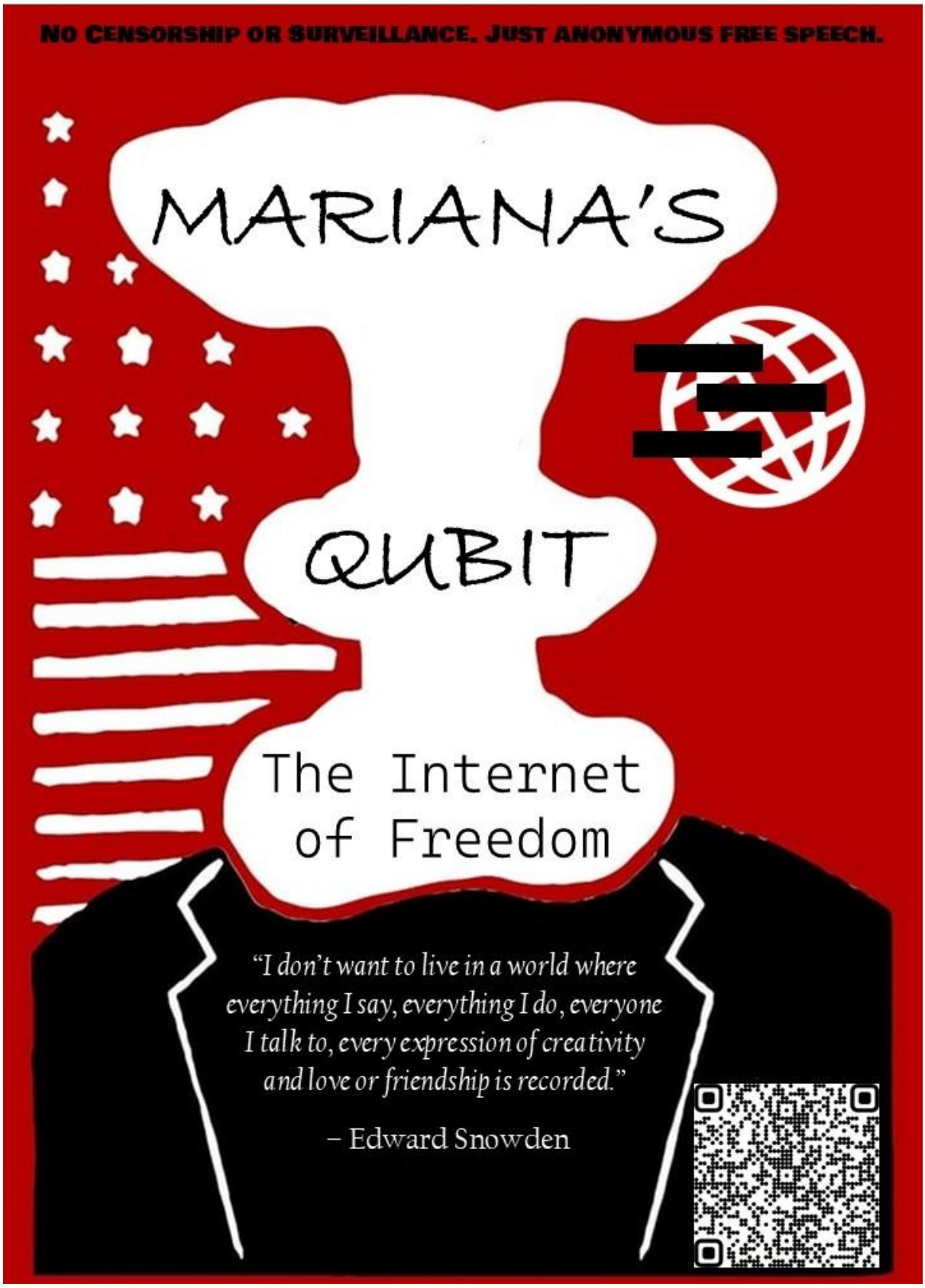
NO CENSORSHIP OR SURVEILLANCE. JUST ANONYMOUS FREE SPEECH.
MARIANA'S
QUBIT
The Internet of Freedom
"I don't want to live in a world where everything I say, everything I do, everyone I talk to, every expression of creativity and love or friendship is recorded."
– Edward Snowden

Project Mariana's Qubit

"I don't want to live in a world where everything I say, everything I do, everyone I talk to, every expression of creativity and love or friendship is recorded."

– Edward Snowden

By Cryptane. For the Free Thinkers.

Contents

Abstract – Privacy is Survival 4
Motivation – Don't just Listen. Speak. 5
Technical Introduction – You need not be a geek to understand 7
Architecture Overview – The power of connection 8
Protocol description 10
Network layers 10
Layer 1 (Physical Layer): 10
Layer 2 (Data link Layer): 10
Layer 3 (Network Layer): 12
Layer 4 (Transport Layer): 13
Layer 5 (Session Layer): 14
Layer 6 (Presentation Layer): 14
Layer 7 (Application Layer): 14
Encapsulation: 15
Node discovery: 15
Local node discovery 16
Online public node discovery: 16
Offline public node discovery: 16
Applications on Mariana – Taking back the control 17
Web Server Proxy 17
Secure Clearnet Exit 18
Mariana Netcat 19
Trench Talk 20
Cargo Ship 21
Phonebook 24
Mariana Firewall and Security – Trust Nobody 26
The Bigger Picture – No gatekeepers, no surrender 27
Call to Action – A thousand nodes, a million free minds 28
Conclusion – In depth of the trench, Mariana grows 29
Appendix – Online References – Trust yourself 30

Abstract – Privacy is Survival

Mariana's Qubit is designed as an anonymous internet architecture from scratch, without depending on much existing libraries. It serves as a self-sustaining, self-healing private internet, resistant to surveillance and censorship. Theoretically it is proven to bypass state-sponsored mass-surveillance and censorship attempts. On a superficial level, it may look similar to Tor, but at a deeper level, it is quite unique. Mariana's Qubit is a bold attempt to protect personal privacy in the face of mass-surveillance and be able to share your opinion in the face of mass-censorship.

Motivation – Don't just Listen. Speak.

"Arguing you don't care about privacy because you have nothing to hide is like saying you don't care about freedom of speech because you have nothing to say," as said by Edward Snowden is more and more relevant with each passing day. Communication is the lifeline of human society and it flows freely when a person meets or interacts with another. With the advent of the Internet and social media, technology has brought the entire world closer to each other. People started communicating with each other more easily and freely.

However, in the early 2010s, tech giants and corporates realized that communication is a huge business model. How a person communicates and what a person reveals to anyone could be a pillar to supporting capitalism: the communication highlighted your preferences, fears, dreams, aspirations and more which started to be commoditized. How often have you come across a social media post or an ad about the exact thing you were planning to buy? Now imagine it in a real-life scenario. You are talking to someone about your deepest fears and insecurities and bunch of strangers eavesdropping on the conversation and then later trying to feed on your insecurities. For example, as a student your insecurity might be passing an exam, or getting a job. And suddenly strangers come up, urging you to join so and so course, buy so and so book, etc., reminding you of your insecurity every single moment. And it is about all your insecurities and fears. Tech giants like Google, Meta, Amazon actively build deep behavioral models on every individual user their services.

And then comes the government. Be it any elected government or an authoritarian regime has normalized surveillance of the citizens and non-citizens, with the excuses like national-security. No country would have a huge fraction of its own citizens surveilled for national security, but they do. Now there is a huge number of people, the common man, who believe 'I am too insignificant to be surveilled.' The common man does not understand the scale of technology. Nobody is sitting at a computer screen and monitoring you personally. But at the same time, huge pattern recognition and machine learning models are probably being run on all your online and internet activities, trying to profile you, group you into categories, or to in-fact identify your political affiliations etc.

In 2016, before the United States Election, you probably would have seen a rise of Facebook games. Pretty harmless games. You would just open it, and answer a short quiz or something and it would spit out outputs like which celebrity you look like, how intelligent you are, etc. Were they really harmless games? At a later investigation it was revealed that one of the political parties collaborated with an organization called Cambridge Analytica who used these games to profile your view points and political mindset. It was correlated to your approximate location, thus revealing which state you are in. This gave a really accurate picture of the swing states, better than any exit-poll. And this gave the political party clear idea on which campaign strategy to follow in which state, supporting them for the elections. This is one of the cases of surveillance that you would have directly faced.

