Workgroup:
More Instant Messaging Interoperability
Internet-Draft:
draft-ralston-mimi-matrix-framework-01
Published: 13 March 2023
Intended Status: Standards Track
Expires: 14 September 2023
Authors: T. Ralston
The Matrix.org Foundation C.I.C.

Matrix as a Messaging Framework

Abstract

This document describes how Matrix, an existing openly specified decentralized protocol for secure interoperable communications, works to create a framework for messaging.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://turt2live.github.io/ietf-mimi-matrix-framework/draft-ralston-mimi-matrix-framework.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ralston-mimi-matrix-framework/.

Discussion of this document takes place on the More Instant Messaging Interoperability Working Group mailing list (mailto:mimi@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/mimi/. Subscribe at https://www.ietf.org/mailman/listinfo/mimi/.

Source for this draft and an issue tracker can be found at https://github.com/turt2live/ietf-mimi-matrix-framework.

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 14 September 2023.

Table of Contents

1. Introduction
2. Overall model
   2.1. Eventual Consistency
3. Rooms and Events
   3.1. State Events
   3.2. Room Versions
   3.3. Mapping Features to Events
4. Users and Devices
5. Room Version I.1
6. Federation API
7. Identity
8. End-to-end Encryption
9. Security Considerations
10. IANA Considerations
11. References
   11.1. Normative References
   11.2. Informative References
Author's Address

1. Introduction

Matrix is an existing open standard suitable for Instant Messaging (IM), Voice over IP (VoIP) signaling, Internet of Things (IoT) communication, and bridging other existing communication platforms together. In this document we focus largely on the IM use case, however the concepts can be applied to other forms of communication as well.

The existing Matrix specification [MxSpec] is quite large, yet modular. Here, we can focus on the reusable portions that cover messaging, and leave the rest out of scope, leading to more effective implementations.

This document assumes some prior knowledge of federated or decentralized systems, such as the principles of email. This document additionally references concepts from [I-D.rosenberg-mimi-taxonomy] to build common understanding.

2. Overall model

At a high level, Matrix consists of 4 primary concepts:

- Homeservers (also called "servers" for simplicity) contain user accounts and handle the algorithms needed to support Rooms.
- Users produce Events which are sent into Rooms through their Homeserver.
- Rooms are a defined set of algorithms which govern how all servers in that room behave and treat Events. They are similar to channels, group chats, etc from other protocols.
- Events are pieces of information that make up a room. They can be "state events" which track details such as membership, room name, and encryption algorithm or "timeline events" which are most commonly messages between users.

Homeservers replicate events created by their users to all other participating homeservers in the room (any server with at least 1 joined user). Similarly, events are retrieved on-demand from those same participating homeservers. The details regarding how this is done specifically, and how a server becomes joined to a room, are discussed later in this document.

A 2 homeserver federation might look as follows:

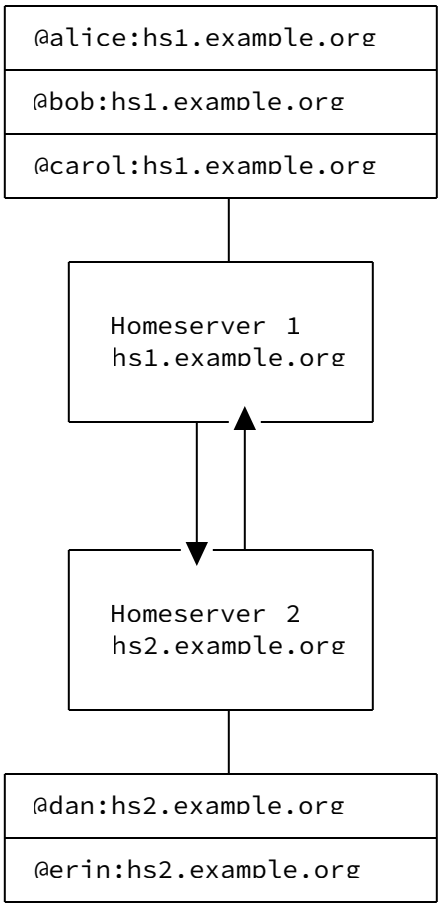

Figure 1: Simple Network Architecture of Matrix

In this Figure, Alice, Bob, and Carol are on "hs1", with Dan and Erin being on "hs2". Despite both having the root domain "example.org", they are considered two completely different homeservers. Typically, a homeserver would use a domain which was closer to the root (ie: just "example.org"), however for illustrative purposes and having two homeservers to work with, they have been "improperly" named here.

If Alice creates a room and invites Bob, Alice and Bob can communicate without hs2. If Bob invites Dan or Erin, hs2 joins the room when either accepts the invite. During the join process hs1 replicates the current state of the room (membership, room name, etc) to hs2. After this initial replication, both homeservers replicate new events from their side to the other. This replication includes validation of the events on the receiving side.

2.1. Eventual Consistency

In federated environments it is extremely likely that a remote server will be offline or unreachable for a variety of reasons, and a protocol generally needs to handle this network fault without causing undue inconvenience to others involved. In Matrix, homeservers can go completely offline without affecting other homeservers (and therefore users) in the room - only users on that offline homeserver would be affected.

During a network fault, homeservers can continue to send events to the room without the involvement of the remaining homeservers. This applies to both sides of the fault: the "offline" server might have had an issue where it could not send or receive from the federation side, but users are still able to send events internally - the server can continue to queue these events until full connectivity is restored. When network is restored between affected parties, they simply send any traffic the remote side missed and the room's history is merged together. This is eventual consistency: over time, all homeservers involved will reach a consistent state, even through network issues.

3. Rooms and Events

Rooms are a conceptual place where users can send and receive events. Events are sent into the room, and all participants with sufficient access will receive the event. Rooms have a unique identifier of !opaque_localpart:example.org, with the server name in the ID providing no meaning beyond a measure to ensure global uniqueness of the room. It is not possible to create rooms with another server's name in the ID.

Rooms are not "created on" any particular server because the room is replicated to all participating homeservers equally. Though, at time of creation, the room might only exist on a single server until other participants are invited and joined (as is typical with creating a new room).

Rooms are not limited in number of participants, and a "direct message" (DM, 1:1) room is simply a room with two users in it. Rooms can additionally have a "type" to clearly communicate their intended purpose, however this type does not fundamentally change that events are sent into the room for receipt by other users. The type typically only changes client-side rendering/handling of the room.

Events are how data is exchanged over Matrix. Each client action (eg: "send a message") correlates with exactly one event. Each event has a type to differentiate different kinds of data, and each type SHOULD serve exactly one purpose. For example, an event for an image

might contain a "caption" (alt text), but should not contain a text message to go along with the image - instead, the client would send two events and use a structured relationship to represent the text referencing the image.

Through the use of namespaces, events can represent any piece of information. Clients looking to send text messages would use m.message, for example, while an IoT device might send org.example.temperature into the room. The namespace for event types is the same as the Java package naming conventions (reverse domain with purpose).

3.1. State Events

Within a room, some events are used to store key/value information: these are known as state events. Alongside all the normal fields for an event, they also contain a "state key" which is used to store similar information of the same type in the room.

Such an example of a state event is membership: each member, once involved in the room in some way, has a dedicated m.room.member state event to describe their membership state (join, leave, ban, etc) and a state key of their user ID. This allows their membership to change and for other clients (or servers) to easily look up current membership information using the event type and predictable state key.

Other examples of state events are the room name, topic, permissions, history visibility, join constraints, and creation information itself (all with empty/blank state keys, as there's only one useful version of each). Custom state events are additionally possible, just like with custom events.

3.2. Room Versions

Rooms have strict rules for what is allowed to be contained within them, and have various algorithms which handle different aspects of the protocol, such as conflict resolution and acceptance of events. To allow rooms to be improved upon through new algorithms or rules, "room versions" are employed to manage a set of expectations for each room. New room versions would be created and assigned as needed.

