ActivityPub in Go

Lessons from a journey

2 July 2018

Cory Slep

Preface

Preface

Goals are to answer the following questions:

- "Why create a library?"
- "What challenges are there when implementing ActivityPub?"
- "What challenges are specific to golang?"
- "What is go-fed and its future?"

Preface

Non-goals of this presentation are:

- "What is ActivityPub and ActivityStreams? Can I have an introduction?"

This deserves to be its own presentation, blog post, or website. This presentation will focus on the meta: How I went about learning and the conclusions I drew. Some general technical advice will be given out with concrete examples, but nothing comprehensive.

- "How can I use `go-fed/activity` to build a federated application?"

This also deserves to be its own presentation. You can use that library to create any federating app you desire. Or to add federating functionality into an existing application.

Preface

This is not a blog post so readers can clone it and read at their own leisure.

This presentation is meant for individual reading. Be prepared for lots of words!

Quick Definitions

Quick Definitions

An *actor* is the basic unit of identity in ActivityPub that can send or receive information. (ex: Accounts on Mastodon are *actors*).

An *end-user application* is a piece of ActivityPub software that has *actors* that belong to human users (ex: Mastodon).

A *library* is a piece of software that software engineers use to build *end-user applications*.

A *server* is a machine running an *end-user application*.

A *client* is a machine a human uses to interact with their account that lives on a *server* (ex: a phone, a browser).

A *peer* is one of two *servers* talking to each other via ActivityPub.

An *Activity* is the basic unit of information sent via ActivityPub to *servers*.

Motivation

Motivation

I would usually tinker with various self-started projects that were interesting only to me. This was my way of exercising my coding skills for personal enjoyment but not for solving a problem.

However, I've always wanted to solve something in the open source world. Having dreams are fun (until gritty reality seeps in).

I just needed a kick in the pants to get moving.

Motivation

Cue scene of a person huddled indoors in snowy Switzerland's winter of 2017.

Motivation

I was in the mood to improve my blog. I was wondering what would let my blog be a part of a social network, POSSE style. This lead me to ActivityPub, which had not yet been blessed as a full W3C recommendation.

To choose this was a significant risk. I had no prior contacts to the W3C social comittee. I had no context over how close it was to being finalized. I had no insight into its adoption.

Motivation

I did basic research online by catching up on previous W3C meeting notes and joining Mastodon to engage with some of the authors and implementors.

Major kudos to Mastodon for taking the early leap to adopt it. This convinced me to keep examining ActivityPub. Without a major adopter like Mastodon, I probably would not be here typing this.

Motivation

So I made the call to implement ActivityPub as a library in golang, after confirming no other implementation in golang existed. Otherwise, I would have wanted to reach out to the previous author to get the kind of knowledge I am dumping here.

Little did I know what I was signing myself up for.

Motivation

So I created the go-fed organization and began to read a lot about:

- JSON-LD (https://www.w3.org/TR/json-ld)
- ActivityStreams (https://www.w3.org/TR/activitystreams-core)
- ActivityPub (https://www.w3.org/TR/activitypub)

There are other additional normative and non-normative documents, too:

- Social Web Protocols (https://www.w3.org/TR/social-web-protocols/)
- ActivityStreams Terms (https://www.w3.org/ns/activitystreams)
- ActivityStreams Vocabulary (https://www.w3.org/TR/activitystreams-vocabulary)
- Authentication & Authorization (https://www.w3.org/wiki/SocialCG/ActivityPub/Authentication_Authorization)

For Mastodon interoperability:

- WebFinger (https://tools.ietf.org/html/rfc7033)

Why a library?

Why a library?

I had no interest in creating an end-user application for two reasons:

1) Due to my personal circumstances, I needed to ask my employer for the copyright to the code. This is easier to get when working on a library.

2) I wanted to ease the burden of writing a new federated application and/or retrofitting in federating functionality.

Why a library?

Also, there are incentives to build a library:

I could focus on the specification and ignore conventions. Let the end-user applications worry about the conventions and let the library solve the specification conformance problem.

Reusability: a statically-typed language with no generics meant that the startup cost of getting a conforming golang implementation is very high, and amortizing that cost across multiple applications significantly reduces the barrier to entry (hopefully).

