Design and Evaluation of a Multi-source Multi-destination Real-time Application on Content Centric Network

Asit Chakraborti, Syed Obaid Amin, Aytac Azgin and Ravishankar Ravindran
Huawei Research Center, Santa Clara, CA, USA.
{asit.chakraborti, obaid.amin, aytac.azgin, ravi.ravindran}@huawei.com

ABSTRACT

Multi-source multi-destination class of applications can range from interactive AR/VR games and high-definition video conferencing to non real-time file sharing applications; and also includes similar applications in other environments including IoT and data center. These applications require efficient synchronization among the end points with varying level of loss and delay tolerance. Also application specific requirements like throughput, end-to-end latency and mobility of end points impose even more stringency. IP based solutions often fail to meet these requirements due to lack of efficient multicast. In this paper we study this problem in the context of multi-party video conferencing system, and propose an information-centric solution based on content-centric networking (CCN). CCN, with its innate support for multicast delivery and service centricity, is used to deliver content and control state in a secure and reliable manner. We address the challenge of synchronizing participant states for a real-time experience through a media agnostic notification framework aided by service functions in the CCN network. Then, we implement a proof-of-concept testbed appropriate for this architecture and demonstrate its impact using scalability, QoE and reliability metrics.

1. INTRODUCTION

Peer-to-peer architectures for content distribution has been defined and classified in [1], furthermore there are commercial solutions employing them as demonstrated in [2]. In this class of applications, file sharing is the most common use-case and the scenario could vary from synchronizing files between multiple devices of a user (or synchronizing data across a disruptive network) to the distribution of software updates and patches to vehicles.

While users of file sharing applications can tolerate longer synchronization latency, more stringent requirements are posed by participants who collaborate using real-time media elements which involve audio, video or whiteboard sessions. AR/VR applications have been gaining popularity in the recent past and many of them require social or virtual entities to interact with multiple other entities, bringing in the aspect of extreme network bandwidth requirements making the problem of synchronizing application states even more challenging. This trend suggests an increasing demand for such advanced services that will necessitate the use of a more efficient and robust networking architecture as the current Internet lacks a general support for multicast and large scale content distribution features.

We define Multi-source Multi-destination (MSMD) as a class of applications where, a set of end points are simultaneously consuming and producing data, while trying to achieve a common set of application objectives. The main underlying problem shared by the MSMD applications is the need to efficiently synchronize the data being produced among all the interested consumers using minimal service and forwarding plane overhead, while adhering to application requirements, as some applications would be intolerant to data loss, while others such as real time applications could tolerate it to a certain degree. For instance, the video conferencing participants typically require voice packets to be synced within 150ms [3, 4], and this becomes an order smaller, for participants in a virtual reality (VR) session. We have selected the case of MSMD video conferencing as our use-case as it combines the essence of most of these applications.

There are many commercial audio/video conferencing systems (such as WebEx, Gotomeeting, Google-Hangout) that leverages the cloud infrastructure to offer their conferencing services. While cloud-based platforms allow these providers to multiplex virtualized resources efficiently, these architectures continue to carry drawbacks such as: (*i*) being a centralized architecture, they are prone to single-point of failure; (*ii*) centralized media processing limits number of participants per session, especially for video sessions. High availability can be achieved by using load balancers and hot-standby replicas, but introducing them can be complex for a service like video conferencing. To circumvent the core compute and bandwidth resource issues providers often use techniques resulting in suboptimal user experience: (*i*) participants are constrained in views they can choose (*i.e.*, one high-quality video and multiple thumb-view videos); (*ii*) data streams from multiple sources are first combined into one stream by a centralized entity and then sent to participants. Mainly through restrictions imposed on conference sessions as agreed by service provider and users, compute and bandwidth resources can be kept under control so that the system can scale linearly in number of participants.

Earlier studies have shown the scalability limitations for IP-based conferencing systems (*e.g.*, [5, 6]). While peer-to-peer solutions avoid some of the challenges faced by the centralized approach, they are discouraged by the lack of easily deployable dynamic IP multicast enabled across multiple domains that can optimize network bandwidth usage; also these solutions offer poor design choices towards enforcing trust and security [7], or handling mobile end points.

