StrongSalt:

A Decentralized Communication Network for Data Privacy and File Sharing

Overnest Team

March 18, 2018

Abstract

Internet traffic will be very different in ten years. Currently, most traffic is still directly or indirectly requested by humans, but in the near future, most communication will be between smart devices controlled by complex AI constructs without any human intervention. Current internet protocols were designed with human consumption in mind and are ill-suited for power-conscious, chatty, and security-focused smart devices. And with the advent of devices and machines powered by AI and algorithms that change dynamically as they sift through more data, network security and digital privacy have become less attainable for individuals.

The recent craze for blockchain and other decentralized services demonstrates that centralized and proprietary services are being challenged by alternative based on decentralized and open architectures. The reason is that most legacy infrastructures are based on centralized, unencrypted hub-and-spoke database architectures. Those traditional services are inefficient, brittle and, most importantly, expensive and vulnerable to cyber-attacks. The cost of maintaining such architectures is growing at an alarming rate.

The advent of blockchain technology along with innovations in token economics, especially at the protocol level, has solved two fundamental problems of computing: 1. How do you establish trust among strangers without a trusted third party? 2. How do you incentivize strangers towards a common goal without incurring a Prisoner's

Dilemma? By providing a framework upon which to programmatically address these two concerns, we now have a chance at providing the most important resource of the modern age: secure data communication without a need to trust the network itself.

StrongSalt is a decentralized low-latency data communication protocol stack that builds security from the lowest level of network protocol all the way to the application layer so that data privacy can be preserved even in a potentially hostile network environment. The communication protocol relies on four conceptual layers: the transport layer, the blockchain layer, the messaging layer, and the application layer. The transport layer proposes secure datagram-based connectivity for highly efficient networking by lowering latency compared to traditional TCP-based networks. The blockchain layer provides a native protocol token (StrongSalt Token), which miners earn by providing trust and coordination between the various participating nodes in the network, which in turn provide data transmission and data storage services. The tokens can then be used to store and share data, reserve QoS (quality of service), and purchase additional security and privacy related features. The blockchain also provides trust establishment among the nodes that are providing the core routing and data retention services. The miners are compensated for providing bandwidth and/or storage.

The reason storage is considered in a network protocol and often we talk about communication and data storage interchangeable is because storage is a special case of messaging in that storing a message is equivalent to sending a message to your future self. In practical terms, for a message in a communication platform to be delivered from Alice to Bob, someone will have to retain the message after Alice send out the message and before Bob retrieves it and this retainment of the message can be considered a storage service.

The API layer is the lowest level that most programmers will deal with in order to utilize the protocol. The programming API is conceptually a publisher-subscriber model to ensure decoupling of the senders and receivers to further preserve privacy of the participants of the network. The application layer is an application built on top of the API layer that fully takes advantage of the platform. StrongSalt also provides a demo service (currently called simply the StrongSalt App) in the form of a file storage and sharing app to demonstrate and test the network. The reason for file storage is obvious as it is the most easily understood use case of a data privacy application.

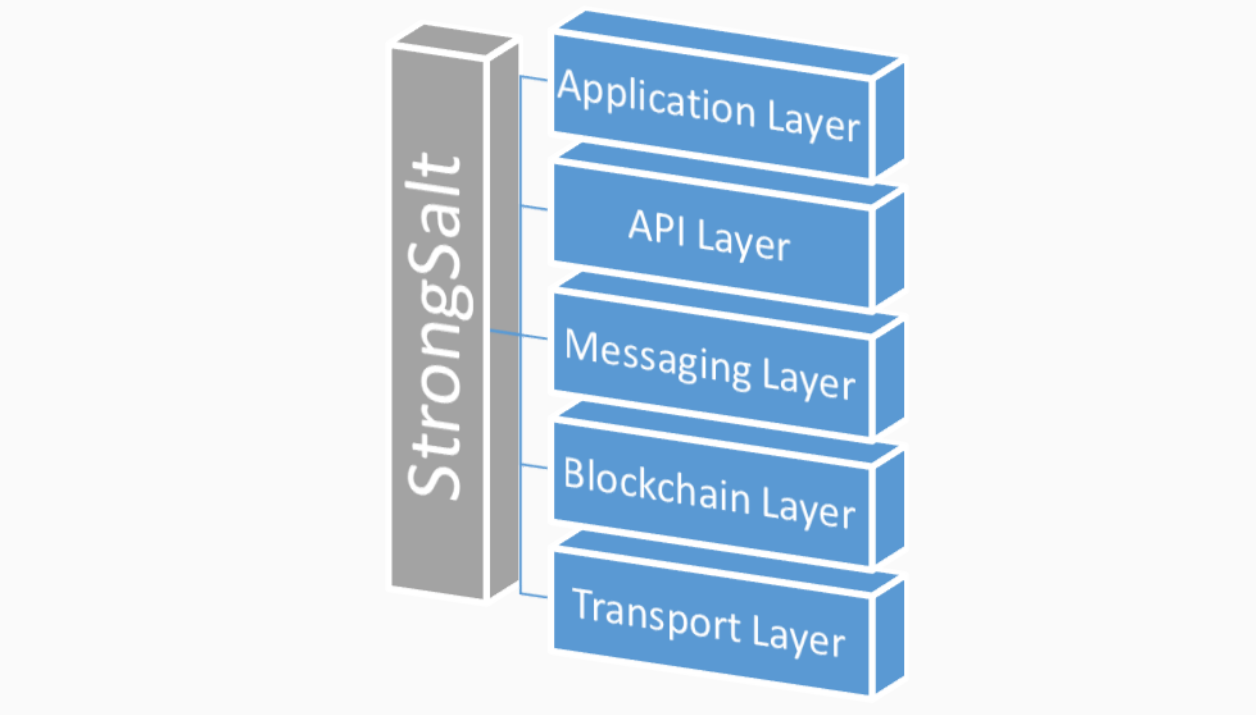
Note: StrongSalt is a work in progress. Due to the complexity of both the network protocols, the token economics and the security and cryptography involved, we are still actively researching various topics in each layer. For comments and suggestions,

contact us at info@strongsalt.com. All future versions of this paper will appear on https://www.strongsalt.com.

Introduction

The StrongSalt protocol is composed of multiple layers of abstraction similar to how the OSI (Open Systems Interconnection) and TCP/IP protocol stacks are divided into multiple layers. Although the StrongSalt protocol does not dictate much of the lower 2 layers (typically considered the hardware layers) in the OSI stack, the StrongSalt protocol by its nature does concern itself with much lower protocol stacks than almost all other blockchain related projects and even most secure messaging protocols. In particular, we are concerned with the transport layer, which is equivalent to both the OSI network and transport layers because we do believe there is a fundamental way our protocol can help solve one of the most difficult problem of how internet routes traffic currently—latency.

StrongSalt protocol consists of the following layers: Transport Layer, Blockchain Layer, Messaging Layer, API Layer and Application Layer. The purpose of the Transport Layer is similar to the layer of the same name in both the OSI and TCP/IP protocol stacks to make efficient use of the network connectivity.



The StrongSalt network protocol made a conscious decision not to go any lower on the network stack to maintain compatibility with all existing network hardware and to take advantage of the research that is being done on Internet2; however, as we will discuss later, the StrongSalt protocol does plan to take a different approach in the transport layer.

The Blockchain (or Decentralization) Layer is very important both philosophically and technically because it's the layer that made decentralization a reality and lays the foundation for the token economics that are inherent in most real blockchain related projects.

The Messaging Layer is the layer that are most similar to the level of abstraction that most other secure messaging protocols concern themselves with. It is the layer that data security is of utmost importance.

The Blockchain and Messaging Layers are a little different from the other layers in that they actually are less higher and lower layers with each other but two parts of the same layer. The Blockchain Layer provides the token economics and keeps track of the message passing between nodes done by the Messaging Layer. In turn, the Message Layer provides the messaging passing so that the Blockchain Layer can function.

The API layer is separated out mostly for convenience because it is easier to discuss programming abstractions separately from the lower layers and the application layer.

