Secure Scuttlebutt: An Identity-Centric Protocol for Subjective and Decentralized Community Applications

ABSTRACT

TODO rework, shorten

Secure Scuttlebutt (SSB) is a novel peer-to-peer information-centric event-sharing protocol and architecture for social apps. In this paper we describe SSB's features, its operations as well as the rationale behind the design. We also provide a comparison with a traditional information-centric networking architecture and discuss SSB's limitations and evolution opportunities.

ACM Reference Format:

. 2019. Secure Scuttlebutt: An Identity-Centric Protocol for Subjective and Decentralized Community Applications. In *Proceedings of ACM XYZ conference (XYZ'19).* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

A simple conceptual architecture for community applications consists of a global data pool to which every person can contribute and where every person can tap into the shared data – data sharing being the purpose of such applications. This model still is valid if one adds access control to the picture, either tied to the data (encryption giving access to content only to entitled holders of the decryption keys) or encrypting data in transit (login and TLS). Facebook and other centrally organized social app service providers fit well under this global data pool model but have been strongly criticized for abusing their central provisioning position. The "decentralized web movement" [13] is the most visible technical response to this critique, pointing out implementation alternatives.

One of these alternatives is a project called Secure Scuttlebutt (SSB) that started in 2014. After several iterations of protocol design and implementation, SSB has become a stable service for over 10,000 users offering them rich media community applications with strong cryptographic protection (end-to-end encryption and metadata privacy) and running in pure peer-to-peer mode.

Selective Complete Log Replication

SSB relies on the core insight that each participant is only interested in a *subset* of the global data pool, thus it is feasable to locally store all the data a participant is interested in. To partition the data pool, all data is associated with the *identity* that produced it. Participants select their slice of the data pool by specifying the set of identities whose data they care about. This creates a "social graph" along whose edges data flows (Figure 1).

Each participant can publish data to their single-writer, append-only log. This choice of data structure allows efficient replication and verification of the integrity of received data. Replicating these larger slices of the data pool comes with an unusual set of tradeoffs, discussed throughout the paper. As it turns out, replicated logs form a solid foundation for implementing many classes of applications.

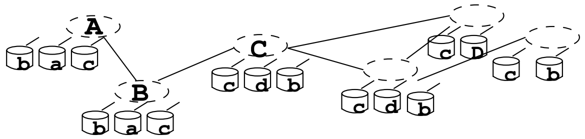


Figure 1: SSB's "Internet of Identities" – Users A, B and C replicate logs (*a, b, …*) based on whom they follow i.e., content flows through intermediary friends: *C* does not follow *A*, hence has no log *a*. *A* and *B* follow each other such that when *A* follows *C*, *A* will get *C's* log *c* via *B*.

Subjective Reader

Because replication in SSB is selective and driven by a peer's social graph, different end devices will have access to different sets of log replicas, leading to different views of the world, which we call a "subjective reader" approach. SSB's considers this a desirable property: each peer is free to consider data sources of its own choosing instead of having to feed from a centrally provisioned or otherwise converged view. While it is possible to implement consensus protocols over SSB, or to designate central data aggregators from which many peers consume the consolidated outputs, the SSB network itself deliberately doesn't offer consensus services nor central content (directories etc). In Section 3.3 we will show the implications of this technological choice on the structure of distributed applications that can only read from and write to local logs.

XYZ'19, Sep 2019, Place, Country

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Novelty

Putting complete replication of individual append-only logs at the core of SSB's protocol avoids several hard problems in distributed systems. **First**, it is a radically decentralized approach requiring only minimal specification-level coordination among the participants but no run-time checks or configuration management. **Second**, although append-only data structures are well known for their benefits and are at the core of crypto currencies' consensus finding, SSB uses logs without any consensus properties. The issue is deliberately sidestepped, but all necessary building blocks are provided to higher layers. **Third**, crafting a cryptographic ID system and maintaining a social graph that informs routing creates a very narrow filter: it implements a receiver-driven approach (similar to e.g. Named Data Network, but at a higher object granularity) where data only flows where it is needed and without imposing any fine-grained request/reply protocol. Instead, any dissemination technology is adequate, including broadcast and push mode, because network elements can verify data validity (due to the logs' signed entries) and monotonicity of the updates without additional key and certificate material, guaranteeing that only conformant traffic is propagated at any forwarding step. **Fourth**, it makes every peer a publisher by design. This property goes beyond the decentralized approach like DAT [] or IPFS [] which assume that there exist replication servers but keep the separation between a data transport network and a server layer. **Last but not least**, log replication leads to a distributed system with inherent high resilience as any communicating element MUST replicate the others' logs. In traditional distributed systems, coordinating the data persistence as a basis for resilience is often an add-on task, or requires at least a special recovery service.

Comparing SSB to NDN

A core contribution of this paper is a direct comparison of SSB with *named data networking* (NDN). Our analysis reveals strong incompatibilities in at least two dimensions. The first is the content dissemination strategy where NDN is well known for its PULL pattern while SSB follows a PUSH model.

	PULL	PUSH
name-centric	**NDN**	*NDN-over-SSB?* *Sect. 4.5*
identity-centric	*SSB-over-NDN?* *Sect. 4.3*	**SSB**

The second difference is more subtle and seems to be rooted in what NDN considers the main focus of networking. In NDN, repositories are an implicit assumption on which the scalability claims of NDN rest, and consequently, these repositories — or, to be more precise, their global prefixes and all names attached to them — are points of centralization. SSB, on the other hand, has no global constructs, true to is decentralized point of view, and can be called *identity-centric* in order to contrast it from *name-centric* NDN.

We dedicate a full Section 4 to this discussion where we examine whether one could either "drag" SSB into NDN (SSB-over-NDN), or what SSB would need to emulate NDN (NDN-over-SSB). But in both cases it seems that none of the emulated ICN styles could benefit much from the hosting substrate, thus remain incompatible. Still, there could be an intermediate territory ("SSB-aware NDN", or vice versa) where in the future a synthesis of the two approaches may emerge.

Structure of this paper

TODO (!)

...

2 SSB ARCHITECTURE AND PROTOCOL

In SSB, each user is identified by an ed25519 [6] keypair. Since anybody can generate a random keypair with very low probability of multiple peers generating the same keypair, no central authority is necessary for introducing users to the system. Conceptually, SSB is a network of identities that connect to each other (physical topology) and share mutual or unilateral interest in the other peer's data (social graph), as shown in Figure 1. A node running the SSB protocol is called a *relay*.

