MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis
„Exploring ActivityPub Interoperability for Bluesky"

verfasst von / submitted by
Martin Sonnberger BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of
Master of Science (MSc)

Wien, 2026 / Vienna, 2026

Acknowledgements

Thank you very much!!

Abstract

This LaTeX template provides example on how to format and display text, mathematical formulas, and insert tables or images. There is a lot more you can do with LaTeX, for more information check out https://en.wikibooks.org/wiki/LaTeX.

Kurzfassung

Das ist eine deutsche Kurzfassung meiner in Englisch verfassten Masterarbeit.

Contents

Contents

List of Tables

List of Figures

List of Algorithms

Listings

1. Introduction

2. Background and Related Work

2.1. AT Protocol

2.2. ActivityPub

ActivityPub [5]

2.3. Related Work

2.3.1. Bridgy Fed

2.3.2. Wafrn

3. Fedisky – An ActivityPub Federation Sidecar for Bluesky PDS

Fedisky is a sidecar service that a Bluesky PDS operator can deploy alongside their PDS to enable federation with the Fediverse. It acts as a bridge between the AT Protocol and ActivityPub, allowing users on Mastodon instances to discover, follow, and interact with users on the Bluesky PDS. Fedisky is designed to be a lightweight and modular service that can be easily deployed and maintained by PDS operators, without requiring any modifications to the PDS itself. In this chapter, we will provide an overview of the design and implementation of Fedisky, including its architecture, data model, key components, and operational considerations. Fedisky's source code is available on GitHub at `https://github.com/msonnb/fedisky`.

3.1. Overview

3.1.1. System Context

Figure 3.1 shows the wider system context of Fedisky and how it interacts with other systems and users. At the core of the system is the PDS host, which runs both the ATProto PDS and the Fedisky sidecar and is managed by the PDS operator. Fedisky uses ATProto XRPC APIs to read and write records from the PDS, and subscribe to its firehose endpoint to receive a real-time stream of all new and updated records.

Fedisky exposes ActivityPub endpoints, which are used by external Fediverse instances to deliver ActivityPub content to their users and to receive incoming activities from the Fediverse. These endpoints include a Webfinger endpoint for user discovery, an actor endpoint for representing Bluesky users as ActivityPub actors, and an inbox endpoint for receiving incoming activities from the Fediverse.

To fetch Bluesky records from users on other PDS instances, Fedisky fetches records from the Bluesky AppView. In addition, Fedisky periodically polls the ATProto Constellation API[1], an external service that aggregates and indexes backlinks across the entire AT Protocol, allowing Fedisky to discover interactions such as replies from users on other PDS instances, and federating them to the Fediverse.

---

[1] `https://constellation.microcosm.blue/`
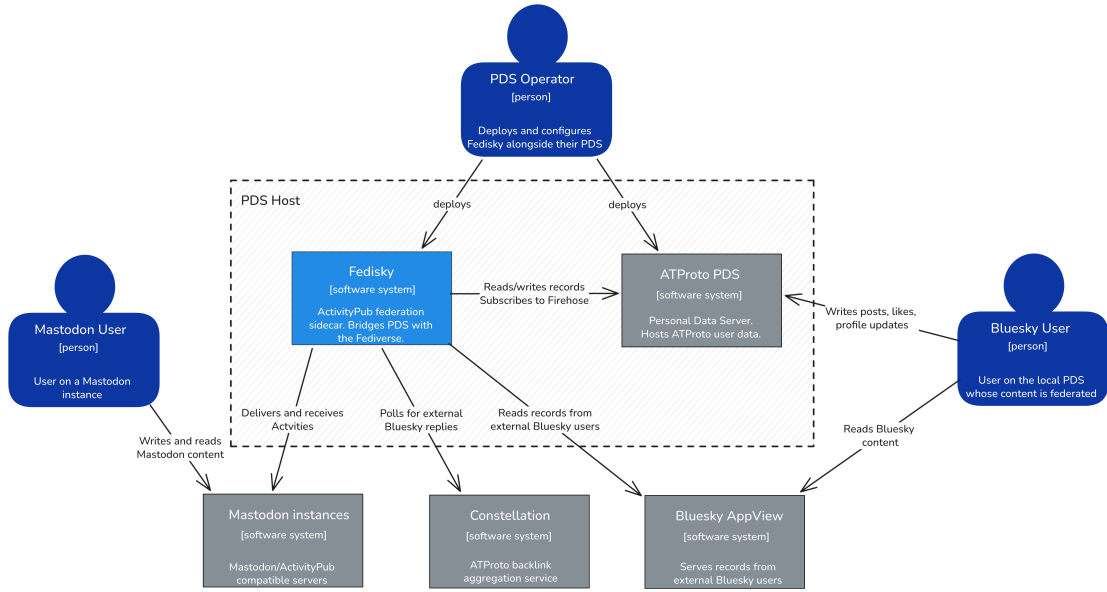
Figure 3.1.: System Context Diagram showing how the Fedisky sidecar interacts with other systems and users.