Why a library?

But the consequences of choosing to build a library:

There is the risk that no one uses the library. A risk resulting in a sad, grim reality.

Imposter syndome actualization: people realize I am faking being a software engineer.

Binary and build time bloat. I knew that code generation was going to be a handy tool (more on this later), which results in large amounts of code to compile. This could drive people away from adopting this library and towards a *roll your own* solution.

Why a library?

From reading the specs alone, I made the early decision to focus on three areas corresponding to various documents I had read:

- `activity/vocab` for the complete ActivityStream vocabulary API
- `activity/streams` for a "simpler"* ActivityStream vocabulary API
- `activity/pub` for the Social and Federated API

*I consider myself to have failed in making this API actually simpler.

What challenges are there when implementing ActivityPub?

What challenges are there when implementing ActivityPub?

ActivityPub allows different clients and servers to communicate with each other without requiring all these machines to be owned by the same person or organization. These machines federate with each other. No two machines contain all data stored in this federation.

Therefore the federation is distributed and decentralized. As opposed to something like Bitcoin, which is distributed but still centralized (everyone has the same global ledger).

What challenges are there when implementing ActivityPub?

These are the major meta-things I had to come to terms with when examining the federation:

- Understanding the ecosystem (going from zero knowledge to passing familiarity)
- Understanding the least-implemented-peer (most-buggy-peer) problem
- Convention vs Specification
- Time is not kind

These problems have a lot in common with the early web and browsers, so lessons learnt there are also applicable here.

I'll go into these in this presentation. Later, I'll address some more technical points.

What challenges are there when implementing ActivityPub?

Understanding the ecosystem (going from zero knowledge to passing familiarity)

Unfortunately, as someone completely new and unfamiliar to this space, I had a lot of trouble really understanding the nuances of the federated behavior. The specification is vague and broad enough to be flexible. This is great as new behaviors and data can be shared within the ActivityPub framework. It makes for dense reading and lots of re-reading, however.

But this flexibility comes with tradeoffs...

What challenges are there when implementing ActivityPub?

Understanding the least-implemented-peer (most-buggy-peer) problem

Keeping the individual pieces of the federation in synchronous understanding of one another is hard. It means that the most limited of implementations will be the one to limit overall progress. As long as users of federative software care about the software, this should not be a problem. In practice, this can be difficult as the number of users scales up.

What challenges are there when implementing ActivityPub?

What do I mean by "the most limited of implementations"?

The least-feature-rich or most-divergent-from-ActivityPub software will be the one that peer software engineers will have to look at and decide "do I care about breaking them and all their users?"

What challenges are there when implementing ActivityPub?

The simplest example today may be the fact that an `Activity` can contain multiple `objects` within it, as permitted by the ActivityStreams Core Vocabulary. However, as long as one major implementation can only handle one, this degree of flexibility cannot really be used. And by convention, it isn't.

Of course, nothing stops someone from saying "screw that" and doing it anyway, breaking the Federation for others.

What challenges are there when implementing ActivityPub?

Convention vs Specification

As a consequence, when developing federating end user applications, one must take great care to understand the limitations of the major players as well. This is the conventional part of the federation.

However, library implementors only have to worry about providing the functionality outlined in the specification. The library will be within spec, or it wouldn't be an implementation. Library implementors therefore absolve themselves of the responsibility of the conventional part of the federation.

What challenges are there when implementing ActivityPub?

This doesn't preclude library implementors from participating in the convention issues of the federation. But it does mean libraries can only guide their users towards those conventions, not enforce them.

Applications that end-users use, therefore, have power proportional to the size of their user base. The power to keep conventions in-spec or the power to diverge from the spec completely.

What challenges are there when implementing ActivityPub?

Time is not kind

Finally, there is the concept of *time*. Once a federated end-user application is released with limited behavior, there is no way to force an administrator to upgrade to software with the newer behavior. This problem does not change regardless whether the end-user application is using its own ActivityPub implementation or an ActivityPub library.

It's the Internet Explorer problem.

What challenges are there when implementing ActivityPub?

Another problem with time is when a federating server goes down. Like the web today, that canonical part of the federate network is simply missing. However, copies of its data may live in its federating peers.