To address these problems and others, future networking architectures with focus on content delivery have been considered to replace the current IP [8]. For instance, conferencing applications can leverage Information-Centric Networking (ICN) features such as in-network multicast, caching, and computing [9] to offer improved services. These benefits are visible even in an overlay deployment mode, as

978-1-5386-4870-4/18/$31.00 ©2018 IEEE

demonstrated through this work. Among the existing proposals for ICN, a popular architecture that has gained significant attention is *Content Centric Networking (CCN)* or *Named Data Networking (NDN)*[1] architecture [10], which aims to replace the current IP's host-centric design with a content-centric one. CCN utilizes a *pull-based* content delivery framework with Interest/Data primitives to efficiently support non-realtime services like video streaming [11, 12]. However, the stringent latency requirements associated with real-time applications bring forth additional challenges for CCN, as they seek *push-based* network services. Furthermore, random join or leave by a participant requires actively syncing a producer's state with each consumer, while taking into account the QoE requirements for the participants. Additionally, as consumers can go out-of-sync with producers during an active session, we need mechanisms to handle such scenarios to re-sync user states. Existing ICN-based approaches (such as [13]) that address these challenges to some extent typically lack a large scale study in regards to quality of experience for real-time MSMD communications.

This paper proposes a scalable and reliable video conferencing solution over CCN leveraging its ability to multicast content and host services in the network. The service in the network provides a generic notification framework to synchronize producer and consumer states. We realize our solution over a programmable CCN network that is controlled by a CCN aware network virtualization framework, which allows it to be amenable to edge deployment, and thereby addressing the short-term deployment challenges for ICN. Our contributions are listed as follows:

- We propose a CCN based conferencing architecture that takes advantage of CCN capabilities at the network layer (such as multicasting) and combines it with an in-network service framework, that not only allows for efficient multicasting of MSMD content but also allows for offloading complex system components from user entities to the network, to achieve a scalable and resilient solution.
- We present an in-depth study of the proposed architecture in order to evaluate its unique features and understand its operation. We utilize a large-scale performance study to investigate the impact of various MSMD communication scenarios that demonstrate its robustness, while addressing potential bottlenecks critical to future adoption of similar schemes.

2. CONFERENCE SYSTEM DESIGN

We show in Figure 1 the proposed conferencing architecture, which is illustrated in the context of an overlaid CCN deployment. Here we assume an operator or an enterprise manages a set of CCN Service Routers (CSRs) that are strategically placed at the network edge. These edge nodes are capable of hosting applications, in our case the conferencing service components. The end hosts (or user entities, UEs) simultaneously operate as both producer and consumer, and require discovery of the service components hosted by the CSRs to connect to and participate in the conference sessions.

2.1 Service Orchestration

[1]Hereafter, we use `ICN`, `CCN`, and `NDN` terms interchangeably.

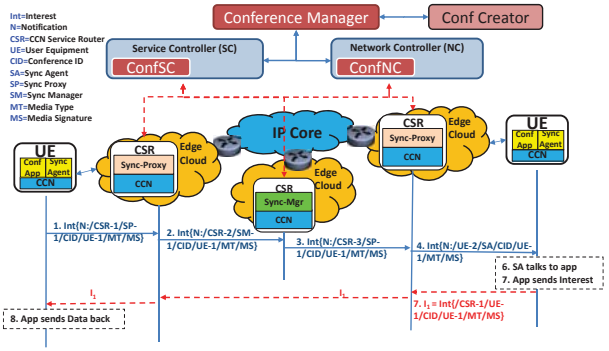


Figure 1: Conferencing System Architecture.

The management of the compute and bandwidth resources is performed by the service orchestration layer, that leverages concepts from SDN and NFV, and comprises of a *Conference Manager, Network Controllers* and *Service Controllers*. *Conference Manager (CM)* is a logically centralized entity in a control plane that analyses conference requirements (*e.g.*, bandwidth, computing) during provisioning and determines the number and locations of the conference service components. The CM can also play important security functions, like managing group keys for each session in order to establish provenance, integrity and confidentiality of the producer's content.

Service Controller (SC) manages the compute virtualization of the CSR resources, by monitoring the compute resource usage and provisioning service functions accordingly at the CSRs. The *Conference Service Controller (ConfSC)* has direct knowledge of the conference service specific components and it monitors and provisions them, reacting as necessary to events (*i.e.*, failure or request to provision more instances from the CM). The *Network Controller (NC)* adapts the underlying network for the name based applications, and also performs the overlay bandwidth management to serve the hosted application network controllers. The *Conference Network Controller (ConfNC)* manages the virtual topology for each conference session, i.e. dynamically programming the ICN network reachability, inter-connecting the service components and connecting the UEs to them using named flows. It also handles UE join/leave events for conference sessions. We use OpenStack [14] and ONOS [15] to enable these functions.

