RightMesh™

‘17

TECHNICAL WHITE PAPER

rightmesh.io

September 7, ‘17

Version 1.0

This is a draft document for open community review.
Subject to change.

Technical White Paper
Connecting the Next Billion Users with an Ad Hoc Wireless Mesh Networking Platform

Authors: Dr. Jason Ernst, Dr. Zehua (David) Wang, Saju Abraham, John Lyotier, Chris Jensen, Melissa Quinn, Aldrin D'Souza

Change Log

Date	Version	Comments/Edits	Contributor
Sept 7, 2017	v.1.0	Initial version of the technical paper	JE

TABLE OF CONTENTS

THIS IS NOT A PROSPECTUS OF ANY SORT

This document is a supplement to the RightMesh Whitepaper and neither document constitutes a prospectus of any sort*; it is not a solicitation for investment and does not in any way pertain to an offering of securities in either Canada or the United States, and Canadian and United States residents are expressly excluded from purchasing any RightMesh Tokens. This document constitutes a description of the RightMesh platform and the functionality of the RightMesh tokens.*

____

Introduction

This technical paper is intended as a supplement to the *RightMesh White Paper*, which more broadly provides an overview of the opportunity for RightMesh and the RightMesh Token Generating Event. This technical white paper gives more details and assumes in-depth knowledge of complex systems, networking, encryption, and cryptocurrencies. Throughout each section, the general high-level architecture is presented, along with justifications for design decisions. Our progress to date will be presented, and areas where future work is required will be specified in each section. We will conclude with a summary of the ongoing technical roadmap which we believe helps to support the original paper.

The document makes the technical case for a tokenized mobile mesh network where the devices are mostly smartphones, IoT devices, sensors, automobiles, and other devices that have had difficulty connecting traditionally since they may move into (or exist in) parts of the world that are connected to infrastructure poorly--and are likely to remain as such for some time. The token system is designed to incentivize devices and people to join a network, stay within a network, and act as virtual infrastructure. It has been designed so that it can work immediately in today's cryptocurrency environment. That is, the system operates under the assumption that smartphones are still not capable of participating as full crypto nodes. At the same time, decisions to "future proof" the technology have been made with the assumption that this will not always be the case.

MeshID

Given that RightMesh is able to support connecting together multiple Wi-Fi hotspots—each with their own set of IP addresses (both IPv4 and IPv6), as well as connecting via Bluetooth (with MAC addresses), and in the future other networks where there may be even more different addressing schemes—one the first problems we aimed solve is how to identify devices in the network uniquely. While building YO! (one of the company's first offline apps), this was handled in the typical manner where unique IDs are generated from an Internet-connected server. In a mesh where the devices may never touch the Internet, or may not for a long period of time, this makes it impossible to form the identity. It is a chicken and egg problem. With RightMesh, we needed to generate the ID on the device without requiring the Internet to do so. Researching this problem led the company into cryptocurrencies, where this problem had already been addressed and proven to do so in a way where there is almost no chance of ever generating the same ID twice.

Currently, RightMesh generates an Ethereum account using the Ethereum-j library. The account generated is compatible with any popular Ethereum client, such as geth, mist, or myetherwallet. This identity is generated once when the RightMesh library first launches on the device, and remains on the device until the user has removed it manually. This means if RightMesh apps are installed and uninstalled, the same identity will be present unless the user removes it. All RightMesh apps that are running at the same time also share the same identity, similar to how your computer has the same IP address for your web browser, your Slack client, and your games (unless you leave your house and travel to work with your computer and get a different IP address at your new location). With RightMesh, despite all the movement and changing IPs while you move, your RightMesh identity will stay constant. This permanence is how other devices in the mesh will always be able to deliver data to the right device.

The identity on the mesh is the public Ethereum address that gets generated along with the account. The generated account is encrypted using a default password that RightMesh provides (as a proof that the service can do the encryption correctly). It also contains the public and private key associated with the account to ensure it works on the Ethereum network when executing transactions. At launch time, RightMesh aims to have the MeshID functioning such that a user will be able to set their own password for the encryption so that their account is locked and protected by the password as they would expect with an account generated by any other client.

Ethereum was selected as a platform mainly for its smart contract capabilities which the company uses for incentivizing the mesh (more details in later sections). However, Ethereum could be swapped for any cryptocurrency that operates in a similar manner. There are several drawbacks currently with using Ethereum, such as needing Ether in the account in order to transfer tokens (to pay for the gas), the high cost of small transactions, and the scalability problems that have been seen in some recent ICOs and TGEs. However, many of these problems are actively being worked on by the Ethereum community. For instance, it will soon be possible to pay for the gas of a transaction with the token rather than Ethereum. There are further enhancements coming that will enable microtransactions to occur more affordably.

Despite the ongoing development to Ethereum to address these issues, it may also be possible for some aspects of RightMesh—such as fast, small transactions—to use other similar decentralized solutions such as Lightning, Ripple or Raiden networks. We have built proof-of-concepts for incentivizing the mesh using Ethereum as a base for now; however, we recognize our expertise is in mesh networks, and while we are learning quickly, the company would like to work with the decentralized community to develop the incentivization aspects of the solution in the best possible way and are flexible to change this aspect of the solution. All of the portions of our library that touch the cryptocurrency and incentivization side will be open sourced, along with all of the sample apps built by RightMesh, so the community can understand deeply how our solution works and work with us to build something incredible.

RightMesh System Overview

Figure 1 shows the high-level system overview and tries to capture all of the possible layers and roles within our system from the perspective of a data-sharing network where there are some devices that wish to sell their data for RightMesh Tokens (MESH); some that wish to purchase; and some that wish to participate as forwarding (infrastructure) nodes. At the highest layer, we also provide some additional infrastructure that presently cannot be eliminated until it is more feasible to run full Ethereum nodes directly on the mobile devices themselves (and to get around things like firewalls put in place by existing ISPs).

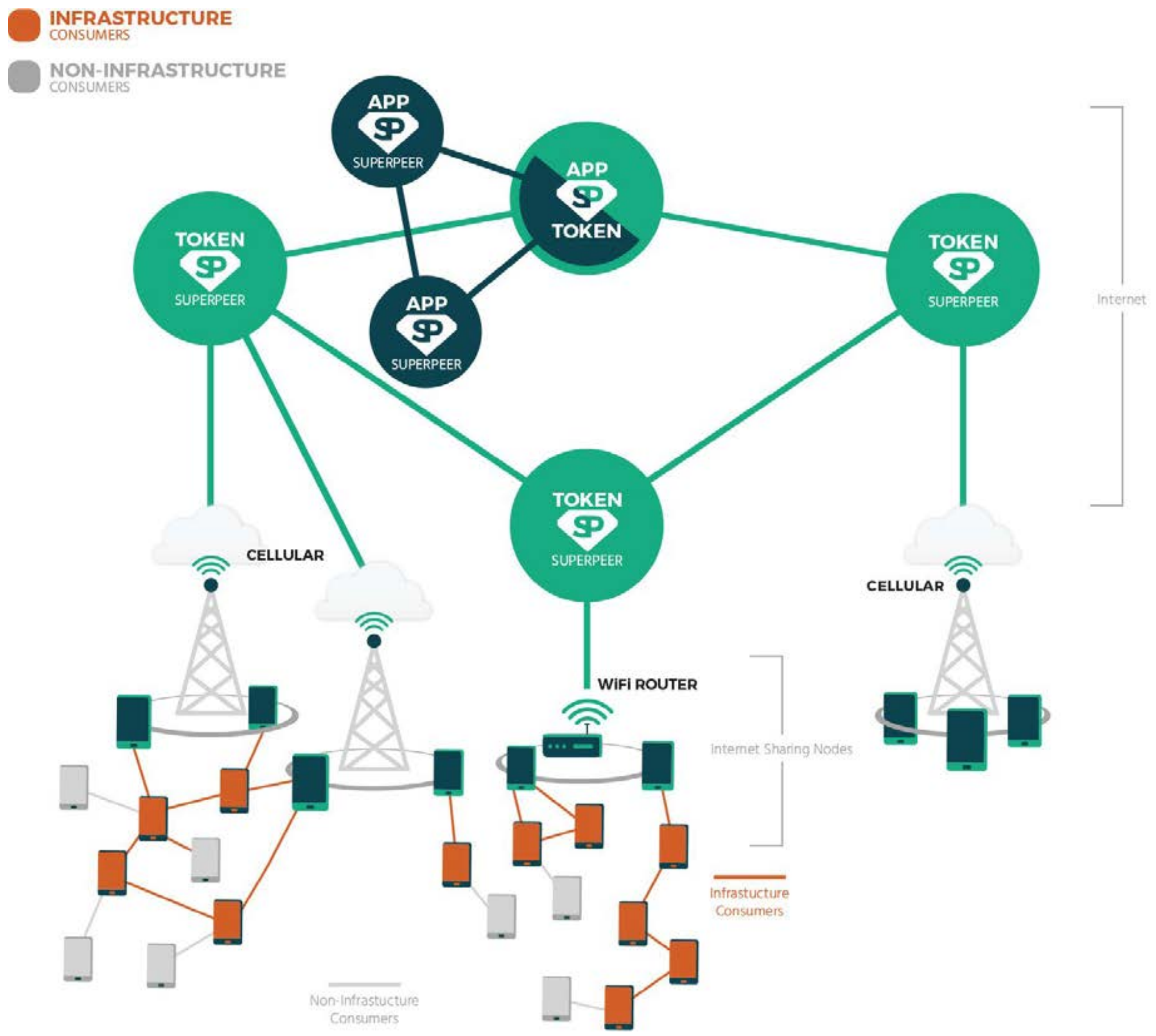


Figure 1: High-level system overview

Roles

In all of the following discussions, the roles and layers are specific only to data traffic. In some cases, devices which are unwilling to forward data can still earn tokens in other ways—such as providing storage, processing, or other resources.

In *Figure 1*, there are a few key pieces and layers to this system. There are token superpeers (labelled as such), which are responsible for the routing of packets between geographically separate meshes, and for relaying Ethereum transactions from devices in the mesh to the real Ethereum network. There are also app superpeers (labelled as such - more details to follow). It is also possible for app superpeers and token superpeers to exist on the same node. The phones directly connected to cell towers, Wi-Fi hotspots and other Internet connections are Internet sharing devices, which may or may not actually share their Internet access. In orange are infrastructure nodes (which may also be consumer nodes). In grey are non-infrastructure consumer nodes.

Token superpeer devices are running full Ethereum nodes, and execute signed transactions on the Ethereum network on behalf of the participating devices. These nodes also act as the main linkage between geographically separate meshes. For instance, on the far left connected to the cellular network, we can consider that mesh to be made of devices in Canada, while the mesh connected to the Wi-Fi router in the middle could be a mesh formed in Bangladesh. Since they are too far away for a single mesh to form, we have to rely on existing infrastructure (ISPs, wires, etc.), and use token superpeers to facilitate this communication. There are two ways in which this can be achieved.

First, the method that is working right now is to simply act as a forwarding node, and directly forward all traffic from one mesh to another. The second method, is to act more as a lookup service. In this case, the token superpeer would be able to determine which other Internet sharing device (in blue) traffic needs to be routed to. After this is determined, the information is sent back to the Internet sharing device on the source side where it will communicate directly with the Internet sharing device on the destination side. This is more complex since it must be able to get around firewalls in order to function on all networks. This second method is left as a future exercise that will bring our costs down (since we will lower our data costs from operating the token superpeer on AWS or some similar service).

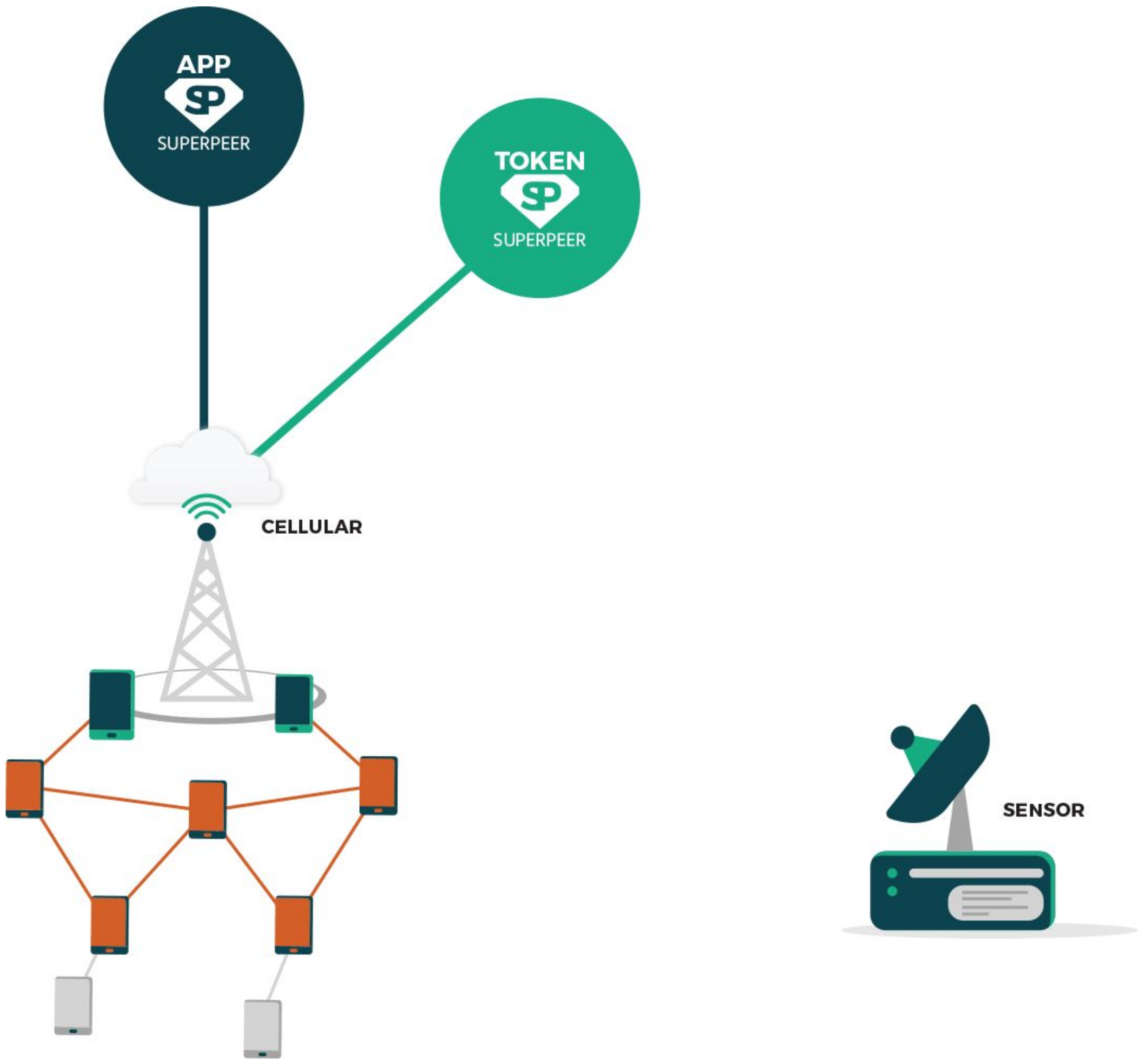


Figure 2: "Data-mule" capability linking geographically separate meshes where some lack Internet connectivity

In addition to token superpeers, it is also possible for app developers to provide their own superpeers. In this scenario, the app superpeer would act as a trusted device (trusted only by the app which is making use of it). For instance, an app which allows people to play geocaching may also deploy a geocaching superpeer. While the app lets people discover geocaching locations, where the location is actually a remote sensor, which relays data into the mesh eventually towards the geocaching superpeer, where the data may be stored and presented on normal traditional websites. This "data-mule" behaviour is demonstrated in *Figure 2*. The app superpeer would be able to provide some further infrastructure needs that a normal mesh app could not provide. For instance, consider a mesh app where there are remote sensors, but in a location where people occasionally pass (e.g., in the mountains or in northern Canada). An app superpeer would be used as a collection point where the data would be stored for visualization, analysis, etc., and the RightMesh library would handle routing, despite the phones not always being connected to the Internet. In our remote sensor example as a hiker passes by, it would forward the collected data the next time the phone had a connection to the app superpeer through RightMesh automatically. Note that app and token superpeers may or may not exist on the same, physical AWS instance.