The application layer is a placeholder for all application related security concerns such as logins, key distribution, etc.

StrongSalt Token is a protocol token whose blockchain establishes trust amongst the miners that provide the core routing functionality in the network. It is also the in-house token used for all services rendered on the decentralized communication platform.

Mission Statement

We believe that everyone is born with certain unalienable rights, one of the most important being the right of privacy. Besides its many functional benefits, privacy is what allows us freedom of expression and individuality. This is why we created StrongSalt. Your data is safe from everyone, including us, and your privacy is protected from

everything including AI-powered IoT devices. A future where each of you and your devices have complete control of data begins with StrongSalt, a secure, decentralized communication platform.

Definition of a Decentralized Communication Network

We first want to define a Decentralized Communication Network (DCN) to differentiate ourselves from a typical messaging app because we address many different problems on different levels. A DCN is a secure data storage and sharing platform that allows the creation, transmission, and the reception of the messages using a decentralized messaging backbone to allow anyone anywhere the ability to send a secure message such as an encrypted file through a network of participating nodes that are organized by a decentralized ledger.

In our definition of the DCN, traffic is not limited to a text message, picture, or video, but comprises any arbitrary data. This is best exemplified by the widespread success of WeChat which began as a messaging app with social functionalities but quickly spread to payments, deliveries and other transactional Mini Apps. The app's transactions and commands are messages that need to be sent out and delivered securely. In particular, we focus on the particular use-cases of centralized file sharing such as DropBox(™) to compare with our service. Our goal is not to compete in the same generic file sharing space but to provide an alternative service for those highly sensitive files that many feel uncomfortable to store on existing file sharing services. These centralized file sharing services are subject to not only security concerns but also data privacy concerns because they have complete control of customer data and files. No existing popular file storage or file sharing service provides data privacy where encryption happens on the client side and that the customers have full control of the encryption keys. We strive to provide a zero-trust data sharing platform where all files are encrypted on the client side and that encryption keys are never shared with the platform or the servers that store the files.

The Decentralized Messaging Platform (DMP) is the communication backbone of the DCN. It consists of the layers below the API Layer and its purpose is to allow core routing nodes to coordinate with other participating nodes that provide the actual message storage, message duplication, message transport, and message retrieval. The clients are concerned with the creation and final consumption of the messages al

ong with some higher-level security constructs and use the messaging API to send and receive messages via the messaging backbone. The combination of the clients, the messaging backbone, the participating core and network nodes, along with a decentralized ledger that coordinates the nodes make up the DCN.

One thing to note is that the coordination by the protocol via the blockchain is decentralized and does not require trusted parties. The blockchain and the protocol establish the trust among the participating nodes and enforce the token distribution (spending, earning, staking) but the messages themselves are not stored on or distributed by the blockchain. This is in fact very different from earlier attempts by others to use blockchain as a message storage and distribution mechanism. We believe placing messages on the blockchain for everyone to see not only is a waste of space and makes the whole protocol difficult to scale but also misuses the semi-permanent nature of blockchain and has too much privacy ramifications even if the content is encrypted.

A typical messaging app including the secure messaging variants mostly are only concerned with the equivalent of our application layer. The existing messaging backbones are usually provided by a centralized server or a cluster of servers controlled by a central entity. These single points of failure are easily controlled, taken down, and subject to cyber-attacks. In fact, we believe it is possible in the future for existing messaging apps to migrate to the DCN using the same API without concerning themselves with the lower level abstractions.

Low level protocol (and why we care)

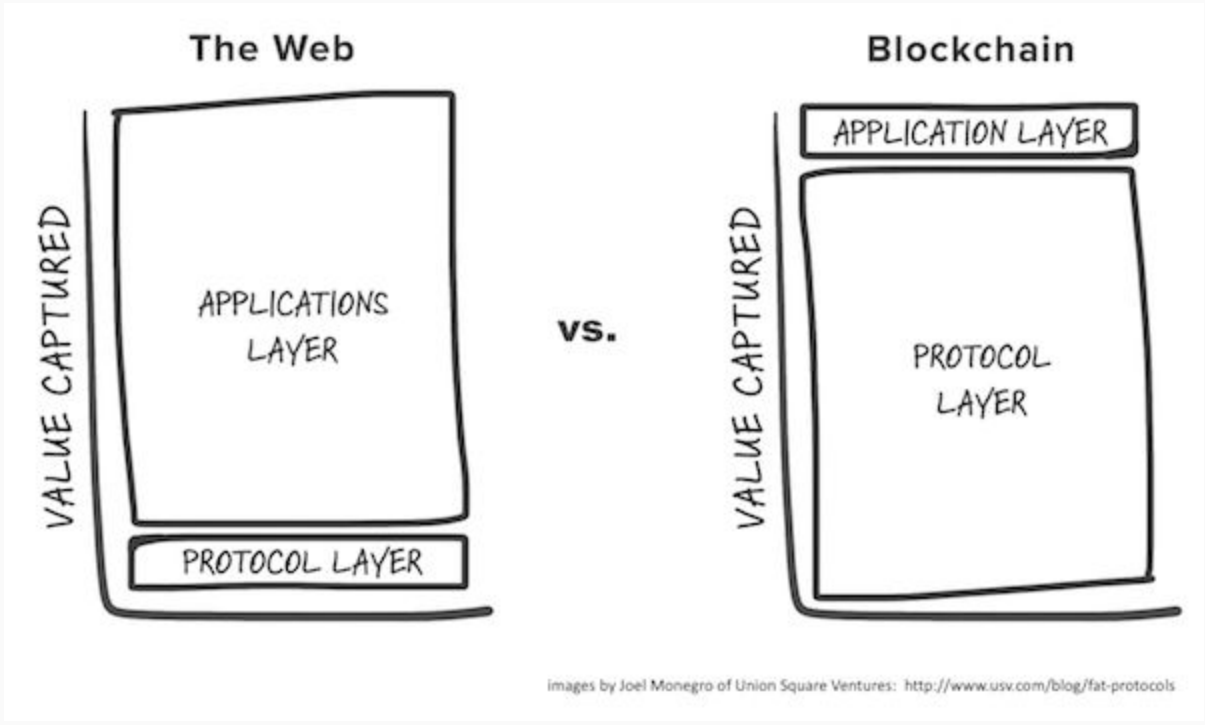
As discussed earlier, the StrongSalt protocol compared to almost any other messaging app or protocol is concerned with much lower level detail in the networking stack. This is due to both necessity and optimization. Because StrongSalt is foremost a decentralized messaging protocol it cannot rely on having a trusted party to store and pass messages. As such, there needs to be a protocol to coordinate between the untrusted participants in the decentralized network. One way is to overlay the decentralization protocol on top of existing networking protocols.

However, we cannot blindly layer on top of network protocols as if we are just an application because fundamentally we need to provide a secure and fast transport between the message sender and the message receiver. The transport may route the message through a combination of routing nodes that are not part of the decentralized messaging backbone, as such we will run into the same performance problem encountered by protocols such as Tor due to latency issues between nodes.

For a generic secure networking protocol, such latency will negatively impact user experience because users have associated messaging protocol with near-real-time performance. For a storage network, this is less problematic because a large file transfer is more concerned with bandwidth than latency. However, for a communication platform, latency is as important as bandwidth because we humans can “feel” latency and, although we associate low bandwidth with “slow,” we associate high latency with “broken.”

Because latency is bound by the speed of light, the only way to realistically reduce latency is to reduce the number of round trips for handshakes or overhead in the network stack. We therefore propose to use a secure UDP-based transport mechanism and to eschew the typical HTTP protocol in favor of a binary based protocol. Of course,

the protocol will fall back to TCP based communication if UDP communication is blocked.



As we see from this diagram, it's no longer limited to innovate only on the application layer as the values of lower protocol layers can be captured by the token economics.