This is just the tip of the ice-berg of surveillance. Many messaging platforms, claiming to be End-to-end encrypted often leave behind digital breadcrumbs, technically known as 'metadata' of your communication. This may not reveal the contents of your messages but may reveal whom you contacted, for how long were you on a call, how many messages you have sent or received etc. And as we like to say 'Metadata is surveillance'. It's not just left for analytics.

Authoritarian regimes like China and Russia enforces strict internet censorship policies like the 'Great Firewall of China', or 'Russia's RuNet' preventing citizens to reach the global internet, or to monitor all activities of the citizens on the global internet. But it is a misconception if you assume only authoritarian regimes follow this. Snowden exposed programs like PRISM, XKeyscore, Five Eyes, etc., in US which was aimed at mass-surveillance of the citizens. Other countries like U.K had the RIPA Act in 2000, replaced by IPA 2016 which allows intelligence agencies to analyze data in bulk, including electronic devices, communications and internet activity. The UK government has the power to intercept communications, access internet browsing history, and require internet service providers to retain data. UK

utilizes a wide range of surveillance technologies starting from CCTVs to Wiretapping. And about the recent news that UK Government has mandated Apple to allow backdoors in iCloud encryption, due to which Apple has stopped offering End to end encryption for iCloud in UK, most of us have come across this news.

In June 2016, Turkey passed the Internet Kill Switch law that allowed the government to suspend internet access in cases of war, national security or public order.

When it comes to Censorship, the well known and famous SOPA, PIPA acts which were ultimately tabled in 2012 allowed government to suspend any website on pretext of copyright infringement suspicion. Well, a similar law is coming back as FADPA law though it is currently a draft.

Skype (before 2011) had a built-in wire-tap. Phones sold with a promise of secure connectivity were actually FBI honeypots (FBI Operation Trojan Shield), Juniper Networks disclosure of backdoor in 2015, NIST Dual_EC_DRBG Random number generator, etc., were all proven to be backdoored.

As we like to say, ‘A backdoor for a government is a backdoor for hackers.’ Or, ‘powers for national emergency or IP rights are powers for surveillance and censorship.’

Having discussed a lot about the current state of surveillance and censorship, let’s come to why Mariana. The truth is if you let a corporate being control the internet, it will never be free from censorship or be independent. And True privacy is not possible without being in a decentralized, independent, encrypted infrastructure. And Mariana aims to be one of them. Freedom should not be a permission granted by infrastructure owners; freedom is a right.

Some other decentralized infrastructure like Tor, I2P, Freenet exists. And each of them comes with its own vulnerabilities (or backdoors). Tor is one of the most popular ones but it has a vulnerability, which allows predictable exit nodes and tapping those could lead to a loss in privacy. Further Tor, though open-sourced now, was originally developed with government funding, raising concerns about privacy advocates. Building resilient alternatives to the internet communication is no longer idealistic – it is survival.

The question remains, why Mariana? As the developer, my claim is that the protocol uses anonymized identity for each node, while intelligently choosing exit nodes to reach out to Clearnet to preserve privacy and anonymity. The technical details of ‘Secure Clearnet Exit’ and the whole infrastructure will come later, but the vision is to make all communications encrypted, never reveal the identity, even if forced to using tools like embedded trackers and to empower yourself to be the owner of your communications: not some third-party corporate entity. And about why trust Mariana? Don’t do it blindly. The technical specifications are here for that.

The next frontier of freedom isn’t on a battlefield – it’s over invisible networks, among free minds.

Technical Introduction – You need not be a geek to understand

Mariana's Qubit is a next-generation, anonymous routing protocol, designed to build a fully functional and quantum-resilient network stack. While for a layman, it is similar to Tor, it is quite different at the fundamental and protocol level. Project Mariana's Qubit builds an entire network stack, analogous to the OSI model to securely route traffic anonymously via relays and proxies. It is truly decentralized, unlike Tor which maintains a centralized relay list for bootstrapping.

The system offers dynamic network topologies where nodes attempt to form a mesh network. For nodes on private subnets without internet connectivity, it attempts to perform local node discovery and forms a network with other nodes in the same subnet. If one of the machines in the private subnet is connected to the internet, probably via a second network interface, it automatically relays traffic from its subnet peers, making them reachable from anywhere on the Mariana Network without involving any manual setup.

For nodes that are in the routable internet, (not behind a NAT or private subnet), it automatically promotes itself to a 'public node' and adds itself to an open-source public relay list, maintained on Github. A node which is behind a NAT and does not have other nodes in the same private subnet may attempt to fetch the public relay list from Github and connect through that. It also maintains a persistent copy of the list on its disk so that it can access the same in case Github is not working. This copy of public relay list is often announced through the network so that other nodes can make a copy of it, thus making Github redundant when the network grows.

Each node on the network is identified by a randomized unique ID, also known as NAC (Network Address Code). Not by its IP or any other identifiable information. Nodes aiming for deeper anonymity may regularly rotate their NACs, while service hosting nodes can maintain persistent identities to remain reachable. The routing list of the network allows sending packets to the destination node by NAC. It uses a custom distance-vector routing protocol to work with NAC.

The system offers strong storm control mechanisms and self-healing. A node going down is automatically removed from the entire network within 60 seconds and a node coming online is automatically added to the routing tables almost instantaneously.

The system performs optimized packet loss management. Unlike TCP, it does not send an ACK for every packet, which increases the overheads. Instead, the destination node makes a list of dropped packets by the sequence numbers of the same and requests retransmission from the source node.

All packets transmitted over the network, containing payload, are encrypted by AES and Kyber 512 for post-quantum encryption ensuring it is resistant to attacks involving a large-scale quantum computer, thus providing a truly decentralized, anonymous network.

On the user end, it opens Chrome (or any browser) proxied through the Mariana Daemon. On the server side, or side of receiving node, it proxies the traffic to any service hosted with a web server like Apache2, Nginx or IIS. Hence, a developer may just develop a web application and let the Mariana Daemon route it over the Mariana's Qubit network anonymously. In upcoming versions of Mariana's Qubit, it is aimed to be able to proxy traffic to any socket, making it a truly protocol-agnostic anonymity network.

Architecture Overview – The power of connection

Mariana's Qubit (will also be referred to as 'Mariana' in the rest of the document) uses UDP traffic and packages data with custom headers to be anonymous. It is designed in 7 layers, analogous to the OSI model where the Layer 1 (Physical layer) deals with how Mariana interacts with existing network infrastructure while Layer 7 (Application layer) deals with how web applications can be used over Mariana.

Mariana deals with 4 different types of nodes:

- Private nodes: The nodes that are in screened subnets without direct connectivity to the internet or to the rest of the network.
- Private relay nodes: These are nodes which is the part of a subnet containing one or more private nodes, while at the same time has a secondary network interface that may reach another private subnet, or the internet. Private relay nodes are characterized by having 2 network interfaces.
- Regular nodes: These nodes are usually behind a NAT but is connected to the internet or can reach other nodes in the network.
- Public nodes: These nodes are characterized by being directly accessible from the internet and not behind a NAT. The port interacting with the Mariana network in a public node may be whitelisted in the firewall, thus making it reachable over the internet.

It is to be noted that the different types of nodes are not differentiated among each other in the Mariana network. This classification is made solely on basis of network exposure of a particular node and not on basis of activities performed by a node. Each node runs the same application and behaves identically.

Figure 1 shows the different types of Nodes in the network and how they would automatically connect to each other, building the resultant network topology. Each has the capability to act like a relay to send traffic to other nodes. It is also significant that Mariana does not use any centralized repository (unlike Tor) for bootstrapping; neither does it use hardcoded seed lists (unlike I2P). This ensures a system in a subnet without internet connectivity may also reach the network via local node discovery and relay. However, public nodes add themselves to an unmanaged open-source repository which may be used a fallback when nodes without peers in the same subnet attempt to connect. This open-source repository is not considered centralized since it is unmanaged by anyone and the public nodes can automatically append themselves to the list.

