NDN, Technical Report NDN-0008, 2012. http://named-data.net/techreports.html
Revision 1: October 10, 2012

Chronos: Serverless Multi-User Chat Over NDN

Zhenkai Zhu*, Chaoyi Bian†, Alexander Afanasyev*, Van Jacobson‡, and Lixia Zhang*
* {zhenkai, afanasev, lixia}@cs.ucla.edu, UCLA, Los Angeles, California, USA
†bcy@pku.edu.cn, Peking University, Beijing, China
‡van@parc.com, PARC, Palo Alto, California, USA

Abstract—Multi-user applications are commonly implemented using a centralized server. This paper presents a new design for multi-user chat applications (Chronos) that works in a distributed, serverless fashion over Named Data Networking. In Chronos, all participants share their views by exchanging the cryptographic digests of the chat room data set. A newly generated message causes a change of the digest at the message originator, which leads to retrieving the new data by all other participants in an efficient way and resynchronization of chat room views. Chronos does not have a single point of failure and eliminates traffic concentration problem of server-based implementations. We use simulations to evaluate and compare Chronos with a traditional server-based chat room implementation. Our results demonstrate Chronos' robustness and efficiency in data dissemination. Chronos' approach of replacing centralized servers by distributed data synchronization can be applied to a variety of distributed applications to simplify design and ease deployment.

1 Introduction

Multi-user chat is widely used to share information via text with a group of users. Traditionally, the design of such applications is centralized in nature: a single server is used to host a chat room and maintain a roster of participants in the room. Users send text messages to the server through TCP connections, and the server mirrors all the received messages to all other users on the roster except the sender.

A 1000-foot view of such a system represents a class of distributed (semi) realtime applications, such as file synchronization, collaborative editing, and audio conferencing. These applications share a common requirement that a group of distributed participants must maintain an identical view of shared data set all the time. Most of the existing applications use a central-server based implementation, where every participant synchronizes its data with the centralized server. However, such designs lead to traffic concentrations at the server and make the applications vulnerable to single point of failure. Furthermore, because participants may be distributed over wide areas, no matter where the server is placed, at least some data packets are subject to triangle paths between producers and consumers. Deploying multiple distributed servers can help alleviate the above problems, but it also introduces a set of new problems to be addressed, such as who would provide those servers, and how to keep data synchronization among the servers.

Can we achieve the goal of data synchronization for distributed applications without centralized server(s)? There are a number of synchronization algorithms [1]–[3], most of them addressing the problem of efficient data synchronization between a pair of nodes. Hence, one could achieve data synchronization among a set of distributed participants through a pair-wise, full-mesh synchronizations between every pair of nodes. However, it is unclear how a new participant may learn about all the existing participants in order to set up the pair-wise synchronization with them. Furthermore, letting every node communicate with every other node through unicast leads to inefficiency. A more practical way is to let the participants share their views of the data set and exchange the actual application data directly once they figure out the differences in their views, as suggested in [4], [5], both done in a multicast fashion. However, the designs in [4], [5] either report a full-list of statuses in order to figure out the difference, which may be costly, or have a limited view on how large the difference can be between the data sets.

In this paper, we evolve the distributed data synchronization idea to design a serverless multi-user chat application over Named Data Networking (NDN) [6].[1] In NDN, every piece of data has a unique name, thus one can represent a chat room's state as a collection of all the data names. Each client in a chat room can hash its collection of names to a cryptographic digest to be exchanged with other clients in the same room. In the steady state, everyone has the same digest. When a new piece of data is generated, it leads to a difference in the digests and notifies others of the new data. A good design of the naming rules facilitates participants to quickly figure out the names for the missing data; the actual data can then be fetched from the data producers directly and efficiently through built-in multicast data delivery in NDN.

We implemented Chronos as an open-source *serverless*

1. NDN project is one of the four flagship projects funded by NSF Future Internet Architecture program, aiming to transform the host-to-host communication paradigm of the current Internet to a data-centric, receiver-driven model in order to accommodate emerging demands on scalable data dissemination, mobility support, and secure communications.

multi-user chat package that runs over NDN and works with any existent chat client that supports XMPP protocol. We also adapted the implementation Chronos to NS-3 NDN simulator to evaluate Chronos' performance and compare it against that of traditional server based implementations.

Our contributions in this paper are two-fold. First, we explored a novel design of multi-user chat, which eliminates the need for a central server and demonstrated its robustness and efficiency through simulation evaluations. Second, we believe that the distributed multiparty data synchronization approach used in Chronos can serve as a first example for a class of simple and elegant solutions to various distributed applications running on top of NDN, such as collaborative editing, audio conferencing, and file synchronizations.

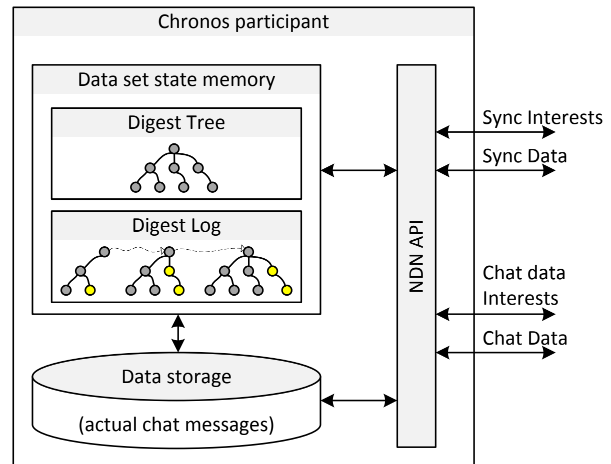


Fig. 1: Chronos design components

2 NDN Background

In this section we briefly go over a few basic concepts of NDN [6] that are essential to describe the design of Chronos.

Communications in NDN are driven by data consumers. Each piece of content is cryptographically associated to a data name. To retrieve data, a consumer sends out an *Interest* packet, which carries a name that identifies the desired data. A router remembers the interface from which the Interest comes in, and then forwards the Interest packet by looking up the name in its *Forwarding Information Base*, which is populated by routing protocols that propagate name prefixes instead of IP prefixes. Each received Interest is kept in a *Pending Interest Table (PIT)*. If additional Interest packets for the same data name are received, the router simply records the arrival interfaces in the PIT entry for the name. Once the Interest packet reaches a node with the requested data, a Data packet is sent back.

A *Data* packet carries the name and the actual data, along with a signature created by the original data producer that binds the two together. When a router sees a Data packet, it finds the matching entry in its PIT and forwards the packet to the interfaces where the corresponding Interests come from and removes the PIT entry. As a result of the Interest state that have been set up at the intermediate routers, the Data packet traces the reverse paths back to all the data consumers in a multicast fashion. Each Interest is kept in the PIT until the desired Data is retrieved, or until its time-to-live period expires. It is the data consumer's responsibility to re-express an Interest when it expires, assuming the Data is still desired then.

3 Chronos Design

3.1 Overview

Chronos design has two main components: data set state memory and data storage (see Fig. 1). The data set state memory maintains the current knowledge of the chat data set in form of digest tree, as well as maintains history of the data set changes in form of digest log.