Evolution of (Cloud) Architecture

The architecture of the StrongSalt DCN owes a tribute to the evolution of early P2P systems such as distributed file systems. Similar to how distributed file system first came about in client server model, and then evolved to server clusters, and eventually to a P2P-based, we would venture to say that a DCN is an inevitable evolution once blockchain solved the trust issue of a P2P system.

Even when cloud services took the world by storm, cloud services run in different networks, provide different API, and are basically incompatible with each other. The primary reason for fragmentation is because the lack of trust especially when each cloud service needs to consider the security of its own network, the services it provides, and that of their customers. Although there were attempts to build a meta layer on top the services, the most successful ones are either to use devops tools to unify the

deployment or to use an API abstraction to orchestrate the services. Neither of these work across all cloud services and in essence these are useful only to save some time between migrations and are extremely fragile in reality.

When striping away all the marketing jargons, current cloud services are simply an evolution of the old server cluster model and as such trust is a fundamental building block of a symmetric server architecture such as a cloud system (where service can be provided by any real server without the clients knowing or caring which server actually provided the service).

However, in a decentralized system such as DCN, the services provided must not have any assumption of trust, stability, or even bandwidth. Similar to any P2P network, the system must be autonomous, self-organizing, and heterogenous. The goal of DCN is to make self-organization easier and provide a foundation of the uniformity on top of heterogeneity. In order to provide a stable system on top of a decentralized network where there is no assumption of trust or accounting, the network must provide a protocol incentivizes good behavior and disincentivizes bad behavior without trusting any particular third party.

We have designed StrongSalt DCN to provide the foundation for a decentralized network that can eventually expanded to a DCN of cloud services (messaging, storage, computing, and etc).

Networking vs Storage

Although StrongSalt is a communication protocol, there is a duality between storage and networking. In fact, in typical computer science literature, often storage is not considered because, in some ways, storage is a special case of networking. For example, in a (secure) messaging system, a message is from Alice to Bob, but if Bob happens to be Alice in the future so that the message is from Alice (now) to Alice (3 days later), then we are essentially talking about storage of the message for 3 days.

In the Decentralized Communication Network, storage is actually a central theme of the protocol for two primary reasons: replication and timing. Because of the decentralized nature of the network, the message must be duplicated in order to guarantee the availability of the message. The message also may not be deliverable immediately because the recipient may be offline or may not have enough bandwidth to consume the whole message at the time or simply is unreachable at the moment. In such case,

the message must be stored by the Decentralized Communication Network for eventual delivery and disposal.

Networking vs Computing

Although on the surface level, networking and computing are different services, StrongSalt is designed with a much more generic purpose in mind due to the hierarchical architecture (described later in the document). The “working” nodes are distributed in the network around the core routing nodes. These “working” nodes provide additional services coordinated by the routing nodes with the help of the networking infrastructure. Similar to the how long-term storage can be provided by nodes providing storage and retrieval services, nodes can provide computing services. There are low level computing services that can provide the equivalent of virtual machines (similar to the AWS EC2(™)), a higher level construct (such as AWS Lambda(™)) can be mapped to the same network. The decentralized network provides a natural habitat for nodes across different geographic areas to build arbitrarily complex virtual fabrics. The DCN is the communication and accounting network that abstracts out the API level differences and provide the token economics for current and future cloud service providers. In essence the DCN is providing a meta-cloud overlay network for existing cloud service providers. Whether the API of the multiple service providers would eventually merge into a StrongSalt Decentralized Cloud API or whether the service provider will simply retain the API differences but compete functionality-wise is to be determined eventually by the market.

There is one key difference between the computing abstraction provided by the StrongSalt DCN and that of blockchain-based virtual machines (VMs) running smart contracts: The running code (bytecode or not) is not stored on the blockchain. The blockchain of the DCN provides accounting and a market for the services provided by the working nodes but does not actually contain the code to be run by the computing services. The code are delivered and distributed by the messaging layer and stored (whether on disk or in memory only) on the working nodes providing the computing service. There are myriad benefits for designing the network this way but the biggest reasons are scalability and security. By not incorporating the code into the blockchain itself, blockchain is free to do what it’s good at--providing token economics and sharing a distributed accounting ledger. The actual service is provided by the working nodes and scalability can be taken care of off the blockchain by the computing fabric. The blockchain also does not become an attack vector for the code. Of course, security is a complex topic when it comes to distributed computing services but at least it can be

isolated to the nodes providing the specific computing service rather than the whole DCN.

Decentralized Messaging Protocol

We can categorize most messaging apps into three categories: 1. Generic chat applications such as Facebook Messenger, WeChat, Kakao, Line, Slack, and so on. 2. End-to-end encrypted messaging apps such as WhatsApp and Signal. 3. The decentralized messaging apps such as StrongSalt alone (as of January 2018). In other words, every other messaging app is centralized. The primary reason usually stated is that users don't care. Because for any app to survive, the primary motivation of any messaging app is to get users and as a result non-user-facing features such as decentralization and even in many cases security and/or privacy take a backseat to adding more features such as video chatting or sending stickers/gifs. Security, especially encryption, is often used sparingly for a different reason, too: it's very difficult to serve targeted ads if the conversations cannot be scanned by the company providing the app.

Another reason for the lack of decentralization is simply that decentralization is actually very difficult to achieve unless you can provide a decentralization network with a clear API so that the app developers can just call the API to send and receive messages without being cumbersome or slow often associated with decentralized apps.

Secure Communication Protocol

By 2014, the number of connected devices had surpassed the number of people on the planet. In fact, most of us now have a device that can record and monitor every moment of our lives—our phone. As we introduce more and more "smart" IoT devices into our lives, not only will we have to deal with privacy from human to human communication, but will also need to consider human- to-machine and machine-to-human (when initiated by machine directly) communications. The most significant change is that we now have true machine-to-machine communications where a human has no input into the operation of and communication between devices. Often those devices are controlled by algorithms or even AI constructs that are so complex that even the designers of these systems would have a hard time understanding the reasons for all of the actions taken by these devices.

The two fundamental problems we will be facing in such scenarios are the issues of trust and privacy. How do we trust that the machines are doing exactly what they were originally intended to do, and how do we prevent a breach of privacy when we have all those devices surrounding us and monitoring our lives at each moment?

How do these billions of connected things communicate while carrying privacy-laden information about us? Currently most of these communication channels are not protected from hackers in any way. Many of these devices are low-cost, battery-powered, and have limited memory and processing power. As such, the cost of traditional security has been considered to be “overweight” and “overkill” for the manufacturers. Even if these devices are not always connected to the internet directly, as long as they have some connection to a device that can eventually access the internet, we have an IoT node and a new attack vector for hackers.

Most of these devices have different network requirements than those of more powerful counterparts because the constraints are different. Those devices need a lighter-weight protocol that doesn’t require large amount of resources.

StrongSalt: A Decentralized Communication Network

The StrongSalt Communication Platform is a Decentralized Communication Network that is designed to provide a decentralized network for secure messaging. It brings together a network of miners (core routing nodes) and participating nodes (messaging nodes) for message routing and message storage respectively. A client connects to any messaging node in the network and registers the channels that they are interested in with the messaging node. The client subscribes to or publishes to channels on the messaging node. For example, if Alice needs to send to Bob, Alice will publish her message to a channel to which Bob subscribes.

When the client needs to send a message, the client sends the message to a channel on a messaging node which in turn then forwards the message to the core routing node. The core routing node will then replicate the message by forwarding the message to additional message nodes.

When a client needs to receive a message, the client is first connected to a messaging node which will notify the client if there are messages in the channels that the client is interested in. When the client is notified that there are messages in the channel, the client will download the message from the channel. The core router and the messaging

nodes work together to forward messages between messaging nodes because it's unlikely that the sender and the receiver happen to utilize the exact same messaging node. The core router thus serves as a broker for the messages between the messaging nodes.

Application Layer: StrongSalt Video App

There are many good reasons to work on an app while designing a decentralized communication platform but the two most important reasons are: 1. To best design an API, you must *use* the API or, to put it bluntly, "eat your own dog food." 2. As stated earlier, there is a lack of good decentralized communication apps and we see this app as a real-life use case.

