Introducing a data sync layer

Status Network
protocol

---

oskarth 2018-12-11 12:44:01 UTC #1

Introduction

We want Status to be decentralized and censorship-resistant. Thus we must design our systems to work in offline environments, as well as for mesh settings.

Right now messaging is relying on 3rd party services like mail servers. They fulfill a role, relaying messages between nodes that aren't online at the same time, but it is fragile. The system should work without it.

Data sync layer

By introducing a data sync layer in our protocol, we can improve on this. It decouples message sync state from transport and message content logic. It thus gives us more freedom going forward:

- to swap out Whisper as a transport privacy protocol
- build various communication mechanisms on top of this data sync layer

All a data sync layer cares about is keeping track of the state of messages. The above layers, like 1:1 chat or Tribute to Talk, then uses this sync protocol. The main idea is that each node is self-sufficient and equal in a true peer-to-peer fashion. They keep track of everything they know, including what they think other nodes know.

Rough protocol stack

Layer	Purpose	Example
Sync Clients	End user functionality	1:1, 1:N, TtT, Public
Data Sync	Syncing data/state	BSP
Secure Transport	Confidentiality, PFS, etc	Status PFS spec
Transport Privacy	Metadata protection	Whisper, PSS, Mixnet
P2P Overlay	Overlay routing, NAT traversal	devp2p, libp2p

Transport/network layer below. There's also a key agreement/exchange protocol.

Specific proposal

Introduce **Bramble Synchronization Protocol (BSP)** into the Status app. This acts as a in-between layer between Whisper and 1:1/group/public chat.

BSP doesn't know what a valid message is, or what the relationship between messages is. That's dealt with by data sync clients.

How does syncing work?

A device stores **synchronization data** for each message and each peer. This includes information such as: if a peer is holding a message, if ACK is required, when a message was last sent, etc.

Peers synchronize data by **exchanging records**. A record contains a header along with a payload.

The header includes protocol version and record type. A record type is things like:

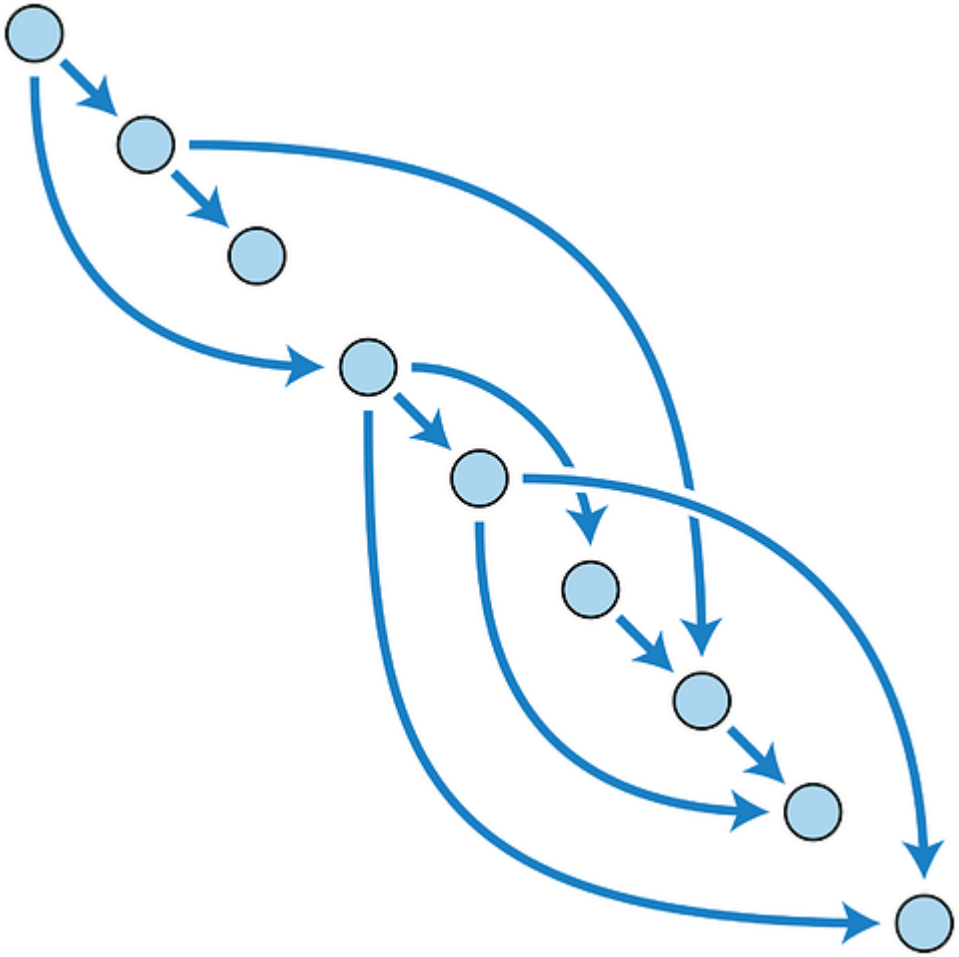
- ACK (list of messages to be ACKed)
- MESSAGE (group id/sync scope, timestamp and message body)
- OFFER (offer to share some messages)
- REQUEST (list of message ids desired).