Chronos participants interact using two types of Interest/Data message exchanges: synchronization (sync) and chat data. A sync Interest represents the sender's knowledge of the current chat data set in form of cryptographic digest, obtained using digest tree, and is delivered to every other participant. Any recipient of the sync Interest who detects that it has more information, satisfies this Interest with a Data packet that includes the missing part of the data set. Common state and knowledge difference discovery is performed using the digest log.

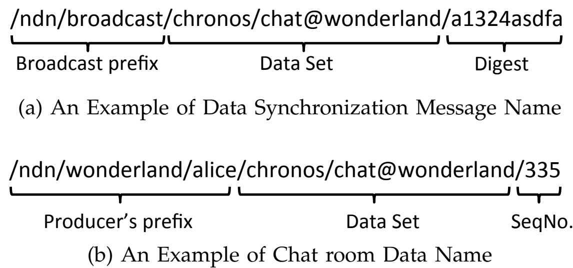
As soon as a participant discovers new knowledge about the chat room state, it sends out chat data Interests to pull actual messages from their originators. Chronos guarantees that a user can learn and infer all the names for the chat data produced by the participants of a room. However, as there is no central storage server, participants should setup or choose a persistent content storage for their data if they wish the data to be accessible after they get offline.

Chronos uses a soft-state method to manage the roster. A user is added to the roster when his presence message to the chat room is received. The participants periodically send "heartbeat" messages if they have no chat messages to send. If nothing is heard from a user for a certain amount of time, the user is no longer considered as a current participant of the chat room.

3.2 Naming Rules

Naming is an important aspect of any NDN application design. There are two sets of naming rules in Chronos. One set determines names for sync exchanges, while the other regulates the naming of actual chat data.

The name for a sync message starts with a broadcast prefix, following by the data set identifier, which represents the chat room, and the digest of the data set (Fig. 2a). Any Interest with a broadcast prefix reaches all the nodes within the broadcast domain, which is a



(a) An Example of Data Synchronization Message Name

(b) An Example of Chat room Data Name

Fig. 2: Naming in Chronos

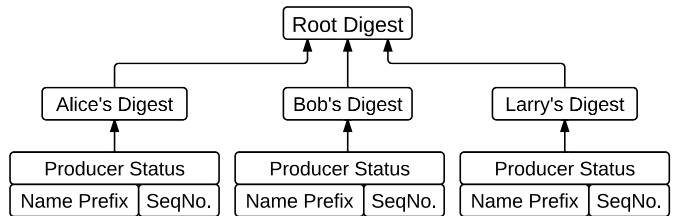


Fig. 3: An Example of Digest Tree

simple and effective way to share digests among all participants in a small network, because NDN suppresses identical Interests. If the participants are distributed over large networks, such as the Internet, simple broadcast may no longer be feasible. For clarity, we describe Chronos design using broadcast for Interests, and in Section 5 we discuss solutions to scale Chronos sync Interest distribution.

The chat data name consists of a unique prefix that identifies the producer and is used to efficiently route the Interest, the chat room identifier, and the message sequence number (Fig. 2b). The initial message sequence number is zero and whenever a participant generates a new message, be it a text message, a user status, or a heartbeat, the sequence number is increased by one. Different from the fixed-length sequence number field used in traditional protocols such as TCP, the sequence number in Chronos has variable length and no upper bound limit by virtue of NDN's flexibility in naming, effectively ruling out problems caused by sequence number wrapping.

3.3 Data Structures

It is critical to maintain an up-to-date view of the chat data in order to generate digest. Chronos uses digest tree to achieve this goal. A digest log, which records the changes of the digests, provides a way to find and quickly resolve the difference of views.

3.3.1 Digest Tree

The overall knowledge about the chat room can be represented by the set of statuses of all participants (*"producer statuses"*), i.e., what data each participant has produced so far. As a result of the naming rule, whenever a participant learns a sequence number N for another participant, he knows for sure that data packets with sequence numbers smaller than N from that participant must have already been sent to the chat room, regardless of whether he has already fetched them or not. Hence, the producer status of a participant can be effectively represented by the latest known data name of that participant.

Inspired by the idea of Merkle trees [7], we use digest tree to organize the producer statuses for quick and deterministic digest generation, as illustrated in Fig. 3. Every participant in this tree is represented as one node, holding a crypto digest, calculated by applying, for example, a SHA-256 [8] hash function[2] over participant's name prefix and the most recent sequence number. The root node keeps the digest for the whole chat room data set, which is computed by applying the same hash function to all of its child nodes. The child nodes of the root are kept sorted in lexicographic order according to their name prefixes, so that every participant generates exactly the same digest as long as they have the same view of the producer statuses.

The digest tree is kept up-to-date to accurately reflect the knowledge about the current chat room data set. Every time a participant generates a new chat message or learn about new messages from remote participants, the digest tree gets updated.

3.3.2 Digest Log

Digest tree is constantly updating as the conversation in the chat room progresses and keeps track only of the current state. However in some cases, recognizing previous digests may be helpful to resolve the difference in views. For example, a participant recovered from a temporary disconnection may send out a sync Interest with an out-of-date digest. Existing participants, recognizing the old digest, are able to reply with the producer statuses that have been changed since then, without resorting to more costly recovery means.

Hence, each participant keeps a *"digest log"* along with the digest tree. The log is a list of key-value pairs, where the key is the root digest and the value field contains the new producer statuses that have been updated since the previous state. The digest log always starts with the empty set and a new item is appended after each change in the digest tree, as depicted in Fig. 4. The size of the digest log can be constrained by a desired upper bound using periodical purging of old entries.

3.4 Chat Data Synchronization

Every participant always keeps one outstanding sync Interest with the current root digest to probe for new chat room data. Anyone with new data to the chat room can satisfy this sync Interest, and the new producer status

2. Note that SHA-256 has negligible probability to create a digest collision [9]. However, Chronos is able to recover even if the collision occurs, as the digests will be recalculated as soon as any new data is sent to the chat room.

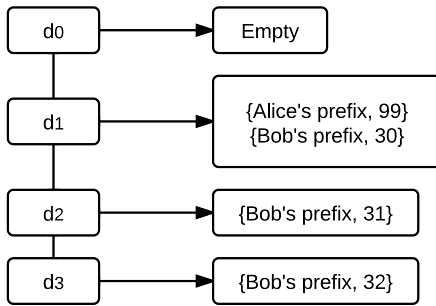
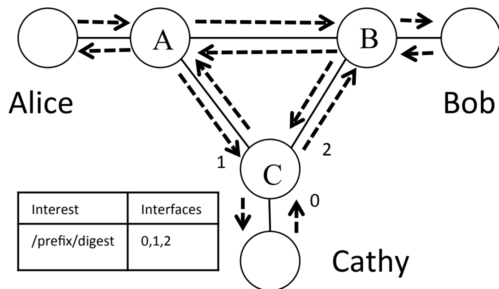
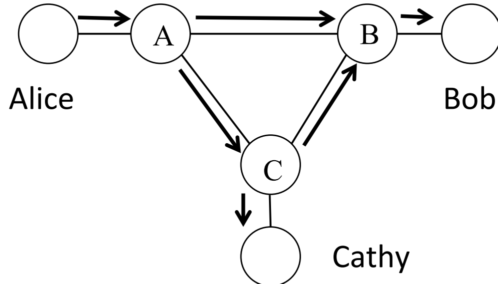


Fig. 4: An Example of Digest Log



(a) For each chat room, at most one sync Interest is transmitted over a link in one direction. There is one pending sync Interest at each node. Due to space limit, only the PIT of router C is depicted.