2.2 Notification (or Name-Sync) Framework

Notification framework is used to sync the state of a producer's media namespace with participants in a given conference session's context. Figure 1 also shows the notification framework, which consists of three main functional components: *Sync-Agent*, *Sync-Proxy*, and *Sync-Manager*. Sync-Agent is hosted at the UE and interacts with UE's producer application, which has multiple media streams to synchronize. Similarly, at the other end, the Sync-Agent is responsible for delivering all the updates in the form of notifications to the consumer application.

Sync-Proxies and Sync-Managers are hosted at the CSR and provisioned as virtual service components by the SC to concurrently handle multiple conference sessions. Sync-Proxy participates in conference management by notifying the ConfNC whenever a participant registers or un-registers with it.

Sync-Proxy is also responsible for managing the local notification state of each hosted conference session by keeping a history of updates received from the hosted Sync-Agents and the Sync-Manager. This state can be used for recovery from disruptions as discussed in [16]. Such a notification state is also managed by the Sync-Agent and the Sync-Manager. Sync-Manager is responsible for managing the notifications for the hosted conference sessions by relaying notifications among the distributed Sync-Proxy instances in a hub-and-spoke manner. Anytime the Sync-Manager receives an update from a Sync-Proxy, the update is pushed to the remote Sync-Proxies. The forwarding mechanism used for the notifications by the SC can be smart to the extent that notifications are only pushed if there is a need for them at a given Sync-Proxy.

The frequency at which the notification mechanism operates depends on the media type. In our solution, text-driven notification is for every chat text committed by the participant, whereas real-time content (audio, video) driven notification is provided at a configurable periodic interval (to manage the associated overhead). In the case of video content, Sync-Agent sends notifications per multiple *group-of-pictures* (GOP) intervals. For example, if the GOP consists of 25 frames, notifications can be sent for every key frame, *i.e*, for every $[25k, 25(k+1))$ frames, where $k \in Z^+$.

2.3 Naming

In the proposed conferencing framework, we define two namespaces: (*i*) *content* (or *data*) *namespace*, which is used for user data exchange, and (*ii*) *notification namespace*, which is used for the name-sync protocol. Data namespace (or names, for brevity) correspond to the media content generated by the UE, and they are prefixed by the CSR, which the UE connects to, and follows the basic format of */<CSR-ID>/<CID>/<UE-ID>/<MT>/<[MS]>*. CID (Conference-ID) is a unique ID corresponding to the given conference session. UE-ID identifies the producer application. MT (Media-Type) identifies the type of content, which is either audio, video, or text. MS (Media-Suffix) includes Media-Type specific sub-components (for example, <frame-id> and <chunk-id> for a video segment where a video frame has been broken into multiple chunks). Note that the proposed architecture is not limited to static hosts, the naming convention can be modified to support mobile UEs as proposed in [17].

Notifications are point-to-point messages exchanged between the service functions Sync-Agent, Sync-Proxy, and Sync-Manager and follow the convention of */<CSR-ID>/<UE-ID>/<SFN>/<[Fingerprint]>*. SFN (Service Function Name) identifies the service type. The Fingerprints are derived from the application components of a UE's data name that uniquely identifies the Conference-ID, UE-ID, Media-Type, and Media-Suffix. The information contained in the fingerprint enables consumers to accurately track the status of a producer.

2.4 Conference Data Plane

As the CCN protocol is typically optimized for non-realtime content delivery services, services with stringent latency requirements (*i.e.*, $\leq 150ms$ and $\leq 250ms$ for audio and video services, respectively [3], in addition to the relative audio/video sync requirement of $+45ms$ to $-125ms$ [4]) cannot be supported efficiently by applications running on a pull-based framework of CCN.[2] Considering the strict QoE requirements for the realtime conference service, proposed architecture requires the consumer to issue proactive requests for content from the producer, with guidance from the notification framework. To properly handle the proactive content requests, producer implements an application layer buffer to hold the proactive Interests and to enable the content to be pushed to consumers as soon as a content object is generated. We next provide a high level functional design of the producer and consumer nodes with reference to Figure 2.

