The above Step 1 and Step 2 allow the organizer to create a conference and designate the conference control server to manage the conference, e.g. distributing conference agenda. The agenda may include conference name, schedule, moderator list, speaker list, announcement, and pointer to the post-conference information (feedback form, proceeding, recording etc.). Step 1 and Step 2 usually happen just once. Step 3 may occur multiple times throughout the conference.

A typical series of operations for participant may look like:
1. query conference list (optional)
2. request conference registration
3. request agenda
4. query announcement and/or request update
5. submit feedback

The above Step 1, 2 and 3 usually happen once. Step 1 is optional for those who have not yet known the conference through external means. Step 5 usually occurs at the end of each session and the end of the conference when participants are encouraged to submit feedback.

Step 4 occurs throughout the conference. Participants can periodically query the server for update announcement. The query frequency can be as low as once every session, e.g. right at the beginning of the session. Participants are also allowed to request updates any time, e.g. when they misplaced their agenda.

An example of floor control is the Q&A session. For those who want to ask question at a Q&A session, permission needs to be obtained from the moderator first. In addition, whoever wants to be heard will need to have their names known by the audience before they speak. See details in subsection 2). Addition steps will be required as following:
1. submit Q&A permission request (for those asking questions) or request Q&A permission request list (for moderators)
2. submit Q&A list (for moderator) or request Q&A list (for non-moderator)

Step 1 allows the control server to compose a Q&A permission-request list (from those who has requested permission) which the moderator reviews and then composes a Q&A list for those with permission granted. Step 2 allows the audience to retrieve the Q&A list. Due to the time constraint, the moderator may or may not grant permission to all, such that the Q&A list may be shorter than the Q&A permission-request list. As an option, a list of questions may be attached to the Q&A permission request so that the moderator may choose to append the Q&A list with a list of questions. Any Q&A permission request arrived after the moderator has retrieved the Q&A permission-request list will not be granted permission.

The audience will have full control as whom to listen to. Often times, a Q&A session can easily go over time. When the next scheduled speaker starts to talk, the audience can choose to tune out of the Q&A session (stop sending interests to the names on the Q&A list) and tune in to the speaker (start sending interests with the name of that speaker) (details in section B). As the audience tuned in to the next speaker, the overdue Q&A session automatically turns into a private discussion if desired. The moderator could "ban" a particular participant by not granting permission and excluding specific name from the Q&A list. A participant should not ask question if his/her name is not on the Q&A list because he/she will not be heard.

Many of operations do not require reliable communication. In the case of message lost, a participant simply repeats the failed operation. For example, request agenda again if not received. On the other hand, e.g. a Q&A permission request lost, it may be too late to repeat the operation of Q&A permission request.

2) Control Message Exchanges in NDN

In general, control operations consist of two types of message exchanges, namely Type 1 and Type 2. Type 1 is for a participant node to request data from the conference control server, such as query conference list, request conference agenda, request Q&A list, request update, etc (see subsection 1). Type 2 is for a participant node to submit something to the conference control server, such as submit conference agenda, submit updates, submit Q&A list, submit feedback, etc.

The message exchange uses NDN primitives, interest and data. We assume a name, \texttt{conf\_service}, for the conference control and management service. The control server responds to a participant’s request for conference information using the name prefix of \texttt{conf\_service}. We assume external means to discover the conference service name \texttt{conf\_service}.

For Type 1 message exchanges, the participant node first issues a CCN interest [5] with the prefix, e.g. /\texttt{conf\_service}/\texttt{conf\_list}. The interest is routed to the control server, which responds with data packet of the same name, see Figure 2.

NDN allows data to be cached inside network so that interests may be answered by nearby routers with cached data (see Figure 2). NDN cache also reduces the server load because most of the interests would not even reach the server. In the meantime, PIT aggregates interests with same prefix from different faces [5], and forwards only one interest towards the control server. This could greatly reduce the interest implosion towards the control server, which reduces both server load and traffic concentration in network.

Type 2 message exchanges usually include a two-step process. For a participant to submit anything to the conference control server, NDN requires the control server issue interests first. We let the participant take the control and notify the control server when it is ready to submit. This is the first (trigger) interest A in Figure 3.

It does not matter whether the control server acknowledge the trigger interest A as long as the server sends out the interest B. If the participant did not receive the interest B after a predetermined time period, it will resend the trigger interest A.

Many control operations involve multiple steps and consist of combination of Type 1 and Type 2 message exchanges. For example, the registration operation can include three steps, as shown in Figure 4.

1. request registration form (Type 1)
2. submit registration form (Type 2)
3. request registration confirmation (Type 1)

Often times, an optimization can be obtained when a Type 1 message exchange is followed by a Type 2 message exchange. For example, in Figure 4, after receiving an interest from Step 1 (Type 1), the control server would expect a registration form submission (Type 2). In addition to sending back a registration form, the control server can start Step 2 by sending the interest B without waiting for the trigger interest A.
operation of request C-Info update. This requires a typical Type 1 message exchange with the name as /cluster/conf_name/C-Info.

Local Information or L-Info can include registrant list and Q&A permission-request list which are collected locally. Organizer and moderators collect L-Info by an operation of request registrant list or request Q&A permission list, typical Type 1 message exchanges, with a common name, e.g. /cluster/conf_name/L_Info, similar to the way the speaker list is obtained in [12].

Some servers maintain only C-Info. Those servers are relay servers helping to distribute only cluster information. A relay server does not handle registration or facilitate Q&A sessions.

The cluster join process is an operation of request cluster update. This is the same as sharing cluster information. A non-relay server may need an additional step, e.g. notify organizer to submit its name, a typical Type 2 message exchange with the organizer. The organizer may maintain a list of non-relay servers for administration purpose, or may collect registrant list directly from non-replay servers.