We intend the StrongSalt Video App to be a secure video sharing app that concentrates on the concept of video on a space-time continuum or more plainly: "send a disappearing video that you can time exactly when will it be shown to the receiver."

The StrongSalt App will fully utilize the full protocol stack of the StrongSalt Decentralized Communication Network. The network will deal with deliverability (sending, receiving, and temporary storage) of the messages and the transport level scalability and security.

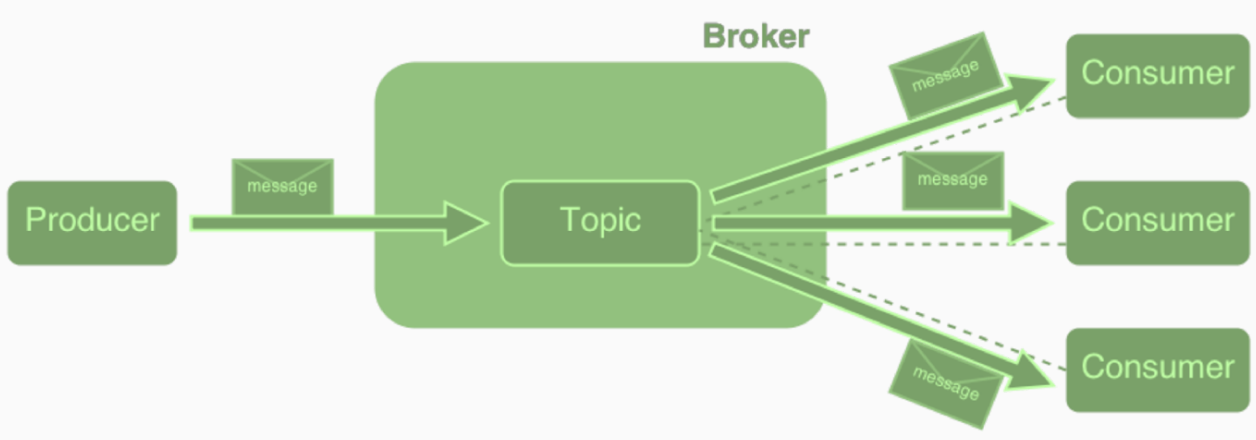
The application layer will deal with video creation, video playback, the user management and user experience. The application layer can choose to create additional application utility token on top of the protocol tokens.

API Layer:

The API Layer is a thin layer on top of the messaging layer. We decided to use a publish-subscribe API primarily because a pub-sub system allows us to decouple the publisher of the message (sender) from the subscribers (receivers). For a decentralized system, this decoupling is highly desirable both conceptually and technically. Because the API sits below the Application Layer, it does not define for example the idea of user the same way as the application. In fact, a user is no more than an opaque string for the API layer.

Another reason for the API Layer is to fully isolate any changes to the backbone from the Application Layer so that we can fully swap out the implementation as we find more appropriate solutions as the protocol evolves.

A good side effect or benefit of this pub-sub system is that this is compatible with the concept of event driven programming paradigm, so this will greatly enhance the scalability of the underlying asynchronous system by adopting a pub sub API. The queue-based system would allow non-intrusive load-balancing of the pub sub queues.



Messaging Layer:

The Messaging Layer is the implementation of the API Layer based on transport provided by the bottom two layers. In other words, the Transport Layer provides the networking between any two of the nodes and the Blockchain Layer provides the mesh network among all of the nodes. The Messaging Layer sits on top of the networking already provided by the bottom layers and fully reify the services provided by the API layer. This is the critical layer that have to deal with the scalability and the security of the overall system. As such we have adopted the Reactive System Manifesto (https://www.reactivemanifesto.org/) and because how applicable this is we have reproduced in part below:

> *Responsive*: The system responds in a timely manner if at all possible. Responsiveness is the cornerstone of usability and utility, but more than that, responsiveness means that problems may be detected quickly and dealt with effectively. Responsive systems focus on providing rapid and consistent response times, establishing reliable upper bounds so they deliver a consistent

quality of service. This consistent behavior in turn simplifies error handling, builds end user confidence, and encourages further interaction.

Resilient: The system stays responsive in the face of failure. This applies not only to highly-available, mission critical systems — any system that is not resilient will be unresponsive after a failure. Resilience is achieved by replication, containment, isolation and delegation. Failures are contained within each component, isolating components from each other and thereby ensuring that parts of the system can fail and recover without compromising the system as a whole. Recovery of each component is delegated to another (external) component and high-availability is ensured by replication where necessary. The client of a component is not burdened with handling its failures.

Elastic: The system stays responsive under varying workload. Reactive Systems can react to changes in the input rate by increasing or decreasing the resources allocated to service these inputs. This implies designs that have no contention points or central bottlenecks, resulting in the ability to shard or replicate components and distribute inputs among them. Reactive Systems support predictive, as well as Reactive, scaling algorithms by providing relevant live performance measures. They achieve elasticity in a cost-effective way on commodity hardware and software platforms.

Message Driven: Reactive Systems rely on asynchronous message-passing to establish a boundary between components that ensures loose coupling, isolation and location transparency. This boundary also provides the means to delegate failures as messages. Employing explicit message-passing enables load management, elasticity, and flow control by shaping and monitoring the message channels in the system and applying back-pressure when necessary. Location transparent messaging as a means of communication makes it possible for the management of failure to work with the same constructs and semantics across a cluster or within a single host. Non-blocking communication allows recipients to only consume resources while active, leading to less system overhead.

Hierarchical Routing

We therefore designed the Messaging Layer based on a publish-subscribe-based messaging pattern. Our protocol differs from a typical publish-subscribe model in that

ours is a hierarchical model where there are multiple layers of publisher and subscribers. The higher-level brokers will route traffic between on the higher level and the lower level brokers will route between lower level. In our current architecture, the higher level routing nodes are called Routing Nodes and the lower level routing nodes are called the Messaging Nodes. The distinction will be discussed more in the Blockchain Layer as well.

Conceptually if a message is going to /A/B, the higher-level broker will first route to A and then within A, the lower-level broker within A will then route to B. This is different from a typical publish subscribe model in that in those models, even though the channels /A/B and /A/C share a common prefix /A, it has no relevance and the channels are treated as if there is no common prefix. The reason for this is because most publish subscribe queues use text as queue names and, although the queue names look like directory structures, they really are in a flat namespace. In our design, because we do not rely on text in our queue naming at all (and in fact expect the naming to be pseudo-random), we call these queues channels and we can use the channel naming structure as a real tree-like routing mechanism.

This hierarchical design allows much more flexibility in designing the scalability of the underlying messaging backbone in the lower layers as we will see.

Another important factor of the Messaging Layer is that of providing a timely notification service. When a client connects to the system with a list of queues it's subscribed to, the system would notify the client of any new messages in the queue. The effect of this is that notification is a push model although you could consider the initial subscription when the client first connects to the system as an initiation event of a pull model.

Although conceptually close to other messaging queuing protocols, because we cannot rely on a typical HTTP or TCP/IP network stack, our protocol does not typically require long-lasting connection to the system. We made this decision because client connections will continue to be ad-hoc and mobile.

Generality of Nodes

One conscientious design decision and benefit of the hierarchical node setup is that by separating the core Routing from the additional message data services is that we can easily provide additional node types in the secondary level. For example, in additional to Messaging Nodes, we can provide Storage Nodes to provide semi-permanent storage

services. We can also provide Backup Nodes for long term backup services and/or Compute Nodes for computing services for Big Data or Machine Learning.

We started with Messaging Nodes because communication is the basis of any network but any type of service-oriented node types can be added to the network to take advantage of the network and economic fabric provided by the DCN.

With such architecture, we are much more flexible when it comes to isolating the existing service providers from the users. In essence we are providing a Service Level Agreement (SLA) between the service providers and the users of the network. For example, enterprise service providers such as Amazon(™) or Google(™) can certainly provide their expertise in providing storage, computing, and myriad other services and compete with each other or other providers fairly in DCN "market."