3.1.2. Technology Stack

Fedisky is implemented in TypeScript and runs on Node.js. It stores its data in a SQLite[2] database using Kysely[3] as a type-safe query builder. For federation functionality, Fedisky uses Fedify[4], a TypeScript library for building ActivityPub servers.

Fedify provides a high-level API for defining ActivityPub actors, registering dispatchers for handling incoming activities, and sending outgoing activities to other Fediverse instances. It also handles the underlying ActivityPub protocol primitives such as signing and verifying HTTP requests, providing type-safe objects for Activity Vocabulary [5] such as `Create`, `Follow`, and `Note`. In addition, Fedify includes scalability and reliability features such as retry logic for failed deliveries, a message queue for processing incoming and outgoing activities, and a KV-store for caching and storing federation-related data such as public keys and remote actor information.

Actor Discovery: WebFinger & Actor Resolution

ActivityPub Instance	Fedify Router	Dispatcher	KeyPairs Dispatcher	Database	PDS Client

WebFinger Lookup

Request WebFinger for alice@hostname

Map handle to bsky hostname

Resolve handle "alice.hostname"

Return DID for alice

Return JRD with subject and self link

Actor Fetch

Request actor profile for DID

Dispatch request to actor service

Validate DID (400 if invalid)

Skip if bridge account DID

Get profile for DID

Return profile info or not found

404 if not found

Get display name, bio, and images

Return profile text and image refs

Get avatar image URL

Get banner image URL

Return image URLs

Get actor key pairs

Get RSA key pair (if any)

Get Ed25519 key pair (if any)

Return key pairs or none

alt [missing keys]

Generate RSA key

Generate Ed25519 key

Export JWKs for new keys

Store new key pairs in database

Import JWKs for crypto keys

Return cryptographic keys

Return actor profile with keys, images, and endpoints

Return actor JSON-LD

Followers Collection

Request followers collection for DID

Get followers count

Get followers (with cursor and limit)