It doesn't concern itself with what is in the message body, as that's a data sync client concern.

Message graph

A message is only delivered once all its dependencies have been delivered. This ensures a consistent view of messages. This means each message must have some information about "parent"/message dependencies. Exactly how this is provided is up to the specific data sync client.

This provides for casual consistency. Using topological sort, the user is presented with a consistent view. Since there's no single source of truth in a p2p system, this is done with partial information. How this is rendered is up to the end client.



What would this mean for data sync clients?

Ideally this written as a specification in e.g. protobuf. This ensures multiple end clients can implement it and be compatible. Another way to think of this is as a form extensions. E.g. one end user client might only care about 1:1 chat or public chat, and ignore all the other mechanisms.

This requires some logic for what is inside a message and how you share. Each message needs to keep track of its parent messages, aka its dependencies.

Examples:

- **Private group chat**
- **Introducing users to each other**.

This is relevant for e.g.: Tribute to Talk, Hamburger Chats, Public Visibility Stake, **Spam prevention** etc.

Briefly on why BSP in particular

- Supports similar use cases
- Is actually used and reasonably specified
- Doesn’t have anything immediately incompatible with us

The main alternative would be something homegrown. This reasoning might be incorrect if:

- there’s a better, more well-designed alternative out there somewhere
- there’s something undesirable in BSP that would warrant our own spec (at this layer)

Briefly on network incentivization

This section isn’t yet fleshed out, but merits a mention.

Since this abstraction provides information about messages ACKed and clients can choose who to offer message, this can work similarly to Bitorrent’s economic abstraction layer. Exactly how this information is leveraged is outside of the scope of this post, but it’s worth pointing out.

Tangential considerations

Some things that may or may not be worth considering at this layer.

This doesn’t preclude the use of mailservers. Synchronization frequency etc can be titrated as well.

- Whisper signature key in messages and implications for messages being agnostic
- How to deal with long public chats with a lot of message dependencies
- UX®: How to render threads in group/public chat to accurately depict state
- Using topological sort to get a canonical representation of past events for all end users.
  – A la git merge of two branches (or not? what does this mean for casual consistency?)
- Group chat different encryption algorithm (non-pairwise)
- Privacy preservation implications for underlying layers

- [cammellos]: bandwidth usage & offline/asynchronous messaging performance (i.e how reliable is retrieving a message chain when devices are often offline, with spurts of online activity, mobile), as those needs to be tested with a potentially much larger audience than briar's use case, as briar seems to be more focused on close proximity interactions.

Further reading

- Log based comms: **Log-based comms**
- BSP: **https://code.briarproject.org/briar/briar-spec/blob/master/protocols/BSP.md**
- Briar Client examples **https://code.briarproject.org/briar/briar/wikis/home#clients**

Summary and next step

The introduction of a data sync layer simplifies the protocol design. It also gives us more options to do interesting stuff on top of and at the bottom of it. It is a good fit for true p2p and delay-tolerant/mesh networks due to it not making too many assumptions. Assumptions such as: messages being received, or relying on a 3rd party to sync messages.

Next steps

- Protobuf specification
- PoC for 1:1 chat

Additionally, this above work should be phrased as research and specification and documented as part of swarm framework. Either as part of Protocol Engineering Swarm or as part of Core/Data Sync.

Thoughts and feedback welcome!

Hello Stratus - toying around with Nimbus and QML

Data sync - next steps and considerations

Bootstrap Nodes Smart Contract

Hello Staples - Swarm PSS/Feeds PoC for #buidlweek

(Mostly) Data Sync Research Log

Minimal Viable Data Sync Research Log

adam 2018-12-14 08:41:44 UTC #2

I have a few questions:

1. Do I need to trust all my peers? If I want to sync messages with them, I must tell them which messages I need. This can reveal which public chats I participate in. It seems worse than bloom filters.
2. If BSP is a layer above Whisper, peers need to know how to open messages. How is that gonna work (encryption agreement)?
3. It might be pretty inefficient in case of public chats where one can lag hundreds of messages and discovering message IDs might require N requests in the worst case scenario.