The *single-writer append-only logs* of SSB consist of messages that include a *backlink* in form of a cryptographic hash of the previous message (or a special indicator for the first message of a log). The most distinguishing feature from a regular blockchain is that each SSB user unilaterally maintains their own log and cryptographically signs all their (and only their) messages. Messages whose backlink points to a message in a different log (i.e. from a different author) are considered invalid and will not be processed by SSB peers.

Concretely, each log entry, which may not exceed 4 KB, contains the following pieces of data [21]:

- the *backlink* to the previous message, or a null value
- the public key of the message's *author*
- the *sequence number* of the message, which must be one more than the sequence number of the previous message, or exactly one if it is the first message of the log
- a claimed *timestamp* of when the message was created
- a *hash* indicator that specifies the concrete hash function that was used to compute the backlink
- the *content* of the message
- the author's *signature* over all the previous data

The *content* is a JSON object that must contain a *type* key that serves as a hint for how the content should be interpreted. SSB enforces that the content is valid JSON by rejecting any malformed entry. Instead of structured JSON data, the content can be a base64-encoded string of encrypted data, together with a tag that signifies which encryption algorithm was used.

SSB defines a format for encoding specifically the public keys of identities and the hashes of messages and blobs (see below) as strings. This allows applications to scan the content of log entries from other peers for such references, e.g. in order to create database indices (see Sect. 3.3).

Replication

The principal function of SSB relays is to connect to other relays and exchange log *updates*. To do so, they maintain a point-to-point encrypted [3] overlay network over which they run a gossip protocol. When two relays start gossiping, they exchange the current sequence numbers of all logs they are interested in. If a peer receives a lower sequence number for a feed than it sent, it transmits the log messages that the peer is lacking. If at a later point a new message of the feed becomes available to a relay, it is automtically *pushed* to the connected peers. As an optimization, this *eager* gossip is only performed over the edges of a spanning tree, which itself is maintained via the plumtree [16] protocol. In classic peer-to-peer fashion, clients (leaf nodes) are no different than relays except that they usually include some graphical user interface and perform application logic.

In addition to this primary replication mechanism, SSB provides two other ways of exchanging information. *Blobs* are content-addressed pieces of free-form data, typically images or other documents larger than the 4KB limit, that are referenced from log entries but are not part of any log. They are not widely disseminated automatically, but rather fetched on demand via a simple request-flooding protocol. *Out-of-order messages* are a similar mechanism to address and fetch log entries on demand via their hashes.

SSB Relays as an Application Platform

Beyond replicating logs and checking the validity of update messages, an SSB relay offers an API to its peers. Peers can host arbitrary programs that issue RPCs to the relay over an IPC mechanism. The exposed functionality includes appending to the peer's log, reading from logs, specifying which logs a relay should replicate, and fetching blobs and out-of-order messages.

The reference implementation of the SSB relay [28], written in JavaScript, also includes a mechanism for loading *plugins* into the relay to extend its functionality. There are a few default plugins: Conceptually these can be thought of as client programs that are always running. Of particular importance are those that guide the replication process. The *friends* plugin [29] scans the relay's log for specific messages that indicate which other authors the identity *follows*. The plugin then instructs the relay to fetch and replicate these logs. These other logs might of course also contain some of these messages. The friends plugin transitively replicates these friends-of-a-friend logs as well, up to a configurable maximum distance in the friends graph.

Beyond "befriending" other users through `follow` messages, a SSB user can control the shape of its social graph via special `block` messages which limit the transitive log replication. Both the `follow` as well as `block` messages are overheard by relays, through scanning all received log updates, and inform them about where updates should be delivered (or not). Decisions about whom to replicate can be –and in the current system is– guided by the content of the very data that is replicated. By storing the relevant information inside the author's log (as opposed to a local or central database), other peers can use this informtion to guide their decisions.

The overlay network also makes use of logs to store configuration information, in this case the SSB_ID-to-IP_address mapping: operators publish the static ip addresses of highly-available relays (called *pubs*) to their log. When a SSB leaf node needs to connect to the overlay, the responsible plugin can scan any locally available log replica for this information.

SSB's Layered Architecture

It is worth noticing that SBB spans three independent layers of protocols. The most fundamental protocol is the message format: All peers need to agree on what consitutes identities, valid messages, and how to compute hashes to address messages and blobs. This is the "thin waist" of SSB (see figure 2).

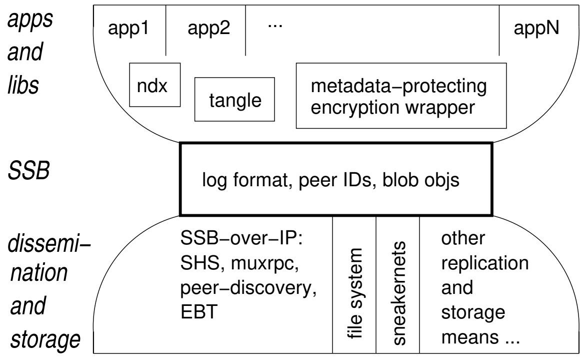


Figure 2: Secure Scuttlebutt's protocol stack.

Next (below) is the specific mechanism by which relays exchange data. The default RPC mechanism is one option, but alternative mechanisms such as distribution via a sneakernet could also be deployed. Different peers that do not share a common replication mechanism could still interact indirectly, as long as there are some relays that understand multiple replication protocols.

In other words, the core logical replication protocol by which a relay serves its clients is fully independent from the actual dissemination protocols. And finally, the publishing and interpretation of data in the form of messages is again a separate affair that is layered on top of the thin waist.

3 DISTRIBUTED APPS AND DATA STRUCTURES OVER SSB

TODO: rewrite/update this introduction *Wordsmithing needed:* The goal of this section: show how SSB’s distributed apps can reconstruct the app’s state by parsing the logs, and operate on the app state by writing to the peer’s log. We give more details for SSB’s user directory and also list and quickly characterize other existing SSB apps. A crucial (performance) aspect is incremental indexing which we expand on in the subsection about the Kappa approach. More node-local support in form of libraries and conventions at the level of log entries is discussed in the subsection on tangles.

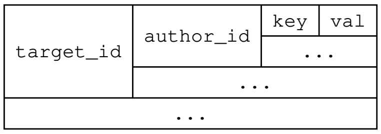
3.1 Example: SSB’s user directory

‘`about`’ is SSB’s user database i.e., an application that associates cryptographic IDs with (typically) human-readable attributes. A single log entry format has been defined to this end:

```
'content': {
  'type'  : 'about',
  'about' : target_id,
  attr_name : attr_value   // multiple times
}
```

The `about` app scans all logs for all entries of type ‘`about`’ and constructs a database as shown in Figure 3, retaining the most recent attribute assignment found. To each target user ID we associate a directory where key/value pairs are collected on a per author basis (which is extracted from the log entry’s envelope).