Room versions do not have implicit ordering or hierarchy to them, and once in place their principles are immutable (preventing all existing rooms from breaking). This allows for breaking changes to be implemented without actually breaking existing rooms: rooms would "upgrade" to the new room version, putting their old copy behind them.

Upgrading a room is done by creating a new room with the new version specified, and adding some referential information in both rooms. This is to allow clients and servers to treat the set of rooms as a single logical room, with history being available for clients which might wish to combine the timelines of the rooms to hide the mechanics of the room upgrade itself.

Rooms can be upgraded any number of times, and because there's no implicit ordering for room versions it's possible to "upgrade" from, for example, version 2 to 1, or even 2 to 2.

Later in this document is a description of a room version suitable for MIMI.

3.3. Mapping Features to Events

To achieve proper interoperability it is important to consider which features the other clients (and sometimes servers) in the domain support, and how to represent them using a common format [I-D.ralston-mimi-messaging-requirements]. Matrix represents everything either as Events, per earlier in this section, or as Ephemeral Data Units (EDUs) [MxEDU] when the data doesn't need to be persisted to the room.

This structure of having everything being a genericised event or EDU allows Matrix to represent nearly every messaging feature as a content format problem. Servers additionally do not generally need to do much processing of events in order for the clients to operate, and can even be purely store & forward-like nodes for clients. The interface between client and server (also called the Client-Server API) is out of scope for this document. The Matrix Client-Server API [MxClientServerApi] may be a good reference for building a Matrix-native client or server implementation.

In Matrix, the following is how some common features would be represented:

Feature	Representation
Direct/Private Messages, 1:1 chats	A room with 2 users in it
Message history	Natural consequence of Matrix's room algorithms. Stored on the server.
Text messages	Timeline events
Multimedia (images, video, etc)	Timeline events
Message edits	Timeline events
Redaction/Removal	Timeline events
Reactions	Timeline events

Feature	Representation
Replies & rich markup	Timeline events
VoIP	Timeline events & WebRTC
Threads	Timeline events
Encryption	Timeline events containing an encrypted plaintext event payload
Typing notifications	EDUs
Read receipts	EDUs
Presence/online status	EDUs
Invites/membership	State events
Room name, topic, etc	State events

Table 1: Examples of IM features mapped to Matrix

Note: some features have not been included for brevity. The features in the table above should represent enough of a baseline to determine whether another feature would be a timeline event, state event, EDU, or something else.

In Matrix's content format, updated and defined by MSC1767 [MSC1767], fallbacks to rich text are common to ensure clients can particpate as best as realistically possible when encountering features they don't support. For example, a client which doesn't support polls might represent that poll simply as a text message and users can react to it (or simply reply with more text) to "vote".

4. Users and Devices

Each user, identified by @localpart:example.org, can have any number of "devices" associated with them. Typically linked to a logged-in session, a device has an opaque ID to identify it and usually holds applicable encryption keys for end-to-end encryption.

Multiple algorithms for encryption are supported, though the Matrix specification currently uses its own Olm and Megolm algorithms for encryption. For increased interoperability, Matrix would adopt MLS [I-D.ietf-mls-protocol] instead, likely with minor changes to support decentralized environments [DMLS].

5. Room Version I.1

The room version grammar [MxRoomVersionGrammar] reserves versions consisting solely of 0-9 and . for the Matrix protocol itself. For purposes of MIMI, a reservation of versions starting with I. and consisting otherwise of 0-9 and . is introduced.

The first version under this reservation would be I.1, described as follows.

I.1 is based upon Matrix's Room Version 10 [MxRoomVersion10], and MSC1767 [MSC1767] is incorporated to provide better content format primitives. Note that while the Matrix specification references clients and HTTP/JSON APIs, neither of these details are strictly relevant for this document.