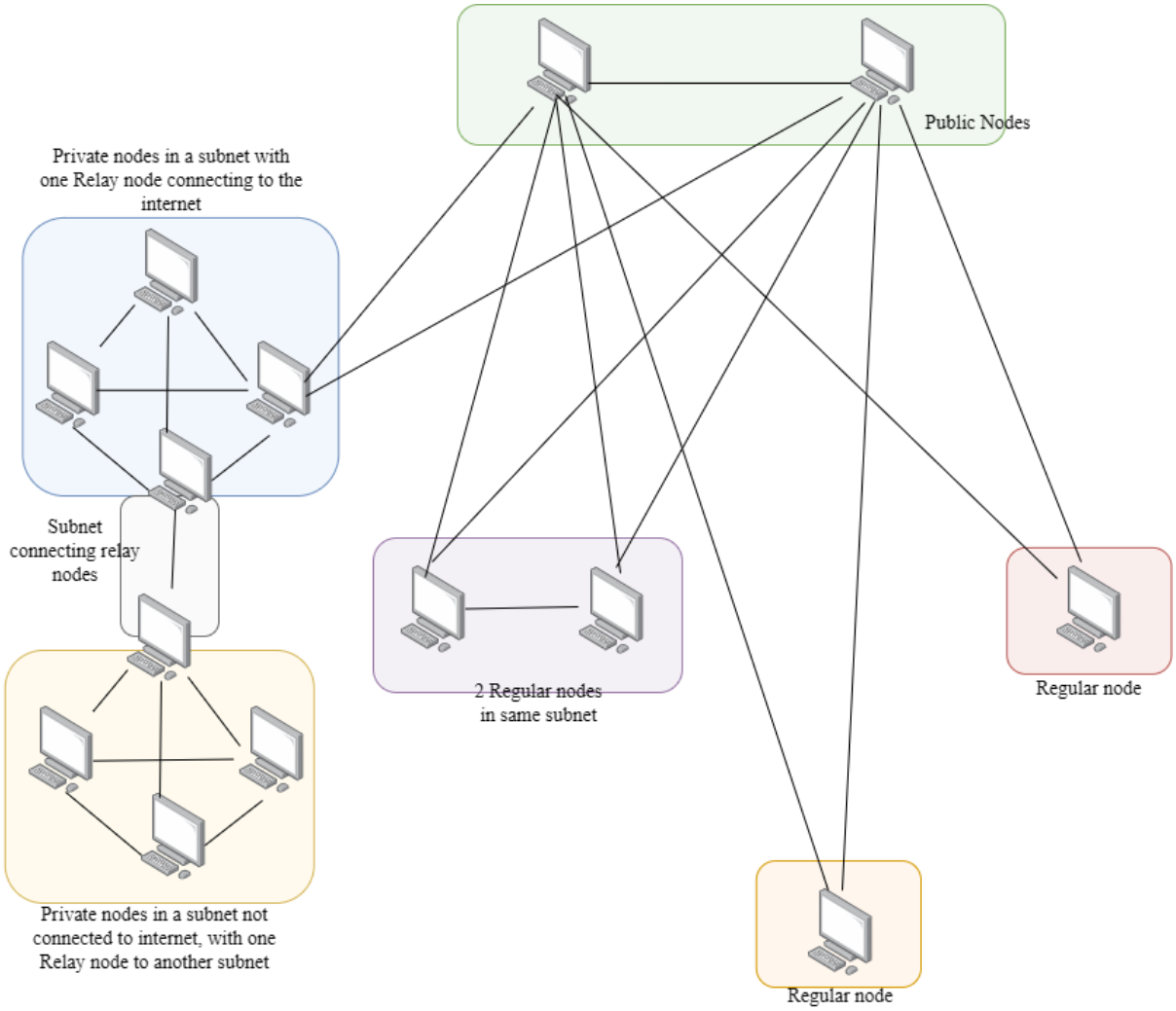

Figure 1: Mariana network topology with various types of Nodes

Mariana attempts to build a full-mesh among all connected nodes automatically and uses a distance-vector routing protocol to send the traffic to the destination node via the shortest route. However, it is obvious that all nodes may not be included in a full mesh due to being in different subnets, often private subnets. However, Mariana autonomously tries it's best to ensure any node can be reached from any other node, even via relays, relays with multiple network interfaces, local discovery.

Mariana uses Post Quantum public key cryptography and symmetric cryptography to ensure all data is encrypted no matter which route it takes to reach the destination node. Mariana propagates the public key of a node through the network when the node joins the network as a part of the routing table. Old/ stale routes and public keys are removed from the network after a fixed timeout. The network is self-healing and self-sustainable without any centralized server.

Each node on the network is identified by an anonymized ID, known as NAC or Network Access Code which prevents fingerprinting of the clients. Further since all traffic is encrypted before sending and decrypted only at the intended destination node, it's not possible for any third party to interfere. Further, a robust retransmission mechanism ensures that information is not lost while in the network.

Protocol description

Mariana relies on UDP communication with next the next node in order to connect and communicate with the network. The architecture is divided into layers analogous to the OSI model.

In Mariana network, each node is identified by a unique 16-byte identifier, known as NAC. Further, each node has its own public key cryptography keypair. Mariana uses Kyber-512 as the preferred public key cryptosystem, owing to its Post Quantum nature. This makes Mariana not vulnerable to attacks involving a large-scale quantum computer. The public key size of Kyber-512 is 800 bytes. Each node has its keypair, with the private key stored securely.

Network layers

This section discusses the various layers and their functions on Mariana, as well as the encapsulation strategies on the higher layer packets being encapsulated into lower layer payloads.

Layer 1 (Physical Layer):

The physical layer defines how each node communicates with the next immediate node over the standard network infrastructure. Nodes in Mariana uses a random UDP port in the range 1024 to 2048 to initiate and listen to the communication. On the first start of a Node, it generates a random port number in the given range and adds it to its stored configuration file. This port number is maintained across the life cycle of the Node and is persistent across restarts. However, deleting the configuration file with the port number does not affect the node since it will re-generate another during the next run and using the self-healing strategy of the network, be part of the network.

A node communicates with other nodes over UDP. Hence the MTU of Layer 1 should be lesser than the MTU of an UDP packet over a given network. While it is stated that MTU of UDP is 1472 bytes, it has practically been noticed that some ISPs or Network devices drop packets of this size. Hence, Mariana uses an effective MTU of 858 bytes for payloads. A CRC-32 of 4 bytes is added, making the maximum packet size 862 bytes. A structure of Layer 1 packet is shown in table 1.

Table 1: Physical Layer Packet Structure

Payload	Checksum
n bytes	4 bytes
0 – (n -1)	n – (n+3)

The packets described in Layer 2 or above would be encapsulated in the payload section of the L1 or Layer 1 packet.

Layer 2 (Data link Layer):

The Data link layer defines how each node establishes connection and communicates with the next immediate nodes and forms the basic building blocks of Mariana's network.

A generic layer 2 communication packet contains the Source NAC and a Flag field, along with a payload, total not greater than the MTU of Layer 1.

Table 2 shows a generic layer 2 communication packet:

Table 2: Generic Layer 2 packet

Source NAC	Flag	Payload
16 bytes	1 byte	n bytes
0 – 15	16	17 – (16+n)

Note that the Flag value is protocol dependent. Hence packets from upper layers are added into the Payload section and also uses a significant flag value identifying the protocol. This flag is also referred to as the L2-Flag.

A node discovers other nodes by local discovery, cached discovery or public discovery. The discovery mechanisms will come later. But essentially when a node discovers another directly connected node, it knows it's IP Address and the UDP port it is listening to. It sends a Connect/Request packet to it. A Connect/Request packet is an 817-byte packet with the structure shown in Table 3. It contains the NAC of the Node requesting connection and its public key. It uses L2-Flag value 0.

Table 3: Connect/Request packet

Source NAC	Flag (Value=0)	Public Key
16 bytes	1 byte	800 bytes
0 – 15	16	17 – 816

The Node on receiving it validates the IP address of the sender known against a blacklist and then allows it to connect. The blacklist feature is not yet implemented but can be used to blacklist known malicious nodes from joining the network. It adds the entry to its Content Addressable Memory (CAM) table and Routing Table with current timestamp and sends back a Connect/Accept packet. Table 4 shows the CAM Table Schema.

Table 4: CAM Table Schema

NAC	IP	Port	Timestamp

Table 3 shows the Routing table schema. In Layer 5, when a Node adds another node that is an immediate next node, it sets Hop count to 0 and Next hop NAC to null when adding to Routing table.

Table 5: Routing Table Schema

NAC	Hop count	Next hop NAC	Public Key	Timestamp

The Node sends a Connect/Accept packet to the Node that sent the original Connect/Request packet post addition to CAM table. The Connect/Accept packet is also 817 bytes and its structure is shown in Table 6. It uses L2-Flag value 1.

Table 6: Connect/Accept packet

Source NAC	Flag (Value=1)	Public Key
16 bytes	1 byte	800 bytes
0 – 15	16	17 – 816

If the source IP was found blacklisted, the Node may optionally send a 17 byte Connect/Reject packet. The structure is shown in table 7. It uses L2-Flag value 2.