What challenges are there when implementing ActivityPub?

Now that we've examined the meta-challenges of the federation, let us look at some general technical challenges.

I will be going over these at a very high level. Future presentations should address the finer points.

What challenges are there when implementing ActivityPub?

Before we get to them, note that there are *two* solutions to some of the challenges presented:

- A SocialAPI solution for client-server interactions
- A FederatedAPI solution for the server-to-server interactions.

What challenges are there when implementing ActivityPub?

Now, there are three broad problems to solve in order to arrive at an ActivityPub solution:

- Serialization and deserialization into JSON-LD ActivityStreams
- Core Activity semantic behaviors for actors, objects, and collections
- Additional core behaviors (ex: Inbox forwarding, Public objects)

What challenges are there when implementing ActivityPub?

Furthermore, there are a breadth of optional behaviors to implement. Some are specified in the spec as optimizations while others are outside of the spec and are conventional:

- Shared inboxes for bundling HTTP requests
- Extended Activity semantic behaviors for actors, objects, and collections
- Other behaviors (ex: Authentication, Authorization)
- Testing Everything

What challenges are there when implementing ActivityPub?

OK, time to gloss over these at a very high level...

What challenges are there when implementing ActivityPub?

Ready?

What challenges are there when implementing ActivityPub?

Set?

What challenges are there when implementing ActivityPub?

```
go
```

What challenges are there when implementing ActivityPub?

Serialization and deserialization into JSON-LD ActivityStreams

ActivityStreams is built on a stack of technology like

1) JSON
2) JSON-LD
3) ActivityStreams

Note however that fully understanding JSON-LD is not required, as the ActivityStreams specification is more strict about the JSON-LD structure of its data. Implementations can do the following instead:

1) JSON
2) ActivityStreams

This kind of serialization and deserialization is easier than also having to deal with JSON-LD's compaction and other algorithms.

What challenges are there when implementing ActivityPub?

Note also that unknown fields in the original JSON *must be preserved* when re-serializing.

Finally, JSON's `null` does have significance to ActivityPub.

These all can add additional complexity.

What challenges are there when implementing ActivityPub?

Core Activity semantic behaviors for actors, objects, and collections

Predefined actions are to be done by an initial set of Core `Activity` types. These must be supported for basic functionality in ActivityPub. However, the exact behaviors differ between the SocialAPI and the FederateAPI.

What challenges are there when implementing ActivityPub?

Additional core behaviors (ex: Inbox forwarding, Public objects)

There are additional *on my honor* functionalities needed to ensure that an end-user application does not break the federation for others or its own members. Devils lurk in these details when it comes to authentication & authorization, retries, and so on.

What challenges are there when implementing ActivityPub?

Shared inboxes for bundling HTTP requests

This is an optional optimization that helps prevent DOSing a peer who has a lot of recipients on an `Activity`. Since only one `Activity` can be sent per HTTP connection, sending a single `Activity` object to 2+ actors who live on the same peer means 2+ HTTP requests.

Shared inboxes reduces that number to 1.

What challenges are there when implementing ActivityPub?

Extended Activity semantic behaviors for actors, objects, and collections

There are additional behaviors that are optionally suggested to be implemented by the ActivityPub specification. These often require application-specific meaning to justify their use.

What challenges are there when implementing ActivityPub?

Other behaviors (ex: Authentication, Authorization)

At the time the specification was drafted there was no consensus as to how to best tackle security. Thus, it is up to the community and conventions.

Today, it seems HTTP Signatures and/or OAuth 2 are favored.

What challenges are there when implementing ActivityPub?

Testing Everything

There are some repositories with test data as well as test.activitypub.rocks (https://test.activitypub.rocks). However, this testing also requires some honest answers and is not fully automated. Many seem to enjoy testing in production. Be careful and don't DOS peers.

What challenges are there when implementing ActivityPub?

Whew! Time to look at golang specifics now.

What challenges are specific to golang?

What challenges are specific to golang?

We are going to wade through the same six items from before, but in golang specific detail.

What challenges are specific to golang?

Serialization and deserialization into JSON-LD ActivityStreams

Serializing and deserializing to and from JSON is by far the most difficult and frustrating problem, due to several design choices made by the golang language.