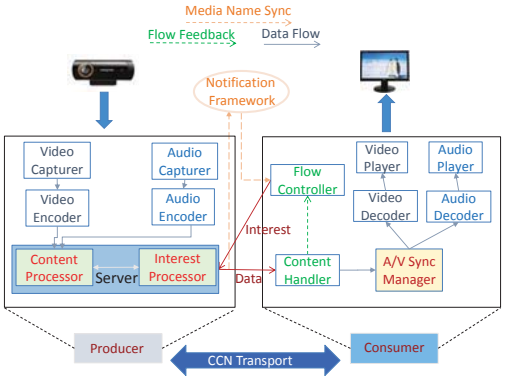


Figure 2: Producer and Consumer architectures.

At the producer side, video-encoder, which receives the live content from the video-capturer, provides the application with the encoded key/delta frames. Encoded frames are then chunked, appended with video metadata, named and published at the server-module's content-processor for consumption. The metadata comprises of *current frame index, frame type and number-of-chunks. Current frame index* and *frame type* provide application level sequencing of packets; while the use of *number-of-chunks* is discussed later in the consumer design section. With this design, an explicit per-frame manifest is not required, and a manifest with coarser granularity will perform poorly in a real-time environment. This server-module also includes an interest-processor function, which holds application-level Pending Interest Table (A-PIT) that stores the pro-active Interests. As soon as it is produced, the content is matched against the A-PIT to satisfy the consumer requests. We use a similar pipeline for the audio content, except that, as the payload size per frame typically varies within the $10-50$B range, the frames do not need to be broken down into chunks.

At the consumer side, the process to display a participant's video is initiated by the Sync-Agent through its communication with the Sync-Proxy, as part of the notification framework. In doing so, Sync-Agent acquires information on the producer's latest video/audio name state, which is sufficient for the consumer to start expressing Interests and sync with the real-time content generated by the producer.

The flow-controller in the consumer is responsible for issuing an estimated set of Interests periodically to fetch the real-time content. Here, one of the challenges is to correctly determine the variable number of chunks corresponding to a video frame as video frame size changes with the activity level of the remote participant. We address this problem using the frame descriptor metadata embedded by the producer in each video chunk, which is extracted and provided

[2]Latency requirement for text chat is much less stringent and is in the order of $\leq 1s$.

to the *flow-controller*, after receiving the chunks from the *content-handler*. The flow-controller can then issue additional Interests or clear out states for unneeded Interests locally. Another challenge in our design is to emulate a 'push' model over a 'pull' based architecture. To address this, the consumer expresses *proactive* Interests; these are issued before the corresponding content objects are generated. The proactive Interest expression process is guided using the notification messages from the notification framework. Here the challenge is to determine how far in future should these proactive Interests be: if the consumer is too aggressive it will result in wasted resources in the network and the producer; on the other hand if these Interests are issued not sufficiently in advance, content would be produced and not sent towards the consumer immediately, as there would be no pending interest in A-PIT.

During significant network disruptions that can last seconds, flow-controller starts from the latest content state learned by the notifications, while clearing out the outstanding Interests for the earlier video frames. If the content-handler module does not acknowledge the flow-controller in regards to receiving a given chunk, flow-controller may initiate retransmission to potentially recover from the network cache or directly from the producer. To support a smooth playout, a de-jitter buffer can be used by the content-handler. An A/V Sync-Manager module (AVSyncManager) can use the timestamp information from the audio/video frames to sync the audio/video playback before handing the content to their respective decoders.

3. PERFORMANCE EVALUATION

In this section, we evaluate the performance of the proposed architecture. We start by explaining the framework used for our analysis and then present the experimental results for the performance metrics considered.

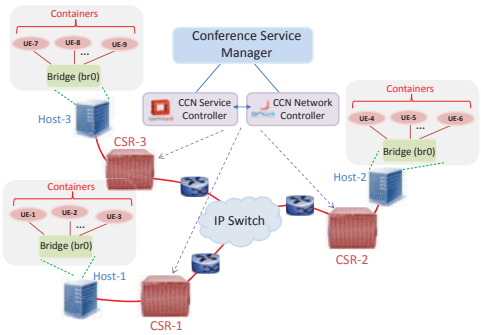


Figure 3: Emulation test bed topology.