(b) The reply from Alice is multicasted to Bob and Cathy.

Fig. 5: Synchronizing the Views of Chat room Data

will be brought to all other participants immediately. Whenever sync Interest gets satisfied or expires, one sends out another one that carries the latest root digest.

When all participants have the same knowledge about the chat room data set, their sync Interests carry identical digests. We call it the *"steady state."* Fig. 5a shows the steady state in a simple topology. Although everyone sends out a broadcast Interest to exchange digest, there is only one pending sync Interest on each node in the steady state, as the identical Interests are be suppressed by the intermediate routers.

When a difference between the digest in the received Interest and the root digest in local digest tree is detected, a participant prepares to handle the sync Interest. For example, in Fig. 5b, Alice sends a new text message to the chat room. Alice noticed that the digest of others is the same as her previous one after her digest tree has been updated, and hence she immediately replies the sync Interest with the data name of her new message. By virtue of the states that have been set up in the routers, her reply is multicasted to all other participants. Whoever receives her reply updates the digest tree and sends out a new Interest with the updated digest, reverting the system to steady state again. Meanwhile, other participants may send separate Interests to retrieve Alice's text message. However, given that generally the text message is small, it is reasonable to piggyback the data to the reply of the sync Interest, eliminating an extra round trip delay.

Normally, participants recognize the digest carried in the sync Interest: it is the same as either the participant's current root digest, or the previous root digest if this participant just generated chat data. However, even in loss-free environments, participants may receive an Interest with unknown digest due to out-of-order packet arrivals. For instance, in Fig. 5b Cathy's Interest with the new digest (after incorporating Alice's new producer status into her digest tree, not shown on the figure) may reach Bob before Alice's sync reply with the old digest, due to the possible out-of-order delivery in the transmission. Hence, Chronos employs a randomized *"wait timer"* T_w, whose value should be set on the order of the propagation delay. In other words, a participant sets up the wait timer when an unknown digest is detected and postpones the processing of the sync Interest until the timer goes off. In our example, Bob's root digest becomes the same as the new digest in the Interest after Alice's reply reaches him, moving system to the steady state.

3.5 Participant Join/Leave

Participants come and go in a chat room. A newcomer needs to learn the current producer statuses in the room before he can participate, and the events of both join and leave of a participant need to be reflected on the roster.

A newcomer probes for information about a chat room by sending out a sync Interest with the reserved digest of the empty set[3]. If the newcomer is the only participant in the chat room, the Interest will not bring back anything. By the time the Interest expires, the newcomer updates the empty digest tree with his default initial producer status with sequence number being zero and sends out an new Interest with updated digest. Otherwise, the existing participants recognize the reserved "empty" digest and reply with all the producer statuses, which will be used by the newcomer to build his own digest tree. The producer statuses of the current participants are marked as active in the reply, so that the newcomer can construct an initial roster.

3. Digest of the empty set is reserved so that sync approach of Chronos is possible to support pure listeners who do not generate any data

Note that the newcomer may had been in the chat room before, and others may still remember his previous status. In such cases, the newcomer increments his previous sequence number by one and use that as his initial sequence number. Meanwhile, the newcomer also receive the sync Interest from others. He waits until the reply from others has been received to determine his initial status and replies with the proper status, prompting others to update their digest trees. In any case, the newcomer's initial producer status is the data name for an "available" message,[4] which prompts others to update rosters. By the end everyone has the same view of participants in the room.

Participants should inform others in the room before actually leaving. That is, the participant that intends to leave should reply to the sync Interest with the data name for an "unavailable" message. On receiving the "unavailable" message, others remove the participant from their rosters. Sometimes a participant may not have the chance to leave with grace. In these cases, other participants would notice this event when they miss a certain number of heartbeats from the participant who had left.

3.6 Handling of Complex Scenarios

While we optimized Chronos for operations under normal network conditions, we also took care of more complex scenarios where normal operations are no longer adequate. Networks may fail, participants may generate data simultaneously, packets may be dropped. The following discussion shows how Chronos copes with such problems.

3.6.1 Network Partitions

The server-less design of Chronos ensures that participants can communicate with each other normally as long as there is network connectivity. However, there could also be network partitions. In other words, the participants can be divided into two or more groups, in which participants in one group do not have network connectivity to participants in other groups. Fig. 6 illustrates one of such situations, where participants on the left can not communicate with participants on the right due to a router failure.

Although participants within each group may continue to chat without requiring any special handling, there may be problems when the partition heals. Depending on the activities of participants, there can be three different types of scenarios when the connectivity resumes.

First, participants may not send any data (including heartbeats) to the room during the period of network partitioning. As a result, everybody keep sharing the same view of the chat room data set and no special action is needed.

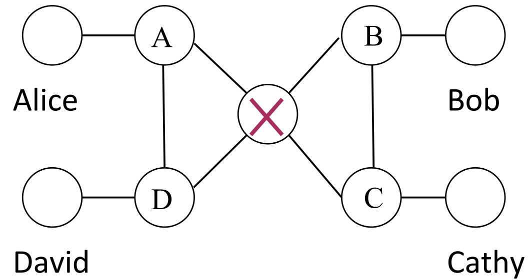


Fig. 6: Network Partition Example

Second, data might have been generated only in one group. In this case, the wait timer and digest log solve the problem efficiently. For example, in Fig. 6, participants on the right side may have had several rounds of conversation while those of the left side remained silent during the partition. When the connectivity resumes, Bob and Cathy receive sync Interest from left side with digest d_L and they immediately find d_L in the digest log, recorded just before the partition. Hence, they reply the Interest with any producer statuses that have updated afterwards. On the other hand, Alice and David see a digest d_R which is unknown to them at first, so they start the wait timer T_w. Before the timer expires, the reply from Bob and Cathy comes in and triggers the digest tree updating processes, after which their root digests also become d_R and the system resumes to the steady state.

Third, participants in all groups may have produced data during the network partitioning. Under such conditions, the participants in any of the groups cannot recognize the digests of the participants in any other group after the partition heals. Hence, no one has enough information to infer the chat room state differences using digests from the received sync Interests. Chronos resorts to the following way to break the impasse.

When the wait timer T_w expires and the unknown digest is still unrecognizable, a participant proceeds to send a recovery sync Interest as shown in Fig 7. It has a component "recovery" to differentiate from normal sync Interests and includes the unknown digest at the end. The purpose of such a Interest is to request missing information about the data set from those who recognize digest. Therefore, when a recovery sync Interest is received, those who have the digest in their digest log reply the interest with all current producer statuses[5], while others should silently ignore it. Upon receiving a recovery reply, a participant compares the producer statuses included in the reply with those stored in the digest tree and updates the tree whenever the one in the reply is more recent. This guarantees to revert the system into the steady state.