We believe net neutrality is not just the networking pipe but also the services that sits on top of the basic networking stack. We hope to help the cause of net neutrality by equalizing the service providers without the vendor lock in.

Actions

The Messaging Layer works like any other queue-based systems in that when a client is connected to the messaging backbone, the client only needs the following actions:

1. *Connect*
2. *Disconnect*
3. *Publish*
4. *Subscribe*

Quality of Service

There is also a level of control in the Quality of Services (QoS) for each published message:

1. *At most delivery*
2. *At least once delivery*
3. *Exactly once delivery*

The Messaging Layer will present this abstraction to higher level but will achieve this using the lower layer. For example, messages will need to be duplicated to ensure

deliverability and stored temporarily on multiple nodes. When the time expires on the message, the messages will need to be expired. Keeping track of duplications and expirations of messages is a task for both the Messaging Layer and the Decentralization Layer. The Messaging Layer deals with the technical aspect of the protocol for message scattering and gathering. The Decentralization Layer

We understand in a distributed system it is actually difficult to have the Exactly once delivery so it needs to be approximated by the system by tossing away messages that are delivered twice if that QoS level is chosen.

https://bravenewgeek.com/you-cannot-have-exactly-once-delivery/

Message Scattering

A long message will need to be chopped into smaller message fragments and scattered within the network. This is to ensure that messages are of similar length for manageability and ease of storage needs.

Message Gathering

A long message that has been split into smaller fragments will eventually need to be gathered back, merged, and delivered to the client who are subscribed to the channels.

Message Duplication

Messages will need to be transparently duplicated in order to preserve the messages in case of node and network failures. This is particularly important in a decentralized network such as StrongSalt where no one controls all the storage nodes.

Message Expiration

All messages have an expiration because StrongSalt is not a permanent storage network. Although similar to how DRAM can be refreshed to work as if the storage is permanent for all practical purposes, a higher level application can potentially build a highly scalable storage network on top of StrongSalt.

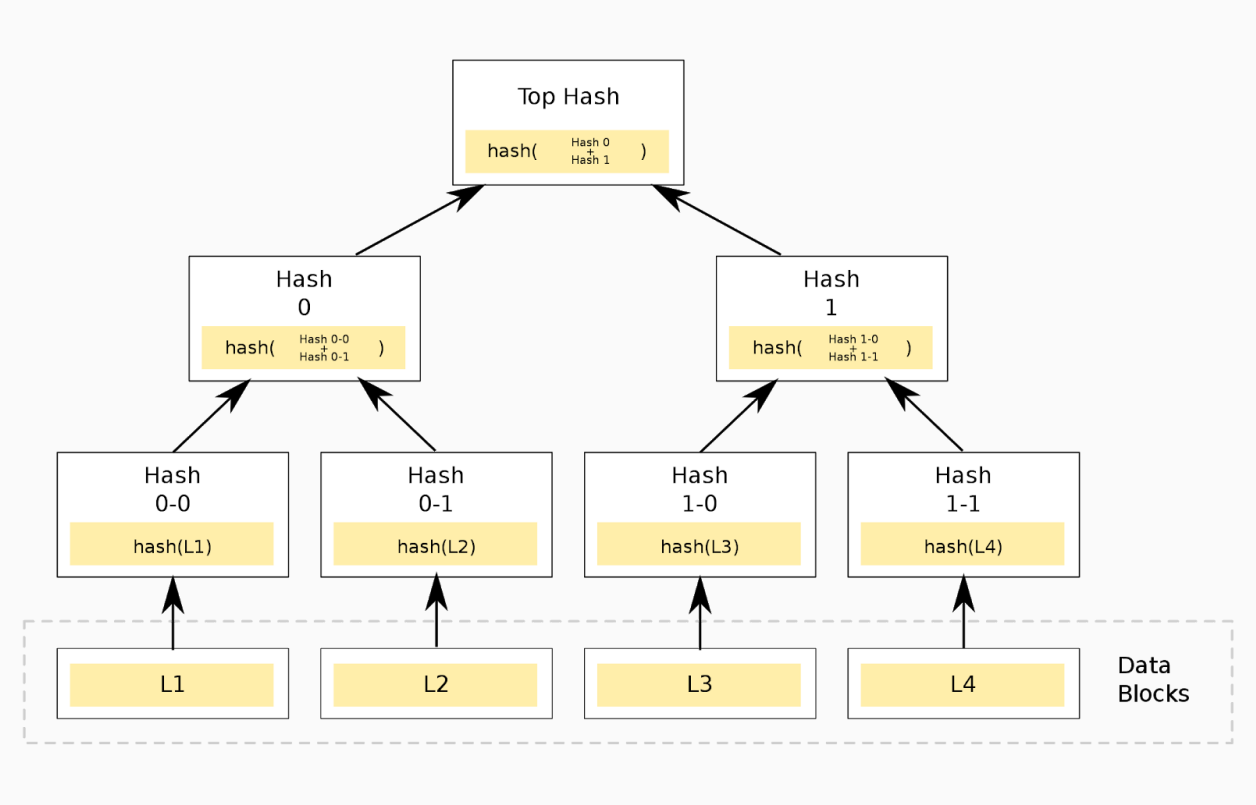
Because messages can expire, the system will over time “garbage collect” expired messages and discard them. Because we are on a decentralized network, it is costly to test whether the messages are indeed removed. This is one of the reason why messages need to be scattered and encrypted so that messages cannot be easily deciphered by anyone collecting expired messages.

Merkel Tree

The message scattering and gathering are kept track in a Merkel Tree structure similar to how a tracker tracks all the pieces of a Bittorrent file. The reason a Merkel Tree is used is because it allows tracking of all the different pieces of the file while verifying there is no tampering or destruction of message fragments.

The Merkel Tree is also instrumental in getting the accounting of the temporary storages by the Message Nodes in the Blockchain Layer. The Blockchain Layer keeps track of the work done by each of the nodes via the mapping of hash of the messages to the Messaging Nodes. A Merkel Tree hash of multiple messages can be used for each transaction instead of having all the fragments of each message be a transaction. This significantly reduces the amount of extraneous information on the blockchain. Not only does this reduce the size of the blockchain to help scale the network faster but also provides a certain level of privacy for the transactions without harming the auditability.

Another reason Merkel Tree is important is when the actual message was scattered into fragments and needs to be gathered and merged from the fragments. By having a Merkel Tree structure to represent the fragments, it’s always possible to know which pieces are missing from the fragments instead of asking for retransmission of the full message which can be arbitrarily large.



Example of Merkel Tree from wikipedia

Isolation of Concerns

The Messaging Layer is responsible for routing traffic between the nodes to be described in the Decentralization (Blockchain) Layer. It does not rely on any particular network transport so ideally, it can use any network transport whether it's TCP or UDP and whether SSL is used because at this level, it does not matter and the protocol at the Messaging Layer has no visibility into the lower Transport Layer.

Communication Pattern

The simple case of Alice communicating with Bob starts when Alice publishes to a channel and that Bob subscribes to that same channel. Alice and Bob do not need to reside anywhere close from a networking perspective and they do not need to be connected to the same Routing Nodes or Messaging Nodes. When Alice connects to a Messaging Node M1, she registers the channel ("C1") with with Messaging Node M1 which then forwards the registration to the Routing Node R1. The Routing Node R1 takes note of this and marks that C1 is on M1. At the same time, Bob subscribes to

channel C1 independently and sends that information to the Messaging Node M2 which then forwards the information to the Routing Node R2 to take note that M2 is subscribed to C1. At this point, Alice and Bob have no physical network connection at all but Alice can start publishing any messages to C1 which are then stored by M1. As the Routing Nodes communicate and exchange channel registration information, R1 is notified that R2 is subscribed to C1. R1 then subscribes to the C1 from M1. At this point, a virtual circuit is established. As soon as Alice publishes any message to C1, because R1 is subscribed to /M1/C1, R1 will be notified of these messages. Because R2 is subscribed to /R1/C1, R1 will notify R2 of these messages. And because M2 is subscribed to /R2/C1, M2 in turn is notified of these messages as well. And finally because Bob is subscribed to /M2/C1, Bob also is notified. At this point the virtual circuit is complete. Any messages published to C1 will now be routed all the way from Alice to Bob. The network can enforce deliverability based on the QoS level selected on the channel.