3.1 Evaluation Framework

In [18] and [19] we presented a prototype of the system; with 4 to 5 participants only. In this paper, to study the system at reasonable scale of up to 40 participants, we instead use *head-less* producer and consumer designs, where the producer emulates generation of media frames, while the consumer emulates the process of receiving these frames without the playback phase. Data from our earlier prototype is used for audio and video traffic modeling [19]. The encoder was set to generate a GOP of 25 frames. The average size observed for *key-frames* was $15KB$, while for *delta-frames* was around 3KB. The size of each Data chunk was $3KB$ that results in 5 chunks for key-frames in average and 1 chunk for delta-frames. The consumer uses this information to maintain the size of Interest window. Any error in estimation is fixed by leveraging the metadata in the received Data packets. The main components for the conferencing system (*i.e.*, Sync-Manager, Proxy and UEs) are implemented in Java using jNDN library [20], while NDN Forwarding Daemon (NFD) is used to achieve name-based forwarding [21] on all network nodes. The main emulation testbed consists of 5 host machines and 3 CSRs overlaid over the IP network. To keep it legible we show the topology at a reduced scale with 3 hosts only in Figure 3. UEs are implemented over Linux Containers [22] and multiple of them can be placed on any of the host machines. To emulate the edge deployment for the CSRs, we set 10ms latency on UE-to-CSR links and 5ms latency on CSR-to-CSR links, depicting a relatively faster backbone connection. Our testbed uses a mix of host machines, ranging from lower-end i5 to higher-end i7s.

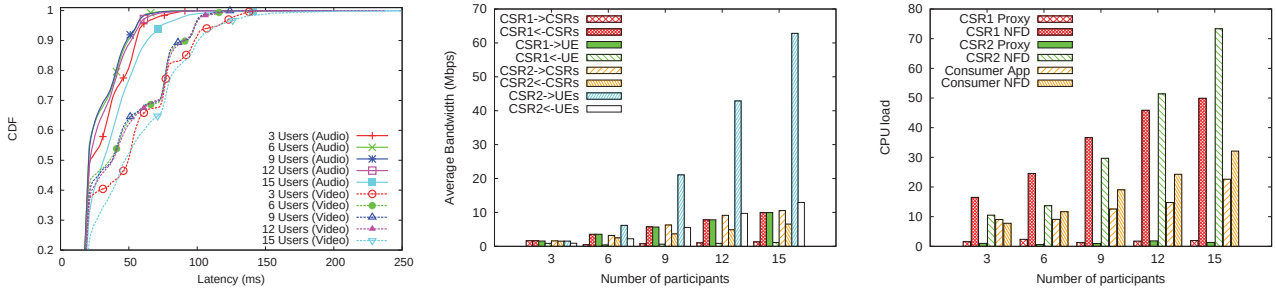
In this section, our primary goal is to demonstrate the scalability and reliability of the proposed MSMD conferencing architecture, by focusing on two main scenarios: (i) all nodes are both consumer and producer (*NCon:NPro*) and (ii) single consumer multiple producer nodes (*1Con:NPro*). To study the network load, we use CPU and bandwidth utilization at CSRs. At the end-hosts, we measure application-level end-to-end latency and CPU utilization of NFD and the conferencing application.

3.2 Latency Evaluation

In Figure 4(a), for the NCon:NPro scenario, we illustrate the end-to-end latency performance for both the audio and video streams from the point of view of a single consumer as we increase the number of participants. Here, latency represents the one way delay from the producer to the consumer, with respect to the audio or video frame generation time up until when the frame is ready for decoding at the consumer side. For both audio and video content, as we increase the number of participants from 3 to 15, the latency values for more than 99% frames stay below the critical threshold based on our definition of service quality (*i.e.*, less than 150ms for the audio stream and 250ms for the video stream). However, after 15 nodes, we observed significant increase in the latency and packet loss. This is because the number of flows served by a CSR towards local UEs grows quadratically as we increase the number of users and single-threaded NFD-based forwarder fails to scale well to the increase in traffic. This issue is also discussed further in section 3.3. We also observe that the latency for the video frames is comparatively higher than that of audio streams. This is because, the video frames may involve more chunks than our estimation of future frames, resulting in a second phase of data retrieval. We also observe that in case of 3 UEs the consumer observed higher packet latency than $6, 9$ and 12 UEs as it was not able to leverage the caching/aggregation feature provided by ICN and most of the Interests are satisfied by the Producer, not by intermediate caches.