Internet sharing devices (SUPPLY) may be connected to the Internet using a cellular network, Wi-Fi, Ethernet (if the sharing device is a router, computer, etc.), and have the ability to share their Internet connection through the RightMesh library. Note: Internet sharing at this time, means that we can use the Internet connection to link to other geographically separate meshes, or to superpeers, but not general purpose Internet access presently. General purpose Internet traffic is on the roadmap, however.

Layers

The app and token superpeers exist on the Internet using more powerful devices than mobile phones (essentially what existing p2p systems require in most cases). We should consider the Internet connection in these cases to be fast and stable, and the devices themselves should be fairly powerful and highly available. The devices should also have the ability to control firewalls and open ports. Currently, these devices would likely be run by RightMesh itself on an interim basis and exist on AWS, Azure, etc. They will also be able to be run eventually by Community Members with systems that match the performance requirements to operate such a node. In the the longer run, these devices will also be located on smartphones with the same characteristics.

The next layer down is the small, blue square devices: the Internet sharing (SUPPLY) devices. These devices are connected directly to the Internet - although they don't have to be all the time. This Internet connectivity decision is left up to the users of these devices. Though the decision may depend on what network they are connected to; for instance, some users may not wish to be Internet sharers on the cellular network, but they will happily do so using their home Wi-Fi network. These Internet sharing devices can have any speed of Internet connection, and the capabilities of the devices may be much lower. They do not need to run a full Ethereum node. Preferably, these devices are not moving too much since many other devices may depend on them for access. However, if other nodes do depend on them for access, a moving device is still better than nothing. The RightMesh library will be able to select which path to take eventually based on how mobile it is, while prioritizing those that will result in the best performance for the lowest cost for everyone.

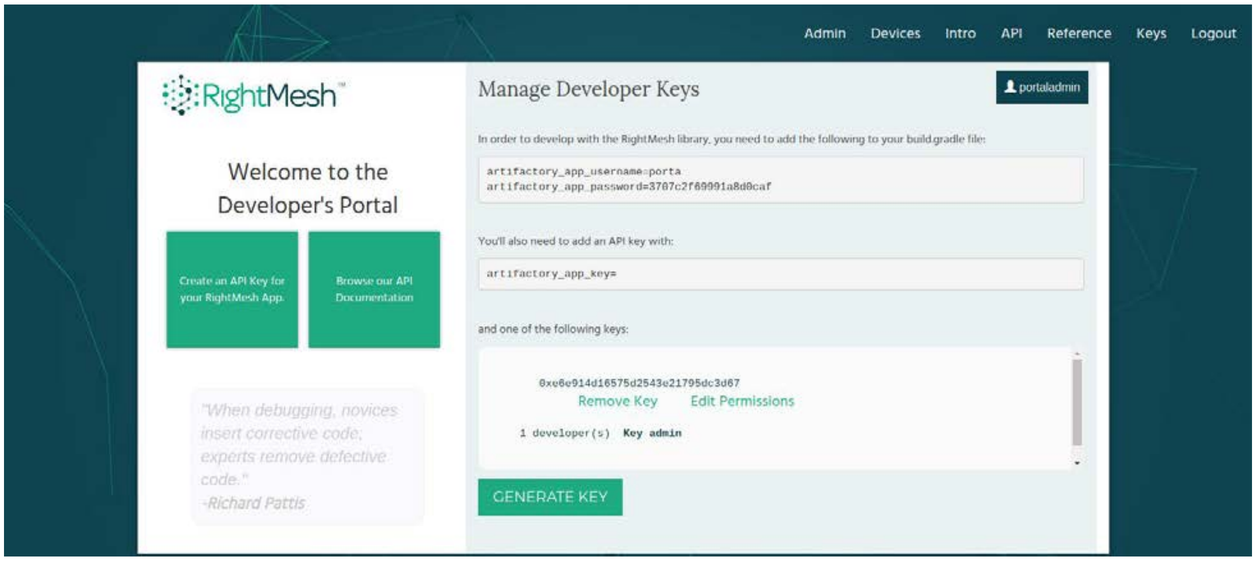
The next layer is the orange squares. These squares indicate devices willing to provide infrastructure. These nodes may also be consumers (DEMAND). The infrastructure nodes do not have an Internet connection themselves, so if they wish to use Internet data that isn't being forwarded on behalf of another, they will also need to pay an Internet seller in RightMesh tokens.

There is also a special type of infrastructure device, which is the orange device in *Figure 2* that showed interfacing with remote sensors. This device is a mobile piece of infrastructure where it is providing forwarding as a result of its mobility. These devices may have an opportunity to earn extra incentive due to the effort of physically moving to provide the data delivery. This could be used to offset costs to travel to the remote location such as gas, or just time and effort for a hiker.

The final layer is the light grey devices. These devices are those which do not wish to provide infrastructure, and they solely want to consume (DEMAND) data on the network.

Mobile / Android Architecture

On Android, the RightMesh library functions as a service. Multiple apps can make use of the service at the same time, provided they are all listening on different mesh ports. The RightMesh Developer's Portal (shown in *Figure 3*) allows developers to register keys which are associated with each mesh app they build. When registering the developer key, they also pick which mesh port(s) the app will be listening on and verify that no other apps are using the same key. The developer who created the key may share access with other developers, so teams can collaborate on the same app without sharing login information. If one of the developers leaves the project, their access to generate valid keys may be removed.



Welcome to the Developer's Portal

Manage Developer Keys

In order to develop with the RightMesh library, you need to add the following to your build.gradle file:

```
artifactory_app_username=porta
artifactory_app_password=3707c2f69991a8d0caf
```

You'll also need to add an API key with:

```
artifactory_app_key=
```

and one of the following keys:

```
0xe6e914d16575d2543e21795dc3d67
1 developer(s)  Key admin
```

"When debugging, novices insert corrective code; experts remove defective code."
-Richard Pattis

Figure 3: RightMesh Developer's Portal

When the app connects to the MeshService, the service parses a cryptographically-signed license key that is generated at compile time and packaged with the app by our RightMesh Gradle plugin. (An example Gradle file for compiling with RightMesh can be found on our "HelloMesh" project on GitHub. The signed license key file requires a valid developer account with access to the key at compile time. Inside this license is the developer key, and mesh ports that the app is allowed to use. Again, similar to the MeshID problem, we tried to design this system so that apps may form meshes without requiring Internet access to validate the library (although we make the assumption that Internet access is available at compile time for the developers).

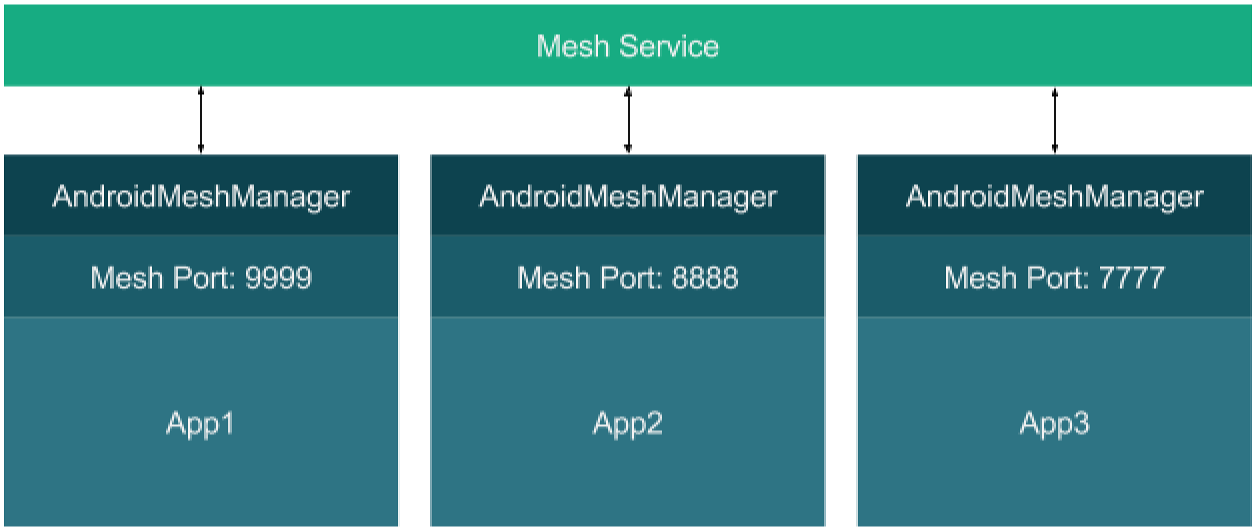


Figure 4: Multiple Apps running on the same device, using a common service

Figure 4 shows the RightMesh Service running on a single phone, with three different apps running at the same time. They are all running on different mesh ports. Each app also comes with a portion of our library we call the AndroidMeshManager, which performs all of the logic to connect to the service, and to fire events such as Peers joining or leaving the network, data arriving, or the result of encryption keys being exchanged.

In Figure 5, a mesh with three devices is shown. Notice that device one has three apps running. Apps are only notified of peers which are running the same app (the developer does not get access to all of the devices on the mesh, even though other devices may act as forwarding nodes in order to reach other devices). For instance, App #1, listening on mesh port 9999 will only "see" peers 1 and 3. App #2, will only "see" peers 1 and 2. App #3 will "see peers 1 and 3 as well. Data being sent from peer 1 to peer 3 will be sent through peer 2, even though it isn't running either of the apps that peer 1 and 3 are using.

An example app which can do basic user discover, send data, and receive data is available on our "HelloMesh" GitHub project.

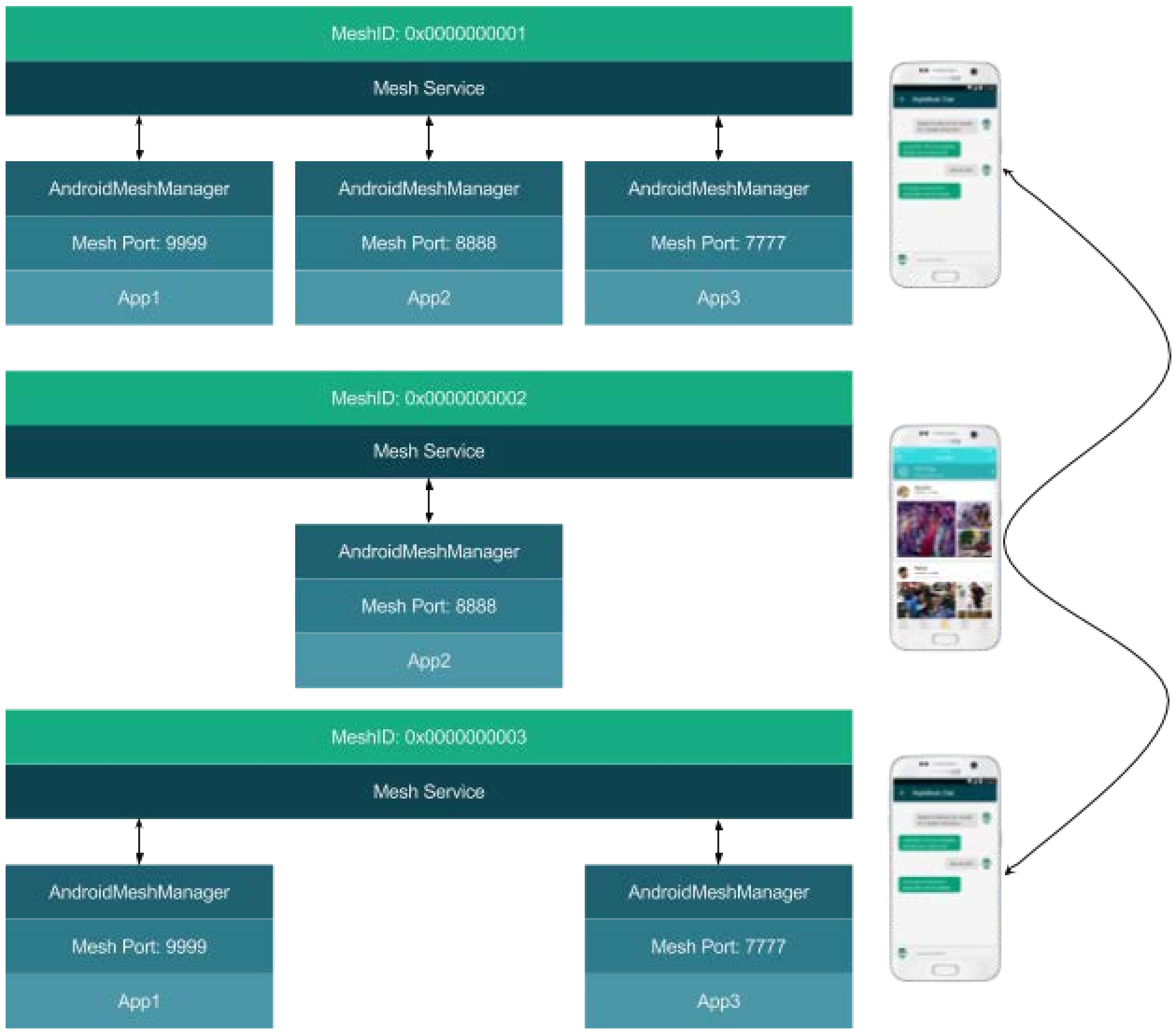


Figure 5: Multiple devices communicate through the service, where each phone may have different apps

Java Architecture

The Java architecture is less developed at this moment, and does not provide a service architecture. As such, only a single mesh app can run on the Java device currently (although turning it into a service is in the works and is on the roadmap). The Java architecture is mostly used to implement the superpeers right now, although it could also be used on IoT devices, routers, and other less mobile equipment.

We have also done proof-of-concepts at hackathons where the Java version acted as an Internet sharing node. It is also quite easy for it to be an infrastructure-less node. Performing tasks—such as linking together Wi-Fi, Wi-Fi Direct, and Bluetooth networks—is somewhat platform dependent and is still in progress at present. The roadmap aims to support Linux and Windows in the near term.

We also have some proof-of-concepts for part of the token-superpeer. We currently do not have it joined with the crypto contract portion, but we do support discovery of devices from multiple geographically separate meshes functioning. With a little more work we can also have data being relayed between these meshes. We also support visualizing the connectivity of the global mesh with the token-superpeer currently (Note: in the link, it shows up to 100 Internet sharing nodes connected to an early version of our superpeer. There is also a video that shows phones coming online in our visualization tool). This will help as the network grows to debug connectivity issues.