Currently the `name`, `description` and `image` attributes are understood by most SSB user interfaces and are used to substitute or decorate the cryptographic ID. If `target_id` and `author_id` are identical, the attributes are self-chosen, and otherwise given.



target_id	author_id	key	val
		...	
	...		
...			

Figure 3: SSB’s user directory data structure (after extraction from the logs).

In terms of CRUD actions, creation happens once a new SSB peer adds its own `about` entry to its log; reading the user database is performed on the above data structure; updates are expressed by adding an `about` entry –regardless whether it relates to the peer itself or to another peer– to one’s own log and all peers updating their extracted database; deleting a user entry is not possible, at least not directly (one would have to block that user ID as well as all IDs which wrote an update for that user).

There can never be confusion about the sequence or scope of attribute assignments because they are orderd by the log (and thus in time) and kept separate, per author ID. Note also the presence of the “subjective reader” property: The content of a peer’s user database is dependent on its position in the social graph. The “subjective” mindset is also visible by letting every user assign attributes to anybody, leaving it to the user interface (and human viewer) to select which of the self-chosen or given display names and images is most suitable for a given ID.

3.2 Profiles of other selected SSB apps

Multiple applications have been written by contributors and are used daily by the SSB community. We briefly present selected examples because they represent alternatives to well-known services and they illustrate both opportunities and challenges of communication through replicated append-only logs.

Git-ssb [15] is an alternative to GitHub [2] that replicates git-based version-controlled code repositories through contributors logs. It provides an encoding of repositories in SSB logs, a bridge to interoperate with git repositories, and a web-based viewer to browse repositories. The object model of Git [9], based on immutable hash-referenced objects organized in a chain of commits, has a similar structure to SSB’s logs, making the mapping natural. Hash objects are blobs and commits are messages in individual append-only logs. Other git operations, such as creating a repository, creating merge commits, creating a branch, or requesting the merging of an alternative branch (*pull-request*) to a core maintainer, are all SSB messages. This model provides automatic distribution of code repositories and their updates through the replication of SSB logs. Many developers can also update and perform operations on the *same* repository, as defined by its creation message, independently. Consensus on the "official" master branch and its latest commit is enforced through social coordination because the developers of the community know and trust each other. Nonetheless, in case of concurrent updates to the same branch in the same repository [23], the git bridge will create multiple branches when a local git repository is updated. A user can then resolve the fork by merging the diverging branches and updating the repository. Referencing both concurrent updates in a later merge commit in effect resolves the ambiguity through a tangle extension (Section 3.4).

Ssb-chess [17] is a correspondence chess application in which players can invite one another to play, alternatively share their next move until the game ends, and external observers can comment on the game. The core data structure that represent a game is a linked list with nodes representing chess moves alternating between the two participants' logs. The detection of invalid moves is performed by the user interface using a validation library, because participants are trusted to only publish valid moves on their feed. This latter assumption was made because games are played between friends in a non-competitive setting. In effect, the validity of the latest move is implicitly confirmed by the next player choosing to follow with another move because if they were to disagree on the validity, the conflict can be handled outside the game and the game can be abandoned. Moreover, a chess application is easy to encode in append-only logs because the rules of chess preclude concurrency, i.e. at any time there is always only one of the two participants that is permitted to modify the state of the chess board by making a move. The game state also cannot be corrupted by external parties because only the participants, explicitly mentioned in the original invitation, are allowed to modify the state of the game. Any non-participant extending the game with their own move is simply ignored.

Gatherings [11] are alternatives to Meetup [1] that enables participants to signal their intention to attend or not attend to physical events. Gatherings can be public, in which case anyone that replicates a log will see them, or private, in which case only explicitly-invited people will be notified of the event and may signal their intention. A gathering is defined by its creation message but otherwise has no fixed properties. Anyone that has a reference to the creation message may change its properties, such as location, start and end dates, description, and image, by publishing an update message. The value of those properties are the most recent set by anyone. Initially, recency was determined by the time of creation, as reported by the user's client implementation (*self-stated creation time*). While this required trust in other users, practice has shown that the hypothesis was reasonable. To be more robust to potential invalid timestamps however, some client implementations have started using the time at which message updates are *received*, then disambiguate using the self-stated creation time.

These applications show how the SSB communication model greatly simplifies the infrastructure required to build social applications since none of the previous examples have to deal with issues of distribution of messages. The fact that a small community with a dozen or so of core developers, which are self-funded and working mostly voluntarily, could produce alternative applications that work well enough to be used daily suggests the SSB communication model does make the implementation of common social applications simpler.

3.3 Running Distributed Applications over Replicated Logs

"Infrastructure-less" distributed application as presented above become possible because central servers can be fully replaced by each peer working on its local set of replicated logs. In this subsection we discuss the particularity of this approach and its constraints.

A common pattern of SSB's applications is that they heavily rely on local database support for organizing the data contained in the logs. Typically a map-reduce strategy is used where the map phase filters the logs and the reduce actions computes the latest application state.

In the user directory application (Sect. 3.1), the filtering is done by selecting only `about` entries for a specific target ID and the reduce action consists in accumulating the latest key-value pairs such that a more recent key-value pair replaces an older one if it was signed by the same author_id. The size of the replicated logs, although locally stored, would lead to very long response times if the map-reduce would be executed at render-time. Instead, almost each application will build indexes and aggressively cache state that was already aggregated. Should the indexes ever become corrupted (e.g. because the user interface app crashed in the middle of a complex indexing step), they can be fully regenerated from scratch, an approach that has also become known as the Kappa architecture [].

An important aspect is whether the reduction step can be done in an incremental fashion by reusing previously computed application state. For example, counting the "likes" that a post receives works fine: incoming log extensions are indexed and if they are of the like type, the counter corresponding to the referenced message in incremented.

Other applications, however, may need a *full* re-evalution of the reduce function each time the underlying index changes. An example for this case is a chat in form of sequence of post messages: if some peer was added to the set of followed peers, its log gets incrementally replicated, and so are the posts of this peer. For each incoming new post, which may have been written very long ago, one has to insert it at the right place. This problem is shown in Figure 4 where a log entry has not yet been replicated to a client and, once it arrives, has to be properly inserted into the application-level data structure.