6. Federation API

In order to replicate a room across other homeservers, an API must exist to federate accordingly. This document defines a transport-agnostic API for federation in Matrix.

Matrix aims for a "Multilateral" federation described by [I-D.rosenberg-mimi-taxonomy], where servers can implement their own non-standard checks on requests to ensure their server is operating safely. For example, while this document describes an "invite API", a server might choose to block invites from a particular server or user for any reason it feels is reasonable (preventing the receiving server from participating in the room).

The major APIs needed for federation are as follows. Note that where Matrix specification already exists, transport details in that specification are out of scope of this document.

* A way to invite users to a room (when their server isn't already participating in the room). This is specified by Matrix already [MxInviteApi].

* A way to join a room the server doesn't yet already participate in. This is specified by Matrix already [MxJoinApi].

* A way to knock or request an invite to a room (when the server isn't already participating in the room). This is specified by Matrix already [MxKnockApi].

* A way to reject invites when the server isn't already participating in the room. This is specified by Matrix already [MxLeaveApi].

* A way to retrieve individual and missing events from other participating servers, subject to history visbility and authorization. This is specified by Matrix already [MxEventsApi] [MxBackfillApi].

* A way to send events to another server. Matrix currently describes this as a transport-level detail in the form of transactions [MxTransactionApi].

Note that the membership APIs above only apply when the server isn't already participating in the room. If the server is already

participating, the server can simply generate an appropriate membership event for the user and send it to the other participating servers directly - it does not need to handshake a valid event with a resident server.

Matrix defines many more federation APIs such as to-device messaging, ephemeral event handling (typing notifications, presence, etc), and encryption-specific APIs, however these are out of scope of this document.

7. Identity

Matrix relies on identifiers being user IDs (@user:example.org), however in the wider scope of MIMI it is expected that a user might be trying to message a phone number instead of knowing the user ID. This document does not define how an identifier like a phone number is resolved to a user ID, but expects that a process exists to do so.

Such a service might resolve +1 555 123 4567 to @15551234567:example.org, for example.

8. End-to-end Encryption

Encryption of events generally happens at the Content Format level, with key exchange happening over a transport-level concern. Matrix currently uses a dedicated set of APIs for key exchange, though with the adoption of MLS by MIMI there are expected changes [MxDevicesApi] [MxEncryptionApi] [MxToDeviceApi].

9. Security Considerations

Not formally specified in this version of the document, Matrix has several threat model considerations to ensure feature development does not make these threats easier to achieve. They are currently specified in v1.6 of the Matrix specification under Section 6 of the Appendices. [MxSecurityThreatModel]

10. IANA Considerations

This document has no IANA actions.

11. References

11.1. Normative References

[I-D.ralston-mimi-messaging-requirements]
Ralston, T., "Requirements of Interoperable Messaging", Work in Progress, Internet-Draft, draft-ralston-mimi-messaging-requirements-00, 13 March 2023, <https://

datatracker.ietf.org/doc/html/draft-ralston-mimi-messaging-requirements-00>.

[I-D.rosenberg-mimi-taxonomy] Rosenberg, J., "A Taxonomy for More Messaging Interop (MIMI)", Work in Progress, Internet-Draft, draft-rosenberg-mimi-taxonomy-00, 24 October 2022, <https://datatracker.ietf.org/doc/html/draft-rosenberg-mimi-taxonomy-00>.

[MSC1767] Hodgson, M. and T. Ralston, "Extensible event types & fallback in Matrix (v2)", 2023, <https://github.com/matrix-org/matrix-spec-proposals/blob/01654eb2dec7769daf1d8d7a25c04cb70a1ac9f4/proposals/1767-extensible-events.md>.

[MxBackfillApi] The Matrix.org Foundation C.I.C., "Matrix Specification | v1.6 | Federation API | Backfill", 2023, <https://spec.matrix.org/v1.6/server-server-api/#backfilling-and-retrieving-missing-events>.