4. Status messages, including "available," "unavailable," and others are treated in the same manner as any normal text chat message.

5. They have no idea what information is missing at the other end, and hence replying all the producer statuses is the only way to do it.

/ndn/broadcast/chronos/chat@wonderland/recovery/142a2sdzd

Broadcast prefix | Data Set | Unknown Digest

Fig. 7: An Example of Recovery Sync Interests

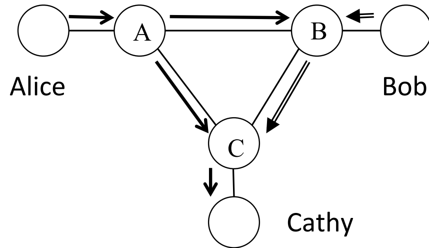


Fig. 8: Alice and Bob simultaneously generate chat data. They can not receive each other's reply to the sync Interest, and Cathy can only receive reply from one of them.

3.6.2 Simultaneous Data Generation

Chronos depends on successful multicasting of the new chat data from a participant to all other participants to quickly resume to the steady state. When simultaneous data generation happens, i.e., more than one participant sends a chat message to the room almost at the same time, this assumption no longer holds since one sync Interest can only fetch one Data packet [6].

Once a router has forwarded a Data packet, the corresponding PIT entry is removed. Fig. 8 illustrates how data flows when two participants simultaneously send messages to the room. When Alice's reply reaches router B, it will be dropped as the corresponding pending Interest has been satisfied by Bob's reply. Consequently, Alice and Bob cannot receive each other's reply. Similarly, Cathy only receives reply from one of them. Thus, when they send out new sync Interests, Alice and Cathy can not recognize Bob's digest, and vice versa. This scenario is the same as the third case in network partition handling, and the same technique can be used to solve the problem.[6]

3.6.3 Random Packet Loss

Similar to the current IP networks, Interests and Data transmissions are best-effort only in NDN.

When a sync Interest or reply gets dropped, it is possible to break the steady state if the loss happens to occur on a link that is the only link between two subsets of participants[7]. For example, in Fig 9, participants in group A have an out-dated view of the data set because the sync reply from a participant in group B gets dropped on the bridging link. However, participants in group A will send a sync recovery Interest shortly after the sync Interest from participants in group B arrives, as they do not recognize the new digest. The information brought back by the recovery Interest would restore the system to the steady state.

6. According to a previous study about the chat systems on the Internet [10], including IRCs, web chat and popular instant messaging applications, the occurrence of simultaneous data generation (chat messages to the room with small interarrival gap) is rare and hence this technique, despite of the perhaps unnecessary inclusion of older producer statuses in the reply, is sufficient in handling the problem.

7. Otherwise, the reply could go through another link and reach every participant

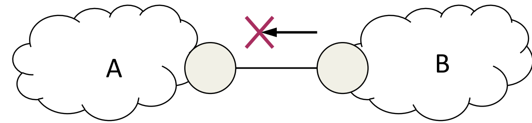
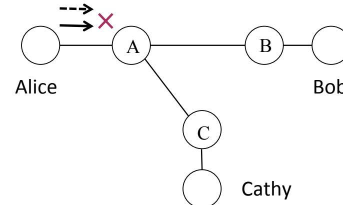
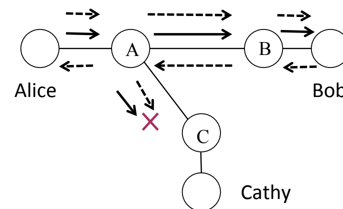


Fig. 9: Random Packet Loss



(a) The loss near the producer



(b) The loss near the receiver

Fig. 10: Both Data and Interest Lost on a Bridge Link

If the recovery Interest also gets dropped, it will be re-expressed in a exponentially-backoff fashion. Because it is certain that there is unknown information, the initial re-express interval could be set to the estimated round-trip delay (RTT), which each participant maintains independently using some algorithm similar to TCP's smoothed RTT calculation.

A more hazardous kind of scenarios is when both the new Data and the new Sync Interest get lost in a bridge link. Fig 10 depicts two possible scenarios of such loss.

In Fig 10a, both the new Data and the new Interest get lost near Alice. Consequently, neither Bob nor Cathy is aware of the new Data. This gives Alice a hint about the loss, as Alice is expecting the new Sync Interest from others. If the new Interest does not come within a time interval (approximately the same as the RTT), Alice would start retransmit the new sync Interest in an exponentially-backoff fashion. When Bob and Cathy receive the retransmitted new Interest, they send recovery sync Interest to figure out the difference.

The situation in Fig 10b is more subtle. The loss happens near Cathy, and hence Bob receives Alice's new

Data, and Alice receives the new sync Interest from Bob. As a result, there is no hint to anyone about the loss: Alice and Bob have the same view of the data set, and Cathy thinks she also has the up-to-date view because neither new Data nor sync Interest with unknown digest comes. There are two ways to get out of this situation: 1) Cathy's old sync Interest expires, and she re-expresses the Interest and fetches Alice's Data cached in router A; and 2) some participants generates new data, and the new Interest with digest unknown to Cathy prompts her to send recovery sync Interest.

The situation depicted in Fig 10b is the only situation where Chronos is not able to infer the loss within approximately RTT delay, and hence the time needed to recover depends on the re-express interval of the sync Interests and the rate of new data generation. In all other situations, Chronos would infer the loss and recover from it reactively.

However, the problem in Fig 10b is not really caused by Chronos design. Rather, it is a more general problem of multicast. It is always hard to detect loss immediately and efficiently. For example, in the reliable multicast work of S. Floyd et al. [4], receivers send low-rate, periodic, session messages that announces the known highest sequence numbers of others, and receivers compare the sequence numbers to detect loss. Hence, it is also possible there for some messages to experience long delay. In fact, the periodically re-expressed sync Interests have the same role as the session messages in [4]. The sutle difference between the two is: the session messages are multicasted and will not be suppressed, whereas the identical sync Interests would be suppressed by the routers. Hence, receivers in [4] may have slightly better chance to detect the loss when the number of participants is small,[8] but Chronos would detect the loss faster when the number of participants is large, as the sync re-express interval is independent of the number of participants.

IRC on the other hand uses unicast and the sender is responsible to ensure the data delivery to all receivers. Hence, it is easier for the sender to detect the loss, but the price to pay is the overhead in the redundant transmissions.

However, in any situation, Chronos will resume to steady state deterministically and reliably. To avoid the rare cases of long delay caused by situation in Fig 10b and at the same time avoid re-expressing sync Interest too frequently, the re-express interval of sync Interest should be dynamically adapted depending on the loss probability and the rate of new data generation.