One thing to note is that at each level of pub-sub messaging, the messages can be scattered and gathered back depending on the level and the need. For example, Alice sends the full message to M1 and Bob receives the full message from M2, but the communication between R1 and R2 may consists of information regarding the multiple message fragments and R2 will not gather messages directly from M1. In fact, R1 will most likely select the other messaging nodes received pieces of the messages and create a metadata message to send to /R2/C1 which in turn will notify M2. M2 can then use the metadata to gather up all the fragments and assemble the message to forward to Bob. Note that the ideal case is that messages are not copied unnecessarily between M1 and M2 that the metadata about the fragments for the messages are sent between R1 and R2 so that M2 can independently receive the full message payload by receiving the fragments from the Messaging Nodes.

Conceptually it’s similar to R1 eventually sending a bittorrent tracker to R2 which then passes to M2 which then gathers up all the fragments and then assembles into a message to send to Bob.

Blockchain or Decentralization Layer

The Blockchain Layer is composed of at the moments three types of nodes: Routing Nodes, Messaging Nodes, and Client Nodes.

Participants

Routing Nodes:

Routing nodes were originally designed to be mining nodes when we launch our own blockchain. However, initially we will be relying on an existing blockchain implementation, so Routing Nodes do not need to participate in the mining. The mechanism described would utilize a smart contract to distribute the tokens instead.

The purpose of the Routing Nodes is to establish a routing table for some of the “neighboring” Routing Nodes in order to to route messages to appropriate channels. Routing nodes are the higher-level brokers mentioned earlier. They are responsible for routing traffic based on the channels between other Routing Nodes and route lower-level messages (those destined for the network in which this particular Routing Node resides) amongst the Message Nodes.

The Routing Nodes are responsible for keeping track of the contributions done by the Messaging Nodes based on the number of messages being stored and retrieved from the Messaging Nodes and write out transactions in the blockchain compensating the Messaging Nodes. The Routing Nodes also splits a percentage of the tokens submitted for these messages among other Routing Nodes. Because the blockchain is public, the Routing Nodes can audit the transactions published on the blocks to makes sure every Routing Node is playing by the rules.

Messaging Nodes:

Messaging Nodes are nodes that forward client messages to Routing Nodes and provide message storage for the message duplication and scattering processes. Message Nodes will notify the connected clients of messages in the channels that they subscribed to or forward messages to the Routing Nodes when the clients publish to the channel.

The Messaging Nodes are not necessarily the same ones who provide the message retrieval as when the messages are forwarded to the Routing Nodes, the Routing Nodes can decide to split the messages into fragments and send the fragments to be duplicated by other Messaging Nodes. However, when the channels that the client is interested in have new messages, the clients will notify any connected clients that have subscribed to such channels. The channels will have information on how to retrieve the

messages from the Routing Nodes which in turn will send gathering messages to the Messaging Nodes.

All the communication between all the participating Messaging Nodes is done via the same pub-sub messaging backbone except that the messaging nodes act as brokers for the clients and that the Routing Nodes act as brokers for the Messaging Nodes.

Blockchain Implementation

The questions for any new decentralized protocol are: 1. Do you use a blockchain? 2. Which blockchain? For 1, our answer has been yes.

However, question 2 is where choices become a problem. We believe there are two answers to question 2. The long term answer to question 2 is that we will likely eventually create our own blockchain because almost all current blockchains are meant for financial transactions and as a result they are scaled in a way that's suboptimal for a network-based communication transactions. In the short term, we can certainly use existing blockchain implementations because fundamentally the DCN does not rely on the blockchain to function from a technical perspective. This is another important reason why we do not rely on blockchain for the actual message passing itself. However, because token incentivization is important for the overall security and scalability of the system, we still need to pay attention to the Blockchain Layer because all accounting of the token spending and token earning are tracked by the blockchain.

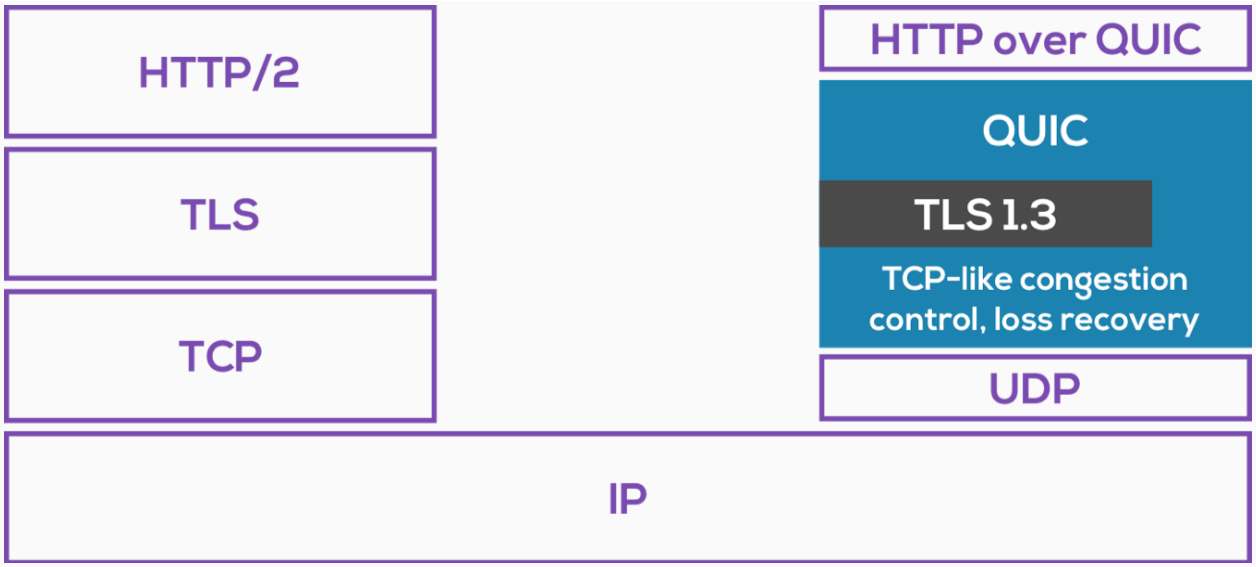
Looking towards the future. We have a different kind of blockchain in mind. We do not think a simple signature-based (it still has a state machine) blockchain (those derived from Bitcoin blockchain) is sufficient in the long run for a DCN. Alternatively, the smart contract based blockchains are even more problematic when it comes to security and long term scalability challenges. We do not currently believe a Turing complete blockchain is necessarily or beneficial for the DCN. We therefore take a middle ground that our future blockchain is a little more complex than a signature-based VM but much simpler than a full Turing-complete VM. Our approach is actually called if-this-then-that which is highly suitable for routing and connecting smart devices. We believe such network would make DCN much more generic and offers functions that are powerful enough for DCN use-cases while providing scalability and performance.

Network Transport Layer

We plan to run our protocol on top of QUIC instead of the typical TCP protocol. We can significantly increase the security of the overall system by utilizing the security primitives on QUIC because, by default, QUIC encrypts all the application data and most of the protocol headers.

QUIC is initially developed by Google to incorporate ideas from TCP and TLS and placed them on top of the much lighter UDP protocol. It improved upon the RTT connection establishment by allowing a client to start a new connection without the typical three-way handshake if the client has previously communicated with the server. This reduction in initial latency along with the introduction of multiple streams per connection and improved acknowledgement information allows QUIC to much more efficiently utilize the connections between devices.

What makes QUIC so suitable for our DCN is because of the short messages that are passed not only between core Routing Nodes but also between the Message Nodes. Such traffic pattern is highly subject to latency concerns in a decentralized network. Our adoption of the QUIC protocol significantly decreases latency for these communications.