Throughout the conference, participants may interact with different servers at different times. For example, a participant may register with one cluster server and later interact with another cluster server with more servers joining. For example, in Figure 1, C4 may have registered with S1 before S2 joined the cluster. Later, C4 may request update from S2 because it is now closer than S1. If, at a later time, S2 failed and/or overflowed, or link to C4 failed, C4 may then retrieve conference updates from S1.

The server cluster is loosely structured with no membership or fix binding with participants. The cluster coverage grows with the footprint of the participants. NDN will ensure that the interest from a participant be routed to the nearest available cluster server. This approach provides robust and efficient conference control and management that is resilient to single point of failure with elastic scalability.

3) Server Cluster

Server-based approach in general inherits the problem of single point of failure. To address this problem, we extend single server to a cluster of distributed control servers (see Figure 1) and allow a dynamic and distributed cluster management.

A cluster is formed when a conference organizer registered with a control server. That server became the first server of the cluster. A non-cluster server is triggered to join a cluster when nearby participants joined the conference. The cluster expands as more participants joined. For example, in Figure 1, S2 sees a request from C4 to /conf_service/conf_name to which it does not belong. That can trigger S2 to join the cluster /cluster/conf_name. The join operation is explained later.

Each cluster server maintains Cluster Information or C-Info shared with all other cluster servers and Local Information or L-Info useful only to organizer and moderators. For example, C-Info can include conference agenda and updates. The cluster server can use /cluster/conf_name to share C-Info by an
C. Implementation

We have experimented with XMPP based conference applications with CCNx to study our hybrid design. We employed an audio [2] and a whiteboard [1] application from MobileSocial Lab [11]. Both applications are built on a Junction platform [1] which uses an open source XMPP server Openfire [9] for conference control. We have ported the media plane of both audio and whiteboard to CCNx [10] on the Android platform.

For audio application, voice data is captured using Android API AudioRecord with mono channel configuration and 16 bit PCM encoding. These raw data is saved in a buffer and then passed into byte CCNx API. In the whiteboard application, each stroke on the touch screen is converted into a text string, which is wrapped around JSON object and XML stanza. Then it is passed down to string CCNx API.

For control plane, XMPP’s message stanza can be used for message exchanges. For example, in XMPP, Alice requests the roster information from the server by specifying the following URI in its XML stanza:

\[
<iq from='alice@ucsc.edu/whiteboard' id='id' type='get'>
\]

the server replies with the following URI:

\[
<iq id='id' to='alice@ucsc.edu/whiteboard' type='result'>
\]

In NDN, the above URI can be translated as follows. Alice sends an interest to the conference control server of name prefix of conf_service:

\[
/conf_service/conf_list/get
\]

and the server responds with the following prefix:

\[
/alice/conf_list/result
\]

Given the complexity of Openfire implementation, we have yet to have time finishing the control plane in CCNx. Future implementation will convert control plane to CCNx.

IV. Experiments and Discussion

Figure 6 presents a demo of our CCNx based whiteboard on Android phones. We also have deployed both audio and whiteboard applications on our lab CCNx test-bed shown in Figure 7.

For audio application, the voice quality is audible and between moderate and good. With some preliminary tests, we recorded the average audio packets per second on each phone and compared the results of our CCNx port against the original implementation over IP from [2]. Figure 8 showed that the number of packets delivered through CCNx is slightly lower than the original application through IP. It is probably due to the overhead that current CCNx is running overlay over IP. This is encouraging that NDN can probably performance no worse than IP if not better when the overlay implementation is removed.

Figure 6 Whiteboard demo

Figure 7 CCNx test-bed for conference applications on Android platform for phones, tablets and smart devices.

Figure 8 The CCNx implementation is slightly under performed than stock application in IP.
It is said that one Openfire server can probably support 50,000 concurrent users [9], which is already sufficient to serve small size conferences with single server. With our participant-driven, distributed server cluster, it can support dynamic server cluster deployment, ideal for hosting business model.

We believe that the control traffic should be quite small, because most control plane operations usually occur once. Only the conference update and cluster update require periodic pulling, and we can reduce the update frequencies to the minimum without breaking the control logic. We plan to analyze the impact that the amount of the control traffic might have on control logic, either through real implementation or simulation.

Security and trust can also play an important role in conference control and management. It is orthogonal to our design and can be integrated in further studies.

V. Conclusion

NDN enables a content-centric communication model for the needs of today's and future Internet applications. NDN supports native multicast and caching inside network which will provide better supports for conference applications than in IP networks. We have proposed a hybrid approach for conference applications in NDN networks. This approach combines a participant-driven, distributed server-based approach for conference control and management with a server-less approach for media forwarding. The server-less media forwarding helps to reduce the traffic concentration and congestion in the network. The participant-driven, distributed server-based approach provides robust and efficient conference control and management that is resilient to the single point of failure and can scale to any size of conferences. We have carried out experiments with XMPP based conference tools over CCNx and have successfully built the media forwarding for two conference tools (audio and whiteboard) on Android platform. Our hybrid approach leverages NDN to support robust and efficient conference applications.

Acknowledgments

We would like to thank Lixia Zhang of UCLA for extensive and insightful discussion, and T.Y. Huang and B. Dodson of Stanford MobiSocial Lab for providing Android conference applications. We are grateful to Albert Tzeng and Soundar Ramaswamy for their enormous contribution with implementation, system setup and testing.

References

http://www.igniterealtime.org/projects/openfire
http://www.igniterealtime.org/about/OpenfireScalability.pdf