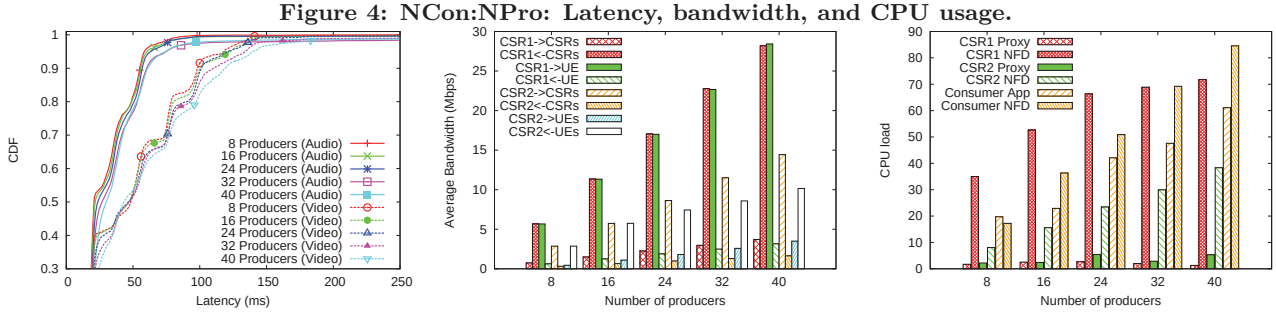
3.3 Bandwidth Utilization

This section discusses the bandwidth utilization as measured by the IP traffic at different network components. For the NCon:NPro scenario, as the contents generated by a client are first transmitted to its servicing CSR, and from there unicast to the other CSRs servicing the other active



(a) Audio/video latency performance (b) Bandwidth utilization (c) CPU utilization

Figure 4: NCon:NPro: Latency, bandwidth, and CPU usage.



(a) Audio/video latency performance (b) Bandwidth utilization (c) CPU utilization

Figure 5: 1Con:NPro: Latency, bandwidth, and CPU usage.

clients, we can approximate the bandwidth utilization at the ith CSR node using the following equations:

$$W_{\mathtt{UE}\rightarrow\mathtt{CSR},i} = w_{\mathtt{UE}}^{(\mathtt{I})} \times \kappa_{\mathtt{CSR},i} \times (\kappa_u - 1) + w_{\mathtt{UE}}^{(\mathtt{D})} \times \kappa_{\mathtt{CSR},i} \quad (1)$$

$$W_{\mathtt{UE}\leftarrow\mathtt{CSR},i} = w_{\mathtt{UE}}^{(\mathtt{D})} \times \kappa_{\mathtt{CSR},i} \times (\kappa_u - 1) + w_{\mathtt{UE}}^{(\mathtt{I})} \times \kappa_{\mathtt{CSR},i} \quad (2)$$

$$W_{\mathtt{CSRs}\rightarrow\mathtt{CSR},i} = w_{\mathtt{UE}}^{(\mathtt{D})} \times (\kappa_u - \kappa_{\mathtt{CSR},i}) + w_{\mathtt{UE}}^{(\mathtt{I})} \times \kappa_{\mathtt{CSR},i} \times (n_{\mathtt{CSR}} - 1) \quad (3)$$

$$W_{\mathtt{CSRs}\leftarrow\mathtt{CSR},i} = w_{\mathtt{UE}}^{(\mathtt{D})} \times \kappa_{\mathtt{CSR},i} \times (n_{\mathtt{CSR}} - 1) + w_{\mathtt{UE}}^{(\mathtt{I})} \times (\kappa_u - \kappa_{\mathtt{CSR},i}) \quad (4)$$

where $n_{\mathtt{CSR}}$ represents the number of CSR nodes, $\kappa_{\mathtt{CSR},i}$ represents the number of clients serviced by $\mathtt{CSR}_i$, κ_u represents the total number of clients serviced by all CSRs (for the given session), $w_{\mathtt{UE}}^{(\mathtt{I})}$ represents the average generated interest stream rate towards a client, and $w_{\mathtt{UE}}^{(\mathtt{D})}$ represents the average per-client generated data stream rate. Here, for the sake of simplicity, clients are assumed to generate traffic streams at similar rates, however, our analysis can be easily extended to cover the case for differing stream rates among clients.

Our results, as shown in Figures 4(b), match closely with the theoretical analysis provided above. Specifically, we observe that increasing the number of clients results in a linear increase in bandwidth use for the CSR-to-CSR links showcasing the benefit of name based aggregation in CCN. The increase in bandwidth use is more pronounced on the CSR2-to-UE links for the NCon:NPro scenario. This is because, CSR2 has to unicast all the streams to all its clients, hence no further downstream aggregation is possible, as shown in Equation 2. The impact of Interest aggregation is more visible in scenarios with higher loads. For instance, for the 15-UE case, there are 7 UEs attached to CSR2, each of which is sending Interests to 14 other UEs, resulting in 98 Interest flows. However, from CSR2 to other CSRs there would be only 14 Interest flows due to Interest aggregation, which provides significant savings in the upstream bandwidth.[3]

[3]Note that, we also observed that the notifications overhead is less than 2% of the overall overhead for all the considered scenarios, limiting its impact on the overall performance.