Why is latency so important for network protocol?

The reason is that by using a decentralized backbone to help preserve privacy, the network protocol can incur significant latency. Unfortunately, even when technology improves the amount of available bandwidth, such as when we go from 3G to LTE,

latency is not significantly improved because we are limited by the speed of light. Unless we have some breakthrough in physics, the only way to cut down latency is to cut down round trips. By implementing our network protocol on top of QUIC we can improve latency significantly by decreasing the number of round trips needed in TCP connections in addition to improving on the security.

Latency is a even bigger performance problem in a decentralized network where by the nature of decentralization there are usually more parties involved especially when messages need to be replicated and scattered and gathered by multiple parties. Because the nodes are not necessarily chosen by geographic locations (but certain mechanics such as *ping* can be used to determine relative latency between nodes), latency becomes a much bigger issue in a decentralized network. We therefore propose to alleviate such problem in our network using our Transport Layer.

Security in layers

Security is a layered approach and because StrongSalt is composed of multiple layers, each layer must deal with security concerns of its own later without duplicating the efforts of the other layer. Each of the layers addresses one particular aspect of security but when the layers are added together, the security of the overall system should be more than the sum of that of each layer.

The Transport Layer will provide transport security between nodes so that the traffic between the nodes are secure and most of the meta data are not exposed to malicious attackers. The Blockchain Layer provides the security needed for the core Routers to work together and to provide the trust needed for the protocol to function without having a trusted third party. The Messaging Layer provides the security for messaging passing to prevent tampering of the messages while it passes from the network from the sender to the eventual receiver. The API Layer actually can provide additional services such as encryption on top of what's provided by the lower layers so that the content security is provided for the application layer without the need of application developers to deal with which encryption and hashing algorithms to use. The API Layer will handle one of the most difficult problem that developers (especially those who are not well-versed in cryptography) have to deal with such as choosing which encryption scheme (for example, AES 128 vs 256, ECB vs CBC vs CTR vs GCM, etc) to use and what if a vulnerability is found (for example, the move from SHA1 to SHA256 etc). The Application Layer then deals with the security of user management and any additional

functionalities such as disappearing messages (this is technically impossible but can be approximated by various cryptographic schemes as we explore in our StrongSalt App).

Certificate

In order to authenticate a device, every client must have a certificate which is signed by a certificate authority (CA) trusted by StrongSalt. When a communication channel is requested by the device, the device must send the certificate which will be verified by a node. The certificate can be assigned by a node or burned in at the factory. The certificate is only used to authenticate the device. Each of the nodes must set up a certificate of its own to be identified as a trusted system for the clients to connect to.

The use of certificate is to approve the devices to connect to the Messaging Nodes in the network.

The policies associated with the certificates determines the authorizations that the devices have for communication. By default, all topics are locked down so that only devices approved for communication can subscribe and/or publish to the channels.

What are the security primitives?

Because complexity is the enemy of execution, we intentionally choose either protocols or cryptographic ciphers that are well understood or researched to make sure our implementation would be secure and easily audited by researchers and technologists both in the academic circles or in the industry. We use the current widespread crypto for security reasons and as any security software, when mathematics, quantum computers, and hardware advances, our crypto will be updated to reflect the latest best practice for strong security.

All numbers are to be encoded using network big-endian order.

We use 256 bit AES in CTR mode along with HMAC for authentication.
We also use SHA256 as our hashing algorithm.
We also use the Curve25519 for Diffie-Hellman key exchange (X25519) by D.J. Bernstein. We do use Ed25519 for signatures which is intended to provide attack resistance comparable to 128-bit symmetric ciphers. Public keys are 256 bits (32 bytes) in length and signatures are 512 bits (64 bytes).

Security Attacks

In 2015, there was an article “How Secure and Quick is QUIC? Provable Security and Performance Analyses” which has been published at the IEEE Symposium on Security and Privacy that has presented a provable security analysis of QUIC in which a new model more suitable for QUIC because QUIC must consider reordering and packet injection issues that are normally handled by TCP to which TLS can hand over.

This new model, Quick Authenticated and Confidential Channel Establishment (QACCE), considers attackers who can initiate and observe communications in addition to intercept, drop, reorder, or modify any exchanged messages. The study found that “QUIC can successfully protect against such strong attackers and provide QACCE security.”

Although QUIC can have performance degradation attacks which do not impact the authenticity or confidentiality of data sent over QUIC. Most of the attacks are due to the fact that in order to optimize 0-RRT connection (the first connection from the client), the 0-RRT data is encrypted under the initial key derived from the server’s config object. Replays of the server config if the server config has not expired.

(https://www.ietfjournal.org/quic-performance-and-security-at-the-transport-layer/)

IoT Messaging Protocol

Some of the criteria for evaluating protocols are:

- Confidentiality
- Low protocol overhead
- Unstable network tolerance
- Low power usage
- Millions of connected clients
- Push communication

https://www.ibm.com/developerworks/websphere/techjournal/1501_maynard/1501_maynard.html

We believe our StrongSalt protocol can demonstrate all of these traits and thus can be highly suited for IoT secure communication in the future.

Why don’t we just use Signal, Telegram, or another existing product?

Although Signal also has an app, the purpose of Signal protocol is to provide the end-to-end encryption protocol for primarily two party real-time messaging. Signal is a great application level protocol and we can and will use some of the concepts of the double ratchet protocol for end-to-end encryption in the Application Layer. However, because Signal is an application level protocol, it does not help with decentralization.

We believe there is significant value in providing a Decentralized Communication Network not just in protocol but also in the Network itself, so others can utilize the same network to send chat messages encrypted end-to-end and many other transactions.

Telegram is actually an application so once again it’s an app on the application Layer. Once again, it’s also centralized. In addition, Telegram does not encrypt multi-party communication.

Pretty much every messaging app whether they claim to be secure or not is based on a centralized model. We decided that it’s more important to design a secure Decentralized Communication Network that eventually other applications can migrate onto without having everyone duplicating the effort of building the lower level protocols and provide network that the community would need in order to build additional decentralized messaging or transactional applications.

Why don’t we just use Tor?

Tor started a while ago and while it can be used for communication, it was originally intended to anonymize access from clients to the server, which asymmetric nature is distinct from a generic communication platform where communication can be p2p and can be initiated bi-directionally.
Another reason is that because of the website use case, Tor is really meant to be used as 1 to 1 communication where the client accesses the information on the server. For a

generic communication, we cannot limit our communication protocol to 1 to 1 communication or even N to 1 communication. It has to support M to N communication. In addition, for generic communication, multiple parties can be involved and that some parties involved may not be online or cannot be reached at the time of communication. For example, when sending a message intended for recipient the recipient may not be online at the time. Tor was not designed to handle such mode of communication. Another reason is because Tor started a while ago, some of the cryptos they use are no longer the recommended settings. For example, Tor uses SHA1 instead of SHA256 that we use. In addition, because Tor needs to maintain backward compatibility it still needs to make sure the old “hybrid encryption” can be used (although it recommends not using it in the Tor specification).

Even more importantly, Tor is designed to anonymize high-latency TCP-based applications so it’s more geared towards surfing the web anonymously and would be considered closer to an application level protocol whereas our intention is to be a modern network level protocol utilizing some of the advances of network technology such as the use of QUIC protocol based on a new TLS on top of UDP protocol that combined the best features of TCP, TLS, and HTTP2 but with significantly improved security and latency. The QUIC protocol is originally designed by Jim Roskind and his team at Google as a new stream multiplexing protocol with the explicit intention of reducing latency compared to that of TCP.

How are we different?

We understand that there are several other projects that sound similar to our DCN on a casual pass on the website. We are glad that there are increasingly more privacy-focused projects because we could always use more privacy in a world where targeted advertising revenue is more important than privacy.

However, to innovate is to show merits so here are some key differentiators of our DCN:

1. We are designed for large binary data streams: Most projects when they mention the word *communication* or *messaging*, they typically mean either email or chat messages which are mostly text-based. We understand that text messages and email are the foundation of human communication but we believe the future of communication is much more data driven. In other words, the communication is not merely text but can be concurrent multi-streaming of video or a stream of never-ending binary data from IoT devices.

