Workgroup: Mimi
Internet-Draft:
draft-rosenberg-mimi-arch-options-00
Published: 24 October 2022
Intended Status: Standards Track
Expires: 27 April 2023
Authors: J. Rosenberg C. Jennings
Five9 Cisco

Architecture Options for More Messaging Interop (MIMI)

Abstract

This document outlines architecture options for managing the state of chats in MIMI.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 27 April 2023.

Table of Contents

1. Introduction

The More Instant Messaging Interoperability (MIMI) working group will specify the minimal set of mechanisms required to make modern Internet messaging applications interoperable. Over time, messaging applications have achieved widespread use, their feature sets have broadened, and their adoption of end-to-end encryption (E2EE) has grown, but the lack of interoperability between these services continues to create a suboptimal user experience. The standards produced by the MIMI working group will allow for E2EE messaging applications for both consumer and enterprise to interoperate without undermining the security guarantees that they provide.

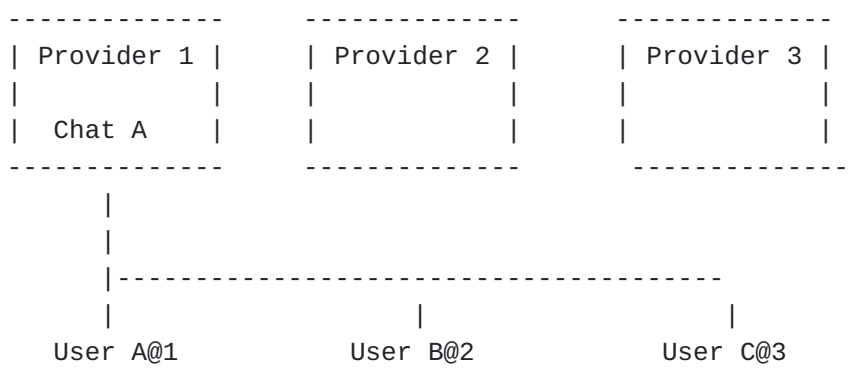
There are a variety of options on how MIMI can work in a federated model. This draft outlines the options and suggests which one to move forward with.

2. Architectural Options

There are numerous architectural models for building inter-provider messaging. One model, implemented in protocols like SIMPLE [RFC6914] and XMPP [RFC6120], consider messaging as a problem of message transport, sending messages from one user to another user. There is no notion of message storage, though in XMPP this can be added through Multi-User Chat (MUC).

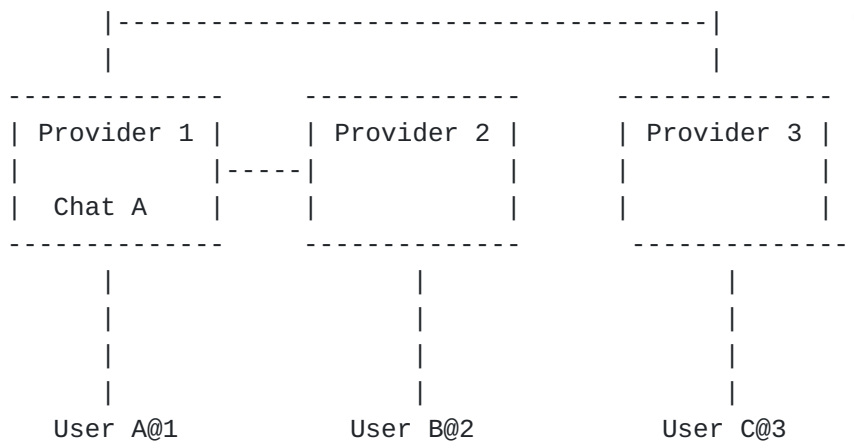
Most modern messaging services implement message persistence, and thus introduce the idea of chats as a stateful resource that live within the messaging provider. A chat resource can be either a 1-1 chat, or a group chat. The state of a chat resource includes the membership in the chat group along, the history of member additions/removals, along with the history of messages posted to it. In that way, 1-1 and group chats are essentially identical. Clients can query the state of their chats at any time to catch up, and they can receive notifications when there are new chats. Sending a message is primarily a transaction between the client and their provider; the provider acknowledges the message, stores it, acknowledges the receipt of the message towards the client, and then begins to replicate the data into databases as needed, while also notifying recipients of the new message.

When considering the expansion of this model to inter-provider communications, there are two ways it can work. In one approach, a particular chat resource lives within a single provider - the one where the chat resource was created. Other users, who may be supported by other providers, connect directly to this single provider to receive information about the state of that particular chat session. In this architecture, the state of a chat resource is not replicated between providers at all - it lives in a single place. Architecturally, it looks like this:

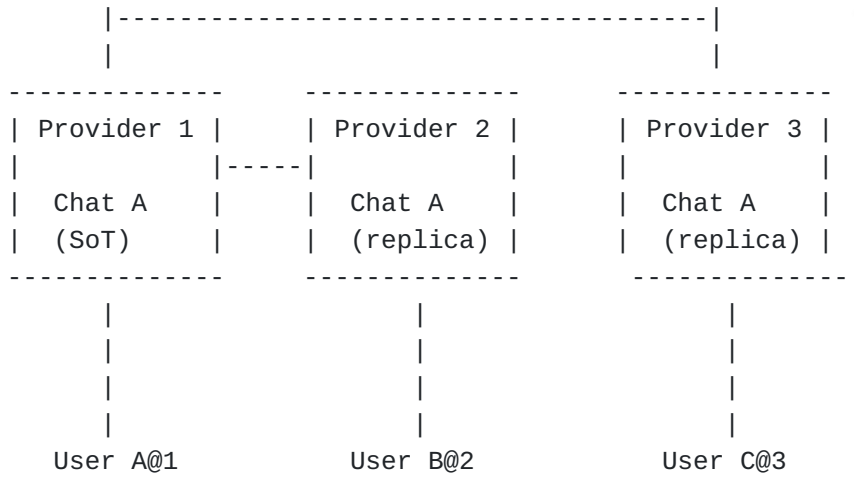


A group chat, chat A, was created by user A, who is a user of provider 1. This chat and its state lives exclusively within provider 1. User B, utilizes provider 2, and user C, a user of provider 3, are members of the group chat A. Through the MIMI protocols, users B and C are able to establish connections to provider 1 in order to retrieve and send messages. We refer to this model as the "guest model", since in essence, users B and C become "guest users" or provider 1. A drawback of this model is that there is no easy way for a user to know the full set of chats - across multiple providers - in which they should retrieve messages. This model is, in essence, similar to the multi-headed chat clients of old, like Pidgin.

A variation of this model is shown below. In this variation, the group chat still lives within provider 1, but users B and C connect to their respective providers to post mesasges, retrieve messages, and get notifications. Providers 2 and 3 do not store contents of the chat - they act as proxies for the purpose of authentication and trust. They also would retain knowledge of the set of chats in which their users are members, including ones like chat A which live within other providers. Let us call this the "proxied guest" model.



The final model - and the one that is used by this messaging format, is shown below:



In this model, the chat resource - its full state - lives within each provider. However, one provider acts as the source of truth. Through the mimi protocols and message formats, the state of this chat is synchronized to the other providers. All write operations happen against the chat resource in provider 1, and the new messages are then delivered to the other providers. For example, if user B posts a message to the group chat, user B sends that message to their provider, provider 2. Provider 2 will deliver this to provider 1. However, provider 2 will not update the state of the chat. Provider 2 may store this new message to ensure delivery to provider 1, but it is not considered as "posted" into the chat yet. Once delivered to provider 1, only then is the message considered posted. This causes a change in the state of the chat, and thus a notification of new message is sent to providers 2 and 3. Both of these providers store the new message and notify their respective

users of the new message. From a user experience perspective, typical implementations have user B's UI show the new message in the chat immediately upon sending. Thus, the notification that the message was posted, acts only as a confirmation to user B.

The astute reader will note there is a fourth model, wherein there is no single SoT and writes can be made to any instance of the chat resource. THis is the most complex solution, and due to the challenges of building such database systems in general - let alone making them work inter-provider - we do not suggest this approach.

3. Recommendations

The third model - where chat resource state is replicated, but there is a single source of truth against which all write operations are performed - seems like the right model.

The guest model incurs a heavy load of long-lived connections on each provider, and requires the client to maintain a connection to each provider. This doesnt seem scalable. The proxy guest model fixes this, but puts the burden of message sync and reliability in the client hands. The replicated state model addresses that, while providing the consistency required for the system to work reliably.

4. Normative References

[I-D.ietf-mls-architecture]
Beurdouche, B., Rescorla, E., Omara, E., Inguva, S., Kwon, A., and A. Duric, "The Messaging Layer Security (MLS) Architecture", Work in Progress, Internet-Draft, draft-ietf-mls-architecture-09, 19 August 2022, <https://www.ietf.org/archive/id/draft-ietf-mls-architecture-09.txt>.

[I-D.ietf-mls-protocol]
Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., and K. Cohn-Gordon, "The Messaging Layer Security (MLS) Protocol", Work in Progress, Internet-Draft, draft-ietf-mls-protocol-16, 11 July 2022, <https://www.ietf.org/archive/id/draft-ietf-mls-protocol-16.txt>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3862] Klyne, G. and D. Atkins, "Common Presence and Instant Messaging (CPIM): Message Format", RFC 3862, DOI

10.17487/RFC3862, August 2004, <https://www.rfc-editor.org/info/rfc3862>.

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, DOI 10.17487/RFC6120, March 2011, <https://www.rfc-editor.org/info/rfc6120>.

[RFC6914] Rosenberg, J., "SIMPLE Made Simple: An Overview of the IETF Specifications for Instant Messaging and Presence Using the Session Initiation Protocol (SIP)", RFC 6914, DOI 10.17487/RFC6914, April 2013, <https://www.rfc-editor.org/info/rfc6914>.

Authors' Addresses

Jonathan Rosenberg
Five9

Email: jdrosen@jdrosen.net

Cullen Jennings
Cisco

Email: fluffy@iii.ca