3.4 CPU Utilization

We illustrate the average CPU utilization performance in Figures 4(c) at the UE and CSRs, where we normalized the values with respect to CPU cores. For the NCon:NPro scenario, we observe that the CPU use for the UE application and forwarder processes grows linearly. CSR1 serves one client and its CPU use is manageable, but CSR2 serves an increasing number of clients resulting in much higher growth for the CPU usage percentage. The reason for such increase can be explained by the unicast transmissions towards the local participants, which also matches the bandwidth use trend we observed earlier, making this CSR the bottleneck for our setup. However, if we use a multi-threaded hardware deployment for the NFD to balance the load of UEs on the CSRs, the scalability performance of the system can be significantly increased.

3.5 Consumer Scalability

To study the consumer UE scalability i.e. number of participants it can support, we conducted different set of experiments under the 1Con:NPro scenario. For this setup, the consumer UE was attached to CSR1 while the producer UEs were distributed to CSR2 and CSR3. By doing so, we removed the bottleneck caused by content replication at the CSRs. In Figure 4(a) we show the one-way latency observed by the consumer. We notice that up to 40 producers, the delivery rate (within play-out deadline) stays close to 100%. The bandwidth utilization performance, which is shown in Figure 5(b), also suggests a linear growth, as we increase the number of producers. This is mainly because the number of replicated streams at a CSR depends on the consumer attached to it. In presence of only one consumer, there is no replication of data streams, hence the growth is linear. Eventually, the CPU usage at the consumer side by the

UE forwarder becomes the bottleneck in this case, as shown in Figure 5(c). Further improvements are possible by taking better advantage of multi-threading or utilizing lock-free data structures, in addition to facilitating more efficient resource utilization schemes. However, such optimizations are currently outside the scope of our research. Please note that due to discrepancy in hardware specifications, CPU load at different network nodes cannot be directly compared with each other. For instance, in Figure 5(c) CSR1's NFD shows more CPU load than that of CSR2, even though CSR1 only serves one consumer. This is because, CSR1 is running on a $1.5GHz$ Intel processor, while CSR2 is running on a high-end $2.99GHz$ Intel processor.

3.6 Recovery Performance

In Figure 6, we illustrate the benefits of caching and the impact of packet loss on end-to-end latency performance for audio and video frames. For this study, we used a simplified star topology with 4-UEs connecting to a single CSR. Packet loss events are triggered using the netem on the link that connects a UE to a CSR. We observe that even for a real-time application with strict end-to-end latency requirements, it is possible to recover and play out a large percentage of these packets using retransmission, which demonstrates a significant advantage for using CCN, as almost all of these retransmitted packets are fetched from a nearby CSR cache. Here, the TCP-like *Interest Retransmission TimeOut* (IRTO) estimation, which changes dynamically with the network state, has an obvious impact on the playability of the recovered content objects. Lower IRTO value may result in a higher recovery rate, but at the cost of increased overhead, due to higher number of overall retransmission requests and possible cache misses, as the requested content might not have been received by the servicing CSR.

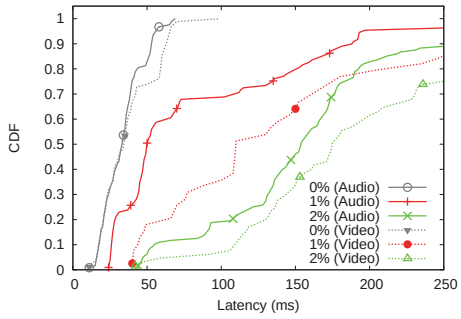


Figure 6: Latency of contents retrieved by *request retransmissions* after IRTO at different packet loss rates.