4 Evaluation

To show benefits of Chronos and compare it to the conventional centralized multi-party chat approach, we conducted a number of simulation-based experiments. We performed our experiments using NS-3 simulator,[9] for which we implemented the NDN protocol module that approximates behavior of the existing code base of NDN project (CCNx project [11]). We also implemented a simple TCP-based approximation of the conventional centralized Internet Relay Chat (IRC) service. For simplicity of the simulation, we did not implement heartbeat messages for neither Chronos nor IRC service. Though heartbeats are necessary components of any chat service (e.g., they allow maintaining an up-to-date roster of the participants), they are handled in the same manner as normal chat messages and do not introduce new challenges for the protocol.

In our evaluations we used the Sprint point-of-presence topology [12], containing 52 nodes and 84 links. Each link was assigned measurement-inferred delays, 100 Mbps bandwidths, and drop-tail queues with 2000 packet capacities.

All nodes in the topology acted as clients for multi-user chat service, consisting of one chat room. The traffic pattern in each room was defined based on the multi-party chat traffic analysis by Dewes et al. [10] as a stream of messages of sizes from 20 to 200 bytes with inter-message gap following the exponential distribution with the mean of 5 seconds (the Poisson process).

In our implementation of Chronos-based text service, the actual data is piggybacked alongside with the state information. In other words, there is no need to request data separately after discovery of a new sequence number for a user.

4.1 Performance under normal network conditions

The first question that we answer is how Chronos compares with the conventional IP chat approaches under normal network conditions (no failures or packet loss) in terms of packet overhead. For this purpose, we performed 20 runs of the chat room simulation with 52-node topology and 1000 messages. Each individual simulation run featured different sets of messages injected to the chat room (different delays, different participant), as well as different choices for the central server in IRC case.

Fig. 11 shows the number of packets transferred over different links (packet concentration) in the simulated topology. When counting the packets, we include both Interest and Data packets in Chronos, and both TCP DATA and ACK packets in IRC. For each individual run, the links were ordered and analyzed by the packet concentration value. For instance, the leftmost point, alongside with 97.5% confidence interval, represents a range of values for links with the maximum packet concentration in a run.

As it could be expected from the centralized architecture of IRC, a number of links close to the server have extreme packet concentration. For example, when the server is single-homed in 52-node topology, a single message causes 51 DATA-ACK transmissions. At the

8. As the session message interval is proportional to the number of participants to save the bandwidth

9. https://www.nsnam.org/

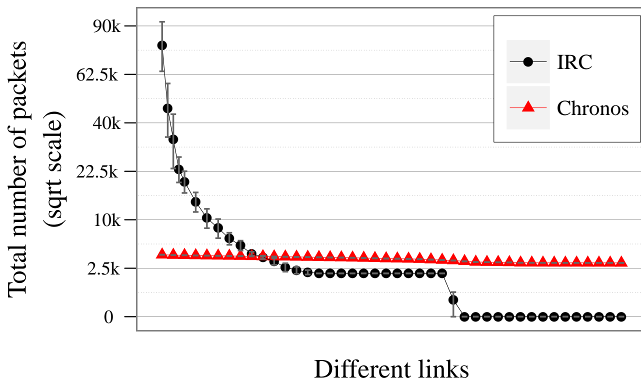


Fig. 11: Number of packets in links (packet concentration)

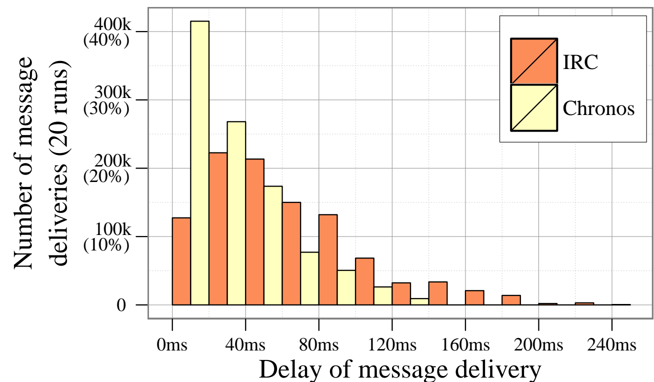


Fig. 13: Distribution of message delays

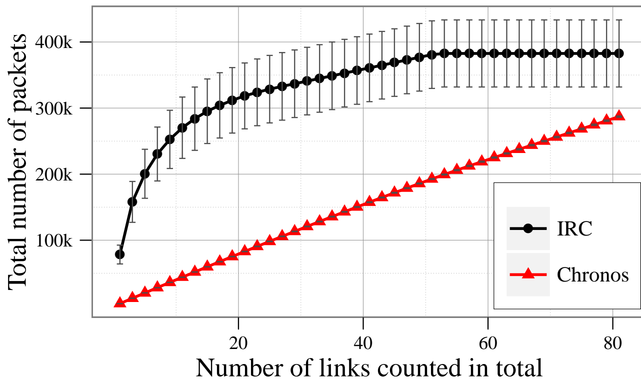


Fig. 12: CDF of per-link packet concentration

same time, in IRC some links are not used at all if they are not on the shortest paths to and from the server.

For the same traffic pattern, results for Chronos are significantly different: all links in the topology have about the same packet concentration, on order of the number of transmitted messages (from 3,000 to 4,000). This result is not a surprise, since for any single link, both nodes send sync Interests to each other and each Interest will yield at most one Data (since Data is not forwarded to the interface it was received from, some Interests will be unanswered).

The fact that Chronos utilizes all the links raises the question of how aggregate packet concentration relates to IRC. Fig. 12 shows a cumulative sum of packet concentrations (e.g., the second point from the left represents the number of packets in two most-utilized links of a run). The results presented in this graph highlight the fact that even with the broadcast-like nature of Interest distribution in Chronos, the overall packet count stays below IRC, while providing the same (and, as we show later in this Section, more robust) text conferencing service.

Another metric that is important for text conferencing is the delay between message generation and delivery to the interested parties. Fig. 13 shows the distribution of the message delivery delays, which include propagation, queuing, and processing components. As it can be seen, delays in Chronos are much shorter than in IRC case (40% cases among 20 runs in Chronos have delay less than 20 ms, compared to ≈13% of IRC cases for the same delay range). The reason for this is that in Chronos packets from the message producer to other participants follow the direct (optimal) paths, while in the IRC paths are usually triangular, as the packets have to go through the central server.

4.2 Performance in face of failures

The key feature of Chronos is its serverless design, which means that Chronos should allow communication between parties whenever there is connectivity between them. That is, even when the partition happens, the group of participants in each connected network should still be able to communicate with each other, and when the partition heals, different groups should synchronize the chat room data automatically.

To verify this property we conducted a small-scale 4-node simulation with link failures and network partitioning (Fig. 14). The total simulation time of 20 minutes was divided into 5 regions: 0–200 seconds with no link failures (Fig. 14a), 200–400 seconds with one failed link between nodes 0 and 1 (Fig. 14b), 400–800 seconds with two failed links between nodes 0, 1 and 2, 3 (partitioned network, Fig. 14c), 800–1000 seconds with one failed link between nodes 2 and 3, and finally 1000–1200 seconds period with no link failures.

We performed several runs of the simulation, all of them giving the same result: until network is really partitioned (period between 400–800 seconds), all parties continue to communicate with each other without noticing any problems. Results from one of the runs presented in Fig. 15, which visualizes the knowledge of node 0 about the current states (i.e., latest messages) of all other participants as a function of the simulation time.