[MxEventsApi] The Matrix.org Foundation C.I.C., "Matrix Specification | v1.6 | Federation API | Event Retrieval", 2023, <https://spec.matrix.org/v1.6/server-server-api/#retrieving-events>.

[MxInviteApi] The Matrix.org Foundation C.I.C., "Matrix Specification | v1.6 | Federation API | Invites", 2023, <https://spec.matrix.org/v1.6/server-server-api/#inviting-to-a-room>.

[MxJoinApi] The Matrix.org Foundation C.I.C., "Matrix Specification | v1.6 | Federation API | Joins", 2023, <https://spec.matrix.org/v1.6/server-server-api/#joining-rooms>.

[MxKnockApi] The Matrix.org Foundation C.I.C., "Matrix Specification | v1.6 | Federation API | Knocks", 2023, <https://spec.matrix.org/v1.6/server-server-api/#a-nameknocking-rooms-knocking-upon-a-room>.

[MxLeaveApi] The Matrix.org Foundation C.I.C., "Matrix Specification | v1.6 | Federation API | Leaves and Rejected Invites",

2023, <https://spec.matrix.org/v1.6/server-server-api/#leaving-rooms-rejecting-invites>.

[MxRoomVersion10] The Matrix.org Foundation C.I.C., "Matrix Specification | v1.6 | Room Version 10", 2023, <https://spec.matrix.org/v1.6/rooms/v10/>.

[MxRoomVersionGrammar] The Matrix.org Foundation C.I.C., "Matrix Specification | v1.6 | Room Versions | Room Version Grammar", 2023, <https://spec.matrix.org/v1.6/rooms/#room-version-grammar>.

11.2. Informative References

[DMLS] Chathi, H., "Decentralised MLS", Web https://gitlab.matrix.org/matrix-org/mls-ts/-/blob/dd57bc25f6145ddedfb6d193f6baebf5133db7ed/decentralised.org, 2021, <https://gitlab.matrix.org/matrix-org/mls-ts/-/blob/dd57bc25f6145ddedfb6d193f6baebf5133db7ed/decentralised.org>.

[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-18, 13 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-mls-protocol-18>.

[MxClientServerApi] The Matrix.org Foundation C.I.C., "Matrix Specification | v1.6 | Client-Server API", 2023, <https://spec.matrix.org/v1.6/client-server-api>.

[MxDevicesApi] The Matrix.org Foundation C.I.C., "Matrix Specification | v1.6 | Federation API | Device Management", 2023, <https://spec.matrix.org/v1.6/server-server-api/#device-management>.

[MxEDU] The Matrix.org Foundation C.I.C., "Matrix Specification | v1.6 | Federation API | EDUs", 2023, <https://spec.matrix.org/v1.6/server-server-api/#edus>.

[MxEncryptionApi] The Matrix.org Foundation C.I.C., "Matrix Specification | v1.6 | Federation API | End-to-End Encryption", 2023, <https://spec.matrix.org/v1.6/server-server-api/#end-to-end-encryption>.

[MxSecurityThreatModel] The Matrix.org Foundation C.I.C., "Matrix Specification | v1.6 | Appendices | Security Threat

Model", 2023, <https://spec.matrix.org/v1.6/appendices/#security-threat-model>.

[MxSpec] The Matrix.org Foundation C.I.C., "Matrix Specification | v1.6", 2023, <https://spec.matrix.org/v1.6/>.

[MxToDeviceApi] The Matrix.org Foundation C.I.C., "Matrix Specification | v1.6 | Federation API | Send-to-device Messaging", 2023, <https://spec.matrix.org/v1.6/server-server-api/#send-to-device-messaging>.

[MxTransactionApi] The Matrix.org Foundation C.I.C., "Matrix Specification | v1.6 | Federation API | Transactions", 2023, <https://spec.matrix.org/v1.6/server-server-api/#transactions>.

Author's Address

Travis Ralston
The Matrix.org Foundation C.I.C.

Email: travisr@matrix.org