Furthermore, BSP might not be a perfect for our use case because:

- “is an application layer data synchronisation protocol suitable for delay-tolerant networks.” – while we aim for short latency,
- “If a message is shared, the device will synchronise the message to any peers with which it shares the group.” – in terms of 1-1 chats, it seems like two peers will create a group. If both are offline how do we want to share the state? In this case, it seems like they need to include one more peer (which is online for most of the time) to allow syncing the messages if one participant is offline.

To me, BSP looks cool for 1-1 and group chats because the client tends to connect to peers with which it shares the same groups (and trusts them as well) so the sync is efficient. If some participants need offline communication, they just need to include a participant which is constantly online and it all makes sense.

What does not make sense, in my opinion, is using the same approach for public chats. I believe that we should completely separate these two scenarios and have different strategies.

oskarth 2018-12-17 04:23:52 UTC #3

Thanks for the critical and great questions **@adam** !

> 1. Do I need to trust all my peers? If I want to sync messages with them, I must tell them which messages I need. This can reveal which public chats I participate in. It seems worse than bloom filters.

What do you mean by trust exactly? If you actively participate in a chat other participants in that chat can see your public key. People not participating in a chat won't be able to see who is participating in that chat, as communication would be happening over a transport privacy layer, such as Whisper or a mixnet. It wouldn't be a direct TCP connection between peers like `markTrustedPeer` for mailservers.

It's also important to remember that who you choose as a peer to sync with is up to a data sync client. So it'd still be possible to just silently listen in, assuming there's no 'join channel' semantics. Bloom filters seems to me to be a different component. You would still use them as today, and you can leverage them for offering messages efficiently.

> 2. If BSP is a layer above Whisper, peers need to know how to open messages. How is that gonna work (encryption agreement)?

Good question, and we'll have to figure out the specifics here, and it depends on what you mean by a message exactly. A Whisper message contains a record, and that has some record ID. You don't need to be able to open that payload in order to state whether you can have that piece of data or not.

> 3. It might be pretty inefficient in case of public chats where one can lag hundreds of messages and discovering message IDs might require N requests in the worst case scenario.

You are right that this isn't a priori optimized for public chats. But I don't think the situation is quite that bad:

1. It doesn't necessarily require N requests. If you have another peer, and they know you don't hold previous messages, they can still offer or directly send these messages to you optimistically. They can also choose to only offer the last or .They can also communicate which messages they can offer via a bloom filter, a la Tribler's Dispersy (**https://dispersy.readthedocs.io/en/devel/system_overview.html**)
2. One can imagine for public chats that only reaching a certain length (either in time or chains) is OK. It isn't quite clear to me how to best render this in a UI, but it seems solvable. And you can communicate this in the UI (parent IDs exist but not fetched).
3. For even further history, one could imagine canonical snapshots that a client can choose to trust or not.

> - "is an application layer data synchronisation protocol suitable for delay-tolerant networks." – while we aim for short latency,

What do you mean by short latency exactly? I don’t think they are mutually exclusive. Delay-tolerant just means the ability to operate over hetergenerous networks that lack continuous internet activity. There’s no inherent tension here, and we can still optimistically send things like we do today.

However, it is true that by aiming to strongly preserve privacy we need to introduce some form of latency at the transport privacy layer. We do this with Whisper, and the same would be the case with a Mixnet. This precludes things like RTC. But if you have a different transport and different environment, like Bluetooth, this wouldn’t be a problem as I see it.

> - “If a message is shared, the device will synchronise the message to any peers with which it shares the group.” – in terms of 1-1 chats, it seems like two peers will create a group. If both are offline how do we want to share the state? In this case, it seems like they need to include one more peer (which is online for most of the time) to allow syncing the messages if one participant is offline.

Correct. And you can also include several peers for this. With things like single-use reply blocks (SURBs) **https://katzenpost.mixnetworks.org/docs/specs/sphinx.html** you can get stronger privacy guarantees for this, but IMO that’s a later concern.

> To me, BSP looks cool for 1-1 and group chats because the client tends to connect to peers with which it shares the same groups (and trusts them as well) so the sync is efficient. If some participants need offline communication, they just need to include a participant which is constantly online and it all makes sense.

> What does not make sense, in my opinion, is using the same approach for public chats. I believe that we should completely separate these two scenarios and have different strategies.