Fig. 15 confirms not only that parties within a connected network continue to communication during the partition, but also the fact that when partition heals, the state is getting synchronized as soon as Interests start flowing into the formerly failed link. This will happen when either a new message is generated by a

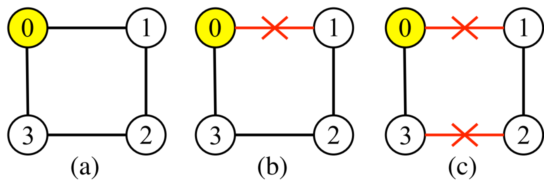


Fig. 14: Simple 4-node topology with link failures (link delays were chosen uniformly at random in the interval 1–2 ms)

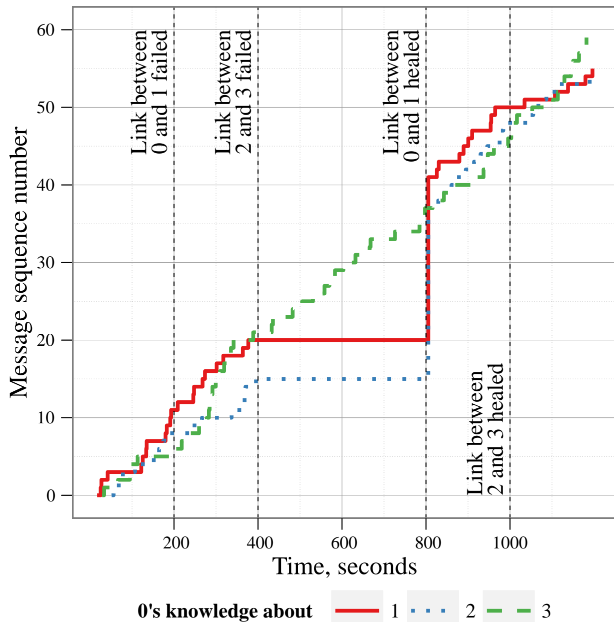


Fig. 15: Chronos performance in face of link failures (sequence number progress)

party inside the healed network, or outstanding Interests are timed out and re-expressed. Alternatively, routers should re-send pending broadcast Interests upon discovery of a new working link. In our evaluations we relied only on application-level Interest retransmissions/re-expressions, which resulted in relatively large partition recovery times (≈5 seconds, which in our simulations is the expectation for both the inter-message gap and the Interest lifetime).

To demonstrate benefits of Chronos on a bigger scale, we used again a 52-node topology, which was subjected to varying level of link failures. In each individual run of the simulation we failed from 10 to 50 links (different set of failed links in different runs), which corresponds to ≈10% and ≈60% of overall link count in the topology). We performed 20 runs of the simulation for each level of link failures, counting the number of pairs that are still able to communicate, despite severe network degradation (Fig. 16). We used the violin plot[10] for this graph to highlight a bimodal nature of the distribution for the percent of communicating pairs in the centralized

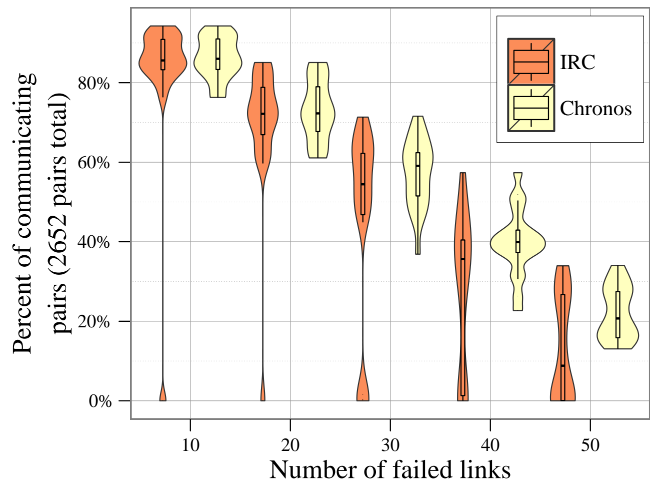


Fig. 16: Distribution of the number of communicating pairs versus number of failed links (violin plot)

IRC service. As it can be seen from the graph, it is quite common for the central server gets isolated from the rest of the network, even with a small level of link failures, which causes complete chat service disruption. In Chronos, there is always a substantial number of pairs able to communicate. For instance, in more than 60% of failed links, at least 12% of node pairs can communicate with each other, inside their respective clusters.

4.3 Performance in face of random loss

We also evaluated Chronos in lossy link environments with various per-link loss rates ranging from 1% to 10%.[11]

Fig 17 demonstrates the packet concentrations in face of losses in IRC and Chronos respectively. The numbers of packets transmitted over different links in Chronos are relatively the same under each loss rate, although the absolute values go up as the loss rate grows, due to the overhead caused by recoveries from the loss. On the contrary, a few links (probably the ones near the server) in IRC always have a huge number of packets, and the numbers grow significantly when the loss rate increases, which may in turn worsen the situation.

Fig 18 compares the delays experienced by packets in IRC and Chronos under different loss rates. To better examine the difference, we blow up the figures using exponential y axis. Regardless of the loss rate, more messages in Chronos experienced smaller delay compared to those in IRC. This trend is more clear as the loss rate grows: the percentage of messages with small delays drops rapidly in IRC and in Chronos it drops more gracefully. However, there is a small percentage of messages in Chronos that experienced delay longer than 10 seconds (probably

10. The violin plot is a combination of a box plot and a kernel density estimation plot. Wider regions represent higher probability for samples to fall within this region.

11. We also evaluated Chronos with lower per-link loss rates, and the performance of Chronos was almost unaffected. Thus, we focus on the uncommon high loss rate to study the effects of random packet loss.