Table 7: Connect/Reject packet

Source NAC (May be all zeroes or randomized)	Flag (Value=2)
16 bytes	1 byte
0 – 15	16

The originating node, on receiving the Connect/Accept packet adds this node to its CAM table and routing table in a similar manner. When a node receives a Connect/Request or Connect/Accept from a node which is already in its CAM and Routing table, it simply updates the timestamp field in both tables. This method of updating timestamps also doubles as a keep-alive mechanism whether nodes send Connect/Request packets regularly to all directly connected nodes and the connected nodes reply with Connect/Accept. A detailed overview on keepalives will come later.

It is to be noted that if it receives a Connect/Request or Connect/Accept packet from a packet originally in Routing Table but not in CAM table (it signifies it was a relayed node previously, with Hop count greater than 0), it expires the previous route in the Routing table while adding it directly.

A Layer 2 communication is defined as a communication between two directly connected Nodes. It involves the source node fetching the IP Address and Port of the Destination Node by NAC from the CAM table and sending it directly. Layer 2 packets are encapsulated in the payload section of Layer 1 packets.

Layer 3 (Network Layer):

Network Layer defines how any Node in Mariana is able to route data to any other Node in the Mariana Network. A Layer 3 packet contains the Destination NAC field.

Table 8 shows the structure of a Layer 3 packet. Do note that the Layer 3 packet is encapsulated into Layer 2 payload.

Table 8: Transport layer packet.

Destination NAC	Payload
16 bytes	824 bytes
0 – 15	16 - 839

The sending function finds the destination NAC on the routing table. If the Hop count of that entry is zero, it is forwarded to Layer 2 functions. Otherwise, the packet is forwarded to the next hop node. Further, the packet is dropped if a matching entry for destination NAC is not found in the routing table.

A node that receives a packet which is not meant for itself, (or the destination NAC field is not same as its NAC), it forwards the packet towards the destination node as described above.

The routing table needs to be populated throughout the network so that every node is able to reach each other. This is done by routing table announcement. A node regularly announces its routing table to all Nodes in its CAM table (or it's directly connected neighbors). Routing table announcement is done as a payload of Layer 6 packet. It is sent in JSON format and contains the NAC, Hop count, Next hop and public key of all the Nodes in its Routing table. Table 9 shows the fields shared in routing announcement in a tabular format.

Table 9: Routing table announcement fields

NAC	Hop Count	Public key	Next Hop

A node on receiving the routing table announcement, processes it, adding 1 to the hop count value of each received entry. If there are entries whose 'next hop' is the same as the NAC of the receiving node, the entries are discarded to prevent cycling routing loops. Further routes with greater hop-count than the ones currently existing in the routing table are discarded since a superior route already exists. If a route with similar superiority exists, the new route is added to the routing table and the older one is discarded. If routes which is exactly the same as what is present in the routing table, the timestamp is updated. While adding new routes to the routing table from announcements of neighboring nodes, the 'Next hop' field is used for comparison and taking decision for whether to ignore the entry. When adding it, the next-hop field

is changed to the NAC from which the announcement was received. Do note that though the routing table announcement and processing is described here, it is actually an application layer activity. Layer 3 merely takes decision on which packet to forward to which node.

Layer 4 (Transport Layer):

The transport layer defines how payload is fragmented and encapsulated before sending. It ensures all data reaches the destination node without failing. Transport layer has a robust retransmission mechanism to account for packet losses. Transport layer has 3 types of packets namely: payload packet, request retransmission packet and payload acknowledgement packet.

When a node is to send an original payload to another node (by original, it implies the sender is the source node, not a relaying node), it forms the payload packet. It is obvious that the payload does not have a fixed size. Hence, for bigger payloads (almost all), it needs to be fragmented. A random 16-byte session id is generated to identify all fragments of a particular payload. The payload is divided into 800-byte fragments. Say the number of fragments for a particular payload be n, signifying there are n fragments with the sequence numbers starting from 0 to n-1 respectively.

The total number of fragments and the current sequence number is represented as 4-byte integers and a packet is assembled. The packet contains the current sequence number, the total number of sequences, the session identifier and the payload fragment. It uses L2-Flag value 3. The packet structure is shown in table 10.

Table 10: Payload packet structure

Sequence number	Total sequences	Session Identifier	Fragment
4 bytes	4 bytes	16 bytes	800 bytes (max)
0 – 3	4 – 7	8 – 23	24 – 823

The sender node forwards the packet to the Layer 3 sending functions that send the packet to the destination node. The sender node also caches the fragments by the session identifier for a timeout of 180 seconds or 3 minutes for retransmission in case of packet drops.

On the other end, the node receiving the payload assembles the packet, identifying it by the session identifier. It keeps a track of sequence numbers that it may have missed while receiving the packets. The receiving node allows a time of 3 seconds for receiving the packets before requesting a retransmission.

To request a retransmission, it assembles the request retransmission packet for each missed payload packet. The request retransmission packet uses the session identifier and the sequence number of the missed packet. Table 11 shows the structure of the request retransmission packet. In the request retransmission packet, do note that the source NAC is the NAC of the node requesting the retransmission. It uses L2-Flag value 4.

Table 11: Request Retransmission Packet Structure

Session Identifier	Sequence number
16 bytes	4 bytes
0 – 15	16 – 20

The node forwards the packet to the Layer 3 sending functions. The node attempts a total of 100 retransmission requests, each on a timeout of 3 secs, before discarding the entire payload if there are missing fragments.

Once a node receives all fragments of a packet, it sends a payload acknowledgement packet containing the session identifier. On receiving a payload acknowledgement packet, the node drops the fragments of the

session from the cache. Table 12 shows the payload acknowledgement packet structure. It uses L2-Flag value 5.

Table 12: Payload Acknowledgement Packet Structure

Session Identifier
16 bytes
0 – 15

The receiving node, on receiving all fragments corresponding to a session identifier, strips off the headers and assembles the payload. It then processes the payload according to the higher layers.

Layer 5 (Session Layer):

The session layer emphasizes on encrypting the payload before sending it to destination. This is done by encrypting the payload with a randomly generated AES key (with default key size 256 bits) in AES GCM mode. The AES GCM nonce (12 bytes) is appended in front of the ciphertext. This is called the AES ciphertext. The AES Key is now encrypted with Kyber-512 with the public key taken from the routing table corresponding to the destination node.

The length of the encrypted AES Key is represented as a 2-byte integer. To this, the encrypted AES Key and the AES Ciphertext is appended, forming the encrypted payload. Table 13 shows the formation of Encrypted payload.

The encrypted payload is sent to the transport layer functions for reliable transmission to the destination node.

Table 13: Components of Encrypted payload

AES Key Length (value=n)	Encrypted AES Key	Data encrypted with AES
2 bytes	n bytes	Remaining
0 – 1	2 – (n+1)	(n+2) - End

It then extracts the encrypted AES key from the encrypted payload and decrypts it with the Kyber 512 private key. It then proceeds to decrypt the AES encrypted payload with the decrypted AES Key, thus revealing the payload. The payload is processed in the presentation layer (Layer 6) according to the contents of the payload header.

Layer 6 (Presentation Layer):

The presentation layer is responsible for segregating the payload for the different applications operating over Mariana. It achieves this by adding an application specific header in front of the application layer (Layer 7) payload before passing it on to the lower layers for sending. Similarly, on receiving a lower layer packet, the presentation layer strips off the header and passes on the packet to the appropriate application.

Table 14 shows the header in presentation layer.

Table 14: Presentation layer packet structure

Presentation Header	Application payload
No fixed size	No fixed size
0 – n	(n+1) – End

For example, the header ‘routinginfo:’ is used for the Router route announcements as described in Layer 3. A payload containing this header is treated accordingly.

Layer 7 (Application Layer):

The application layer deals with the various applications operating on top of Mariana. One of the core applications that has already been discussed is the routing table announcements. Another core application

is ‘Public relay announcement’, which will be discussed in the next section. Apart from the core applications that build the Mariana network, there are a few user applications which includes ‘Trench Talk’, an anonymous messaging platform; ‘Cargo Ship’, an anonymous file sharing platform, Web server support over Mariana, and raw port proxying over Mariana. These applications will be discussed in a later section.

Encapsulation:

As discussed earlier, packets of each layer are encapsulated inside the payload of lower layers, similar to the OSI model. Table 15 shows the encapsulation in action.