A simple solution (adopted in some SSB client software) is to use the timestamp claimed by the author of the post, and in this case one can reuse the existing time-sorted list and insert

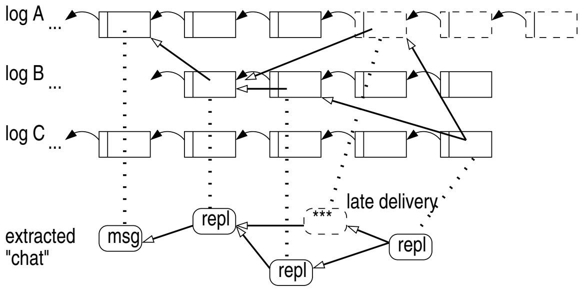


Figure 4: Example of extracting application data (in form of a tangle) and dealing with not-yet-delivered data.

the new post. However, because an author could lie about the timestamp, the reduce function should do a topological sort of the tangle based on the causality relationship with other posts and their replies. Insertion into the dependency graph may or may not lead to having to rerun the sort on the whole graph of postings. Clearly, the lack of a central server hosting the reference list of posts and being able to record a post's submission time, leads to more complex client software that must prepare for and defend against a broad range of adversarial data found in the logs.

Storing this transitive closure of application-level graphs in a space-efficient way such that it can be quickly queried and efficiently updated is a difficult but well-studied problem in the database literature [14, 30]. It should be possible to leverage some properties of SSB to get higher-quality results. The immutability of messages reduces the number of cases when dynamically maintaining the index structure, and the fact that each feed forms a total order can be utilized to efficiently encode the relation. Designing and implementing a general framework for performing causal ordering queries on SSB messages would be both an interesting research topic and a powerful tool for building applications

3.4 Synchronization and Eventual Consistency

SSB's basic log replication service synchronizes peers in a consistent way: due to the hash chaining, events (represented by log entries) will be delivered in the order they happened and replica content will be consistent. This does not instantly lead to consistent shared data structures, though, if the corresponding events are spread over multiple logs. Instead, the natural guarantee is that of *eventual consistency* where all peers will see the same reduced application state (if they share the same log set) after sufficient replication progress.

Eventual consistency is the hallmark of Conflict-free Replicated Data Types (CRDTs, see [26]) which are directly applicable to the SSB setting as they only assume a reliable and (sometimes) in-order delivery of update messages. Potentially, CRDTs permit to implement global data structures featuring eventual consistency without coordination effort (thus are fully scalable). The caveat here is that SSB peers do not necessarily see all involved peers due to their position in the social graph which controls replication wherefore consistency is always modulo that fact. For example, like counts will be eventually consistent with respect to the same set of followed peers but not globally, at least if they are directly counted. Other applications relying on reduction via set union may learn from state that stems from beyond the circle of followed peer. More research is needed to understand the constraints brought by the combination of coordination-less interaction with partial log replication, but SSB's rich set of applications used on a daily basis is an encouraging sign that eventual consistency in combination with subjective replication is a "good enough" basis for real decentralized services.

4 COMPARING SSB WITH NAMED DATA NETWORKING (NDN)

After a brief introduction to NDN we compare and relate SSB to NDN in three different ways: layering SSB on top of NDN, layering NDN on top of SSB, and a hybrid mode where NDN forwarders are made SSB-aware. Some of the encountered problems point to questions that apply to NDN as well SSB.

4.1 Named Data Networking

NDN is a receiver-driven datagram access protocol: The receiver has to request content by name –in a so called `Interest` packet– and at most one matching content is returned in a corresponding `Data` packet. The Data packet includes the content's name and is signed such that a forwarding node can verify the correctness of the name-to-content binding (content was not tampered with). Checking the validity of a signature requires additional certification data which a forwarding node can fetch using the standard `Interest/Data` packet pattern. Validated data packets are typically cached such that subsequent requests for the same name can be served from in-network memory.

```
--> I('/ndn/some/item')
<-- D('/ndn/some/item', data, signature)
```

In order to facilitate routing of Interest packets towards a data source or a data repository, NDN uses a hierarchical name space. Routing rules are based on name prefixes, typically aggregating all data items made available by a publisher. In a forwarder node, incoming Interest packets are matched against these prefixes on a longest-prefix matching basis, yielding the interface(s) to where an Interest has to be forwarded. Interests for the same name that arrive close in time are deduplicated using a PIT (pending interest table).

On the return path, a data packet is copied to all interfaces from where a corresponding interest came in, and the PIT entry is deleted.

In such a pull-only communication model, streaming protocols –as well as fetching content that is larger than the 4KB datagram size– use name prediction (e.g., sequence numbers as part of the name): Several Interests are sent ahead without waiting for the first Interest to be answered. If name prediction is not available, manifests [5] can be used i.e., a datagram is returned in lieu of the content which contains a sequence of names pointing to single Data packets or further sub-manifests.

4.2 Layering (or Merging) of SSB and NDN

In this paper we examine three different configurations: Whether NDN can be used as a transport network for SSB, whether a "SSB-aware" NDN layer would work better, and whether SSB could be used as a support for NDN networking. Figure 5 shows the three different scenarios which we expand in the following subsections. The purpose is to shed light on the sometimes implicit assumptions behind SSB and NDN.

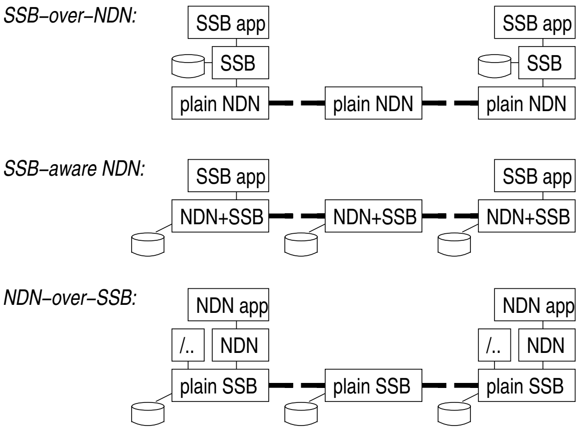


Figure 5: Three different layerings of SSB and NDN, as discussed in this paper

4.3 SSB over NDN

SSB's smaller than 4KB log entries map perfectly well to NDN if one abstracts away from the wire format: A SSB log entry for $\langle id : seqno \rangle$ could be named by
`D('/ssb/logs/<id>/<seqno>', content, signature)`
where `content` includes the name of the previous log entry. A SSB client wanting to replicate a log from scratch would issue Interest packets starting at `I('/ssb/logs/<id>/0')` and increase the sequence number until no more log entries can be fetched.