RightMesh Networking Stack

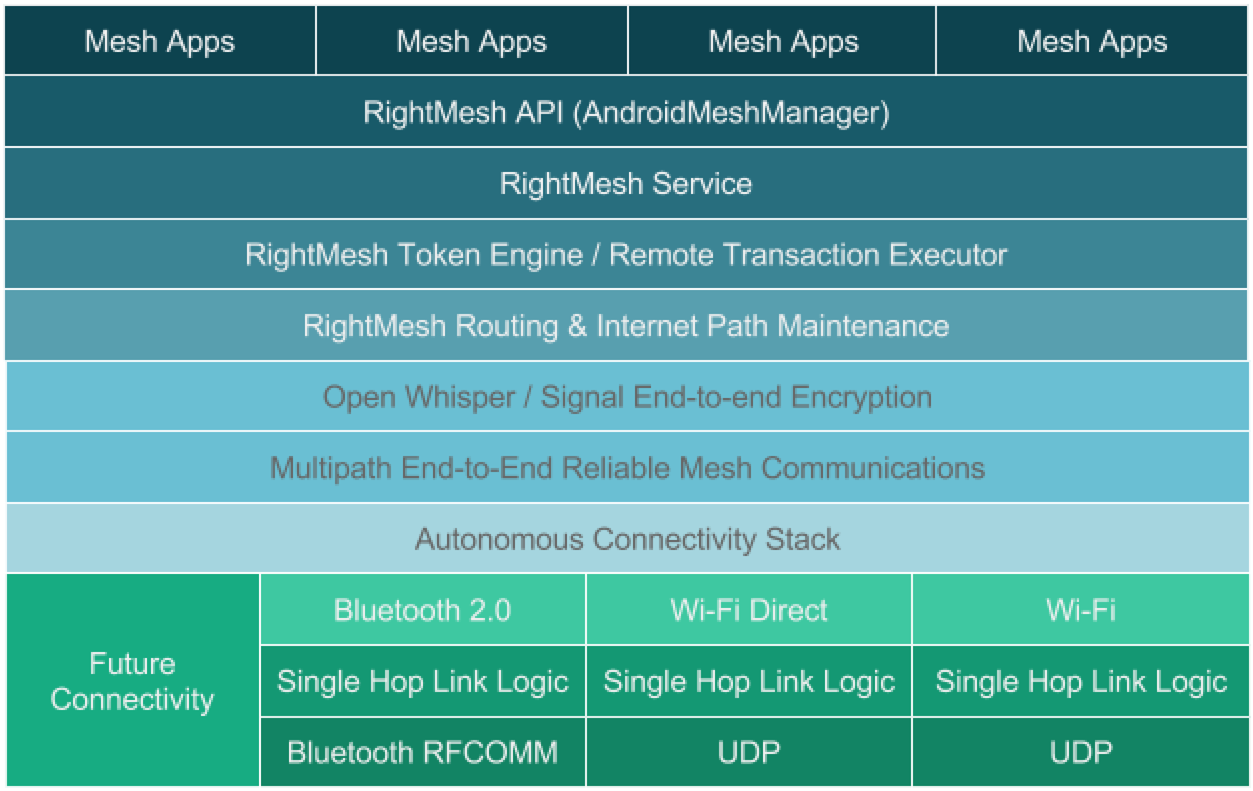

Mesh Apps
Mesh Apps
Mesh Apps
Mesh Apps
RightMesh API (AndroidMeshManager)
RightMesh Service
RightMesh Token Engine / Remote Transaction Executor
RightMesh Routing & Internet Path Maintenance
Open Whisper / Signal End-to-end Encryption
Multipath End-to-End Reliable Mesh Communications
Autonomous Connectivity Stack
Future Connectivity
Bluetooth 2.0
Wi-Fi Direct
Wi-Fi
Single Hop Link Logic
Single Hop Link Logic
Single Hop Link Logic
Bluetooth RFCOMM
UDP
UDP

Figure 6: High-level system overview

Layered Architecture Similar to the Internet

The RightMesh networking stack shown in *Figure 6*, builds off of the success of existing Internet protocols. The design is a layered design, so improvements can be made iteratively on each of the layers without having to do a full redesign of the entire system. Everywhere in RightMesh, next-hop connectivity is formed without user intervention. This is a core guiding principle of RightMesh: the user should not have to approve every single device connection (as one has to do with Bluetooth pairing, for example). The only way a widespread mobile mesh network will scale from a user experience point of view is if the connectivity can be made without user authorization of every connection. If the user is required to approve every single connection to every other device in the network (how some similar libraries work), it would be difficult to get meshes that scale beyond a few devices locally - and most people will be inclined to only form meshes with people they directly know.

One-hop Connectivity

Bluetooth 2.0 was used as a starting point because it was possible to make a connection with other peers without requiring pairing. With our Wi-Fi Direct and Wi-Fi implementations, the same is also possible. RightMesh makes connections programmatically using three different types of next-hop links. Within our design, RightMesh can add support for higher versions of Bluetooth as we figure out ways to make the connectivity occur without user intervention. We may be able to do this by making use of some of the other connectivity pathways we already have as a way to signal information between devices to set up the faster connections.

Everything below the autonomous connectivity stack can be thought of as similar to a layer two protocol in a normal networking stack (think of MAC addresses being mapped to IP addresses). Instead of that, our protocol maps IPv4, IPv6, MAC, and other addresses to a MeshID. This layer is responsible for exchanging local peer information. We make use of a clustered routing where hierarchy is imposed on the devices based on the internal roles the devices are assigned. This means that not every device needs to know the connectivity to every other device, allowing the RightMesh network to scale significantly higher than competing approaches. This same cluster based architecture lends itself well to enabling caching in the network which is also on our roadmap. For instance, the same nodes that are cluster heads would be ideal candidates to also be a cache of common apps, ads, and multimedia content.

Autonomous Connectivity

The autonomous connectivity layer does the job of assigning internal roles to devices in our network. This decides whether a device will be in hotspot mode or not, which devices to connect via Bluetooth, whether to use Wi-Fi, Wi-Fi Direct, both, or all three. The developer does not need to worry about how the autonomous connectivity works. For the developer as far as they know, there is a list of MeshIDs that are either connected, or not, running the same app.

The RightMesh approach to autonomous connectivity is something that will constantly get better, but it was kept simple as a sort of MVP state. In fully-automated mode, the user will only need to select whether they wish to participate as a forwarding node, and whether they wish to share their Internet, or not. If they wish to sell their Internet, they will also need to set the price they wish to sell it for (in Mesh Tokens). Conversely, if the user wishes to consume Internet, they must set the price they are willing to purchase Internet for (in Mesh tokens). If they do not wish to pay any mesh tokens and there are no free paths, the apps will still function, but they will only be able to use a localized version of the mesh.

It is also possible, in advanced mode, for users to control exactly which internal role the device should use. This capability is packaged automatically with every app that uses the RightMesh library (see *End-Users* section).

Multipath, End-to-End, Reliable Mesh Communications

Combined with the routing layer and the one-hop connectivity layer, multipath end-to-end reliable communication forms a really unique and valuable part of the RightMesh solution. This is what distinguishes RightMesh from most of its competition. There are a few ways communication typically occurs in mobile meshes:

1) Broadcast to everyone (either using UDP + existing broadcast / multicast approaches, or TCP on a link by link basis - no end-to-end capability
2) Single-path end-to-end TCP (with many limitations as outlined below)

3) Best-effort UDP communications with no reliability

Many existing mesh approaches simply make use of existing TCP to provide end-to-end connectivity. However, TCP is not well suited to a mobile mesh network for several reasons:

1. Unless rooted, mobile phones do not let the user customize the version of TCP being used.
2. Mobile mesh networks are made of devices that frequently connect and disconnect. When this occurs, a TCP connection would require end-to-end reestablishment of a connection, with something like a 3-way handshake.
3. With IPv4 It is often impossible to make an end-to-end TCP connection because the IP address in one hotspot means nothing to the neighbours three hotspots away. As such, there is no mechanism to exchange this information between hotspots and have the routing automatically work. In this case, it may be possible to have TCP on each single-hop link, but there are still the same drawbacks from some of the other bullet points to be aware of. In the case of performing TCP on each link, there is still no mechanism to prevent an entire end-to-end path from being saturated with traffic (the congestion control will only work on a link-by-link basis, leaving the queues at the intermediary nodes to potentially overload).
4. TCP often gets interference and congestion mixed up. It may start a backoff unnecessarily thinking that congestion is occurring when it is temporarily poor network conditions due to external factors (e.g., a vehicle driving through the path of the signal, a microwave being turned on, a tree being between two people who are moving, etc.).
5. The TCP on most non-rooted phones cannot handle multiple paths. This means even if your device has a Bluetooth, Wi-Fi, and Wi-Fi Direct connection available, it is only able to use a single one of them for one data stream. This also applies to Internet connections out of the mesh as well. There are existing versions of TCP that can do this, notably mTCP; however, it is almost never included in commercial phones.

RightMesh borrows concepts from delay-tolerant network protocols such as LTP and has combined them with multipath-TCP to create a delay-tolerant protocol when the network becomes fragmented, but high performance in the cases where it is well connected. RightMesh also uses concepts from Software Defined Networks, Overlay networks and self-* networks.

Open Whisper / Signal End-to-end Encryption

Rightmesh supports end-to-end encryption using the Open Whisper/Signal library (whispersystems.org). This library has been modified so that it no longer involves the server portion, since that would require Internet access. RightMesh offer two levels of security: one where the key is directly exchanged in one hop (this is the more secure option). The second, where the key exchange occurs through the mesh over several hops, is less secure because it may be subject to a man-in-the-middle attack.

RightMesh does not store any keys in any server, so any key exchange that occurs securely means only the recipients can decrypt the data. For the company, there is no way RightMesh can be compelled to give up the keys because none are stored. We are working on ways to improve this process (e.g., like sending the key split up across multiple paths so that any attacker would need to compromise many devices at the same time). The company is also working on ways to improve the user-friendliness of a secure key exchange such as with a 2-D barcode or NFC. Compared to other mesh platforms which broadcast to every device, RightMesh only forwards directly on a routing path. As as result, fewer devices have data flowing through them, making it much harder to attack.

RightMesh Routing & Internet Path Maintenance

As part of our peer discovery protocol, information regarding possible paths to the Internet are provided to devices in a local mesh. Currently, the library supports a single path to the Internet, but because of our multipath transmission support, it will eventually be possible to also support multiple paths to and from the Internet simultaneously.

RightMesh Token Engine / Remote Transaction Executor

Since devices on RightMesh have an identification based on an Ethereum account (including the private and public key), using the Ethereum-j library, it is possible to generate signed transactions from the account and pass them through RightMesh to the token superpeer. The token superpeer runs a full Ethereum node. When the superpeer recognizes that a remotely signed transaction is being passed to it, the superpeer executes the transaction on its local geth instance and waits for the confirmation. It then passes the result back to the device in the network which executed the transaction. This means that any device within a RightMesh network is able to send Ether to another device, query its own Ether balance and execute contracts on the Ethereum network from potentially many hops away from the Internet. At best, as far as we know, thin clients exist on mobile phones which can perform similar capabilities when the device is connected to the Internet directly (similar to how the blue Internet sharing devices are). However, nothing exists that allows for cryptocurrency transactions from outside of the range of traditional Internet access.

Furthermore, the token superpeer is able to execute token transactions on behalf of clients. More information about this is found in the section: *Token Integration & Token Superpeer Architecture.*

Token Integration & Token Superpeer Architecture

Managing limited resources is a known hurdle for RightMesh, whether in regards to battery life, network capacity, or memory and storage limits. Because of these constraints, it is currently not feasible to run a full crypto node on the mobile phone itself. To allow for Ethereum transactions to occur, the nodes would sign a transaction locally and relay it in towards full Ethereum nodes (i.e., Superpeers), running at the edges of the mesh. Some of these nodes would be run by RightMesh and some by community partners. This model may evolve into an industry on top of our ecosystem since these nodes can earn tokens by performing a mining-like function to keep the network operating. At launch, these Superpeers would be run on computers or on existing cloud providers spread geographically around the world. In the future, as mobile devices improve, we believe it will be possible to eliminate this centralized aspect, as much of this functionality may move directly into the Internet Sharing Devices themselves. At present, the biggest limitation is hardware, as it is not feasible for phones to act as full Ethereum nodes. The Superpeer functions as a proxy, a hole-punching aid, a translator between the IP-based Internet and our mesh protocols, and an Ethereum full-node.

All the code that makes use of Ethereum technology will be open sourced, along with any contracts, so that the community can verify and improve upon the stability of the platform and to encourage the users to trust that things are operating correctly. RightMesh will also be open sourcing some of its own created apps to demonstrate possible implementations of the Mesh SDK: https://github.com/RightMesh. The company will explore open sourcing its core mesh technology once the network becomes hardened and the network established. From a developer's perspective, they simply need to define the transaction they wish to make within their RightMesh application, and issue a call to our library. RightMesh will perform the transaction signing and optimize how to route the packet out of the mesh and into the Ethereum network. The full Ethereum node at the edge of the mesh is also running a Java version of our library. It uses the local remote procedure call (RPC) mechanisms to relay the transaction to a locally running Geth client. The Geth client will verify the transaction and execute it. The result of the verification and execution will then be relayed back to the mesh node which originated the transaction.

Using a similar approach, RightMesh could adapt to support other cryptocurrencies as well, such as Bitcoin.

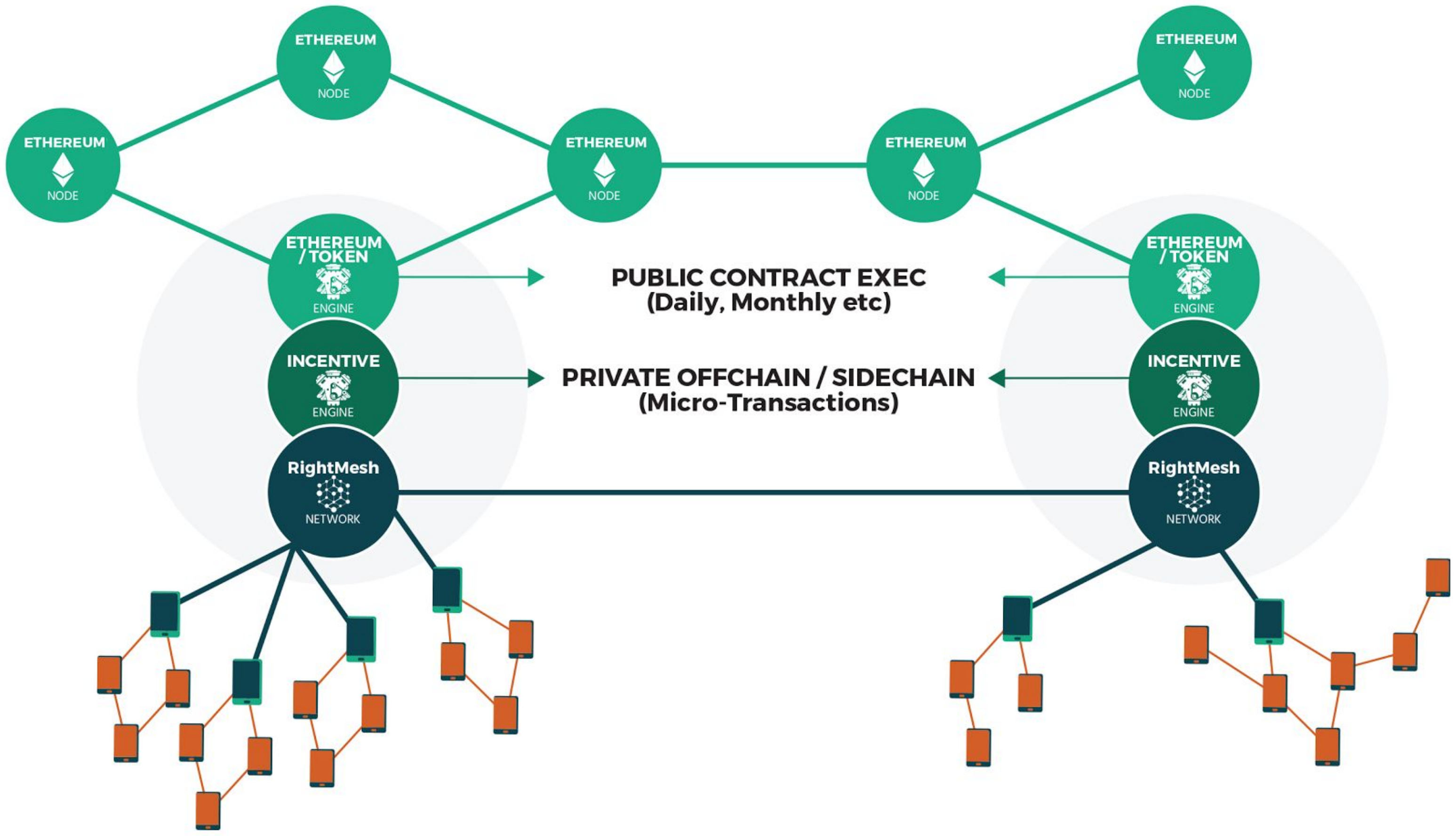


Figure 7: Multiple "token superpeer" architecture to support incentivization of Internet sharing devices