Return follower data

Return OrderedCollection with items and nextCursor

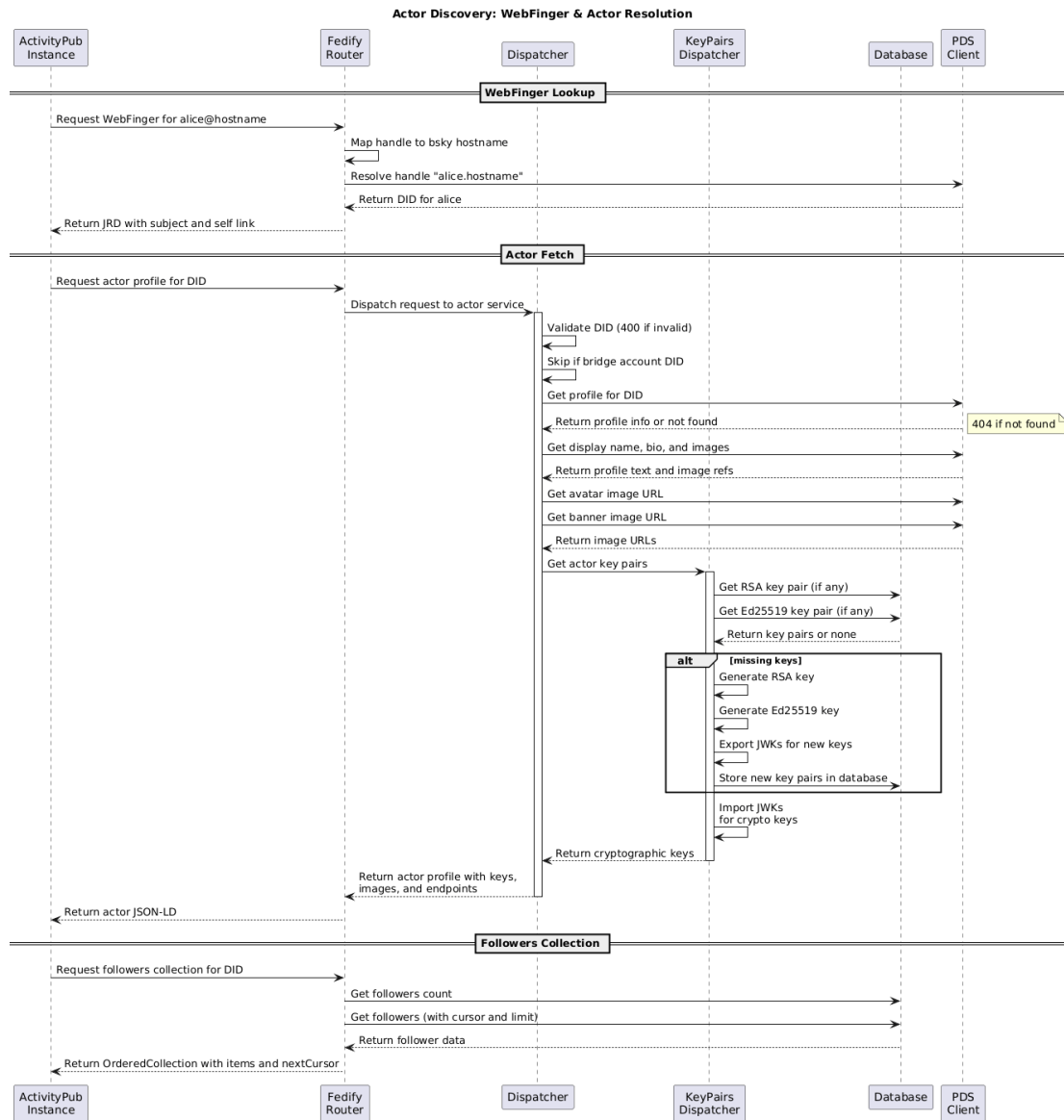| ActivityPub Instance	Fedify Router	Dispatcher	KeyPairs Dispatcher	Database	PDS Client

Figure 3.2.: Sequence diagram showing the actor and identity discovery flow when a Mastodon user tries to follow a Bluesky user on the PDS.

3.1.3. Subsystems

3.2. Core Flows

3.2.1. Actor and Identity Discovery

When a Mastodon user tries to follow a federated Bluesky user on the PDS, the Mastodon instance needs to discover the corresponding ActivityPub actor for that user in order to send the follow request. This flow is illustrated in Figure 3.2. The Mastodon instance first queries the WebFinger endpoint with the user's handle (e.g. `@alice@fedisky.social`) to discover the corresponding ActivityPub actor URL. Fedisky first constructs the AT-Proto handle by prepending the localpart (in this case, `alice`) to the PDS's hostname (e.g. `fedisky.social`), resulting in `alice.fedisky.social`. Note that the Activity-Pub handle domain and PDS domain do not have to match, but in this example we use the same domain for simplicity. Fedisky then resolves this handle using the PDS's `com.atproto.identity.resolveHandle` API, which returns the corresponding DID if a matching user is found. If a user is found, Fedisky constructs the ActivityPub actor URL using the user's DID, resulting in `https://fedisky.social/users/{did}`. This URL is returned to the Mastodon instance in the WebFinger response, as shown in Listing 3.1. In addition to the actor URL, the response also includes references to the user's profile page and avatar.

```json
{
  "subject": "acct:alice@fedisky.social",
  "aliases": ["https://fedisky.social/users/did:plc:
    n3jcidccul6u3lif5q4rh42x"],
  "links": [
    {
      "rel": "self",
      "href": "https://fedisky.social/users/did:plc:
    n3jcidccul6u3lif5q4rh42x",
      "type": "application/activity+json"
    },
    {
      "rel": "http://webfinger.net/rel/profile-page",
      "href": "https://bsky.app/profile/alice.fedisky.social"
    },
    {
      "rel": "http://webfinger.net/rel/avatar",
      "href": "https://fedisky.social/xrpc/com.atproto.sync.getBlob?did=
    did%3Aplc%3An3jcidccul6u3lif5q4rh42x&cid=
    bafkreiahfo2qkotgr475q2kx4psbffojwup6fkchnw43y2i44uhanmy2em",
      "type": "image/jpeg"
    }
  ]
```