What challenges are specific to golang?

First, there are no generics. I know, I hear you say "Duh, it's golang." There is only one way to metaprogram. Kind of. And that is with code generation. Why does this matter? Because JSON-LD permits a JSON field to be a JSON object `{}`, a JSON array `[]`, or an IRI which is just a JSON string `""`. And if it is an array, apply recursively! No amount of struct tagging with `encoding/json` will solve this.

What challenges are specific to golang?

Second, `encoding/json` only supports JSON's `null` when deserializing to `map[string]interface{}`. This tiny detail can cause large-scale havoc in the very few places it is needed, depending on the implementation design.

What challenges are specific to golang?

Thirdly, a reasonable person would think ActivityStream types map to a class inheritance hierarchy, which are either golang struct embeddings or interfaces. So all the parent ActivityStream type's properties flow to the child type, and so on. And all is well in the world.

Well, you can imagine where this is going.

What challenges are specific to golang?

When reading the ActivityStreams specification, everything seems to gracefully inherit everything from a parent type. However, this is not guaranteed. Nowhere is it stated that the specification's parent and child relationship map to any programming language's notion of subclassing, inheritance, interfaces, or embedding. Which is reasonable, but a bummer.

Unfortunately for our reasonable person, not only is this kind of inheritance not guaranteed, it is actually violated. Putting them in a horrible position if not thought of ahead of time.

What challenges are specific to golang?

The core type `IntransitiveActivity` inherits everything from `Activity` except `object`. And yet it is still a very real subtype of `Activity`.

Thus, ActivityStreams does not easily map to a class hierarchy, struct embedding, or interface. Only by checking its actual `type` property and consulting the specification can one know for sure.

What challenges are specific to golang?

So, the base complexity of everything else is dictated by how the ActivityStreams API looks.

What challenges are specific to golang?

Note that there *is* a beacon of hope here. Since ActivityStreams dictates the JSON-LD structure, an implementation can forgo a lot of the JSON-LD compaction and expansion algorithms and go straight from raw JSON to an ActivityStream representation.

Thus, we can skip knowing about JSON-LD entirely.

There is the concern about dynamically knowing what to do with new ActivityStream types via the JSON-LD `@context`. But since this would require coding knowledge during compilation, or fancy ML inference to judge what new fields should meaningfully do in an end-user application, this point is not the most compelling for golang.

What challenges are specific to golang?

No major new golang concerns for the following:

- **Core Activity semantic behaviors for actors, objects, and collections**
- **Additional core behaviors (ex: Inbox forwarding, Public objects)**
- **Shared inboxes for bundling HTTP requests**
- **Extended Activity semantic behaviors for actors, objects, and collections**

These all basically depend on the complexity of the ActivityStreams API.

What challenges are specific to golang?

Other behaviors (ex: Authentication, Authorization)

HTTP Signatures are not a part of any core golang library. As always, be very careful with how you use `crypto/subtle` if deciding to roll your own. Or, use libraries written by others.

I'm not familiar with OAuth 2 golang libraries, so this may also require careful design considerations.

What challenges are specific to golang?

Testing Everything

Part of golang's appeal is its tooling for things like tests. I implore you to take advantage of it to make up for the difficulty of testing live. There's really no reason to have untested code in golang.

What challenges are specific to golang?

Whew!

Still here? Wow! You rock! Onto the last section.

What is go-fed and its future?

I created go-fed on a whim since from the very beginning I wanted to have the library be under an open-source license, free of copyright issues, and not tied to me being a BDFL.

I have no experience running an open-source organization, but am eager to have an ActivityPub golang community grow. The project living under the go-fed name is really just a symbolic gesture up to this point, but with a very real intent:
I wish for those who are slow to trust and are wary of long-term ownership to look at it as an initial goodwill gesture of things to come!

What is go-fed and its future?

And if the consensus is that ActivityPub in golang is too difficult, then no harm was done to the federation or to others.

Instead, I'll finally get to work on my blog.

:)

Thank you

Cory Slep
https://github.com/go-fed/activity (https://github.com/go-fed/activity)
cjslep@gmail.com (mailto:cjslep@gmail.com)
@cj@mastodon.technology (http://twitter.com/cj@mastodon.technology)