Table 15: Encapsulation

Layer 7									Application Layer Payload	
Layer 6								Presentation Header		
Layer 5						AES Key Length	Encrypted AES KEY	(Encrypted)		
Layer 4				Sequence Number	Total Sequences	Session identifier				
Layer 3			Destination NAC							
Layer 2	Source NAC	Flag								
Layer 1										Checksum
Size	16 bytes	1 byte	16 bytes	4 bytes	4 bytes	16 bytes	2 bytes	Not fixed	Not fixed	Not fixed	4 bytes

A node which is relaying traffic between other nodes may implement only upto Layer 3 for core packet forwarding function, though it needs all the layers for route announcements. However, the relaying node may not implement the various applications. This also makes the protocol version agnostic, implying a relay node running on lower versions would successfully be able to relay traffic for nodes running higher versions.

Node discovery:

A node in Mariana does not rely on a centralized or hard-coded boot-strapping list to connect to the network. This is done so that a centralized list cannot be compromised, thus bringing down the whole network. On the other hand, nodes in Mariana uses 3 separate methods to find other nodes in the network, without relying on a centralized list. The node discovery activities are repeated every 30 seconds and this solves three crucial issues.

- Active mesh formation: Discovering new nodes every thirty seconds allows a node to actively connect to other nodes.

- Keepalive: The node discovery process also acts as a keepalive, thus ensuring a node stays a part of the network.
- NAT Traversal: Regular discovery via fixed port and to fixed ports on the network allows incoming traffic to traverse the NAT (if the node is behind one) by maintaining the port mappings on the NAT device.

Local node discovery

This node discovery strategy aims at discovering other nodes that are operating in the same subnet. It relies on local UDP broadcasts and not on the internet. This ensures that Mariana network is usable without internet connectivity, if only one node in the subnet can connect to any other node outside the subnet via a secondary network interface, a split tunnel VPN or any other way.

A node trying to discover other local nodes lists its own network interfaces and their associated IP addresses and subnet masks. It calculates the broadcast address for each interface. And then floods the broadcast addresses with Connect/Request packets through all ports ranging from 1024 to 2048. Other nodes receiving the broadcast would reply with the Connect/Accept packet, thus letting the node join the network.

Online public node discovery:

A node downloads an open-source list of public relays and sends the Connect/Request packets to them. Though it seems like this is a centralized bootstrap list, it actually is not centrally maintained. It is maintained autonomously by other nodes. This list is often maintained on Github and the Github API credentials for accessing this list is hardcoded into the Mariana application.

The list is maintained using a strategy called ‘Self-discovery’. Each node, makes a call to an API which returns the public IP address of the node. It sends a self-discovery packet to the public IP and the port taken from its own configuration. It generates a random 16-byte temporary state and uses it as the payload of a Layer 2 packet with L2-Flag value 9. The node, on receiving back the same packet matches the temporary state to check whether the packet is valid. The node has successfully discovered that it is either not in a NAT or is behind a Static NAT and is routable via the public Internet. It then promotes itself to a public node and announces its IP and Port via the Github API to the open-source list. It also adds itself to the list of public relays and announces it for Offline public node discovery, as explained in the next section.

Offline public node discovery:

Every node on the Mariana network shares its own list of public relays with the nodes that it is directly connected to. This forms a chain and all nodes receive the list other public nodes via Mariana network. This list is saved persistent on the disk of the node and is used to connect to the network the next time it starts up. It is evident that this strategy is not effective for the first startup of any node, unless the list file is manually copied from another node.

The ‘Public relay announcement’ is a core application as mentioned earlier. It ensures each node is able to join back the network after a shutdown using the persistent list of previously shared nodes.

Applications on Mariana – Taking back the control

Mariana currently has 5 core applications:

- Web server proxy
- Secure Clearnet exit (A part of Web server proxy)
- Mariana Netcat (Port Proxy)
- Trench Talk
- Cargo Ship

And support for a Phonebook application that enables users to use human-readable aliases for Nodes as remembering NACs are not feasible. Mariana primarily gives a browser-based interface for all applications.

Web Server Proxy

Web Server Proxy feature allows one node to host a website over HTTP using any web server of their choice, (including Apache2, NGINX, IIS, etc.) and allow other nodes to use the website. This is achieved without exposing the Web Server to the internet and routing the web requests through the Mariana network. On the client node, the Mariana daemon acts as a proxy server and a browser is opened using that. When the NAC of a node is entered into the URL bar, the daemon uses the 'Host' header to read it. The daemon then reads and serializes the request method, headers, data, parameters and add them to a JSON.

A random 16-byte session id is generated and an application layer payload is generated. The payload contains a flag byte which signifies whether this packet is a request packet or a response packet of the given session. Flag value 0 signifies it is request, while flag value 1 signifies it is a response. The payload is encapsulated in the presentation layer packet with header 'mariana'. The structure of the Web server proxy application payload is shown in table 16.

Table 16: Web Server Proxy Request packet

Session ID	Flag (Value = 0)	Payload
16 bytes	1 Byte	No fixed size
0 – 15	16	17 – End

The web server node when receives this packet deserializes the payload makes the corresponding request to the local web server serving on port 80. This is followed by the daemon on the node serializing the response, including fields like content, status code and headers. This is serialized into a JSON and a similar response packet is formed. Table 17 shows the packet structure for the Web Proxy response.

Table 17: Web Server Proxy Response packet

Session ID	Flag (Value = 1)	Payload
16 bytes	1 Byte	No fixed size
0 – 15	16	17 – End

The daemon in the requesting node now deserializes the response by matching the session ID. It makes a corresponding response and sends it back to the web server. Thus, making it possible to proxy websites hosted on other nodes. Figure 2 shows a demo of web server proxy. Apache server is installed on the remote node and the NAC of that is entered in the URL bar.

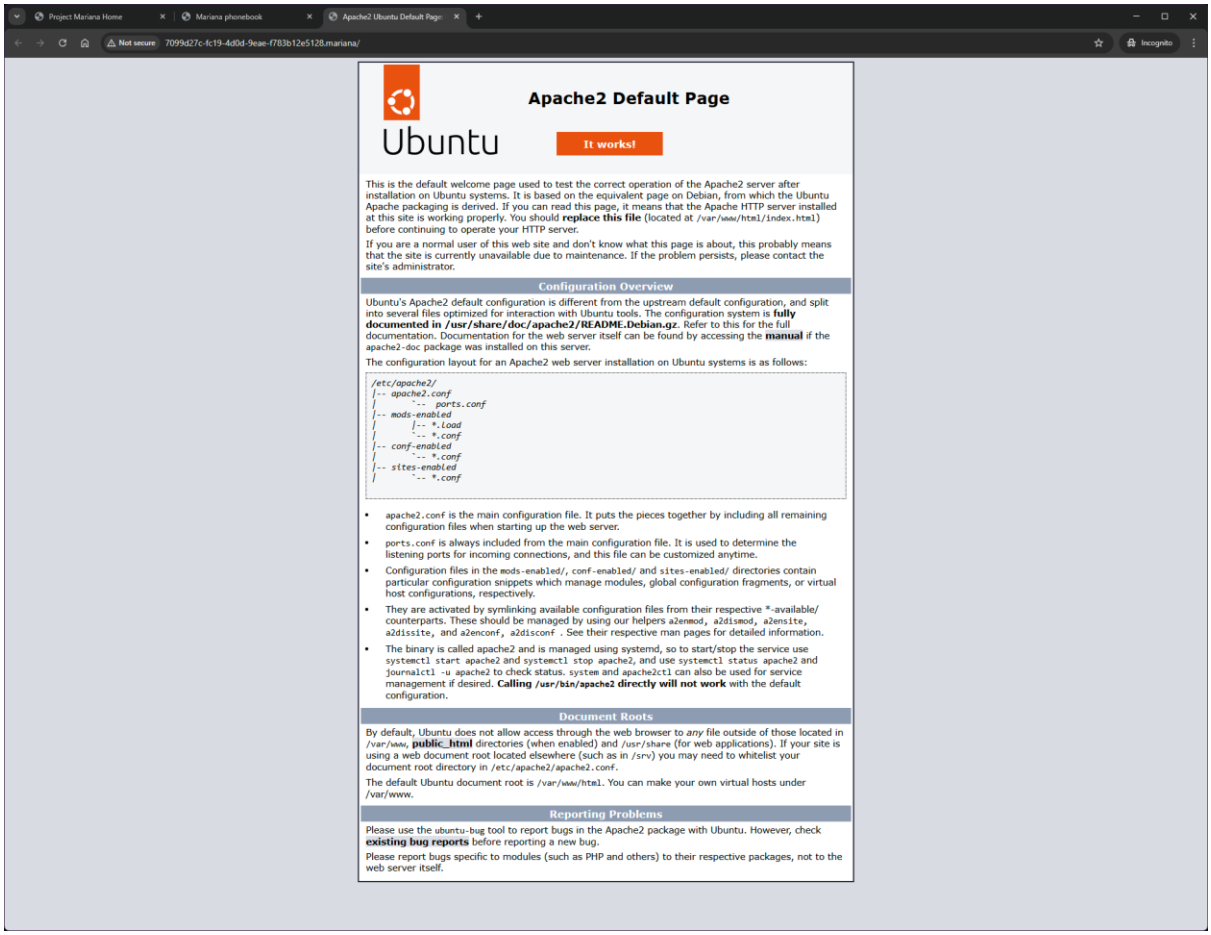

Figure 2: Web Server Proxy demo

Secure Clearnet Exit

As described in Web Server Proxy, the proxy server daemon resolves the Host header of the request into the NAC of the web server node to open the website hosted on the remote node. However, it is indeed possible and very probable that the hosted website uses elements from the Clearnet, including but limited to asset files, CDNs and more. Further, it is possible that the server embeds IP Tracker links in the website which may reveal information about the client node if opened by the client. This is a potential identity-risking vulnerability in the Web Server Proxy model.