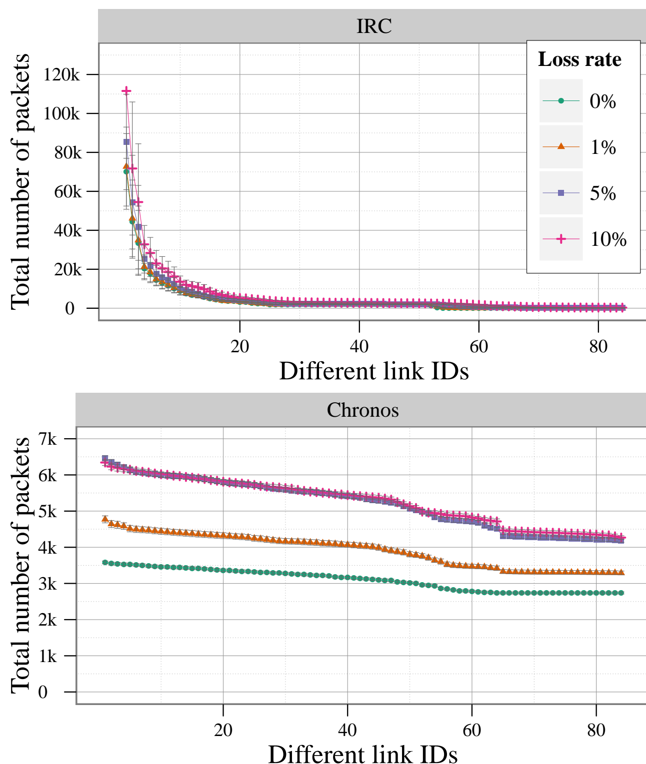


Fig. 17: Packet concentration in face of packet losses

due to the situation illustrated in Fig 10b)[12], whereas in IRC almost all messages experienced delay smaller than 10 seconds. This is because in TCP the sender will keep re-sending the data until the ACK is received, and in Chronos, sometimes it is not possible to detect the loss and has to resort to the sync Interest re-expression to recovery from the loss. To confirm our hypothesis, we compared the CDFs of message delays in Chronos under 5% loss rate with re-expression intervals of 5 seconds and 10 seconds respectively in Fig 19. As it shows, the longest delay experienced by messages dropped to half when the re-expression interval is reduced to 5 seconds from 10 seconds.

Fig 20 compares the total number of packet transmissions. As the loss rate increases, the number of total transmissions increases in both IRC and Chronos. However, the deviation in IRC grows significantly, while in Chronos it does not change much. Overall, the number of packet transmissions in Chronos is lower.

To conclude our evaluation, Chronos, leveraging benefits of the serverless design and NDN primitives (Interest/Data naming, loop-free Interests broadcasting, Data caching), provides a multi-party chat service that has lower aggregate overhead, delivers messages faster, and is robust against link failures and packet losses.

12. In our simulation, the sync Interests re-express interval is 10 seconds

5 Discussions

In this section we briefly discuss how to make Chronos scale well, both with large numbers of participants and in large-scale networks, such as the Internet. We also briefly discuss the security concerns for Chronos and the proposed solutions.

5.1 Support Large Number of Participants

As the number of participants grows, the storage overhead for producer statuses and computation overhead for the root digest at each node grow linearly. Similarly, the overhead due to heartbeats also increases as the number of current participants in a chat room goes up.

5.1.1 *Scale the digest tree*

Participants may come and go in a chat room. Remembering every participant that had ever entered the room is both costly and unnecessary. One way to reduce the overhead is to trim off the nodes in the digest tree for the participants who have left the chat room.

If the number of current participants in a chat room is large, one can effectively control the overhead by adding a level in the hierarchy of the digest tree, as depicted in Fig. 21. Participants can be uniformly distributed to a few groups by using a uniform hashing algorithm over their name prefixes. There is a child node of the root for each group that contains the digest of data generated by all the participants in that group. When a participant sends a new message to the room, only the digests of the corresponding group node and the root need to be changed. However, when handling partition healing, a two-round process may be needed to “walk down” the digest tree [13]. That is, instead of immediately replying to a sync Interest with all producer statuses, participants may first reply with a list of digests of groups. The recipients compare and identify the groups whose digests differ, and then send Interests to retrieve producer statuses of participants in those groups.

Depending on the number of participants, one can easily add more levels to the digest tree as needed.

5.1.2 *Adaptive heartbeat interval*

Heartbeat messages are used to maintain a roster of participants in a soft-state fashion. As the number of active participants grows, more bandwidth is utilized for heartbeat messages, as it becomes more likely that some participant generates a heartbeat at any time. A method similar to the one used in RTP [14] can be applied here to limit the bandwidth consumed by heartbeats. The interval to generate a heartbeat should be inversely proportional to the number of active participants in the roster, which Chronos already maintains. With such a dynamically adjusted interval, the proportion of bandwidth consumed by heartbeat messages can stay constant regardless of the number of active participants in a chat room.

