Workgroup:
More Instant Messaging Interoperability
Internet-Draft:
draft-ralston-mimi-matrix-transport-00
Published: 6 November 2022
Intended Status: Informational
Expires: 10 May 2023
A T. Ralston
u The Matrix.org Foundation C.I.C.
t
h
o
r
s
:
M. Hodgson
The Matrix.org Foundation C.I.C.

Matrix Message Transport

Abstract

This document specifies an openly federated protocol, Matrix, for interoperable message transport.

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-transport/draft-ralston-mimi-matrix-transport.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ralston-mimi-matrix-transport/.

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-transport.

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 10 May 2023.

Table of Contents

1. Introduction

Designing transports for interoperable messaging can often be a race towards the lowest common denominator among the systems one is attempting to interoperate. In addition to considering those systems' capabilities, the design must also account for a number of edge cases and routine failures that often come up during implementation, such as handling of network partitions/disconnects, malicious actors (intentional or accidental), and recovering from failure states.

Matrix solves this by providing a highest-common denominator messaging layer between current real-world messaging systems, which expresses events in an authenticated Directed Acyclic Graph (DAG) that is incrementally replicated between untrusted participating servers, providing decentralized access control without single points of control. This ensures that all participants converge on a consistent view of room history as rapidly as possible, including key/value state, even in the face of bad actors or network partitions - rather than all participants piecing together partial independent views of a room from pubsub streams or other sources. This provides aggressive resilience to network partitions, suitable even for the harshest denied, disrupted, intermittent and high

latency environments (for example, P2P networks overlaying Bluetooth or HF carriers; maritime or space deployments).

Matrix [MxSpec] is an open standard first created in 2014 to define interoperable, decentralized, secure communication. Matrix exited beta in June 2019, and having been actively maintained and improved since under an open governance model [MxFoundation], a subset of the open standard fits extremely well within the More Instant Messaging Interoperability (MIMI) working group's efforts to establish standards for interoperable modern messaging applications. This document focuses on the transport (message- or event-passing) portion of the Matrix protocol as it relates to MIMI.

2. Matrix Primitives

Within Matrix there are four key primitives needed for cross-server communication:

1. Homeservers (or simply "servers"), identified as [RFC1123] Host Names with extensions for IPv6, which act as a namespace for holding Users and replicated copies of Rooms.

2. Rooms, identified as !localpart:example.org, consist of the Directed Acyclic Graph (DAG) for Events sent by Users. The DAG is replicated across all Homeservers participating in that Room. It can be thought of as a pubsub topic with additional semantics to access and authenticate message history and key/value state.

3. Events, identified as $base64HashOfItself, have a type (m.room.message or m.room.encrypted, for example) and are sent by Users, added to the Room (DAG) by Homeservers. State Events are versioned key/value data which are pinned in the room, usually tracking information about the room itself (membership, room name, topic, etc). State Events can be overwritten while other Events can not (though normal Events can be "edited" by sending a new Event which points to the original Event).

4. Users, identified as @localpart:example.org, can have multiple devices (logged-in sessions) to join/leave Rooms and send Events to those rooms.

Other primitives do exist in Matrix, such as Devices for users, however they are not directly relevant to this document's purpose and have been excluded.