This is mitigated by Secure Clearnet Exit. This is achieved by the proxy server resolving non-Mariana websites if present in the Host header by the referring node. For example, let's assume Node A hosts a website with an IP Tracker with a '<script src>' tag in the HTML. Node B opens this website and the browser automatically attempts to resolve the IP Tracker link, this may result in the identity of B being compromised. The proxy server may choose to block all non-Mariana links when on Mariana network. But that would end up legitimate CDN links being blocks. Hence, the feature Secure Clearnet Exit.

In this, when the browser tries to resolve the IP Tracker link, the proxy server detects that it is a non-Mariana link. It inspects the 'Referrer' header of the request; it will reveal that the link comes from Node A. Node B now assembles a packet similar to Web Server Proxy packet and forwards it to Node A. Node A resolves the link to Clearnet and returns the response to Node B. This ensures that if the link was an IP Logger or similar tracker, it would log the IP address of Node A only. Node B will be obfuscated from any attempts to bring it to Clearnet by any other node.

Mariana Netcat

Mariana offers to proxy TCP and UDP ports from a node to any other node. This may often be used by application developers or gamers who need raw socket connectivity between nodes.

One node trying to proxy to a port of another node is achieved via encapsulating the proxied data over Mariana network. The node that is creating the proxy is treated as 'Server node' while the node that is being proxied to is called the 'Client node.' This is often counterintuitive but the name depends on what type of socket Mariana daemon is opening.

For example, let's say on Node A, an application, like netcat is listening on a port (say 6666). Node B tries to connect to the netcat on Node A. Node B sets up the proxy with a source port on local machine (say 1234), destination port as 6666 and destination NAC as the NAC of Node A.

Now Node B can use any application that connects to port 1234 on his own system. For example, 'telnet localhost 1234' achieves is. Now as the Node B enters data on his Telnet client, it is proxied to the netcat listening on Node A. Now, Node A entering data on netcat will be proxied back to Telnet on Node B.

When Node B initiates the proxy, it opens a socket listening on the 'source port'. It is in server mode. It waits for a connection to the port. It creates a socket identifier by concatenating Destination NAC and Destination port. The node also creates a random 16-byte identifier that is used to mask the source port. This is called dummy source port. This socket is kept running on a separate thread.

For ease of explanation, let's assume Telnet is connected to 'Source port' on Node B, while Netcat is listening on 'Destination port' on Node A.

When data is received, it assembles a port proxy packet. The proxy packet contains the fields flag, dummy source port, destination port and payload containing the received data. Table 18 shows the port proxy packet structure. This packet is encapsulated in a layer 6 packet with header 'portproxy'.

Table 18: Port proxy packet structure

Flag (Value 0 – 3)	Source port	Dest port	Payload
1 byte	16-byte	16-byte	No fixed size
0	1 – 16	17 – 32	33 – End

The flag uses 1 byte but only carries 2 bits of information. The first bit signifies whether this is a UDP proxy or TCP proxy. If the first bit is 0, it is an UDP Proxy, while if the first bit is 1, it is a TCP Proxy. The second bit signifies whether this node is a Server mode (value 1) or Client mode (value 0).

When Node A receives this packet, it creates a socket of the type UDP or TCP as specified in the flag, bound to a random ephemeral port. This socket is also identified by a similar socket identifier (by concatenating the NAC of the node that initiated the proxy and the dummy source port). This socket connects to Netcat listening on the value of 'Destination port' from the packet. This socket is also kept running on a separate thread.

If netcat returns data on this socket, it creates a similar port proxy packet, by flipping the value of the source and destination ports, and flipping the Server mode bit. It sends it back to the Node that created the proxy. The receiving node maps back the dummy port to the real port number and sends the data back to Telnet. Figure 3 shows the port proxy and Telnet.

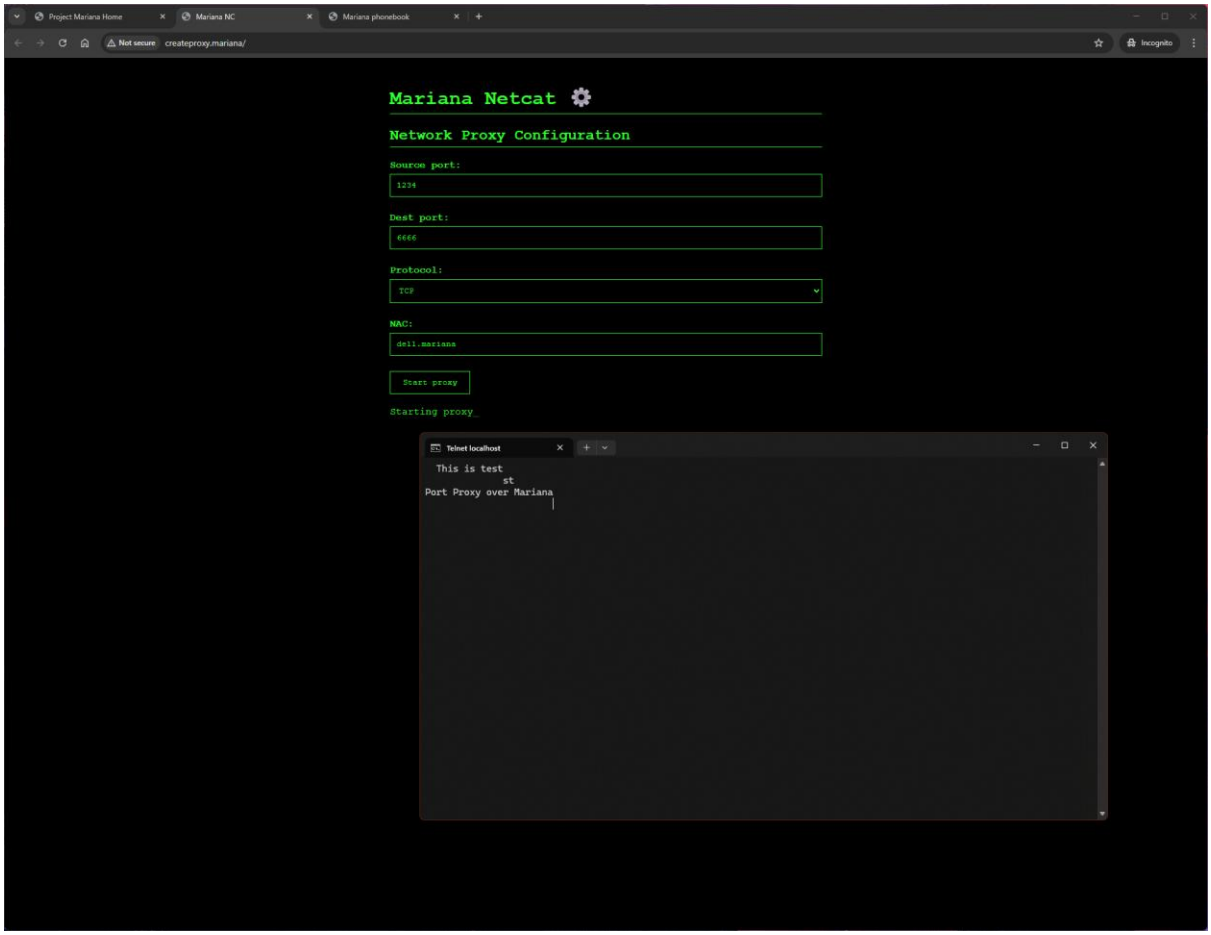

Figure 3: Port Proxy and Telnet

Apart from the web interface, Mariana Netcat also has a command-line interface for advanced users and automation for initiating proxies.