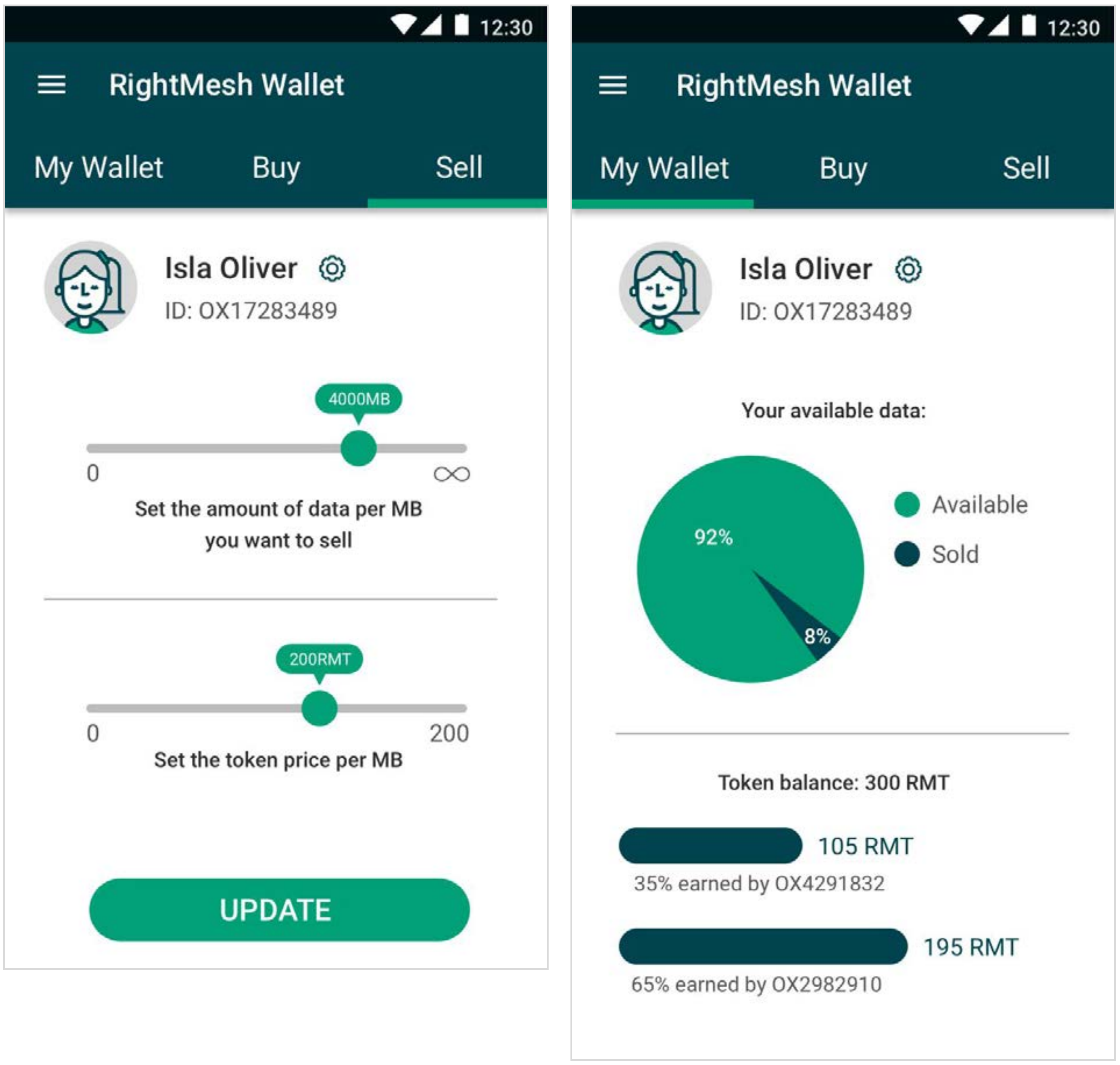
Figure 7 shows a RightMesh network with two participating token superpeers. In the full, high-level diagram at the start of the document, there also may be app superpeers, but these are left out of this figure for simplicity. In this figure, the left superpeer has two geographically separate meshes attached to it while the right superpeer has two different geographically separate meshes attached to it. The superpeers may be deployed all around the world, similar to how AWS instances are deployed in different places based on geography and demand. The left superpeer may be deployed in Canada, and have a mesh in Vancouver and a mesh in Toronto, while the right superpeer may be deployed in Bangladesh and have a mesh in Dhaka and a mesh in Khulna. The green line represents a fast internet connection between the two superpeers. This is important because in the initial implementation, the data will be related directly between the superpeers to reach the farther meshes. In addition, the superpeers are running full Ethereum nodes. On the Ethereum network, the two token superpeers may also be peers of each other. In addition, there may be many public peers on the Ethereum network.

Device Roles for Tokenization

Recall from the system overview, that some portion of the devices provide SUPPLY to the network: these are the Internet sharing devices. In the above figure, these are the devices connected directly to the token superpeers. Devices which aid in delivering the SUPPLY to the consumers (DEMAND) are called infrastructure nodes. The consumers may be scattered throughout the mesh, and at times they may be both consumers and infrastructure nodes. At launch, incentivization will be possible when a device wants to send traffic from one separate mesh to another through the internet sharing devices via the superpeers.

Buying and Selling Resources, Interaction with RightMesh

A device user requesting the resources, whether a client device or service provider, sets how much they are willing to pay to consume a certain number of bytes of traffic. This is the DEMAND side of the marketplace. For instance, people around the world are fairly familiar with the concept of a data consumption rate whereby a user may pay for their data consumption to their ISP or mobile carrier, typically priced per MB of data. This price is set via free market with a minimum floor, where the floor includes the base commission for RightMesh services and for Superpeers who execute the RightMesh token transaction.


12:30

RightMesh Wallet

My Wallet Buy Sell

Isla Oliver

ID: OX17283489

4000MB

0 ∞

Set the amount of data per MB you want to sell

200RMT

0 200

Set the token price per MB

UPDATE

12:30

RightMesh Wallet

My Wallet Buy Sell

Isla Oliver

ID: OX17283489

Your available data:

92%

8%

● Available

● Sold

Token balance: 300 RMT

105 RMT

35% earned by OX4291832

195 RMT

65% earned by OX2982910

Figure 8: Example of Resource Accounting Built into RightMesh

Similarly, the device users—which wish to participate as an Internet sharing device, as a caching device, or mobile delivery device—are able to set their minimum selling price for their resources. To start, it is likely that only buying and selling data within RightMesh will be supported; however, the roadmap also includes possibility to incentivize app and ad distribution; storage on the phone (similar to, or potentially in partnership with Filecoin or like services); usage of the processing resources (similar to, or potentially in partnership with Golem or like services); and enabling token transfers between devices on the mesh. This is why the word *resources* is used instead of *data*.

The RightMesh library would interact with the incentivization model such that it would always be searching for paths to the Internet to minimize cost and maximize performance. Currently, RightMesh selects routes based on the least hop count path. At launch, it would be updated such that there is equal weight between the fewest hops and lowest cost. In time, this algorithm would become increasingly complex so that performance of the mesh may be fine tuned for individual users where a best path uses a combination of speed, cost, performance, and even reputation of participating nodes. There could also be separate route selection depending on the type of traffic, for example. There also could be different costs associated with different types of traffic (e.g., if the user was willing to pay extra to download video, but wanted to use best effort for email).

Figure 9 shows the initial state of a small mesh where some of the users have some account balances purchased on the token generating event, and others have nothing yet (and wish to earn by sharing data).

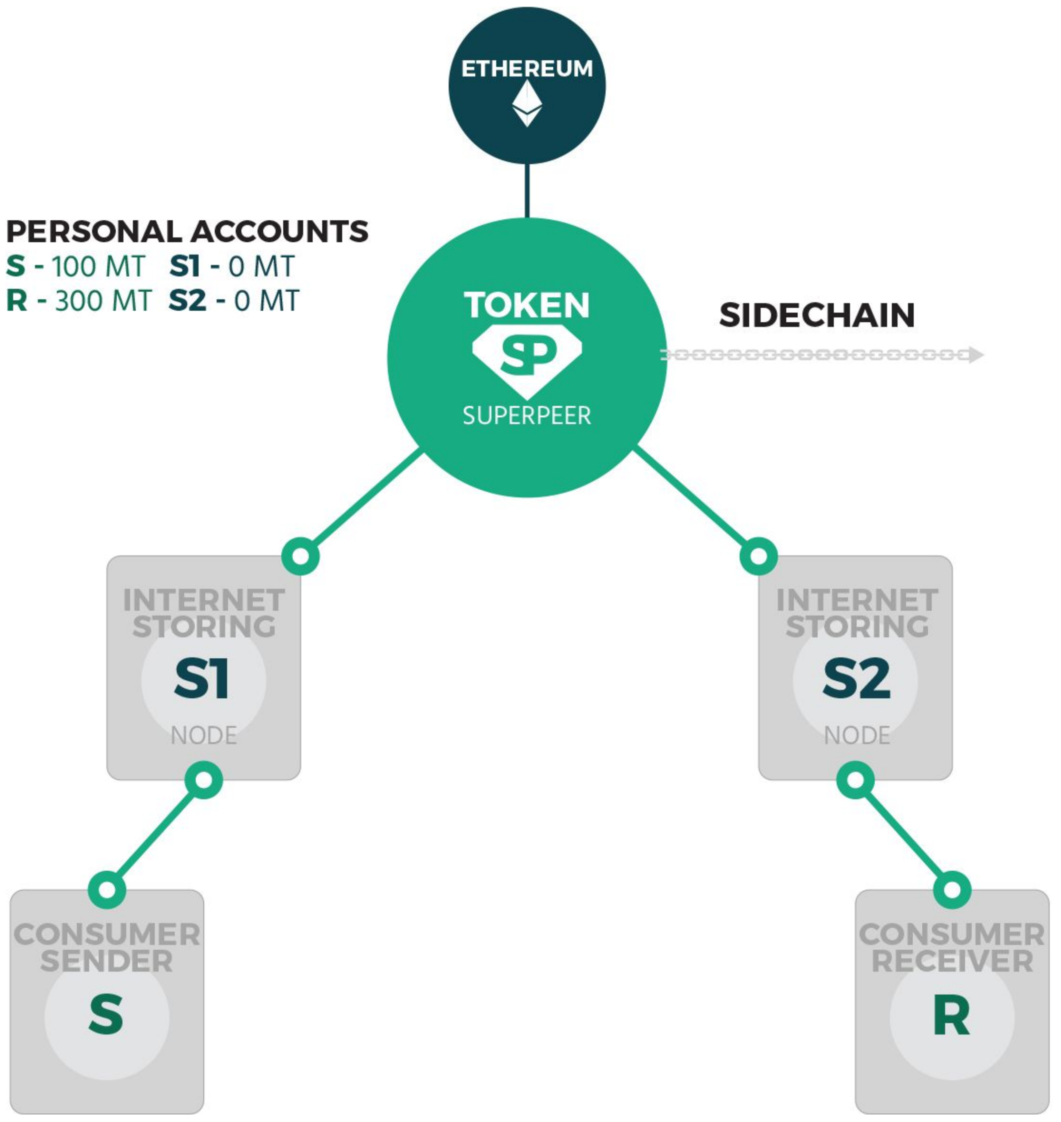


Figure 9: Token Superpeer + Sidechain example

In *Figure 10,* the Internet sharing devices must set how much data they are willing to sell during the billing cycle and at what rate. For example, suppose a S1 wishes to sell 300MB of data for 3 Mesh Tokens. This will trigger the RightMesh library to interact with a contract where 300MB of data is added to the selling pool contract at the price of 1MT per 100MB. The data seller covers the cost of gas in order to add data to the selling pool. This contract keeps track of how much data is available from whom and at what cost. This is executed by relaying a signed remote Ethereum transaction from the selling device into the superpeer.

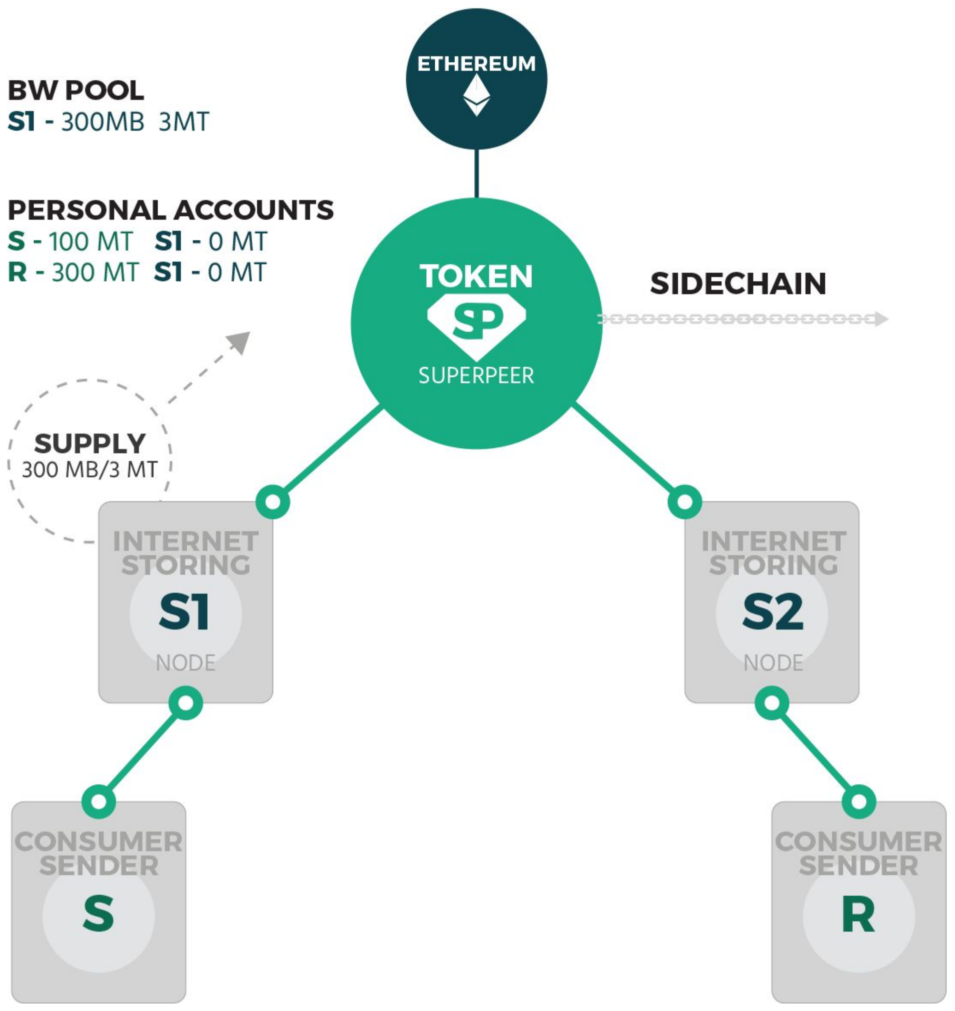


Figure 10: S1 Supplying data at a set rate into the network

Similarly, in *Figure 11*, S2 wishes to sell 500MB of data for 5MT and the BW pool contract is updated accordingly.