3. Federation Basics and Eventual Consistency

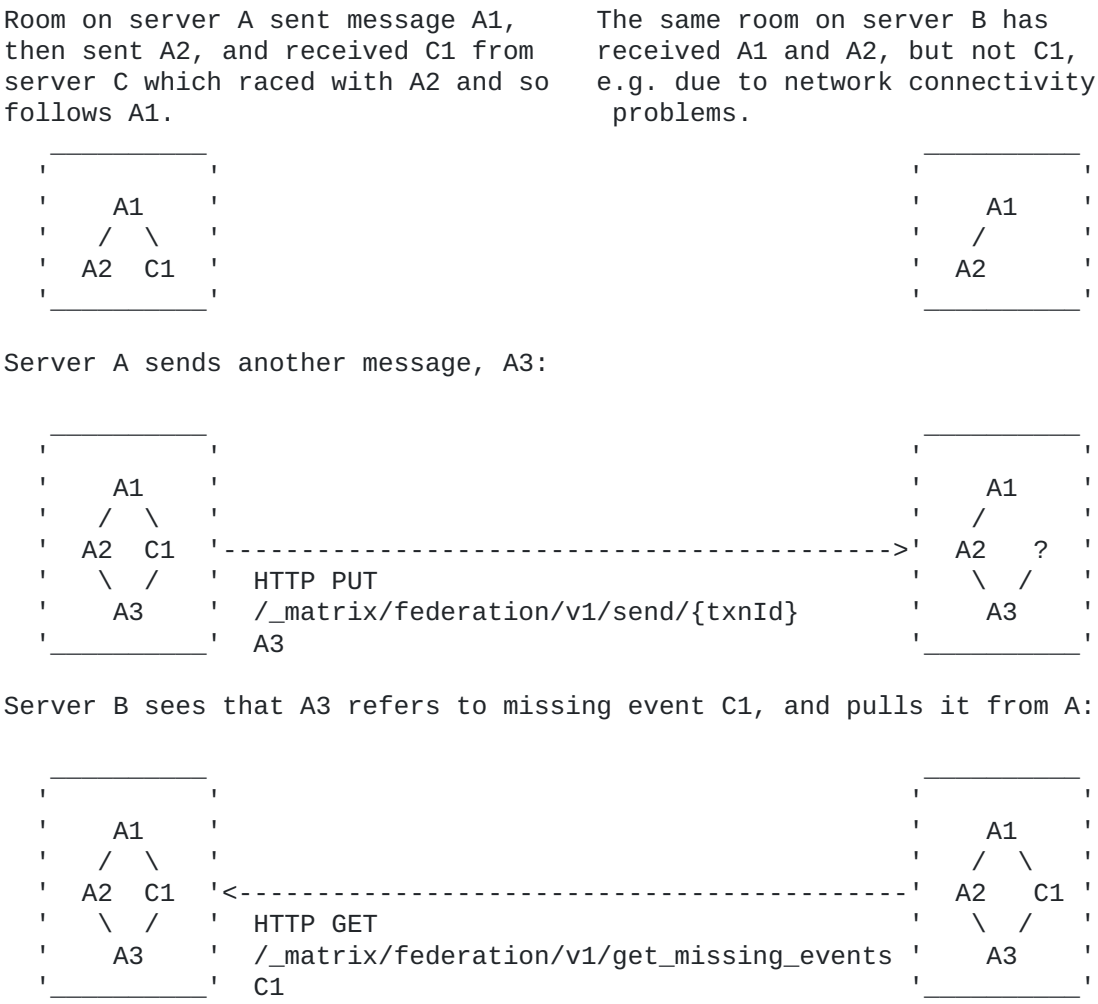
Matrix uses a set of RPC APIs (typically over HTTPS) to pass JSON objects between client and server, and server and server (federation). In its simplest form, a user sends an event to a room using the Client-Server API [CSApi], which servers then forward to each other using the Federation API [SSApi]. For an interoperable transport, the Federation API would be used. The events which users send are populated with information about previous events (as known to the local server), "authorization events" (events which prove the sender is allowed to send the event in the first place), and additional server-specific signatures (to prove authenticity) by the

server before the event is sent to other servers. Receiving servers check the authorization events against their local view of the room to decide whether the received event should be accepted, soft-failed or rejected - thus providing decentralized access control semantics.

An example workflow might be that Alice (@alice:s1.example.org) wants to invite Bob (@bob:s2.example.org) to a room over federation (because the users are on different servers). Alice's client would send an /invite [CSInviteApi] request for @bob:s2.example.org in that room, which causes s1.example.org to send a similar /invite [SSInviteApi] request over federation to s2.example.org. When Bob has accepted the invite (by joining the room, using similar endpoints), the room's state and recent history are replicated to Bob's server and sent to Bob's client. Both Alice and Bob can now send events, including m.room.message events for instant messaging cases, and their servers will build a DAG off of them. Servers by default push events to each other, but can also pull events from each other in order to fill holes in their DAG.