---

[2] `https://sqlite.org/`
[3] `https://kysely.dev/`
[4] `https://fedify.dev/`
[5] `https://www.w3.org/TR/activitystreams-vocabulary/`

```
20 }
```

Listing 3.1: WebFinger response for `@alice@fedisky.social`

After receiving the WebFinger response, the Mastodon instance can then query the actor endpoint to fetch the user's actor document, which includes the user's profile information, public keys, and inbox URLs. The object conforms to the Activity Vocabulary `Person` type as defined in [4]. To construct the actor document, Fedisky fetches the user's profile record form the PDS using the `com.atproto.repo.getRecord` API. From the profile record, Fedisky extracts the user's display name, description, as well as avatar and banner images. Finally, Fedisky retrieves the user's public keys from the database. If no keys exist yet, Fedisky generates new RSA and Ed25519 key pairs and stores them in the database.

In an effort to link accounts referring to the same identity, Barrett [1] proposes the use of account links. In this approach, instead of relying on a separate "meta account" that links all the user's accounts together, accounts reference each other. Platforms can then use these references to show highlighted links to these other accounts on different platforms. Fedisky follows this approach and includes the user's ATProto URI in the `alsoKnownAs` field of the actor document, allowing Fediverse instance to link the ActivityPub actor back to the original Bluesky user and profile.

3.2.2. Outbound: ATProto Post to ActivityPub Note

3.2.3. Inbound: ActivityPub Activity to ATProto Record

3.3. Data Model

3.3.1. Database Schema

Identity & Cryptography

- `ap_key_pair` – Stores the cryptographic key pairs used for HTTP signature signing. Each local PDS user gets two key pairs generated on first access: one RSA for compatibility with older ActivityPub implementations, and one Ed25519 for modern servers. The keys are stored as PEM-encoded strings.

- `ap_bridge_account` and `ap_bluesky_bridge_account` – Singleton tables that store the credentials for the two bridge accounts (see section 3.4).

Social Graph

- `ap_follow` – Records which ActivityPub actors follow which local PDS users. This is the core table for activity delivery, as it determines which users should receive which activities based on their follow relationships. It stores the follower's inbox URL and shared inbox URL for efficient delivery.

Content Mapping

- `ap_post_mapping` – Maps ATProto post URIs to their original ActivityPub Note IDs and author information. This table is essential for correct reply threading.

- `ap_external_reply` – Stores external Bluesky replies from other PDS instances that have been federated via the Constellation processor. Primarily for deduplication as the Constellation API is polled repeatedly.

- `ap_monitored_post` – The work queue for the Constellation processor, which tracks which posts need to be checked for new external replies.

Engagement Tracking

- `ap_like` and `ap_repost` – Stores incoming ActivityPub `Like` and `Announce` activities targeting local posts. The tables main purpose are to track engagements for display (e.g. showing like counts) and batching engagement for DM notifications.

3.3.2. Migration Strategy

Migrations are implemented using Kysely's built-in migration system, which allows us to define schema changes in a type-safe manner. Each migration is defined as a numbered TypeScript file exporting `up()` and `down()` functions. On service startup, pending migrations are automatically applied in order. This approach allows us to evolve the database schema over time while ensuring data integrity and providing a clear history of schema changes.