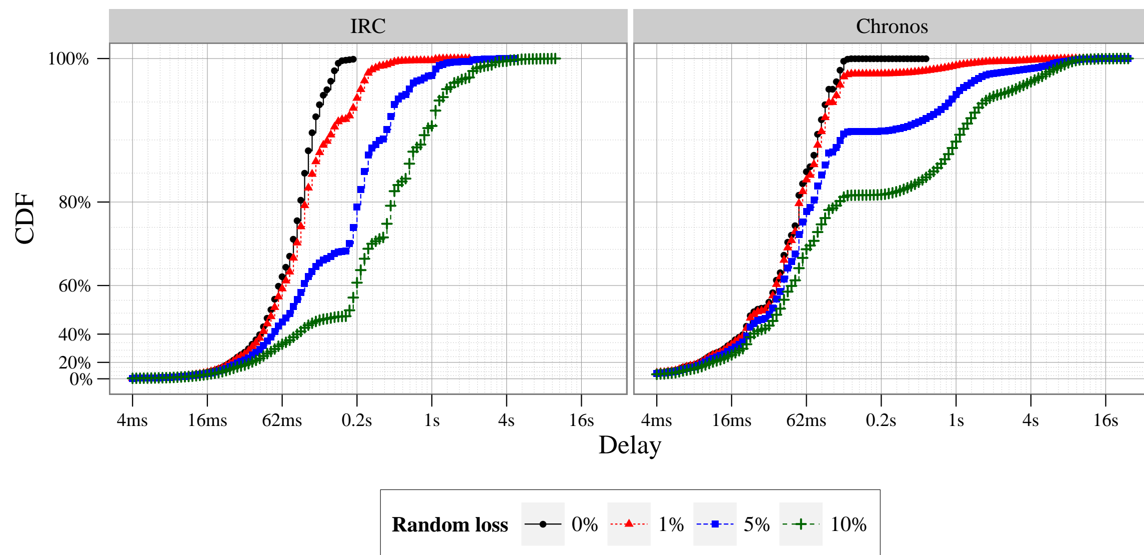


Fig. 18: Packet delivery delay in face of packet losses

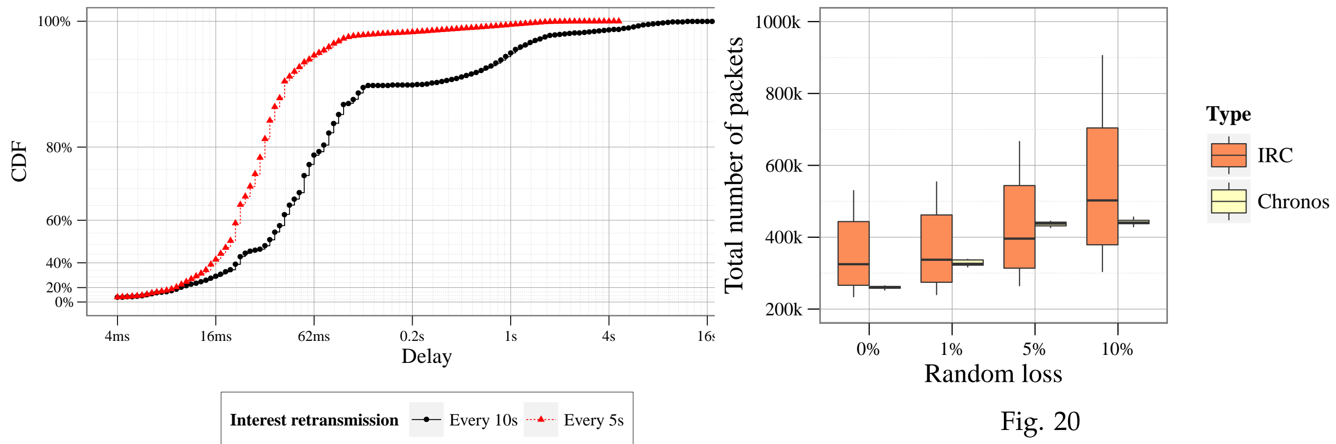


Fig. 19: Comparison of delays with different re-express intervals

Fig. 20

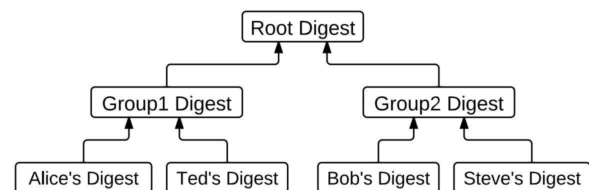


Fig. 21: Digest Tree with Additional Layer for Groups

5.2 Chronos in Large Networks

Chronos relies on broadcasting sync Interests to exchange digests of the chat room data set. However, it is unrealistic to assume that sync Interests could be broadcasted to the participants scattered in large networks, such as the Internet. However, sync Interests can still be broadcasted locally, while an overlay broadcast network can be used to propagate them to remote participants. As shown in Fig. 22, each network with Chronos participants sets up a gateway node, which knows how to forward sync Interests to other Chronos-supporting gateways. The issue of how gateways learn each other's presence is out of scope for this paper.[13] A gateway node relays the sync Interests received from its local network to gateways in other networks. Vice verse, the sync Interests received from other gateway nodes would also be broadcast in the local network. As a result, the broadcast of sync Interests is confined to networks where there are users participating in the Chronos chat service. Furthermore, besides supporting Chronos, the gateway nodes can be configured to forward sync Interests for other applications that use Chronos approach for data synchronization.[14]

13. We are currently working on a parallel project on making such Interest routing strategies scalable.

14. After the successful Chronos development, we are applying the same approach to several other application developments including joint editing and distributed file sharing.

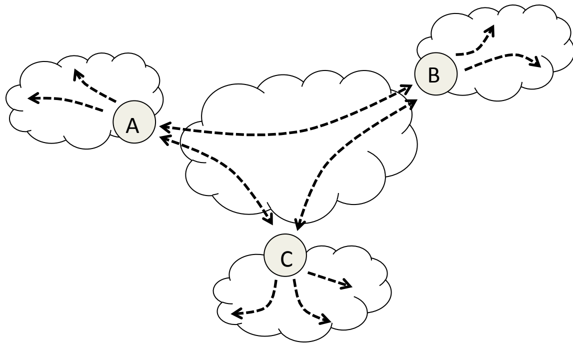


Fig. 22: Overlay Broadcast Network for Chronos

Note that the overlay gateways differ from the distributed servers in fundamental ways. Their role is to facilitate scalable propagation of sync Interests. A failure of any of them has no impact on Chronos service as long as the overlay remains connected.

5.3 Security Considerations

There are mainly two security concerns: how to prevent outsiders from accessing data of a private chat room and how to handle non-cooperative participants, who may disrupt a chat room by injecting false information about other producers' statuses. We assume that Chronos participants can securely obtain each other's public keys through a trustworthy key management system.[15]

To enforce access control, Chronos may employ the encryption-based approach. In other words, each private chat room should have a moderator that decides who are legitimate participants. Each participant publishes an encryption public key. As each Data packet in NDN is signed, the moderator can easily fetch and verify these keys. The moderator uses the encryption public keys of legitimate participants to encrypt a secret key, which then can be retrieved and decrypted by the eligible participants. Both the chat data and producer status data of private chats should be encrypted using the secret key, effectively preventing outsiders from eavesdropping.

Provenance of producer status can be achieved by requiring participants to sign their own statuses. This signature is stored along with the corresponding producer status. Whenever one needs to reply the sync Interest with producer statuses, one always include the original signatures for the statuses as well. Fig. 23 depicts a sync reply to a newcomer that contains the producer statuses for current participants. The recipient can verify the signature of each producer status and update the digest tree only if the signature for producer status is valid.

15. Manually configured key files, certificates signed by a CA, or any other key management system could work here.

/ndn/broadcast/Chronos/chat@wonderland/empty-digest		
Alice's prefix	37	Alice's signature
Bob's prefix	21	Bob's signature
Cathy's prefix	96	Cathy's signature
NDN packet signature		

Fig. 23: A Sync Reply with Signed Producer Statuses

6 CONCLUSION

In this paper we presented Chronos, a novel multi-user chat approach for Named Data Networking paradigm, that provides a way for a group of participants to communicate without relying on any central server. This make the system more robust and efficient compared to the traditional centralized designs. We verified and evaluated our implementation using simulation-based experiments, which demonstrated lower aggregate overhead, while delivering messages faster, and being more robust against link failures and packet losses. The serverless data synchronization approach used in Chronos design can be applied to various applications other than text chat, ranging from real-time audio conferencing to file synchronization applications. We hope our work could spark more discussions on the design space of the serverless data synchronization and further ease the job of application developers.

REFERENCES

[1] A. Tridgell and P. Mackerras, "The rsync algorithm," *TR-CS-96-05*, 1996.
[2] D. Eppstein, M. Goodrich, F. Uyeda, and G. Varghese, "What's the difference? efficient set reconciliation without prior context," *Proc. of SIGCOMM*, 2011.
[3] S. Agarwal, D. Starobinski, and A. Trachtenberg, "On the scalability of data synchronization protocols for PDAs and mobile devices," *IEEE Network*, vol. 16, no. 4, 2002.
[4] S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," in *Proc. of SIGCOMM*, 1995.
[5] X. Zhang, J. Liu, B. Li, and T.-S. Yum, "CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming," *Proc. of INFOCOM*, 2005.
[6] L. Zhang et al., "Named data networking (NDN) project," PARC, Tech. Rep. NDN-0001, 2010.
[7] R. C. Merkle, "A certified digital signature," in *Proc. of Advances in Cryptology*, 1989.
[8] National Institute of Standards and Technology, "Secure hash standard (SHS)," *FIPS PUB 180-3*, 2008.
[9] H. Gilbert and H. Handschuh, "Security analysis of SHA-256 and sisters," *Selected Areas in Cryptography*, 2004.
[10] C. Dewes, A. Wichmann, and A. Feldmann, "An analysis of Internet chat systems," *IMC'03*.
[11] Palo Alto Research Center, "Ccnx," http://www.ccnx.org.
[12] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP topologies with Rocketfuel," *IEEE/ACM Transactions on Networking*, vol. 12, no. 1, 2004.
[13] L. Wang, D. Massey, and L. Zhang, "Persistent detection and recovery of state inconsistencies," *Computer Networks*, 2006.
[14] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," RFC 3550, 2003.