You are right that it isn’t by default optimized for public chat. But I don’t think it warrants a different strategy altogether, just a different way of using the same layer. Here’s an example that isn’t too far from what we have today.

You have a public chat foo with a lot of participants. You are node A. You also use a “mailserver” like node E which include as a peer.

Each node holds the following state: Hold/Ack/Request flag, send Count and send Time (HARCT).

A sends a message foo (appends to own log). Since A is online and uses Whisper, it sends this message straight away (same as now).

A POV:

Message	Node E
a0	H=0,A=1,R=0,C=1, T=t1

Now, E has a lot of peers connected to it and has some previous history. It receives the message from A.

E POV:

Message	Node A	Node B	Node C	…
b0	H=0, A=1, R=0, C=0, T=now	H=1…	…	…
c0	H=0, A=1, R=0, C=0, T=now	H=1…	…	…
b1	H=0, A=1, R=0, C=0, T=now	H=1…	…	…
c1	H=0, A=1, R=0, C=0, T=now	H=1…	…	…
a0	H=1, A=0, R=0, C=0, T=never	H=0…	…	…

Now, E can do one of several things. This might depend on heuristics, or the specific relationship with A (friend who paid or foe who misbehaved in the past?). This choice is **entirely up to the data sync built on top of BSP**.

First, it'd ACK the a0 message. Then one thing it could do is to SEND b1 c1.

A then receives these, and it attempting to render them notices that there's a lack of parent messages b0 and c0. So A choose to REQUEST them. Whereupon E, behaving nicely, sends them.

Alternatively, one could imagine that E sends their whole history to A straight away. Or perhaps it is paginated, based on timestamp etc. E could also send a bloom filter saying which messages they have, a la Dispersy. All of these are optimizations that eventually lead to a casually consistent state.

Additionally, A could have multiple peer. Each peer would roughly have the same state, as they store a partial graph of the entire history. Notice that it doesn't matter here if a single mailserver (aka peer) goes offline, or missed a specific set of messages (though this probability is minimized, since they'd likely have previously requested the full history).

These are all trade-offs that can be reasoned about, assuming there's a fundamental base layer to reason about these things.

---

adam 2018-12-18 08:40:51 UTC #4

> oskarth:
>
> as communication would be happening over a transport privacy layer, such as Whisper or a mixnet. It wouldn't be a direct TCP connection between peers like `markTrustedPeer` for mailservers.

Ok, that's clear but this point was related to the fact that BSP needs more specific information like requesting and acking particular message IDs while Whisper operates only on bloom filters. My concern is that it will be easier to track who is interested in what messages and link peers within the same conversation.

> oskarth:
>
> A Whisper message contains a record, and that has some record ID. You don't need to be able to open that payload in order to state whether you can have that piece of data or not.

Right but then the only advantage will be using a formal spec to sync data (which is good :)). Messages will be broadcasted following the Whisper spec so no gain in terms of traffic usage will be achieved.

The scenario looks like this, I believe:

1. I intend to send a message `a1` to a group chat (1-1 is still a group chat as probably some additional peer being constantly online is required),
2. With BSP I use `SEND` record type and wrap it in a Whisper envelope with some topic `T0`, my signature and encrypt it with recipient's public key,
3. I broadcast the message to my other peers,
4. Only peers from this group chat can decrypt and read my message,
5. After they do, they will send `ACK`.

Now, if a peer from the group chat gets online, how does it request messages? From BSP I don't see a record type allowing that. It only makes sense, if this peer connects to some other participant of a group chat (probably that additional peer being constantly online) and then can receive `OFFER` record.

To sum up, my mind struggles to see any significant advantages of using BSP and Whisper at the same time 😀 BSP + some network topology based on peers you have active conversations with (this is a crucial feature of scuttlebutt) + mixnets seems like something more revolutionary that might bring us on a new level.

If the goal is to get rid of mail servers (or reduce their significance), I also don’t see how it can be achieved. The problem is that if REQUEST record type is broadcasted, who is responsible for fulfilling it and how can we know that it will be done honestly? With trusted peers, this problem does not exist.

---

oskarth 2019-01-14 04:26:03 UTC #5

Follow up PoC here: **https://github.com/status-im/status-research/blob/master/data_sync/sync.py** (code). Basic (and sometimes wrong) protobuf spec here: **https://github.com/status-im/status-research/blob/master/data_sync/sync.proto**

Initial write-up with two example scenarios that encodes unreliable delivery and mobile/desktop nodes with different characteristics: **https://notes.status.im/THYDMxSmSSiM5ASdl-syZg?both**