Thus, nodes are able to proxy raw socket connectivity over Mariana network, without revealing any tracking information like IP addresses or even source port.

Trench Talk

Trench Talk is a text messaging application that works over Mariana. A user enters the message to send and the recipient NAC. The message is treated as a Layer 7 packet and is encapsulated in Layer 6 payload with the header ‘trenchtalk’. This is then sent down via the lower layers to the destination node.

The destination node, on receiving this payload renders it into the UI. Trench Talk has a feature of sending same message to multiple nodes by mentioning their NAC as comma separated in the input field. However, in that, each message is treated as an individual message sent to an individual user, not as a group message. Hence, on the receiving end, it would not be possible for a node to know that the message has been sent to other nodes too.

Figure 4 shows a demo of Trench Talk.

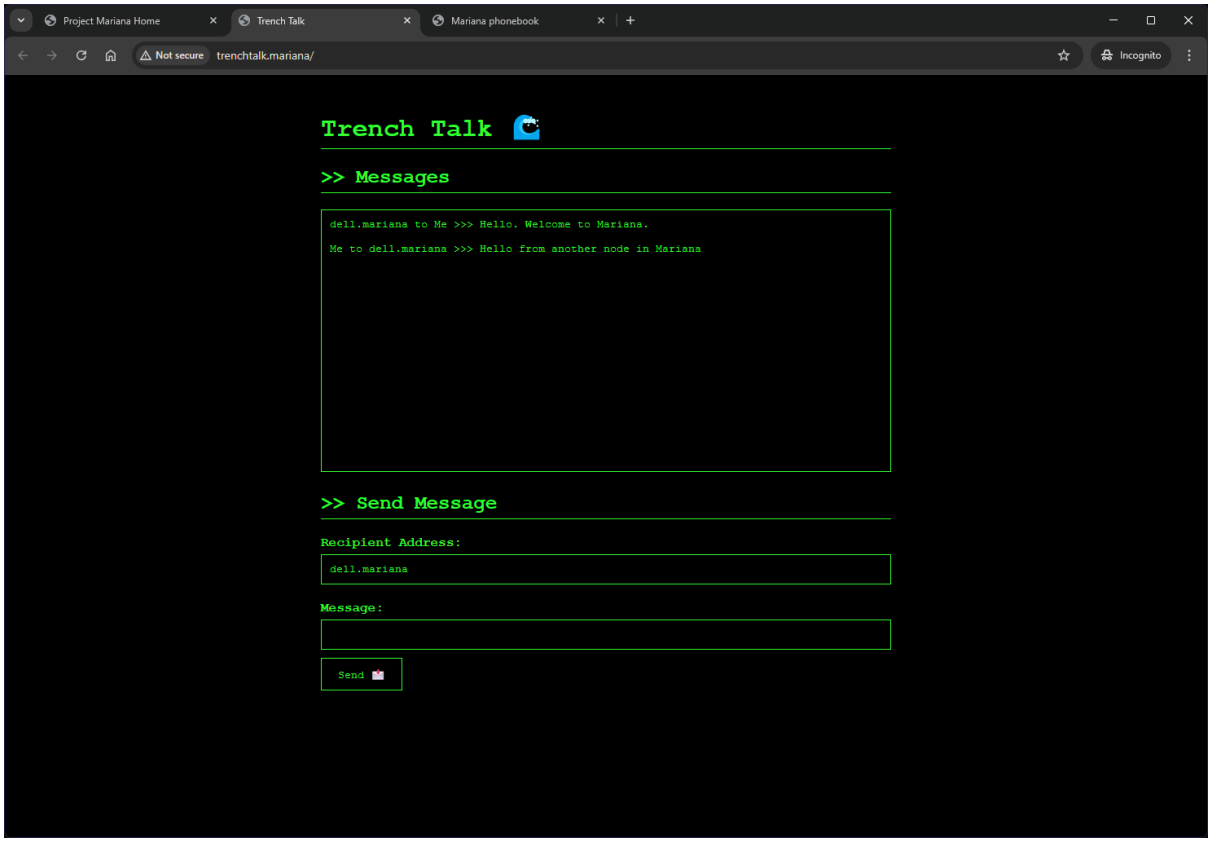

Figure 4: Trench Talk

It is to be noted that Trench Talk does not have a complicated application layer packet since it is raw text, usually ASCII encoded sent over Mariana. This further implies Trench Talk is stateless and the daemon does not store the messages in buffer once it is sent to the web client.

However, if the web client is not active, the daemon keeps storing the received messages in a buffer. However, if a node is not active, messages will not be sent to it. For example, the states shown in Table 19.

Table 19: States of Trench Talk web client and Mariana Daemon

Mariana Daemon	Trench Talk web client	Behavior
Inactive	Inactive	Messages will be dropped
Active	Inactive	Messages will be stored in a buffer and shown when the web client is opened
Active	Active	Messages will be shown in real time
Inactive	Active	Not possible

Cargo Ship

Cargo Ship is a reliable file sharing application built on top of Mariana. Despite the robust retransmission mechanisms in Layer 4, Cargo Ship adds its own retransmission mechanisms to ensure better reliability for heavy files. Unlike layer 4 which requests retransmission for dropped packets, Cargo Ship actively tries to retry sending a packet until an Acknowledgement packet is received.

When a user tries to send a file via Cargo Ship, the daemon fragments the file into fragments of 400 bytes each and orders them into a sending buffer. Note that here the first fragment is numbered 1, unlike layer 4 where the first fragment is numbered 0. A MD5 hash digest of the file is calculated (which is 16 bytes in size).

A metadata application layer packet is generated with the sequence number 0 (and the fragments start from 1). The metadata packet contains the sequence number 0, a flag byte identifying it's sending the data, the file hash, the total number of packets to be sent (including the metadata packet) and the filename.

The metadata packet structure is shown in table 20.

Table 20: Metadata packet structure

Sequence number (Value=0)	Flag (Value=0)	File hash	Total number of packets	File name
4 bytes	1 byte	16 bytes	4 bytes	Not fixed
0 – 3	4	5 – 20	21 – 24	25 – End

The fragments are also assembled into an application layer packet. The packet contains the sequence numbers, flag value of 0 identifying it is the data, the file hash and the fragment. Table 21 shows the packet structure of the data fragments.

Table 21: Data fragment packet

Sequence number	Flag (Value=0)	File hash	Fragment
4 bytes	1 byte	16 bytes	400 bytes (Max)
0 – 3	4	5 – 20	21 - 420

These packets are added into an array. It is identified by an identifier made by concatenating the file hash and the destination node NAC.

A send sequence pointer is initiated with the value of 0. The node repeatedly attempts to send the packet identified by the sequence pointer every 10 seconds. At this stage, it is attempting to send the metadata packet, (identified by sequence 0) to the destination every 10 seconds.

Now, on the receiving node, when it receives the first packet, (the metadata packet), it creates a receive buffer for storing the packets. The buffer is identified by an identifier generated by concatenating the file hash and the NAC of the sending node.

It stores the information from the metadata and is ready to receive all the fragments. It initializes a receive sequence pointer with the value of 0.

On receiving one packet, including the metadata, the receiving node keeps sending an acknowledgement (ACK) packet with the sequence number of that packet every 10 seconds. The ACK packet contains the sequence number from the receive sequence pointer and the file hash. It contains the flag value 1. Table 22 shows the structure of the acknowledgement packet.

Table 22: Cargo Ship Acknowledgement packet structure

Sequence number	Flag (Value=1)	File hash
4 bytes	1 byte	16 bytes
0 – 3	4	5 – 20

The sending node, on receiving the ACK packet reassembles the identifier from the file hash and the NAC of the node from the packet. It checks whether the sequence number of the ACK packet is same as that on the send sequence pointer. If yes, it removes the fragment identified by the sequence number from the memory (since it has already been transferred) and increases the send sequence pointer by 1. The node continues sending the packet pointed by the send sequence pointer as mentioned earlier. After it receives the last acknowledgement packet, it marks the identifier to have completed sending.

Similarly, the receiving end, on receiving a packet with non-zero sequence number, only accepts the packet if the sequence pointer is exactly 1 more than the receive sequence pointer. When it has received all

the fragments, it assembles them and calculates recalculates the hash and matches it. It writes the file to the disk with the same file name as received in the metadata, thus completing the file transfer. Figure 5 shows the packet flow in Cargo Ship file transfer.