2. We are run on lower level networking protocols: Most protocols are based on the application layer. Although some protocols can be used by other applications they are still on top the actual networking stack. Although it takes significantly more work to propose a protocol that goes lower in the network stack (such as our DCN), we do believe there are much more values to do so for a decentralized network because decentralization makes performance optimization difficult if the protocol is application level only.
3. We believe decentralization is not only peer-to-peer: We currently do not use dark routing, onion routing, or other peer-to-peer networking protocol because our goal is not just to preserve privacy at the expense of performance. The purpose of any communication network (decentralized or not) is foremost *efficient* communication. We actually see value in other overlay networks to be built on top of our DCN. Currently, a purely peer-to-peer communication network will not be efficient enough for the high-bandwidth low latency requirements of the existing usage pattern of the Internet.
4. We treat blockchain as first-class citizen of the protocol: Many other protocols require tokens for incentivization but do not require a new type of blockchain. Because our architecture requires message routing based on token distribution and our protocol is designed with blockchain sandwiched between the Messaging and Transport Layers, we have a need for a VM that are closer to the logic of a secure networking switch (if this then that) instead of a Turing-complete VM for smart contracts.
5. We welcome participation of existing cloud infrastructure: The purpose of our DCN is to democratize networking infrastructure and embrace network neutrality. We want to realize this goal by making it more beneficial for the networking and cloud infrastructure companies to participate not by bypassing their network or ignoring the huge computing and storage capability of the existing infrastructure providers. We designed the DCN to provide additional infrastructure services as our network matures.

To summarize, we are not merely a messaging protocol or an overlay network. We designed the DCN foremost as a communication *network* that does not depend only on peer-to-peer communication but on a decentralized network where existing cloud infrastructure are incentivized to contribute. We would like the DCN to be a building block of the future internet where smaller volunteers and larger corporations can compete on merits instead of proprietary fragmentation.

http://ieeexplore.ieee.org/document/4151637/

Future Use Cases

There are some obvious use cases of the DCN such as a decentralized video conferencing application that allows video conferences to be routed and coordinated by a decentralized network of operators. Such application can be used to preserve privacy for the parties involved in the video conferencing. Such

Another use case is in the enterprise group collaboration market. Currently, many cloud-based enterprise group messaging products still hold all data in their clouds. In other words, the companies that offer these services have complete control and visibility to the conversations. Often, these companies would put on words such as encryption and firewall on the marketing website to alleviate such concerns. However, as long as those companies are the ones that do the encryption and hold the keys, there is no data privacy per se. DCN can be used to create an enterprise group messaging application that does not hold enterprise data hostage.

Currently, the DCN is setup for storage service for the need of file sharing. However, in the future, additional nodes can be added to provide additional services on the same DCN. This is similar to how Internet evolved. First you need the network, then you can provide services.

Potentially additional data services such as long term data storage can be layered on top of the DCN so that additional nodes can provide long term storage service in exchange for tokens in the system. Similarly, a decentralized computing service can use the DCN to deliver computational services on the DCN by providing additional computing services.

In some ways, the participating Message Nodes function similar to cloud services so that the DCN can be considered a decentralized AWS(™) or Google Cloud(™) that provides various services via the Message Nodes. We believe this model of DCN is much more efficient because specialized service providers such as Amazon(™) or Google(™) can potentially participate in the DCN because they already do have the expertise in providing computing and storage services and DCN can be a delivery network of their services.

Another interesting use case is an application that explores data escrowing with payments guarded by conditions enforced by smart contracts. The difference of this

service compared to a generic blockchain smart contract is to limit the use case to data escrowing only. A list of pre-coded smart contracts of various functions are hidden behind an user interface so that non-programmers can choose which way to charge for the work done (say a piece of digital art) and whether a flat price or an auction for the piece, etc. The escrow will only release the work or charge the tokens after the conditions are satisfied such as a preview is only downloadable after the token is in the escrow account but the tokens is released to the seller only after the download is completed.

Team

Founder

Ed Yu

Ed Yu was the founding engineer of FireEye and has also worked as VP and Director of Engineering in various enterprise and software companies. He has worked both in startups and large corporations such as Oracle, Sun, SGI, and McAfee.

Ed graduated from the California Institute of Technology with a Bachelor's degree in Computer Science.

Ed loves scuba diving, skydiving, and tai chi but with his current busy life, he spends more time annoying his coworkers with uncommented Haskell code.

Advisors

Dan Boneh

Dan Boneh is a professor of computer science and electrical engineering at Stanford University. He is also the Co-Director of the Stanford Computer Science Lab. Dan's main research focus is applied cryptography and computer security. His work includes applications of cryptography to computer security, cryptosystems with novel properties, cryptanalysis and Web security.

http://crypto.stanford.edu/~dabo/

Matthew D. Green

Matthew Daniel Green is an Assistant Professor of Computer Science at the Johns Hopkins Information Security Institute. He specializes in applied cryptography, privacy-enhanced information storage systems, anonymous cryptocurrencies, elliptic curve crypto-systems, and satellite television piracy. He is a member of the teams that developed the Zerocoin anonymous cryptocurrency and Zerocash. He has been involved in the groups that exposed vulnerabilities in RSA BSAFE, Speedpass and E-ZPass.

https://blog.cryptographyengineering.com/

Mike Weimer

In 2007, Mike co-founded Solar Junction and led the teams to two world records in Solar Cell efficiency (43.5% and 44%) and then sold the company in 2014. Mike is now building a new company. Mike also advises early stage companies on their strategies.

Mike has a BS, MS, and PhD from Stanford University, all in Electrical Engineering. He has patents and papers in Semiconductor Devices & Applications, Silicon Photonics, Materials Integration, Lasers, Solar Cells, Solar Systems, and Analog Circuits. He has given talks on technology and entrepreneurship at Universities and Events in the US, Middle East, and Asia, and speaks regularly at Stanford University.

Conclusion

StrongSalt is a decentralized overlay network designed to incentivize volunteers to build a mesh network of nodes that can preserve privacy, promote net-neutrality, and enhance security for current and future network communication protocols such as video messaging, real-time video conferencing, and social network-based group messaging.

The intent of the StrongSalt Protocol is to build an ever growing decentralized network of volunteers incentivized not just by ideology but also by the rewards of protocol token that can be utilized by the multitude of services on the network. In other words, by

helping out the network by volunteering network bandwidth, storage capacity and computing resources, the relay nodes are rewarded with tokens which in turn can be used to use the network for the services provided by other nodes and the network itself. This creates a virtuous circle that will continue to expand and democratize network reachability and enhance the security and privacy of the “last-mile.”

We therefore fully intent to help build a community of volunteers which will overshadow the involvement of our company in providing the basic network messaging nodes. We do not want to be the trusted Messaging node provider because trusting any one company or an organization can be a risky business for anything as fundamental as network protocol. We believe with the StrongSalt Protocol, it is possible most of the Messaging nodes in the network mesh to consist of mostly volunteers who are not part of our company. Because the tokens are what incentivizes overlaying of the network messaging nodes, it’s an essential that the nodes are rewarded fairly and transparently. We therefore rely on a blockchain-based platform to record transactions related to providing the messaging services in the network. Anyone participating in the network can be rewarded by participation only but can also compete by providing lower network latency, higher network bandwidth, more dense storage capacity and faster computing resources.

StrongSalt is a decentralized communication network built on top of a new type of blockchain optimized for secure communication between either humans or IoT devices such as your mobile phones or autonomous car. Not only will StrongSalt protect data security of the IoT devices but it will help protect the privacy of humans who are using those smart devices. StrongSalt is a combination of a new network transport layer protocol, a blockchain based messaging platform, and an API that takes away the complexity of secure communication for smart devices.

https://www.chromium.org/quic
https://tools.ietf.org/id/draft-ietf-quic-applicability-01.html