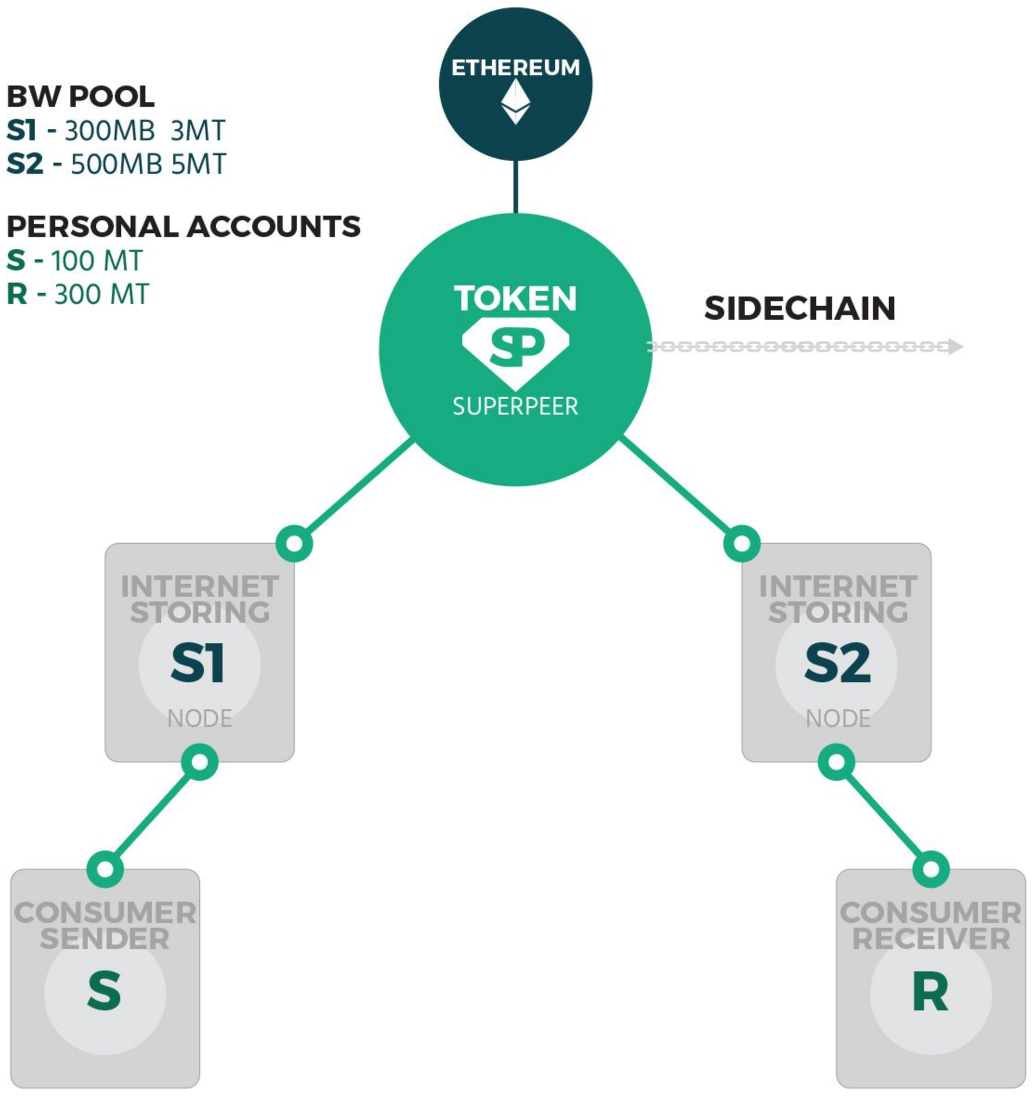


Figure 11: After S1 and S2 have both reserved data to sell

In order to illustrate how the token model might work, the following sections explain using examples as follows. Note that while all of these examples show a single token superpeer, the idea is that many of these token superpeers can be connected together as the figure at the start of the section suggests. For instance, if we find that the token superpeers are too slow at processing the local chain, more could be spun up on demand to help with this aspect. Similarly, since these devices are also responsible for forwarding data, they may become overloaded in terms of throughput, in which case more will need to be added. These are connected both to each other, and to the individual meshes which connect to them.

If the consumer users do not set a price they are willing to pay, they will be able to "see" other users outside of their local mesh, but will need to pay tokens to actually communicate with them (or alternatively, join a network which provides Internet access, such as a free Wi-Fi cafe). When a price is set, this moves the reserved amount of Mesh Tokens from the consumer account into the buyer pool contract. Again, this is executed by relaying a signed remote Ethereum transaction through the mesh, through the Internet sharing device and into the token superpeer where it is executed on the public Ethereum network. As the tokens are passed into the smart contract for the demand pool, an entry is added into the private sidechain with the MeshID and the amount available. *Figure 12* shows consumer devices S and R setting the rates they are willing to pay.

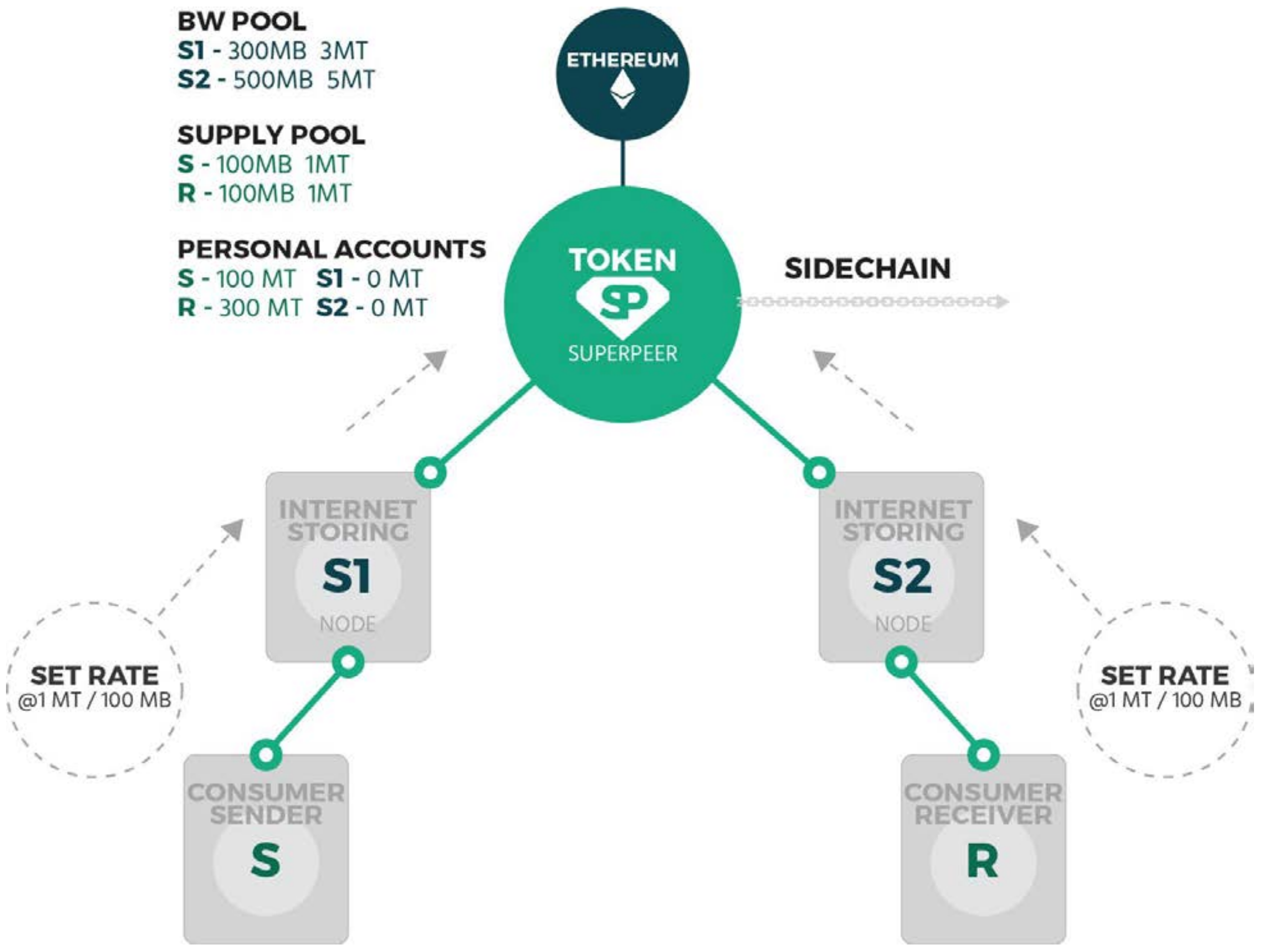


Figure 12:Consumers setting their consumption rate.

The effect of this is to take the tokens out of the user's public wallet and into the contract. The only way to get the tokens out of the contract is by the superpeer which will perform a payout function at the end of every billing period.

Case 1: Both the Send and Receiver Side have enough tokens

Once the consumer sets a price and a maximum limit of how much they wish to consume, the RightMesh library internally knows which paths fall within this limit and will select the lowest cost paths with the best performance to be used. Upon issuing a "send" call, the RightMesh library knows how much data to reserve before allowing it to pass. The RightMesh library on the sending device first performs a check to ensure it can afford the transaction based upon its records. It initiates the start of the send and when it reaches the Internet sharing device (seller), the seller first verifies that it does have enough remaining balance to forward the transaction into the token superpeer. The token superpeer verifies there is enough tokens and bandwidth supply remaining by parsing the sidechain, and if it agrees, it marks the amount to be used in the side chain and sends a confirmation back to the Internet sharing node. The internet sharing node starts forwarding the traffic into the token superpeer using its Internet connection.

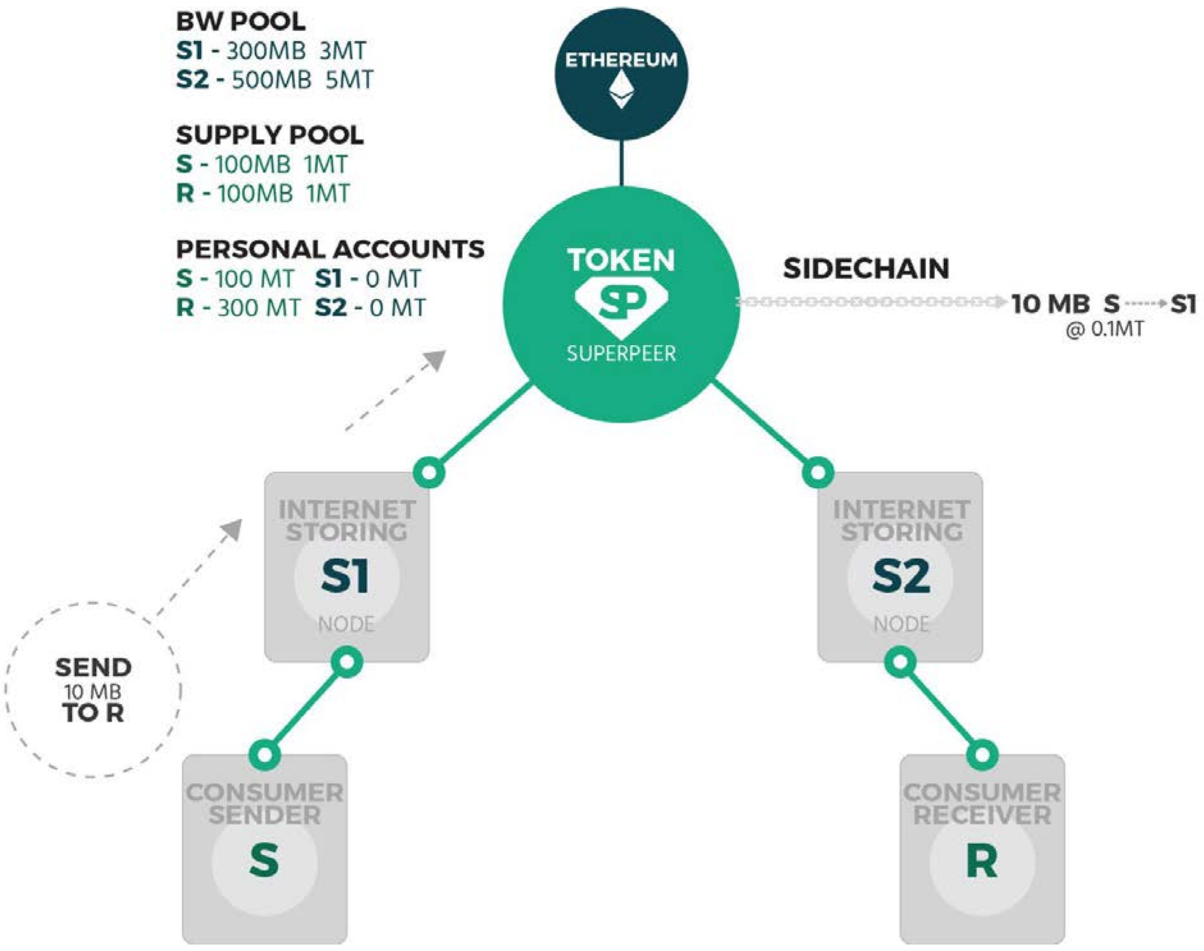


Figure 13: S starting to send 10MB of data to R

Figure 13 shows two geographically separate "meshes", each with two devices. Each mesh has a single internet sharing node, and each has a consumer. Let us assume the sender and receiver (S and R) both already have purchased MeshTokens from a token sale, were given them by RightMesh Foundation, or an app developer. Let us also assume that the two sellers (S1 and S2) have no tokens yet. The first step would be for the sellers to set their price and specify how much bandwidth they would like to sell. This functionality would be packaged into RightMesh similar to how the activity is packaged that allows users to configure their role in the network. From a developer's perspective, they would just show a button on their app that lets the user configure their data usage. When RightMesh is running, there is a tray icon in the notification area on Android, and if a user pulled this down, they would also have access to it.

Bandwidth Pool Smart Contract on Public Ethereum Network

When the user sets their selling price, this triggers a signed remote transaction to be generated by the selling device. This transaction would be relayed by the superpeer into the Ethereum network. The signed transaction would execute a function on the bandwidth pool smart contract which sets a reserve of bandwidth available from the selling device, at the rate set by the seller. Then, since it is free (in terms of gas) to query a value in a contract, the superpeer could later determine if there is bandwidth available by querying the contract. There are a few drawbacks to this contract being located on the public Ethereum network. First, it would cost gas for the selling devices to add their data into the data pool. Second, any further updates to the pool after people consume data would also cost gas. Third, even if the selling devices were willing to pay this, currently with Ethereum, it would require the sellers to have Ether in their accounts (because gas cannot be paid in app tokens yet).

It may also be possible at this stage for RightMesh to charge a fee on this action. This would require feedback from the community as to whether people selling their data should pay at the moment when they wish to make the data available, or should they only capture associated fees when the money is paid out at the end.

Bandwidth Pool Smart Contract on the Private Sidechain

To solve this, we could move the bandwidth pool contract into the sidechain (which we can consider for now to be a private Ethereum network that only the token superpeers participate in together and jointly mine on). In this case, since it is a private network, when we make the genesis for it, we can assign all of the superpeers to have virtually infinite resources and have them pay the gas to execute the transaction (or transfer the correct amount of Ether first to the remote account within the private chain and then have the remote transaction execute afterwards). The effect of this is we now have a record of how much bandwidth is available and at what rate, for all of the sellers in the network. The token superpeer can keep track of this information and use it to decide whether to accept incoming data or forward it between meshes.

Pre-purchase Contract on Public Ethereum Network

The next step is that the consumers/buyers also need to set their price they are willing to pay and how much data they would like to “pre-purchase”. Again this is done with a remote signed transaction where the amount of tokens is moved from the personal account to the smart contract which keeps track of how much data is being purchased and at what rate. In this case, since we can assume that it is required to have tokens anyway to purchase data, the user can afford to pay the gas fee to move the tokens from their own account into the contract. There is still a problem that in the short term it will still require Ether, but we are confident the update will be released soon to address this. Moving the tokens to the smart contract has the effect that the tokens cannot be spent more than once. It also means that until the user either requests the balance back or the billing period ends, the funds are locked within our sidechain (the holding contract).

Pre-purchase Contract on Private Sidechain

To prevent high gas costs on small data transactions (i.e., if RightMesh had to update the contract every time a small data message was sent), we propose to move all individual transactions into the “sidechain” as well. In this case, the public contract only serves to lock in the unspent tokens until after the “billing period”. The billing period could be defined for a fixed interval (e.g., once per day, or once per month… whatever makes sense technically and economically). It could also just be for the time the user wishes to keep their money locked in place. While we may be able to support withdrawal at any time, since there is a fee for us in gas to do so, this would need to be structured carefully.

Now that some buyers and sellers have set their prices, it is possible for devices to send data between the meshes. Let us assume the sender S wants to send data to the receiver, R. S sends using the RightMesh library with R’s MeshID as the recipient. The RightMesh library at S is able to determine that there is a path within the price range because this type of discovery is built into the RightMesh discover protocols. It begins forwarding packets toward R through S1. (note: if there were other paths, and there was a cheaper option with comparable expected performance, it would choose the other path. It is also possible to use multiple paths together if more than one path falls within the price range; however, for simplicity we will consider a single path for now).