3.4. Bridge Account System

3.4.1. Why Two Bridge Accounts?

Fedisky uses two special "bridge accounts" to facilitate bridging between the AT Protocol and ActivityPub. The first is the Mastodon bridge account, which is hidden from users in ActivityPub and is used to post incoming Fediverse replies as ATProto posts on the PDS, so that they appear in the Bluesky user's thread and can be replied to and interacted with like normal posts. Its handle and display name can be configured by the operator, with the default being `mastodon.{hostname}` and "Mastodon Bridge" respectively.

The second one is the Bluesky bridge account, which is used to federate replies from Bluesky users on other PDS instances to the Fediverse. This allows users on Mastodon to see and interact with replies from users on other PDS instances, which would otherwise be invisible to the Fediverse. The Bluesky bridge account's default handle is `bluesky.{hostname}` and display name is "Bluesky Bridge", again configurable by the operator via environment variables.

3.4.2. Attribution Model

Since both bridge accounts post content on behalf on users and do not carry any user identity in their handle or display name, we need to ensure proper attribution of content to the original authors. For incoming Fediverse replies posted by the Mastodon bridge account, we include the original author's handle in the first line of the post content, e.g. "@bob@mastodon.social replied:", followed by the actual reply content. Similarly, federated Bluesky replies from third-party PDS instances sent by the Bluesky bridge account include an attribution line with the original author's Bluesky handle, e.g. "alice.bsky.social replied:", followed by the reply content. In both cases, the handle is a clickable link to the original profile, allowing users to easily find and follow the original author if they wish. Additionally, since Mastodon content is formatted in HTML, we need to ensure all content is properly escaped to prevent Cross-Site-Scripting (XSS) vulnerabilities.

3.5. Inbound Federation

3.5.1. ActivityPub Endpoints

Using Fedify together with its Express HTTP integration[6], we expose the following ActivityPub endpoints:

- `GET /.well-known/webfinger`
  The WebFinger endpoint for user discovery. When a Mastodon instance encounters a handle such as `@alice@fedisky.social`, it will query this endpoint to discover the corresponding ActivityPub actor URL. Fedisky resolves the handle using the PDS's `com.atproto.identity.resolveHandle` API, and if a matching PDS user is found, constructs an ActivityPub actor URL using the user's DID as the unique identifier, i.e. `https://fedisky.social/users/{did}`.

- `GET /users/{did}`
  The actor endpoint. Returns an ActivityPub `Person` object representing the Bluesky user with the given DID. Includes the user's inbox, outbox, followers, and following URIs, as well as their public keys (both RSA and Ed25519), and profile information such as display name and avatar. It also includes a `alsoKnownAs` field with the user's ATProto URI, in order to link the ActivityPub actor back to the original PDS user and identity. The Mastodon bridge account does not have an ActivityPub actor representation.

- `POST /users/{did}/inbox`
  The inbox endpoint for receiving incoming activities from the Fediverse. This is where we receive activities such as `Follow` requests. When an activity is received, we verify the HTTP signature to ensure it is from a trusted source, and then dispatch it to the appropriate handler based on the activity type.

---

[6]`https://fedify.dev/manual/integration#express`

- `POST /inbox`
  A server-wide shared inbox endpoint that some Fediverse instances support for more efficient delivery. Fedisky also supports this endpoint for incoming activities, and dispatches them in the same way as the user-specific inbox.

- `GET /users/{did}/outbox`
  Paginated collection of posts, likes, and reposts, aggregated from the PDS using its `com.atproto.repo.listRecords` API. This allows Mastodon instance to fetch the user's content and engagements for display on their profile and timelines.

- `GET /users/{did}/followers` and `GET /users/{did}/following`
  Paginated collections of followers and following, based on the `ap_follow` table. This allows Mastodon instances to display the user's followers and following lists, and to determine which users they should receive activities from.

- `GET /posts/{uri}`
  Endpoint for fetching a specific post by its ATProto URI, used by Mastodon instances to fetch the content and metadata of a post when displaying it or when a user clicks on a link to the post. Fedisky resolves the ATProto URI using the PDS's `com.atproto.repo.getRecord` API, and returns an ActivityPub Note object with the post content, author information, and any media attachments.

- `GET /nodeinfo/2.1`
  The NodeInfo[7] endpoint providing metadata about the Fedisky instance, such as software name and version, and supported features. This is used by Mastodon instances to determine compatibility and capabilities of the Fedisky bridge.