The ease with which this mapping could be introduced is misleading, as there are several problems with such an approach. First and foremost, implementing SSB's push model leads to continuous polling at the level of pull-oriented NDN. Second, there is no prefix aggregation possible because of SSB's flat ID space.

The latter concern could be addressed in two ways, both having undesirable consequences. The first would be to introduce a NDN routing strategy that mimics SSB's forwarding along the receiver's social graph. Such a modified NDN forwarder would have to either parse all logs or to receive the graph information from somewhere else. Additionaly, Interests would have to carry the ID of the receiver (otherwise the forwarder does not know which graph to use), which is contrarion to interest aggregation, which would have to be changed. Such a special "SSB routing strategy" would have to be deployed globally, essentially converting NDN into a SSB core. We will come back to a SSB-aware NDN layer in the next subsection.

A second approach would be to use NDN's LINK objects in order to redirect Interests towards the location of the storage provider for a given $\langle id \rangle$, using a NDN naming service [4] and a set of permanent storage providers to which the SSB devices would have to upload new content. Even if there are only a small number of storage provider, the NDN naming service would have to handle a global database for the $\langle id \rangle$-to-storage mapping.

4.4 SSB-aware NDN

Assuming that SSB's replication approach (limiting content propagation to a device's social graph works) leads to scalable solutions, NDN could be made SSB-aware. This would require that NDN inspects the logs in order to learn about the social graph. In fact, it would promote NDN forwarders to become SSB nodes that replicate SSB content (instead of a name server infrastructure and dedicated repo providers). A new, corresponding forwarding strategy doesn't require routing entries as it simply floods incoming interests that cannot be satisfied from the local replicas to all SSB+NDN neighbors along the social graph.

However, using such a strategy would be a travesty as it turns NDN into a push network for interest packets (due to the polling, by every node, whether new content is available or not) – only sporadic data packets would be returned during the replication process. An optimization would be to use long-lived interests, turning NDN into a pub/sub system. Finally we point out that some variant of NDN's SYNC protocols could be suitable to capture SSB's goal, namely to sync the replicas across (a part of) the network. Such a variant would be able to benefit from SSB's strict log extension rule. But looking at the way Sync is implemented in NDN, this comes back at a push-style communication where interests are used to poll neighbors about any state change. We think that bringing a SSB mindset into NDN would change NDN beyond recognition, especially flow-balance would be lost during this transition.

4.5 NDN over SSB

An interesting thought experiment is to reverse the layering, stress-testing SSB in this case. Is SSB universal enough so that one can “emulate” NDN over SSB (see the third subfigure of Fig. 5)? SSB would have to implement three distinct NDN features: the hierarchical name space, the pull-model and NDN’s trust system.

Again assuming rough equivalence of NDN data packets and SSB log entries (i.e., a triple $\langle name, content, signature \rangle$), the pull-model part is easy to answer: Either some item is already in one of the eagerly replicated local logs, or it is not available yet (because SSB is push-based).

The major problem is NDN’s hierarchical namespace which is a globally shared construct with the service level agreement that any (existing) item referenced through this tree can be fetched. Even if a delegation model is used, this global resource introduces a central authority, or at least a consensus algorithm, to allocate prefixes. This entity would have a “well-known” SSB id and a log from where the rest of the SSB world would inform itself about its decisions. Once these prefixes are handled (by a special SSB app for supporting NDN’s namespace, as shown in the third subfigure), repo IDs can be introduced such that an end device can address them and request content from them. However, in SSB’s worldview, a repo would have to follow all potential customers i.e., to learn about their IDs, otherwise these customers cannot express interest in some content (which could be delivered through log replication of transient SSB IDs, for example). When looking at NDN we realize that NDN has a social contract along the interest path: a NDN forwarder accepts any downstream node as a friend, accepts its interest packet (= pushed replica of the request), and then relies on a similar contract with its upstream node. Following this insight, our NDN emulation would have to introduce “NDN forwarding providers” at SSB level. Once these “NFPs” are in place, we would also let them implement NDN’s trust model by validating content through NDN’s certificate authorities.

While it doesn’t seem strictly impossible to continue that emulation argument, it is already obvious from the above discussion that one would not benefit from SSB’s social graph replication mindset. We try to explain this result in the following subsection.

4.6 Name-Centric vs Identity-Centric

At first sight, the stark difference of PULL vs PUSH is the main differentiator between NDN and SSB. However, the above layering discussion shows that there is a second, equally strong discrepancy which we pin to the way IDs are handled.

NDN has no notion of receiver ID, by design, which has the benefit of easy Interest aggregation but also is the basis of the social contract with the forwarder (to accept interests from anybody). SSB, however, is deeply ID-centric: Only by following (= declaring interest in) some ID, *all* its content will be pulled towards the interested party. Selective data replication, which requires a bidirectional Interest/Data protocol, only would work in SSB if both sides follow each other - hence the difficulty to implement NDN’s global fetch mechanics over SSB. NDN, on the other hand, introduces repositories (somehow linked to routing prefixes) as quasi-IDs: NDN’s service model is to interconnect (many) ID-less clients with (few) identified repos. As such, NDN has a centralization bias which obviously clashes with SSB’s decentralization stance.

One can also draw the following picture: NDN works with repo IDs (prefixes) on top if which we have IDs for content (= content names extending a repo ID). In NDN, IDs have no role for the receiver or in the replication process except that forwarding validates the origin of data items. On the other hand, these repo IDs must be globally routable through some unspecified routing protocol outside the NDN specs. SSB also has producer-side IDs, but it is mandatory that clients also have an ID because otherwise they could not publish their replication needs (towards SSB’s routing logic).

Finally, IDs have different weights in NDN and SSB when it comes to replicas and packet loss. NDN only validates that a data item’s signature can be traced back (via publisher IDs) to some trust anchor, while ARQ (automatic retransmission request) must be used to recover from lost packets. In SSB, the whole log associated with an ID is validated (assert strict ordering and completness), which factors out difficult tasks to the benefit of the applications (no retransmission logic, no need to SYNC).

As a conclusion, we were surprised that we could not find easy ways how NDN can be made suitable for implementing SSB, or be enhanced by SSB properties without turning NDN into SSB, and how it looks difficult to port NDN to SSB without deviating from SSB’s decentralization agenda. Beside the push/pull theme, SSB’s identity-centric approach seems to introduce a yet unseen element for ICN.