When the first MeshPacket reaches S1, S1 itself should already be keeping track of how much data it is making available, and if it detects it does not have enough, it will reject the data here. If the forward is rejected, S will send no further packets (note: it is also possible to just send the request separate from the data; however, if the data is sent together, it reduces delays waiting for the approval just to start).

If S1 determines it has enough data remaining to sell, it issues a request to the superpeer to check that this is true and that S has enough data remaining in its purchase allocation. The superpeer verifies by parsing the sidechain (or keeping internal state [sidechain would be used more as a historical record]) that S1 has enough data to sell, and S has remaining data to buy. If both are true, it responds to start the transfer and data is actually used on S1 Internet connection from S. At the moment the superpeer verifies there is enough data available to S and S1, it marks down the intended consumption so that it does not count in the available pool of data (for subsequent requests). When the transmission completes, or if one of the nodes leaves the network (after some timeout period), an adjustment is made for the actual amount consumed. The superpeer at this stage could determine if it was the Internet sharing device or the consumer which left the network and has the option to penalize the devices which are purposefully trying to avoid charges. If the user has closed the RightMesh library cleanly, it sends a “goodbye”, whereas if there are communication problems, there would not be. Consequently, we can try to determine if the leaving is due to network problems or user behaviour.

When the packets arrive at the token superpeer, there are two options. First, the token superpeer performs the same operation as the send, but in reverse for the receiver, R, and immediately passes the data through as it arrives. The second option is the case where the receiver does not have enough tokens nor on a path where there is an internet sharer in its budget. In this case, the data is queued indefinitely (or with some set expiry) at the superpeer until R either finds an affordable path and has tokens in the purchase pool, or R moves to a normal internet connection, at which point it is forwarded for “free” (i.e., using only the cost of the normal Internet from the ISP, which could be free if it is a public Wi-Fi network).

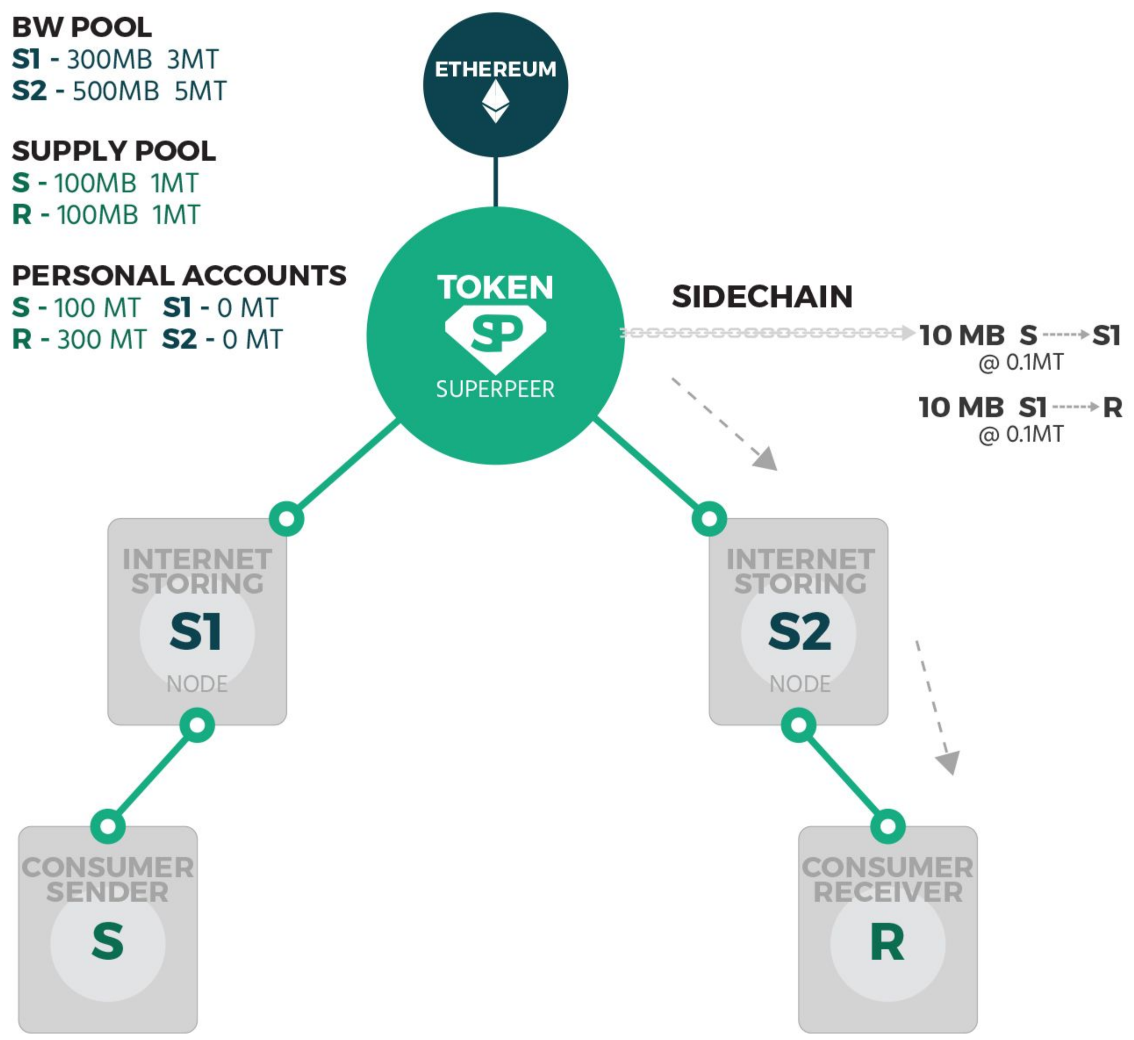


Figure 14: Token superpeer forwarding the second half of the send to the receiver R

On the receiver side, a similar process happens (as shown in *Figure 14*). Before the superpeer relays the data to the receiver, it first determines if there are any Internet sharing devices on a path that the receiver has a budget for. It then determines by parsing the side chain, if there is enough tokens remaining, and, if there is enough bandwidth remaining in the pool for the particular Internet sharing device. If there are enough tokens and bandwidth available, it marks an entry in the sidechain for how many tokens should be consumed and how much data to detect from the pool. In then starts forwarding data over the Internet sharing devices Internet connection towards the intended receiver.

Billing Period Payout

There are two options for the “end of the billing period” (i.e., when a user is withdrawing their tokens from the sidechain). First, at the end of the billing period all accounts that are not empty will have the money transferred from the purchase smart contract and into their accounts. This would be performed by the superpeer parsing through all of the transactions for the billing period, and then transferring money out of the contract and into the accounts. This would be potentially expensive for us since it requires us to pay the gas for this transaction (transactions)? The second option would be that we only process withdrawals when people request it. This could potentially save on gas costs, and leave more “reserve” funds available in the contract until people withdraw. This also makes some sense in reality because people may wish to retain their funds in the reserve rather than having to keep paying the gas to transfer it in if it is paid out every month automatically.

Another option to reduce costs in terms of gas is to set limits similar to how exchanges currently do it. For instance, a user will need to have earned at least <x> tokens before the transfer occurs, withdrawals may only happen periodically, etc.

Further, to prevent the operator of the superpeer from losing money on a transaction, the operator can set a “transfer fee” that would cover the cost of the gas.

Case 2: Sender out of data

In this case, the sender tries to send a new data transfer to R; however, they have already consumed all of the data they have available. The data arrives at S1, and let us assume S1 still has some available connectivity to sell. It will then check the token superpeer, which will process the sidechain, or report its internal record keeping, and reply back that the transaction should be rejected. If this occurs, the packets will be dropped. Since S should be keeping track internally of its data limit, this should never be a case that occurs. However, should this be detected, it could be an indication that a node is malfunctioning. After <x> attempts to send data when there is no remaining data left in a pool, the rest of the nodes in the network could be made aware that a misbehaving node exists on the network, and they would disconnect from it.

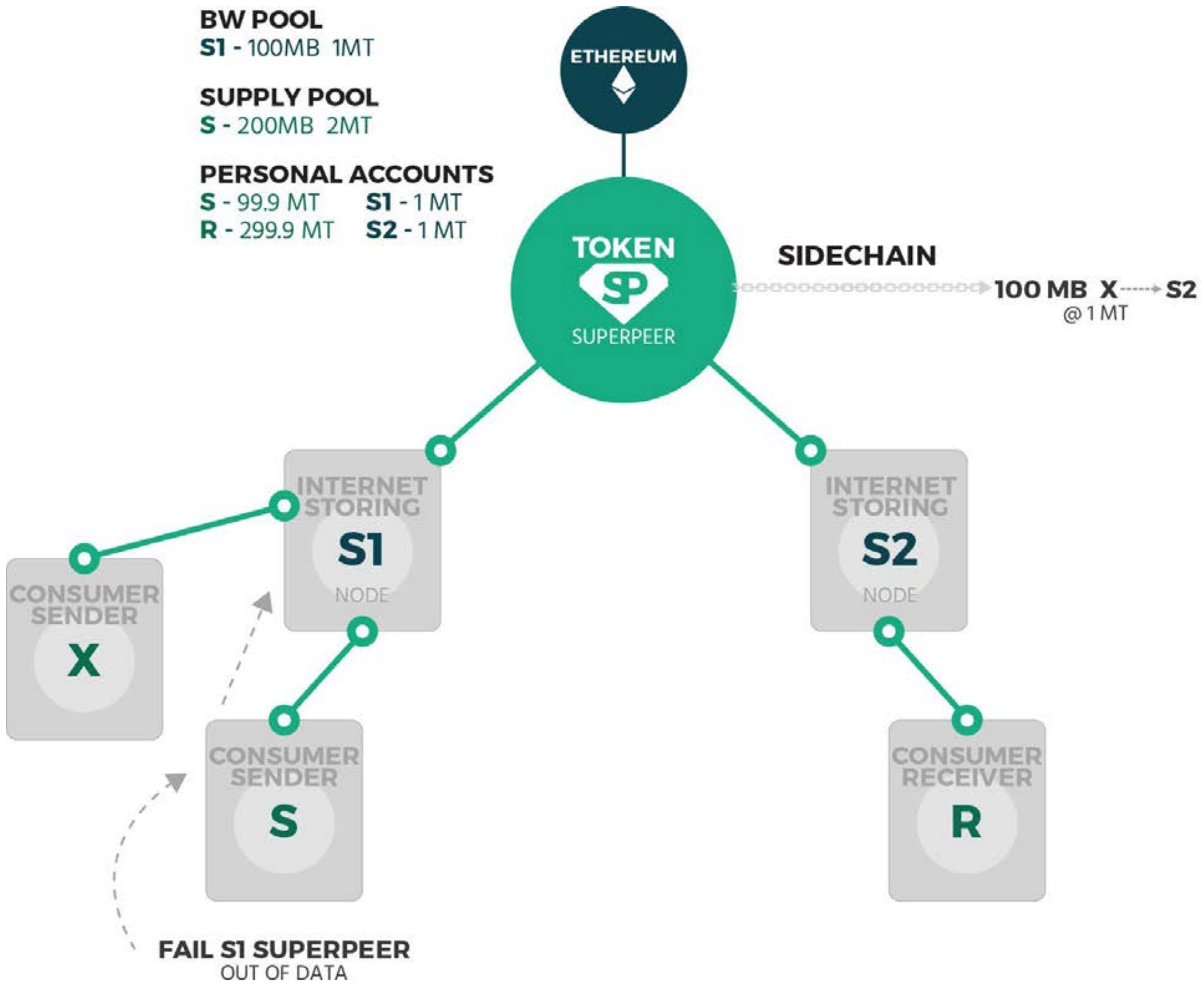


Figure 15: Failure case detected at S1 when S tries to send because all data has already been consumed by X

Case 3: Bandwidth pool for S1 already consumed or Receiver out of data

In this case, S has already consumed all of the data that S1 was willing to sell, but R would like to send data to X which is on the same mesh as S. The data will arrive at the token superpeer successfully if R has enough purchased data; however, it will be held at the superpeer until a new Internet sharing node arrives with data to sell, S1 increases the amount to sell, or X moves to a normal Internet connection.

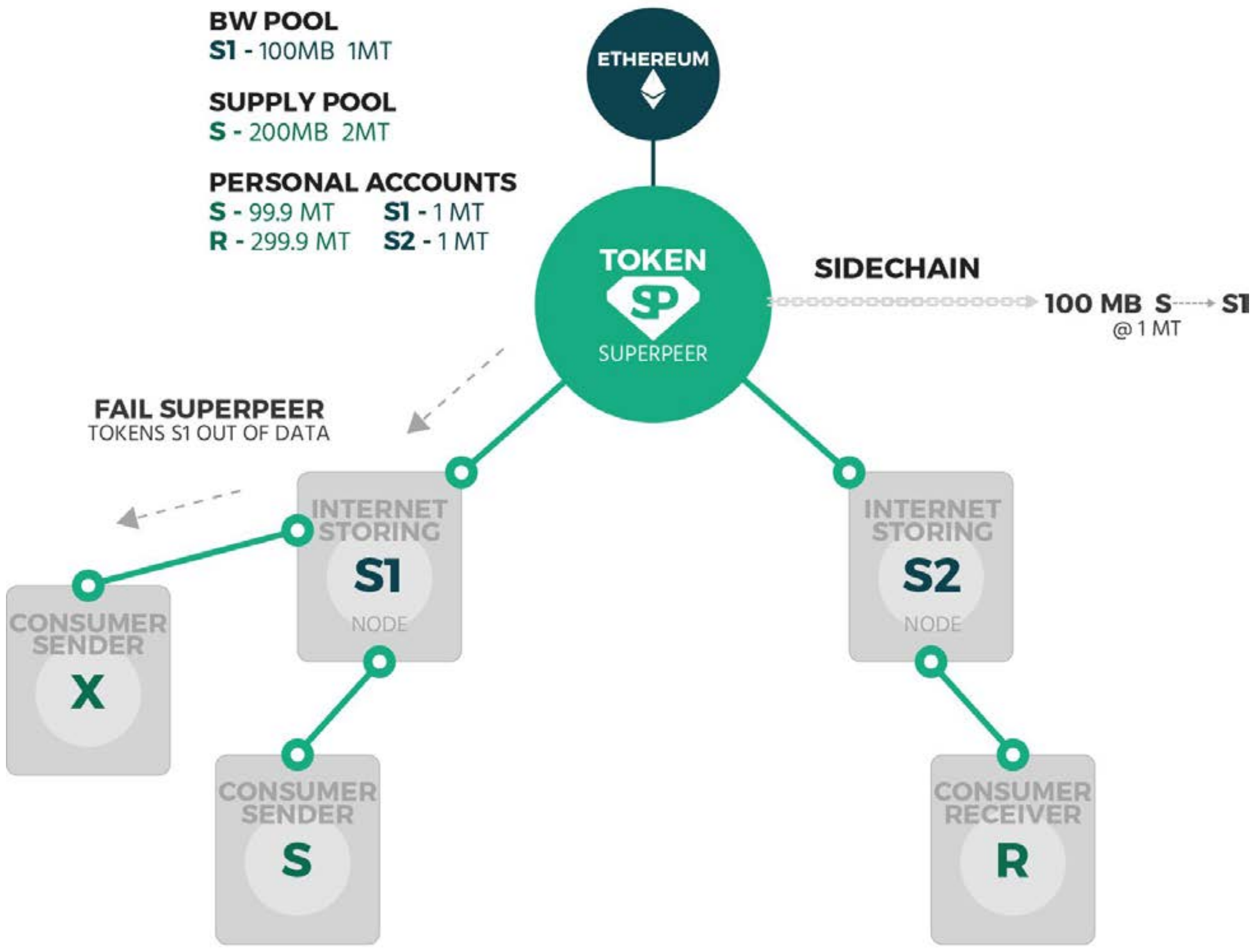


Figure 16: Failure case detected at superpeer because S1 is out of data

There are also equivalent cases on the R side of the network as well; however, the approaches to detecting error cases are similar to the S side of the network, and it is left as an exercise to the reader to determine how this detection would work.