4. RELATED WORK

Real-time communication over ICN has been studied in VoCCN [23], ACT [24] and NDN-RTC [13]. The first two dealt with two-party and multi-party audio-only communication, and established the feasibility of real-time communications in ICN, while pointing out the benefits in terms of scalability, robustness and security over similar IP based solutions. NDN-RTC addressed the additional complexities for generating, publishing, and consuming video content, and provided novel approaches to minimize the latency using a pull based communication framework. However, the MSMD sync features offered by NDN-RTC continue to rely on complex heuristics implemented at the consumer end points, such as measuring response time for Interests during bootstrap phase to determine if the consumer is in sync with the producer state. These works fail to show promising results when it comes to scaling the solution to a large number of participants. Conversely, our solution pushes many of these features (such as discovery and sync) towards the network as in-network services to reduce the end host complexity, which allows for better scalability and reliability.

5. CONCLUSION

In this paper, we proposed a scalable multi-source multi-destination content distribution architecture capable of handling real-time communication challenges such as random join or leave events and fast session recovery after intermittent drops. We utilized an application agnostic notification framework that leveraged the service-friendly CCN transport to support efficient name-sync among participants. We provided an in-depth discussion of the proposed architecture and evaluated its performance using a realistic CCN-based emulation framework by studying the key performance metrics including bandwidth, latency, computing and resilience to packet loss. We demonstrated its capability to effectively handle more than 40 same session participants to support not only multi-party conferencing but also multi-source streaming towards, for instance, efficient realization of Augmented/Virtual Reality (AR/VR) conferencing system using CCN.

6. REFERENCES

[1] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4):335–371, December 2004.
[2] Use Cases Archives - Resilio Blog, https://www.resilio.com/blog/category/usecases.
[3] Yan Chen, Toni Farley, and Nong Ye. QoS requirements of network applications on the Internet. *IKSM*, pages 55–76, Jan 2004.
[4] ITU-R BT.1359-1 1, `https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.1359-1-199811-I!!PDF-E.pdf`.
[5] Y. Lu, et al. Measurement study of multi-party video conferencing. In *IFIP Networking*, pages 96Ű–108, 2010.
[6] Y. Xu, et al. Video telephony for end-consumers: Measurement study of Google+, iChat, and Skype. *IEEE/ACM ToN*, 22(3):826–839, June 2014.
[7] A Survey of Peer-to-Peer Network Security Issues, `http://www.cse.wustl.edu/~jain/cse571-07/ftp/p2p/`.
[8] Jianli Pan, Subharthi Paul, and Raj Jain. A survey of the research on future Internet architectures. *IEEE Communications Magazine*, pages 26–36, Jul 2011.
[9] B. Ahlgren, et al. A survey of information-centric networking. *IEEE Communications Magazine*, pages 26–36, Jul 2012.
[10] Van Jacobson, et al. Networking named content. In *CoNEXT'09*, pages 1–12, New York, NY, USA, 2009. ACM.
[11] S. Lederer, et al. Adaptive streaming over content centric networks in mobile networks using multiple links. In *IEEE ICC'13 IMC Workshop*, pages 677–681, Jun 2013.
[12] Y. Liu, et al. Dynamic adaptive streaming over CCN: A caching and overhead analysis. In *IEEE ICC'13*, pages 3629–3633, Jun 2013.
[13] Peter Gusev and Jeff Burke. NDN-RTC: Real-time videoconferencing over named data networking. In *ACM ICN'15*, pages 117–126, New York, NY, USA, 2015. ACM.
[14] OpenStack. `https://www.openstack.org/`.
[15] ONOS. [Project Website]: `http://www.onlab.us/`.
[16] R. Ravindran, et al. Towards software defined icn based edge-cloud services. In *IEEE CloudNet'13*, pages 227–235, Nov 2013.

[17] A. Azgin, et al. Seamless producer mobility as a service in information centric networks. In *ACM ICN 2016, IC5G Workshop*, 2016.

[18] A. Jangam, et al. Realtime multi-party video conferencing service over information centric network. In *IEEE ICMEW'15*, pages 1–6, Jun 2015.

[19] Asit Chakraborti, et al. Icn based scalable audio-video conferencing on virtualized service edge router (vser) platform. In *ICN'15*, pages 217–218. ACM, 2015.

[20] J Thompson and A Brown. A named data networking client library for java.

[21] Alexander Afanasyev, et al. Nfd developer's guide.

[22] Linux Containers (LXC), `https://help.ubuntu.com/lts/serverguide/lxc.html`.

[23] Van Jacobson, et al. VoCCN: Voice-over content-centric networks. In *ReArch'09*, pages 1–6, New York, NY, USA, 2009. ACM.

[24] Z. Zhu, et al. ACT: Audio conference tool over named data networking. In *ACM SIGCOMM Workshop on ICN*, 2011.