5 RELATED WORK

Some basic ideas behind SSB can be traced back to the nineties, like for example *secure logging* [25] and *secure relative timestamping* [12]. The major innovation of SSB is to use these techniques for disseminating data through a gossip protocol in a network of untrusted peers, effectively implementing a push-based information-centric network. SSB's log entries, named by $\langle id : seqno \rangle$, are a plain copy of DONA's naming schema **??**.

SSB's push-based content dissemintation approach is also underlying middle-ware systems like *Linda* [10]. Linda offers a global data pool abstraction where distributed processes can store and consume objects without locality references: The effects of a `wr()` operation are propagated automatically such that processes being blocked on a `rd()` could be resumed immediately.

The use of logs itself has a long tradition in distributed systems, especially in operating systems (*write-ahead logs* in journaling file systems) as well as distributed databases. More recently, in the cloud context, resilient event ordering protocols like *RAFT* [24] have been proposed that also rely on replicated logs. Although logs are used at various places of distributed systems, this data structure is typically not exposed to the commuicating parties, while SSB exactly rests on letting apps interact directly with the secured single-author logs.

Selective Hearing [19] uses a gossip protocol to disseminate monotonically growing sets of updates to provide a runtime or the Lasp [18] programming language. The general architecture is similar to that of SSB, the most striking difference is that Lasp is by design restricted to CRDTs. SSB can be considered more low-level, developers are free to choose a strategy for dealing with concurrency and eventual consistency. Selective hearing was developed in a more traditional research approach, so it glosses over some of the difficult problems encountered in the "real world" such as user onboarding and byzantine peers. Their "practical large scale evaluation" [20] consists of 1000 well-controlled nodes running a toy application in the cloud, whereas SSB with is roughly 10000 users is more battle-proven.

6 SSB'S "WORK IN PROGRESS"

So far we have mostly restricted our presentation to those features that are implemented today as part of SSB. In this section we will describe further extensions, namely a potential revision of SSB's log format and the operational challenges for scaling SSB beyond its user base of slightly more than ten thousand users.

6.1 Partial Replication

By using a linked list of messages as the underlying datastructure, a message can only be verified in time linear to its sequence number. Since all previous messages need to be available for verification, this also implies linear storage overhead. More sophisticated datastructures could reduce this to logarithmic overhead, both anti-monotone binary graphs [7] and threaded authentication trees [8] would be suitable and would only require a single additional hash per message.

An interesting problem in this context is how peers would indicate the subsets of a feed they want to receive. Specifying individual sequence numbers works fine, but degrades to a pull-based system. Instead semantic criteria are needed, for example subscribing to only messages of certain types. Finding a general framework for specifying partial subscriptions based on semantic frameworks is an interesting task. Care must be taken that malicious peers can not silently drop data that matches a partial subsciption, this could be done by adding additional sequence numbers for each criterium.

6.2 Local Deletion

SSB signatures range over the full content data. Accordingly, the content needs to be available in order to verify a message. If a relay locally deletes the content of a single objectionable message, then it could not replicate the log beyond the point of that message, since the peers could not verify the integrity of newer messages.

This situation could be improved by only including a *hash* of the content in the signature, rather than the content itself. This way, content could be deleted from a local log replica, while keeping the hash, so that the whole log could still be verified and thus replicated.

6.3 Cryptographic Agility

SSB relies on multiple cryptographic primitives (signatures and hashes for the log format, encryption for the replication protocol): best practice mandates that cryptographic agility is supported [22]. All hashes and signatures in the logs include an indicator of the cryptographic primitive that has been used. At least in theory this means that the SSB protocol can introduce the use of new primitives as old ones become broken.

An open problem is how old log entries can be "saved" once their primitives become insecure. The naive approach of republishing old messages with a new key changes the hashes of all those messages, thus breaking inter-message references. A similar discussion (and proposed solution) for NDN can be found in [31].

6.4 Log Management

In SSB, there is no mechanism for terminating a log. But it would be straightforward to add a mechanism that declares that the log will not be extended in the future (and any future extensions should thus be discarded). By allowing this mechanism to carry some payload data, key rotation could be supported: The log termination record would include the public key of a new log that should serve as the extension of the terminated one.

6.5 Multi-Device Support

If two different devices used the same SSB identity to publish messages concurrently, this would result in a forked feed with the consequence that peer relays would stop propagating at least one, if not both log extensions. It is thus recommended to create a distinct keypair per device. But this leads to bad user experience, such as having to follow or block identities on all device.

This could be mitigated by developing schemes that allow sharing the same private key across multiple devices to allow read-access, while enforcing mutual exclusion on writes.

A different angle is to write applications in a way that anticipates that there might be a one-to-many mapping from users to SSB identities. Since the messages in a single feed are totally ordered but messages across multiple feeds might only be partially ordered, it is not sufficient to naively treat a set of feeds as a compound feed. Instead, the application needs to be designed from the ground up to deal with partially ordered sets of messages.

Orthogonal to the issue of *using* data from aggregated, partially ordered feeds is the issue of determining which feeds to aggregate in the first place. Settling on a common scheme for signaling compound feeds will be necessary for SSB to successfully improve on the multi-device situation.

6.6 Access Control

Conceptually, all SSB messages are globally visible. The messages that control replication (*follows* and *blocks*) only specify where data is wanted, but they can't express bounds on how far data should be spread.

To improve privacy, some servers choose to only forward messages of some feed F to identities that are followed by the feed F. This ad-hoc solution overloads the meaning of *follow* messages. Work is underway for specifying dedicated messages that request bounds on how far data should be propagated through the social graph.

For "harder" guarantees, encryption can ensure that only intended recipients can access data. Currently, the only supported mechanism is encrypting a message to a small set of recipients. More sophisticated approaches such as encrypted groups could be adapted for ssb.

6.7 Content Moderation

Users need the ability to shape their virtual space such that they can feel safe. Because nobody has a global view of the system, traditional centralized approaches for moderation can not be applied directly to ssb. In particular, it is not possible to globally "ban" an identity.

There are two fundamental options for dealing with unwanted content. Users can stop replicating a feed and delete it from their local database. Less drastically, applications can choose to ignore specific messages.

While these are powerful primitives that give users full control over their environment, they place the burden on the affected individual. Going forward, it will be important to find mechanisms that allow to share the task of moderation and shift it to users who are privileged enough to be able to invest the necessary energy and time. A simple example could be to automatically adopt the *block* messages of trusted peers. Since human dynamics are very nuanced and every human has their specific needs, we expect a lot of experimentations and different groups of users settling on different tools.