Transaction Resolution and Sidechain Justification

According to the Kik Kin white paper, the Ethereum network can support up to 8.5 transactions per second.[1] By August 2017 it was projected that it would take just under 30 seconds to confirm a transaction. If every data request generated a transaction on the real Ethereum network, Ethereum

[1] Kik Interactive Inc., *Kin: A decentralized ecosystem of digital services for everyday life,* May 2017, (Source: http://bit.ly/2uOUxAE)

would quickly break down as RightMesh grows. And RightMesh would break down because there would be unreasonable waits for verification of basic tasks.

While we realize this might be a good opportunity to explore such things as Ripple, Lightning and Raiden, we would like to spend more time exploring these options and benchmarking before making a full commitment to any of these potential solutions (and also give some time to the Ethereum community to come up with an in-built solution to this scaling issue).

In the meantime, our proposed solution is to make use of a sidechain/off-chain approach, whereby the superpeer makes a daily private chain in conjunction with all other token superpeers together. Since the superpeer is required to relay real transactions anyway into the public Ethereum network, it can be a point where this type of processing may be done. An analogy to how this would work would be something similar to a pre-paid Internet plan where you reserve a certain amount of tokens at the start of the billing period in order to consume data. At the end of the billing cycle, any remaining tokens would be returned to you. As an Internet sharing device, you would receive compensation at the end of every billing period as well. If you participated as both roles, or as infrastructure, the difference between selling and purchasing data would be returned to you at the end of the billing cycle. This is all handled by three separate Ethereum contracts.

Regulating the Payout Process with Game Theory

There will be two distinct smart contracts for the buyer node and seller node respectively. At the end of the wireless data sale, both the seller and buyer are required to call the functions in the smart contract to report how much data (in bytes) has been sold or bought. The amount of sold data is reported by the seller and the amount of bought data is given by the buyer. Thus, two different functions are defined in the smart contract for such a purpose. These two functions can be called in any order, but the Rightmesh token transfer would not occur unless both of functions have been invoked by seller and buyer. The smart contract requires the amount(s) of data claimed by both seller and buyer to calculate how much the buyer needs to pay and how much the seller can earn.

For ease of presentation, let us denote the amount of data claimed by the seller as c_{seller} and the amount of data claimed by the buyer as c_{buyer}. If any one of them refuse to send his claimed amount, the corresponding value would be counted as zero. After the smart contract receives these values, it needs to calculate the amount(s) of data used to charge the buyer and reward the seller. The amount of data used for charging the buyer is represented by d_{buyer} and calculated as follows:

$$d_{\text{buyer}} = \begin{cases} c_{\text{seller}} - k_1\left(c_{\text{buyer}} - c_{\text{seller}}\right), & \text{if } \; c_{\text{buyer}} - c_{\text{seller}} < 0, \\ c_{\text{buyer}} & \text{if } \; c_{\text{buyer}} - c_{\text{seller}} \geq 0, \end{cases}$$

where $k_1 > 0$ is a constant parameter which is referred to as the punishment factor for the cheating buyer.

On the other hand, the amount of data used for rewarding the seller is represented by d_{seller} and calculated as follows:

$$d_{\text{seller}} = \begin{cases} c_{\text{buyer}} - k_2\left(c_{\text{seller}} - c_{\text{buyer}}\right), & \text{if } \; c_{\text{seller}} - c_{\text{buyer}} > 0, \\ c_{\text{seller}}, & \text{if } \; c_{\text{seller}} - c_{\text{buyer}} \leq 0, \end{cases}$$

where $k_2 > 0$ is a constant parameter which is referred to as the punishment factor for the cheating seller.

It should be noted that the seller (or buyer) does not know the amount of data that the buyer (or seller) wants to claim or has claimed. In fact, if they know each other's claim, the peer who first reports his data amount to the smart contract dominates the game, since the second peer has to report the same amount claimed by the first peer to maximize his benefit. The buyer can sign the transaction of the function by calling the smart contract and include raw transaction details within the contract. However, the seller cannot see the amount of data claimed by the buyer (and vice versa). Based on the previous two formulas, we can show that reporting the true amount of data sold and bought is a Nash equilibrium for the seller and buyer.

Note that, in our system, the amount of data claimed by either seller or buyer is not manipulated by user themselves, since related functions have been coded into the RightMesh library itself. The reason we still need the above mechanism is that the user could turn off his or her WiFi or other physical connections to avoid the payment. We have considered this in our design, and as a result, if any of the participants refuses to send a claimed amount, the corresponding value would be counted as zero. Thus, both buyer and seller want to cooperate and the misbehavior of malicious users can be avoided.

Eventually, the actual RightMesh token to be charged to the buyer should be the unit-price of data (per byte) multiply $d$buyer and the actual RightMesh token earned by the seller would be the unit-price of data multiply $d$seller.

Tracking Data Forwarded and Payouts to Intermediate Nodes

All of the previous discussion has made the assumption that the Intermediate nodes along the path would not be compensated. While this will initially be true, we also aim to incentivize these devices as well. The following outlines how this may be achieved.

For each source-destination pair, there may be more than one paths/routes between them. Even if only one route exists, we still have the problem of identifying which nodes participated in the route and how much data they helped in forwarding.[2] If there is more than one path for each source-destination pair, it becomes even more complicated. Thus, it is important to track the contribution of the intermediate peers.

The solution adopted by RightMesh is described in the topology below:

[2] In building the system, we considered multiple methods to solve this problem. One naïve approach considered was that whenever a packet is forwarded, the forwarder appends its MeshID in the data packet. However, two problems exist with this approach. First, since the MeshID is 160 bits, the size of data packet grows fast when going through a long-hop path. Second, we have a maximum packet size for a UDP packet in our RightMesh library, and meanwhile, we always try to deliver data with as less number of UDP packets as possible. That means adding extra MeshIDs into a data packet may exceed the maximum size of a UDP packet and an existing UDP packet may need to be split into two.

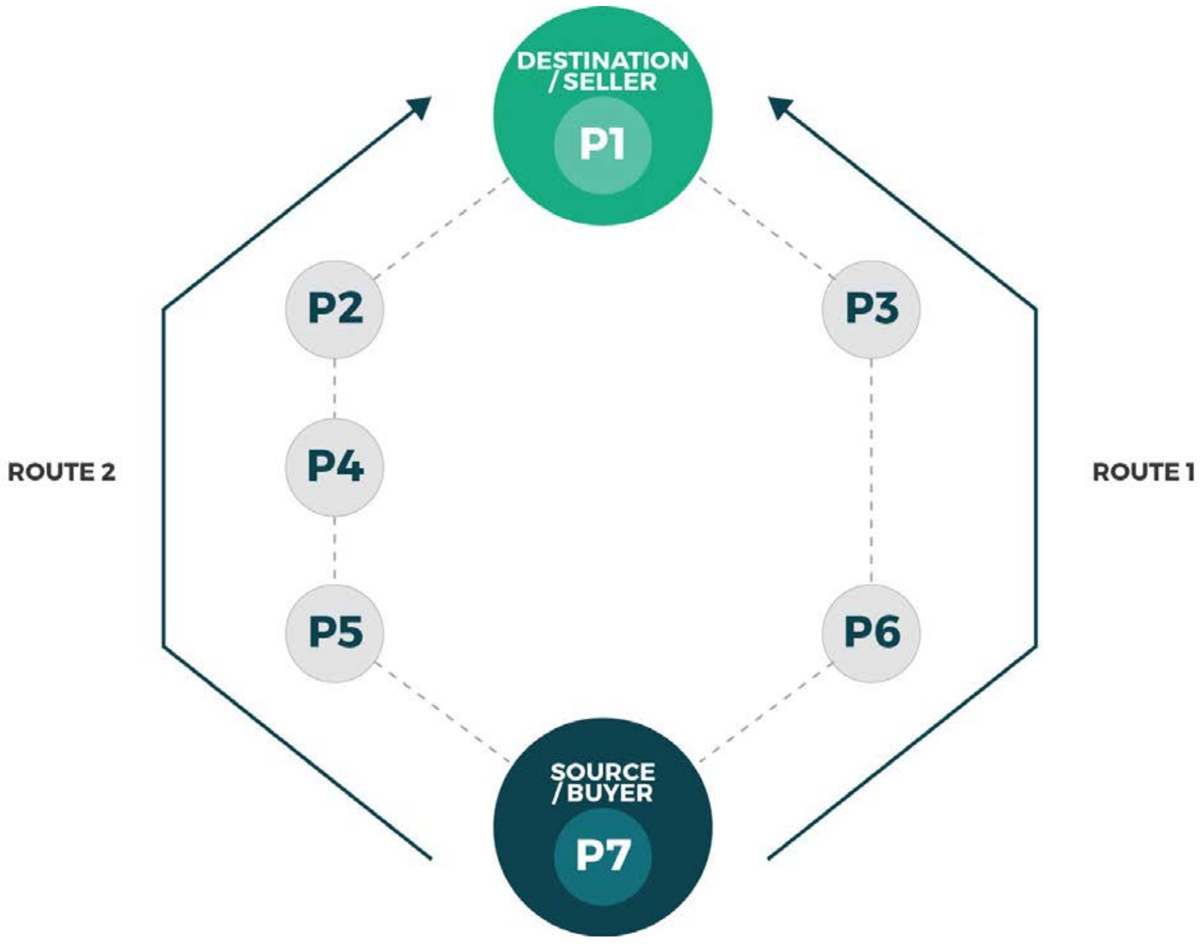


Figure 17: XOR to track intermediaries (to eventually reward them)

It has seven peers. Assume the buyer of wireless data is the dark blue peer, denoted by P_7 , at the bottom of the figure, and the seller is the green peer, denoted by P_1, at the top. The buyer now would like to send some data to the Internet via P_1 . For simplicity, we use P_1, ... , P_7 to represent the MeshIDs of these peers.[3]

Assume P_7 sends out a packet, it first conducts the bitwise XOR operation between 0 and its local MeshID. That is, 0x0^P_7. The result should be still P_7. This result will be included in the packet and will be used by the rest of the downstream peers. When the packet is sent to P_1 via the left route, every intermediate node first uses the existing result contained in the packet to calculate a new result by XOR with its own MeshID. Then, it replaces the original result by the new one and saves the new one locally. For example, P_5 first calculates the result of P_7^P_5 and use it to replace the

[3] A prerequisite to understanding this part is the binary Exclusive OR operation (See: https://en.wikipedia.org/wiki/Exclusive_or)

original P_7. After, it saves P_7^P_5 locally. This process repeats until the packet reaches P_1. The eventual result should be P_7^P_5^P_4^P_2^P_1. Note that, the length of this result is still 160 bits since every MeshID is 160 bits. We refer to this result as X_1. An instance of HashMap at P_1 uses X_1 as the key and save the amount of data traffic as the value of the key X_1 by tracking all the received packets tagged with X_1. Similarly, for the packets going through the right route from P_7 to P_1, the eventual result should be P_7^P_6^P_3^P_1 = X_2. Since there are 2^160 possibilities of the result, the chance of X_1 = X_2 is so small and can be counted as 0.

The keys of X_1 and X_2 together with their values will be logged into the deployed smart contract either in the final stage of the wireless data sale or in a periodic manner. Then, everyone who has X_1 or X_2 can generate some reward by sending a reward request to the smart contract with X_1 or X_2 as the argument.

For each source-destination pair, whenever an intermediate peer finds a new XOR result in the packets calculated sent from its predecessors in the upstream, the predecessor will be used as the next hop to deliver an end-to-end ACK. Note that the end-to-end ACK is sent from the destination back to the source. Inside the end-to-end ACK, we have the result of similar XOR operations from destination node to the source node. Since the XOR operation possesses the commutativity and associativity, an intermediate peer can use both the received XOR result in the end-to-end ACK packet and the previous saved result to get either X_1 or X_2 depending on which route that the intermediate peer is located at. This peer can thus directly request the reward by sending a request to the smart contract with either X_1 or X_2 as the argument. Due to the huge space of MeshIDs, which is 2^160, it is quite unlikely that an attacker finds either X_1 or X_2 by guessing. Thus, such a system is safe and reliable.

Supply of Traffic and Infrastructure

It is expected as the network grows initially, the use cases for the mesh will be primarily around sharing Internet connectivity with very localized parties. For instance, sharing data between a few friends where only one has sufficient Internet connectivity.[4] This will simply be a problem of density. As RightMesh grows in popularity, and as it is adopted in places where a user population

[4] RightMesh could be used to accelerate content delivery to an individual client node. Take the example of three friends each with 3G connectivity to their mobile devices. A client node may compensate the two other nearby nodes to use their Internet connectivity to have multiple concurrent connections to a content source, recombining the file across the mesh.

density is greater and the problem of connectivity is much more pronounced (such as the developing world), RightMesh will start to cover wider areas. The multi-hop, multi-path, multi-technology capability will set RightMesh apart from competitors who have designed solutions which will not scale technically.

As more apps are designed for RightMesh, and as we make more partnerships with device manufacturers, app platforms, and ad distribution providers, there will be more devices with the RightMesh library running passively on a network. This will increase the value of the network, leading to even further increases in the density of devices, and enabling the creation of even more diverse applications powered by RightMesh. This will create a moat to future mesh platform developers should they emerge.

As RightMesh spreads in usage and is applied over wider areas, there will be many opportunities where users will be able to adjust their prices to compete with other providers and client users will have a wider variety of paths to take in the network. RightMesh can join forces with applications which aim to provide WiFi coverage maps, and instead of WiFi coverage, RightMesh could provide pricing maps. Areas with limited connectivity (or expensive coverage) would end up being places where people could physically move to so as to provide coverage for others and make a profit. An apt comparison here is to a ride-sharing driver relocating their stationary vehicle to the most optimum location to maximize their revenues.

Temporary Storage for Mobility

One of the key potentials for increasing the value of RightMesh to users is the delay tolerance which is possible and built into the platform. Data may be sent and preserved for delivery to another peer even when the peers have become separated from each other. That is, the devices may be in separate meshes, or the current mesh has split, or one or both sides lacks an Internet connection. This process means that less Internet data is required to successfully deliver intended content to the recipient. Rather than starting over in entirety, if anything has successfully been delivered, RightMesh will pick up where it left off when the path broke. Furthermore, sending will continue automatically when a new route to the intended receiver has been detected. This type of temporary storage is called “forwarding storage” and is used when a device temporarily disconnects from the network mid-data stream. Forwarding storage is used for faster recoveries during transmission.

Secondly, it is possible that client devices may also reserve some portion of their storage capacity for caching in the mesh. This not only improves performance significantly by reducing the number of requests sent directly to the Internet, it also provides a semi-offline way in which applications, advertisements, Web services, and other content (videos, music, images, maps, and other media) may be distributed. This creates many new opportunities for emerging markets that the fully-connected world already takes for granted. This type of storage is called Offline Transit Storage. With this type of storage, an entire artifact (think: file, video, music, advertisements, etc.), or potentially only a portion of it, is stored on a device indefinitely so that devices that are connected to fully offline meshes may retrieve the artifact as the mobile device moves from one offline mesh to another.

This feature of RightMesh could be greatly enhanced by supporting or collaborating with existing solutions in this space, including IPFS/Filecoin and Storj, which currently operates in the traditional infrastructure based p2p space. A collaboration here could enable these existing solutions to expand to devices and people which are unreachable with existing infrastructure. Furthermore, an IPFS hash could be sent efficiently across the mesh allowing the user to retrieve the file from the most appropriate node.

Developers API

The developers API is meant to be as simple as possible for the developer to use. There is not really any knowledge of networking that is required in order for the mesh to function. The main functionality can be summarized as follows:

- Peer discovery and updates
- Encryption key exchanges
- Sending and receiving data to a mesh peer
- Executing Ethereum transactions and contracts
- Allowing the user to configure their role in the network

All of the code is event driven. When a new device joins the mesh and it is running the same app as another device, upon discovery of each other in the library, an event is fired by our library. The developer must define a handler function for the event. In the case of a peer discovery event, the developer may wish to send a message to the new peer, update a UI element showing a list of peers, exchange encryption keys, etc.

All of our major functionality works in this manner. The sending of data is also quite simple. The developer simply calls the send function, provides the MeshID of the peer to send the data to, and the byte array of the data to be sent. The library will figure out how to split the data up, how to route the data, and how to deliver it successfully. On the other device, a received data event will be generated which can be handled by the developer with their own custom function.