A key part of this eventually consistent model is that a server can go offline (for any reason, including being affected by a Denial of Service (DoS) attack, network partition, or being turned off) and other servers in the room are not affected. The other servers can continue to send events into the room and amongst each other while the other server is offline. When that server comes back online, it will have the events it missed efficiently synchronized to it by the other servers. Similarly, if the offline server was operational but unable to send events to other servers, it can continue sending its own events to the room and have those events be "merged" with the other servers' events when network connectivity is restored. Conflicts with state events (two or more changes to the room state at the same time or by temporarily diverged servers, often due to network connectivity issues) are resolved using an appropriate State Resolution Algorithm [SSStateResAlgo]. The state resolution algorithm is not covered here for brevity, and would likely be its own document.

Practically, this looks like:



Typically, get_missing_events isn't needed, given servers push all events to all participating servers by default.

Figure 1

4. Interoperability

Matrix's split of Federation and Client-Server APIs allow homeservers to implement the API surface which is most relevant for its application. For interoperability, only the Federation API is relevant. The APIs have been designed to intrinsically support load balancing and active/active horizontal scaling - for instance, it's valid for different parts of a server to race together when sending a message in a room (causing a temporary fork in the room's event DAG, same as if the race happened against a remote server), avoiding the need for global locks within the server.

The steps needed for an existing system to be interoperable with another over Matrix would mean implementing the Federation API and storing minimal information about the room's current state (state events (including members), a few of the most recent event identifiers seen, etc) and hooking that up to their existing application. An example of this happening is Gitter showcasing Matrix interoperability back in 2020 [GitterMigration].

As was the case with Gitter, existing messaging applications could deploy a homeserver using software which already exists to rapidly get connected to the Matrix network. Later stages of implementation for messaging applications might include writing proprietary software to handle application-specific traffic on one end and Matrix federation on the other, optimizing for internal scaling requirements.

5. Encryption

End-to-end Encryption is deliberately layered on top of the Matrix transport (Client-Server or Federation APIs). Currently a combination of Double Ratchet (Olm) encryption and group ratchet encryption (Megolm) is specified in the End-to-End Encryption section of the Client-Server API [CSEncryptionApi], but Matrix over MLS [I-D.ietf-mls-protocol] (with minor bookkeeping to compensate for the lack of a centralised sequencing function in Matrix) is being specified as DMLS. [DMLS]

6. Security Considerations

TODO Security. Matrix has its own threat model that needs to be described here to protect against malicious actors.

7. References

7.1. Normative References

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <https://www.rfc-editor.org/rfc/rfc1123>.

[SSApi] The Matrix.org Foundation C.I.C., "Federation API", 2022, <https://spec.matrix.org/v1.4/server-server-api/>.

7.2. Informative References

[CSApi] The Matrix.org Foundation C.I.C., "Client-Server API", 2022, <https://spec.matrix.org/v1.4/client-server-api/>.

[CSEncryptionApi] The Matrix.org Foundation C.I.C., "End-to-End Encryption | Client-Server API", 2022, <https://spec.matrix.org/v1.4/client-server-api/#end-to-end-encryption>.

[CSInviteApi] The Matrix.org Foundation C.I.C., "POST /_matrix/client/v3/rooms/:roomId/invite | Client-Server API", 2022, <https://spec.matrix.org/v1.4/client-server-api/#post_matrixclientv3roomsroomidinvite>.

[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>.

[GitterMigration]
Hodgson, M., "Gitter now speaks Matrix!", 2020, <https://matrix.org/blog/2020/12/07/gitter-now-speaks-matrix>.

[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://datatracker.ietf.org/doc/html/draft-ietf-mls-protocol-16>.

[MxFoundation] The Matrix.org Foundation C.I.C., "The Matrix.org Foundation", 2019, <https://matrix.org/foundation>.

[MxSpec] The Matrix.org Foundation C.I.C., "Matrix Specification | v1.4", 2022, <https://spec.matrix.org/v1.4/>.

[SSInviteApi] The Matrix.org Foundation C.I.C., "POST /_matrix/federation/v2/invite/:roomId/:eventId | Federation API", 2022, <https://spec.matrix.org/v1.4/server-server-api/#put_matrixfederationv2inviteroomideventid>.

[SSStateResAlgo] The Matrix.org Foundation C.I.C., "Room State Resolution | Federation API", 2022, <https://spec.matrix.org/v1.4/server-server-api/#room-state-resolution>.

Acknowledgments

TODO acknowledge.

Authors' Addresses

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

Email: travisr@matrix.org

Matthew Hodgson
The Matrix.org Foundation C.I.C.

Email: matthew@matrix.org