6.8 Replication Improvements

The currently used gossip-based default replication protocol does not protect against malicious intent such as for example eclipse attacks [27]. But whereas it is difficult to defend against these attacks in general, SSB can make use of data such as the friend graph to protect against them. A *follow* message can be interpreted as an expression of trust. Keeping a certain number of trusted peers in the views of the peer sampling service could protect against eclipse attacks.

Another area where the replication protocol could be improved is by using private set intersection when determining the set of feeds that both parties are interested in. That way, untrusted peers would not be able to learn about new ids purely from the replication layer. Combined with an access control mechanism that only forwards data to authorized identities, this would provide resilience against bots "spidering" the network.

7 SSB CHALLENGES

In this section we critically review limitations and challenges faced by identity-centric systems such as SSB. We omit those problems that apply to SSB in its current state but that would be solved by the extensions presented in the previous section.

7.1 Privacy

First and foremost are privacy considerations. SSB is an inherently pseudonymous system, anonymity is fundamentally incompatible with identity-centric message propagation. Furthermore, the architecture discourages ephemeral pseudonyms, favoring the creation of a rather stable network of trust to guide replication. Since all messages are signed, they are not refutable. Finally, all messages are immutable.

The cocktail of pseudonymity, non-refutability and immutability can be a serious risk to users. Personal details could fuel harassment, political statements could justify persecution, all data could serve as the basis of (future) discrimination. The risks can be reduced by taking care that pseudonyms can not be traced to physical identity, compartmentalizing pseudonyms, using encryption, and only giving the messages to trusted parties. Still, participation currently favors users for whom privacy issues are not critical. Consequently, applications must clearly inform users about the peculiarities of the virtual space they participate in to ensure users don't share information that might be detrimental to them later.

7.2 Onboarding

Data can only be propagated to relays that specifically ask for it. When a new node joins the system, it can only participate effectively once someone subscribes to its feed. The SSB community approaches this "onboarding" with multiple techniques. Pubs can issue *invite codes* out-of-band. When a user sends such a code to the pub, the pub automatically follows the user, requesting their messages in the process. As an additional onboarding mechanism, the reference server uses lan multicast to discover nearby peers. This allows local onboarding where an established user can follow a new user in the same lan.

7.3 Coordination

The *type* field of SSB messages can be regarded as a global resource without any central coordination regarding its usage. In the worst case, this can lead to multiple applications using the same *type* but in incompatible ways. Namespacing and random types reduce but don't eliminate this problem.

Non-interoperable *types* are a very tangible symptom of a broader theme, the *plurality* of interpretations of message contents. Taken to an extreme, "the" community of SSB users could splinter into a multitude of mutually non-understanding fractions that use different kinds of messages or interpretations thereof. Supporting divergence can also be considered a feature because it mirrors the informal evolution of human languages over time, a property that is often overlooked or actively shunned in more centralized designs.

8 BENEFITS

Here we summarize some desirable properties of SSB. These go beyond the obvious productivity gains for application developers who don't need to implement encryption, authentication and synchronization, as well as the automatic replication of the content through SSB's push approach.

8.1 Resilience

The design of SSB intentionally avoids "global singletons", or centralization aspects requiring consensus. SSB can therefore be characterized as a "collection of decentralized systems" that overlap to varying degrees. It is consequently highly resilient to failures, whether due to attacks or errors in the code base or in operations. Also, users do not need to depend on any single, privileged central authority, including cloud-based service providers. Going beyond the common mentality of a grand unified social platform, there are deliberately isolated networks, for example limited to a family, or a specific local area network.

Since SSB applications only interact with the local replicas of logs, complete offline operation is automatically built in. Offline operation is simply a special case of a (temporary) network partition. Because this case occurs so often, the protocol is geared towards handling network partitions gracefully, further contributing to the resilience of SSB. In particular, all operations are delay tolerant.

Data loss is highly unlikely since full log replicas are stored throughout the network. Indeed it is common practice when migrating between devices or during development to delete (or loose) the whole local database, keeping nothing but the keypair. Upon starting the node again, all data gets retrieved from the network, more specifically from your friends.

8.2 Efficiency

Due to the "subjective reader" approach, all ssb relays can operate concurrently. There is no need for synchronization across nodes, and the overhead they might incur. Applications can fully embrace this, for example by using CRDTs. The monotonically growing logs are well-suited for implementing CRDTs and similar techniques.

By *leveraging* existing trust behind social interactions, instead of trying to eliminate it (like, e.g. in blockchain systems which establish consensus over trustless nodes), proof-of-work and other computationally-intensive techniques can be replaced by *social signals* encoded as events in a log. The delay tolerance allows routing layers that can optimize for different tradeoffs, for example by minimizing the bandwidth required to disseminate updates rather than minimizing latency. Pushed further, the same approach could lead to infrastructure that is quite efficient in its usage of memory, bandwidth, and energy, making the overall required infrastructure sustainable with less resources than other approaches requiring always available, high-throughput, routing infrastructure. Leveraging existing social trust between participants can therefore provide clear technical benefits.

8.3 Plurality and Disintermediation

The freedom of applications to interpret data in whatever way they see fit increases the agency of users and application writers, to choose how to leverage the data they produce and for what purposes. SSB supports plurality also as a deliberate strategy to drive evolution. Rather than depending on proprietary app-level APIs, interoperability and the ability to create alternative frontends is the default, resting on the direct access to the replicated data. This also fosters the sharing of data between applications. For example, the '`about`' information (Section 3.1) can be reused by all programs, freeing implementors from duplicating work and creating a coherent user experience across apps.

As another consequence of the subjective interpretation of data, there is no need for central coordination to introduce or evolve features: new uses can evolve based on the immediate needs of participants and then spread if the needs are more widely shared. Applications can simply start producing new kinds of log entries, and interoperability works out with all users who share the same interpretation of those messages. Especially when evolving functionality, it is often sufficient to add more fields to a JSON object: Other applications simply ignore them but will continue working.

This results in an organic evolution of features, without "the system" ever shutting down. Like any uncoordinated evolutionary process, this may sometimes lead to dead-ends or less apparent conceptual integrity, but the evolution is open ended, can explore multiple alternatives in parallel, and lower the requirements for governance. Perhaps the value of that approach can best be assessed by the continuous and incremental improvement of SSB applications that has resulted in a thriving ecosystem that is used daily by hundred of users today, and evolves similar to the way the Internet has grown over the last decades.