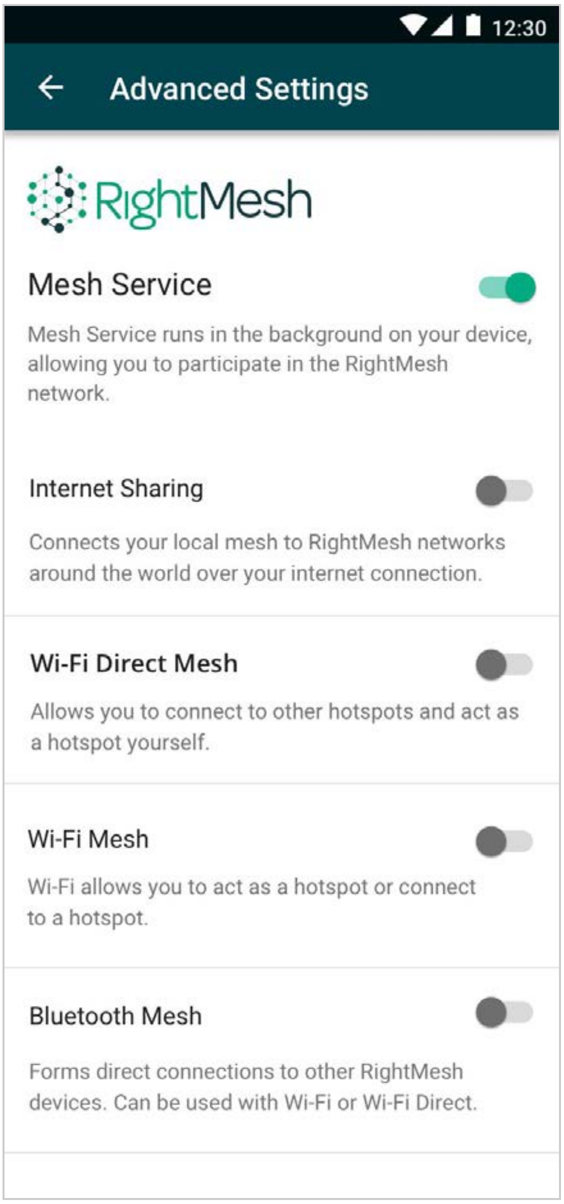
The full developers API is available on our website at: https://developer.rightmesh.io/api/latest/

There is also a detailed getting started guide available at:
https://developer.rightmesh.io/reference/

End-Users

RightMesh from the End-User's perspective is designed to be as easy to use as possible. The mesh by default uses an autonomous mode which tries to form connectivity wherever it can. We have the philosophy that the library should never interrupt the way you are currently using your phone, so it will not switch off of existing Wi-Fi networks to join the mesh (it will attempt to use Bluetooth or Wi-Fi direct instead to form connections).

We also believe in giving the user the choice in how they wish to participate. If the user wishes, they can disable Internet sharing, and even choose not to participate as infrastructure. We also support an advanced mode where power users or developers may control their role in the network at a more fine grained level (selecting whether the device should setup hotspots, perform switching functions, enable / disable bluetooth, Wi-Fi and Wi-Fi direct independently, etc.).


12:30

Advanced Settings

RightMesh

Mesh Service

Mesh Service runs in the background on your device, allowing you to participate in the RightMesh network.

Internet Sharing

Connects your local mesh to RightMesh networks around the world over your internet connection.

Wi-Fi Direct Mesh

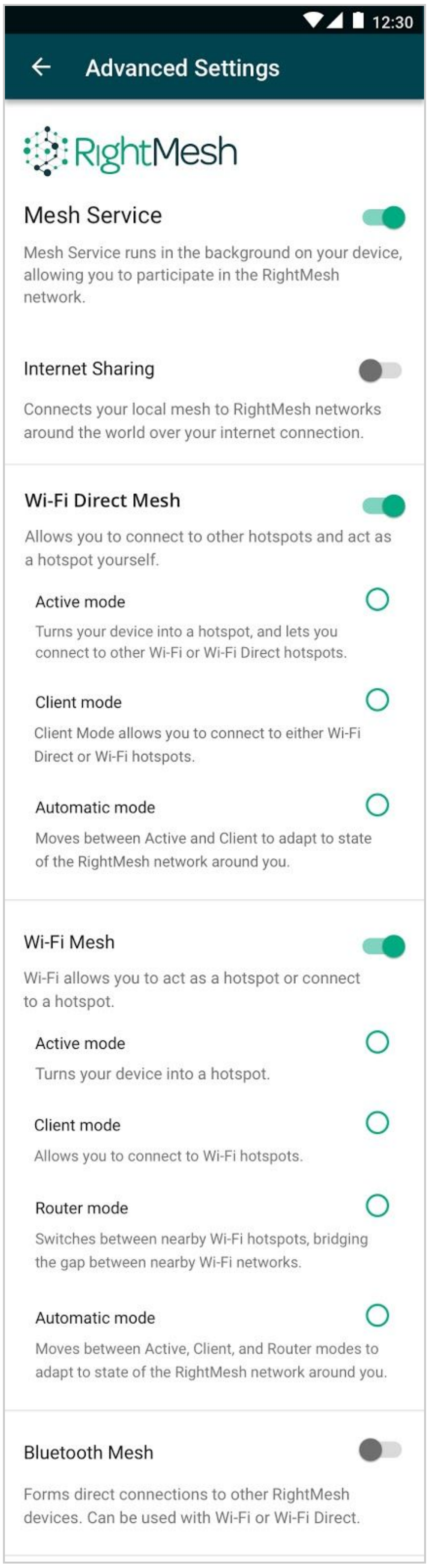
Allows you to connect to other hotspots and act as a hotspot yourself.

Wi-Fi Mesh

Wi-Fi allows you to act as a hotspot or connect to a hotspot.

Bluetooth Mesh

Forms direct connections to other RightMesh devices. Can be used with Wi-Fi or Wi-Fi Direct.


12:30

Advanced Settings

RightMesh

Mesh Service

Mesh Service runs in the background on your device, allowing you to participate in the RightMesh network.

Internet Sharing

Connects your local mesh to RightMesh networks around the world over your internet connection.

Wi-Fi Direct Mesh

Allows you to connect to other hotspots and act as a hotspot yourself.

Active mode

Turns your device into a hotspot, and lets you connect to other Wi-Fi or Wi-Fi Direct hotspots.

Client mode

Client Mode allows you to connect to either Wi-Fi Direct or Wi-Fi hotspots.

Automatic mode

Moves between Active and Client to adapt to state of the RightMesh network around you.

Wi-Fi Mesh

Wi-Fi allows you to act as a hotspot or connect to a hotspot.

Active mode

Turns your device into a hotspot.

Client mode

Allows you to connect to Wi-Fi hotspots.

Router mode

Switches between nearby Wi-Fi hotspots, bridging the gap between nearby Wi-Fi networks.

Automatic mode

Moves between Active, Client, and Router modes to adapt to state of the RightMesh network around you.

Bluetooth Mesh

Forms direct connections to other RightMesh devices. Can be used with Wi-Fi or Wi-Fi Direct.

Creating Feedback Loops

One of the biggest challenges in the mesh will be scenarios where MESH tokens are collected and concentrated and a lack of supply will circulate the mesh. First, having a minting process tied to actions that are critical to the mesh functioning will encourage people not just to speculate, but actually participate in sharing data. Second, by continuing to add more value than just data sharing, it will encourage more usage and feed into a cycle that drives more growth. Some of these initiatives include supporting ad and app distribution, supporting "data mule" behaviour, incentivizing storage/caching, processing resources -- all over the mobile mesh where competing platforms struggle to reach beyond one-hop away from the Internet.

Upon a successful token generation event, some portion of backers will have bought RightMesh Tokens which they intend to use to purchase resources from sellers on our mesh. At launch, there will likely be a mix of both backers and users who have not yet purchased any MESH tokens. Users which have not yet purchased MESH tokens will be able to earn tokens by selling their own data (the fastest way which includes earning MESH tokens from consumers of data, and also in a minting process), or by participating as infrastructure (where MESH tokens will be earned much more slowly).

There will be people who are attracted to the entrepreneurial route of becoming a data seller. These people may not have any MESH tokens initially, but they can accumulate tokens at a higher rate by reselling their data, or through other exchanges of value (viewing advertisements, sharing applications, being a Routing Node). RightMesh will receive tokens as a commission for data selling, relaying and verifying Ethereum transactions (while running a Superpeer), or for facilitating data distribution across the mesh. RightMesh could then perform other actions and feed the tokens back into the network. Furthermore, those who have earned tokens by being a Data Seller or by exchanging value across the mesh (storage, processing power, content creation, attention, etc.), may now choose to sell their RightMesh Tokens to people who would like to purchase more in order to buy more resources on the mesh.

RightMesh Tokens may be used by RightMesh-powered mesh applications and services. Rather than setting up their own token or cryptocurrency, and figuring out how to operate it within the confines of the mesh (including the complexities of setting up the signing procedures and infrastructure to relay requests to Ethereum or some other crypto network), anyone who develops a RightMesh mobile app or game could simply use the MESH tokens as proxy for in-app currency, to trade for in-app items, or even real-world goods and services (e.g., compensating a driver after using a decentralized ride sharing app device-to-device).

Token Minting

If we adopt a token model where the supply of tokens is not fixed, we can reduce the risk that large numbers of tokens will be constrained without liquidity. The main goal of RightMesh is to have people using the tokens to incentivize supply data, forwarding packets, acting a "data mule", providing caching and storage, providing processing resources, etc. in places where existing infrastructure does not reach people. Given that this is the goal, we can introduce a minting mechanism, which we have called "Proof of Forward", whereby a small portion of Mesh Tokens are generated for completing actions core to the platform success, such as: successfully providing a full data send or recv into the mesh. The amount that is minted tokens should scale with the size of the successfully send/recv, so it does not encourage people to do weird things like only send small tiny amounts or very large amounts.

Lastly, for every transaction that occurs within the RightMesh network, a percentage (TBD) of that transaction value in tokens will flow back into RightMesh. These tokens can be recycled back into the network or used to incentivize behavior and further drive action in the network.

Failing all these options to try to prevent the situation where RightMesh tokens stop flowing, it may also be possible to take more drastic action to keep network tokens flowing. This can be discussed and debated with community members in the unlikely event that this is required.

Summary of Technical Roadmap

While there are details and hints at our roadmap scattered through this document, this section collects the most important milestones and future roadmap elements to come in one spot, so that readers can understand clearly what has been built, what will be built, and what needs to be fleshed out.

What Works Today:

As of August, 2017, RightMesh has a basic mesh networking platform developed for Android devices. The platform as it stands now is a library that supports multihop connectivity between many devices (we have had up to 20 so far on a single mesh) and can cover long distances (we have achieved about 1000 feet with only 5 devices; we will need to do further experimentation to cover longer distances).

There are several sample apps available on our public GitHub repository: https://github.com/RightMesh including a "hello mesh" app which gives an introduction to developers, a colour changing demo that illustrates how our library forms routes compared to the broadcasting libraries, a "ping test" type of app which is good for coverage testing, and soon there will be a simple hello Java app which shows the same type of code running on a something like a personal computer, Raspberry Pi, or other device running Java.

The mesh portion of the library is becoming quite mature. The connectivity aspect supports Wi-Fi, Wi-Fi Direct and Bluetooth and has two provisional patents covering it. It is partially autonomous right now, in that once the user selects which technologies to enable and which role they wish to be in (hotspot, switching, or client) the network will self form. We are working on what we call "fully autonomous" mode where the library can figure out the best mixture of these roles and technologies to enable.

We have put quite a lot of time into a Developer's Portal, so our library is easy to program with, and well documented. This includes Javadocs, a step-by-step *Getting Started Guide*, and information about what to include in Gradle.

On the crypto/incentivization side, we acknowledge that we are not experts in this area and have been seeking both talent and advice on best practices in implementation. However, we have been able to achieve a few key milestones. The first question we had was, "Can we execute Ethereum transactions from our mesh, several hops away from the Internet?" Using the Ethereum-j library, plus the Ethereum accounts we generate as our MeshIDs, we can do this using remote signed transactions. These transactions can be relayed into our superpeers which execute the transactions on behalf of the phones. You can think of the phones as specialized thin clients that can operate many hops away from the real Internet connection.

We have also made some inroads with some proof-of-concepts for contracts. We have worked with a sample token contract to represent our app token and issued a pool of tokens. We have created a contract that can act as a data pool where sellers can offer their data for sale for a price. We have also implemented the buyer contract where buyers pre-purchase their data. We've got two apps working on Android where one is a buyer and one is a seller. They are not connected to the mesh yet, but they simulate this and are connected directly to a laptop running a full Ethereum node. The buyer is able to send data by executing all of the pieces of the contracts as spelled out in this white paper. The last step is to integrate this into the mesh platform to fully realize the vision we have presented.

What is Next to Come:

The next major milestone on the mesh side is real performance analysis. While we don't have official performance numbers we can stand by yet (because we haven't obtained them over a wide number of hops using a scientifically rigorous process), we have small scale evidence (over a small number of hops with toy types of applications) that we can achieve high performance (on the order of Mbps). As we go farther down this road, we will release further apps that allow for throughput and delay measurements, so the community can try out the mesh and perform their own tests to see where and how it works best.

Along with the performance analysis is getting a clear idea on how the network performance changes as the network grows. Where are the bottlenecks, how does performance degrade over hops, and how many Internet sharing devices can one superpeer support? How many consumers can one internet sharing device support? Also, with <x> devices consuming data, how quickly will <y> data be consumed? After we answer some of these questions, there will be lots of opportunities to improve the performance as we will learn a significant amount about where the network is weak, while helping us generate ideas on how to improve.

One of the biggest challenges for us on the network side is automated testing of the library. Given that much of the operation of the library depends on complex connection topologies between groups of phones, it is difficult to simulate effectively. There is some portion of the library that we have applied unit testing and limited offline testing to; however, some of it requires real device testing as there are aspects that need to be tested specific to particular Android versions, the orientation of the phones near to each other, and the order in which the devices discover each other. We would like to support phones from Android 4.x to present; however, with the wide range of capabilities, the way Android handles permissions, and other differences between devices, this means there is a huge scaling problem to automating the whole process since the number of testing possibilities grows exponentially as the mesh size grows. We are looking at ways to automate as a much of this as possible so future builds are stable across a wide range of devices.

As mentioned on the incentivization side, the biggest challenge going forward is integrating the contracts with the mesh. We don't anticipate this being particularly difficult because we have already had the mesh execute remote signed transactions, and we can do the same with contracts. The biggest challenge here is getting the sidechain part right: whether this is using Ethereum, Lightning, Ripple, or some other technology. In addition, getting the economics of the system correct so that there is stability, so that tokens don't end up getting stuck and not used in the network, etc. This is where we are looking for the most community feedback and advice as we are more mesh focused than blockchain focused, but we recognize that blockchain technology is the best path toward incentivizing the mesh.

Once the basic data sharing incentive structure is in place and well understood, the next step will be apps for ad and app distribution, "data mule" incentives, storage/caching, processing, and more.

We are also rapidly growing our team in Canada, and are on the lookout for more talent. Whereas others are building walls, do note that Canada has a new two-week processing program for skilled workers under a new initiative to invite the world. We do not discriminate based on location, nationality, religion, or sexual orientation, and are consistently recognized as one of the Best Workplaces in British Columbia, including winning the award in two straight year from the BC Tech Association as the best tech company in BC that balances, "Work, Life, and Play". So if you have experience (either academic or professional), passion, and a commitment to doing things right, we would appreciate hearing from you.

We welcome feedback via our Slack channel, Git, Telegram, and more methods identified below.

Additional Resources & Links

RightMesh White Paper:	https://www.rightmesh.com/whitepaper
Website:	https://www.rightmesh.io
Developer Portal:	https://www.rightmesh.io/developers/
Corporate Blog:	https://www.rightmesh.io/news/
Twitter:	https://twitter.com/Right_Mesh
Slack:	https://rightmesh.slack.com
Telegram:	https://telegram.me/RightMesh
GitHub:	https://github.com/rightmesh
Careers:	https://www.left.io/careers