---

[7] `https://nodeinfo.diaspora.software/`

3.5.2. Inbox Processing

3.5.3. Reply Bridging

3.5.4. Post Mapping for Reply Threading

3.5.5. Engagement Notifications

3.6. Outbound Federation

3.6.1. Firehose Processor

3.6.2. Record Conversion

3.6.3. Activity Delivery

3.6.4. External Reply Discovery

3.7. Conversion Layer

3.7.1. Post Converter

3.7.2. HTML ↔ Rich Text

3.7.3. Media Handling

3.7.4. Edge Cases

3.8. Observability & Operations

3.8.1. Wide Events Logging

3.8.2. Testing

Unit Tests

End-to-End Tests

3.8.3. Deployment

4. Results

5. Discussion

5.1. Sidecar Architecture

As described in the previous chapter, Fedisky is designed as a *sidecar container* as defined in [2, Ch. 3] and runs alongside its *application container*, a Bluesky PDS. Sidecars are a common architectural pattern, where a secondary container provides auxiliary functionality to the primary application container, often without the application knowing. Sidecar containers share system resources and are scheduled to run in sync with the application container [2, p. 21]. This design allows Fedisky to operate independently while still closely integrating with the PDS.

An alternative design could have been a centralized bridging service that bridges *all* of Bluesky into the Fediverse, which is realized by Bridgy Fed (subsection 2.3.1). Compared to a centralized bridging service, a sidecar container running alongside a single Bluesky PDS instance, can share resources with the PDS, most importantly the server's hostname, which allows it to operate under the same domain and thus provide a single shared identity for users across Bluesky and Mastodon, e.g. `alice.fedisky.social` in Bluesky becomes `@alice@fedisky.social` in Mastodon. Additionally, the sidecar pattern allows for operator autonomy, where each PDS operator can choose to deploy Fedisky independently, or not. It also reduces the scope of the bridge, as it only needs to handle traffic for a single PDS instance, simplifying development and maintenance.

Another approach would be to fork and modify the PDS itself, thus integrating bridging functionality directly into the PDS. We considered this approach initially when designing Fedisky, but chose the sidecar pattern instead for several reasons. Forking the PDS would significantly increase the maintenance burden and development complexity, as one would need to keep up with upstream changes constantly to ensure compatibility. The sidecar avoids this by only consuming the PDS's public APIs, which are stable contracts based on AT Protocol Lexicons [3]. In addition, the sidecar pattern allows for better separation of concerns, as all bridging logic is contained within the sidecar, rather than the PDS being responsible for both AT Protocol and ActivityPub compliance and therefore tightly coupling two protocol implementations, making each harder to reason about and test in isolation. Furthermore, the sidecar gives more deployment flexibility, as operators can choose to start or stop the bridge independently of the PDS. Since communication between the sidecar and the PDS is done through HTTP APIs, the sidecar can be developed a different technology stack, and is automatically compatible with any PDS implementation that adheres to the AT Protocol specifications.

6. Conclusion

Bibliography

[1] Ryan Barrett. *Bridging Identity with Account Links*. A New Social. Aug. 13, 2025. URL: `https://blog.anew.social/bridging-identity-with-account-links/` (visited on 02/19/2026).

[2] Brendan Burns. *Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services, Using Kubernetes*. Second Edition. Sebastopol: O'Reilly, 2024. 200 pp. ISBN: 978-1-0981-5635-0.

[3] *HTTP Reference | Bluesky*. URL: `https://docs.bsky.app/docs/category/http-reference` (visited on 02/12/2026).

[4] James Snell and Evan Prodromou. *Activity Vocabulary*. W3C Recommendation. W3C, May 2017. URL: `https://www.w3.org/TR/2017/REC-activitystreams-vocabulary-20170523/` (visited on 02/19/2026).

[5] Jessica Tallon and Christine Lemmer-Webber. *ActivityPub*. W3C Recommendation. W3C, Jan. 2018. URL: `https://www.w3.org/TR/2018/REC-activitypub-20180123/` (visited on 02/19/2026).

A. Appendix

here you can put further things you want to add like transcripts, questionnaires, raw data...