9 CONCLUSIONS

We presented secure scuttlebutt, a fully decentralized, peer-to-peer event-sharing protocol. The core novelty is that data replication occurs at the granularity of complete append-only logs of messages by a particular author. This approach leads to a simple, yet efficient replication protocol that lends itself well to a large class of applications. By embracing *eventual* delivery and subjective interpretation of data, SSB gets to sidestep common sources of complexity. A community of multiple thousand users interacting through a variety of different applications confirms the viability of the approach.

The comparison with NDN shows that SSB's paradigm of push-based, identity-centric data transfer comes with a different set of tradeoffs than NDN's choice of pull-based, name-centric data transfer. Focussing on identities leads to challenges with respect to user privacy, but it also enables elegant, decentralized solutions to common problems with information-centric systems. Whether in the context of SSB or more generally, we believe that further study of identity-centric systems will lead to valuable insights and designs.

REFERENCES

[1] 2002 - 2019. meetup.com. https://www.meetup.com/

[2] 2008 - 2019. Github. https://github.com

[3] 2015. Designing a Secret Handshake: Key Exchange as a Capability System. http://dominictarr.github.io/secret-handshake-paper/shs.pdf

[4] Alexander Afanasyev, Xiaoke Jiang, Yingdi Yu, Jiewen Tan, Yumin Xia, Allison Mankin, and Lixia Zhang. 2017. NDNS: A DNS-Like Name Service for NDN. In *26th International Conference on Computer Communication and Networks, ICCCN 2017, Vancouver, BC, Canada, July 31 - Aug. 3, 2017*. 1–9. https://doi.org/10.1109/ICCCN.2017.8038461

[5] Mark Baugher, Bruce Davie, Ashok Narayanan, and Dave Oran. 2012. Self-Verifying Names for Read-Only Named Data. In *2012 Proceedings IEEE INFOCOM Workshops, Orlando, FL, USA, March 25-30, 2012*. 274–279. https://doi.org/10.1109/INFCOMW.2012.6193505

[6] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2012. High-speed high-security signatures. *Journal of Cryptographic Engineering* 2, 2 (2012), 77–89.

[7] Ahto Buldas and Peeter Laud. 1998. New linking schemes for digital time-stamping.. In *ICISC*, Vol. 98. 3–14.

[8] Ahto Buldas, Helger Lipmaa, and Berry Schoenmakers. 2000. Optimally efficient accountable time-stamping. In *International Workshop on Public Key Cryptography*. Springer, 293–305.

[9] Scott Chacon and Ben Straub. 2014. *Pro git (2nd Edition)*. Apress. https://git-scm.com/book/en/v2

[10] David Gelernter. 1985. Generative Communication in Linda. *ACM Trans. Program. Lang. Syst.* 7, 1 (Jan. 1985), 80–112. http://doi.acm.org/10.1145/2363.2433

[11] Piet Geursen. 2017. path-gatherings. https://github.com/pietgeursen/patch-gatherings

[12] Stuart Haber and W Scott Stornetta. 1990. How to time-stamp a digital document. In *Conference on the Theory and Application of Cryptography*. Springer, 437–455.

[13] Internet Archive. 2018. Decentralized Web Summit 2018, Jul 31 – Aug 2, San Francisco. https://decentralizedweb.net/

[14] Ruoming Jin, Ning Ruan, Saikat Dey, and Jeffrey Yu Xu. 2012. SCARAB: scaling reachability computation on large graphs. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 169–180.

[15] Charles Lehner. 2018. Git-SSB: Social Coding on Secure-Scuttlebutt. https://git.scuttlebot.io/%25n92DiQh7ietE%2BR%2BX%2FI403LQoyf2DtR3WQfCkDKlheQU%3D.sha256

[16] Joao Leitao, Jose Pereira, and Luis Rodrigues. 2007. Epidemic broadcast trees. In *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*. IEEE, 301–310.

[17] Gordon Martin. 2017. ssb-chess. https://github.com/Happy0/ssb-chess

[18] Christopher Meiklejohn and Peter Van Roy. 2015. Lasp: A language for distributed, coordination-free programming. In *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming*. ACM, 184–195.

[19] Christopher Meiklejohn and Peter Van Roy. 2015. Selective hearing: An approach to distributed, eventually consistent edge computation. In *2015 IEEE 34th Symposium on Reliable Distributed Systems Workshop (SRDSW)*. IEEE, 62–67.

[20] Christopher et al. Meiklejohn. 2017. Practical evaluation of the lasp programming model at large scale: An experience report. In *Proceedings of the 19th International Symposium on Principles and Practice of Declarative Programming*. ACM, 109–114.

[21] Aljoscha Meyer. 2018. SSB Specification. https://spec.scuttlebutt.nz/feed/messages.html

[22] David Nelson. 2011. *Crypto-Agility Requirements for Remote Authentication Dial-In User Service (RADIUS)*. Technical Report.

[23] Noffle. 2016. git-ssb-intro. https://github.com/noffle/git-ssb-intro#push-conflicts

[24] Diego Ongaro and John K. Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *Proc USENIX Annual Technical Conference*. 305–319. https://www.usenix.org/system/files/conference/atc14/atc14-paper-ongaro.pdf

[25] Bruce Schneier and John Kelsey. 1998. Cryptographic support for secure logs on untrusted machines.. In *USENIX Security Symposium*, Vol. 98. 53–62.

[26] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-free replicated data types. In *Symposium on Self-Stabilizing Systems*. Springer, 386–400.

[27] Atul Singh et al. 2006. Eclipse attacks on overlay networks: Threats and defenses. In *In IEEE INFOCOM*. Citeseer.

[28] Dominic Tarr, Paul Frazee, Christian Bundy, Matt McKegg, Anders Rune Jensen, Mix Irving, et al. 2014. SSB Server. https://github.com/ssbc/ssb-server

[29] Dominic Tarr, Mix Irving, Christian Bundy, Michael Williams, Anders Rune Jensen, Andre Staltz, and Matt McKegg. 2014. SSB Server. https://github.com/ssbc/ssb-friends

[30] Hilmi Yildirim, Vineet Chaoji, and Mohammed J Zaki. 2013. Dagger: A scalable index for reachability queries in large dynamic graphs. *arXiv preprint arXiv:1301.0977* (2013).

[31] et al. Yu, Yingdi. 2017. NDN DeLorean: An authentication system for data archives in named data networking. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*. ACM, 11–21.