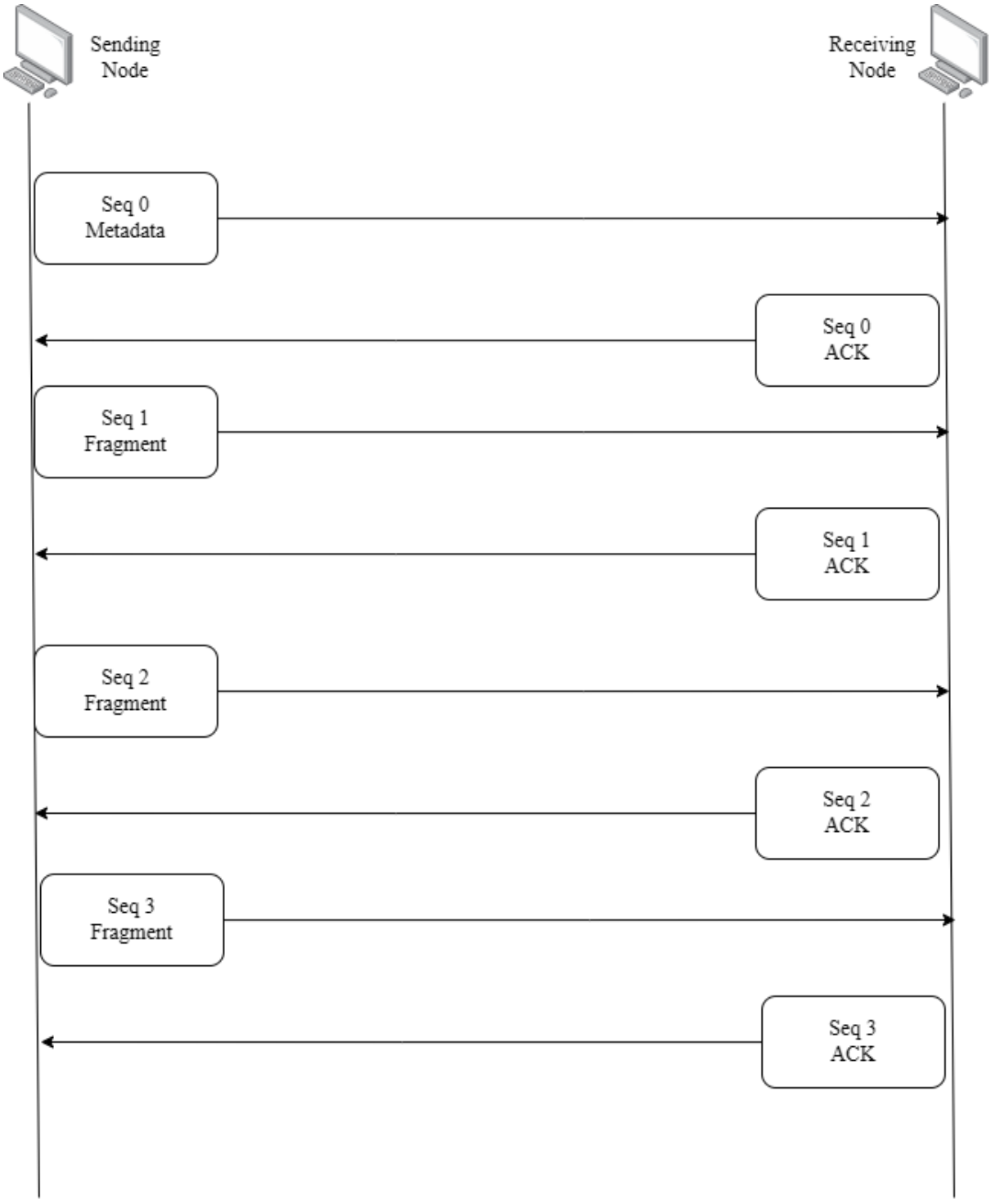

Figure 5: Data flow in Cargo Ship

This acknowledgement packets and automatic retry of packet sending unless an acknowledgement is received increases the reliability of Cargo Ship data transfer. It ensures there is no packet drop during the transfer and even big files can be sent reliably. Further for websites hosted in Mariana, nodes can use Cargo Ship effectively for a file upload or download mechanism in connection with the website.

Figure 6 shows a demo of Cargo Ship in action.

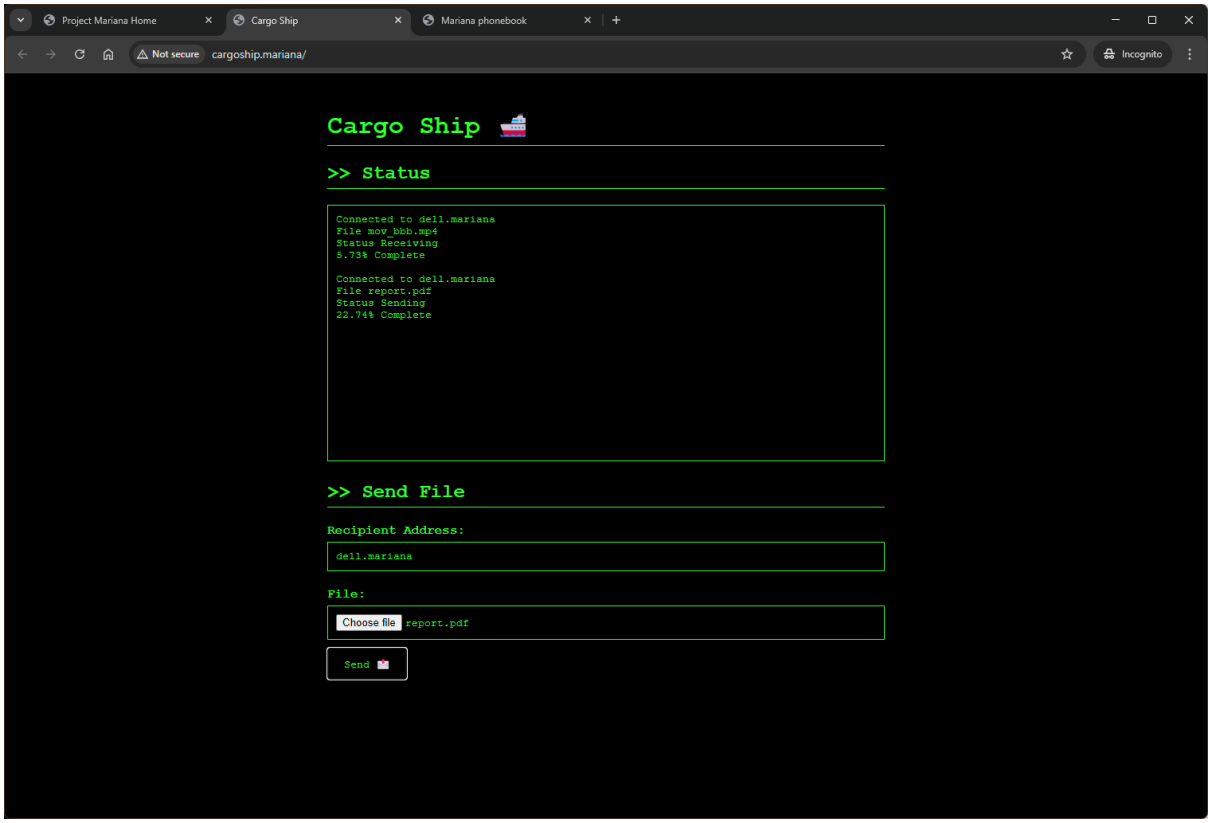

Figure 6: Cargo Ship in File Transfer

Phonebook

It is quite evident that as the network grows, every node will be connected to more and more nodes. It would be practically impossible for a user to keep a note of every node he wants to interact with or to type or copy paste the NACs of every such node. Hence, Mariana features a local phonebook to keep a note of all nodes one would need. It assigns a human-readable alias (in short 'name') to certain NACs that the user wants to remember.

A local phonebook is not synced with any other node. However, the local phonebook acts analogous to a local DNS resolver for the nodes. It resolves the alias of the node to its NAC while sending or receiving data. In figures 3, 4 and 6, it is visible that the alias 'dell.mariana' is used instead of a NAC. This is because the node is saved in the local phonebook by the alias.

A local phonebook is persistent and is stored in the disk. Figure 7 shows the phonebook.

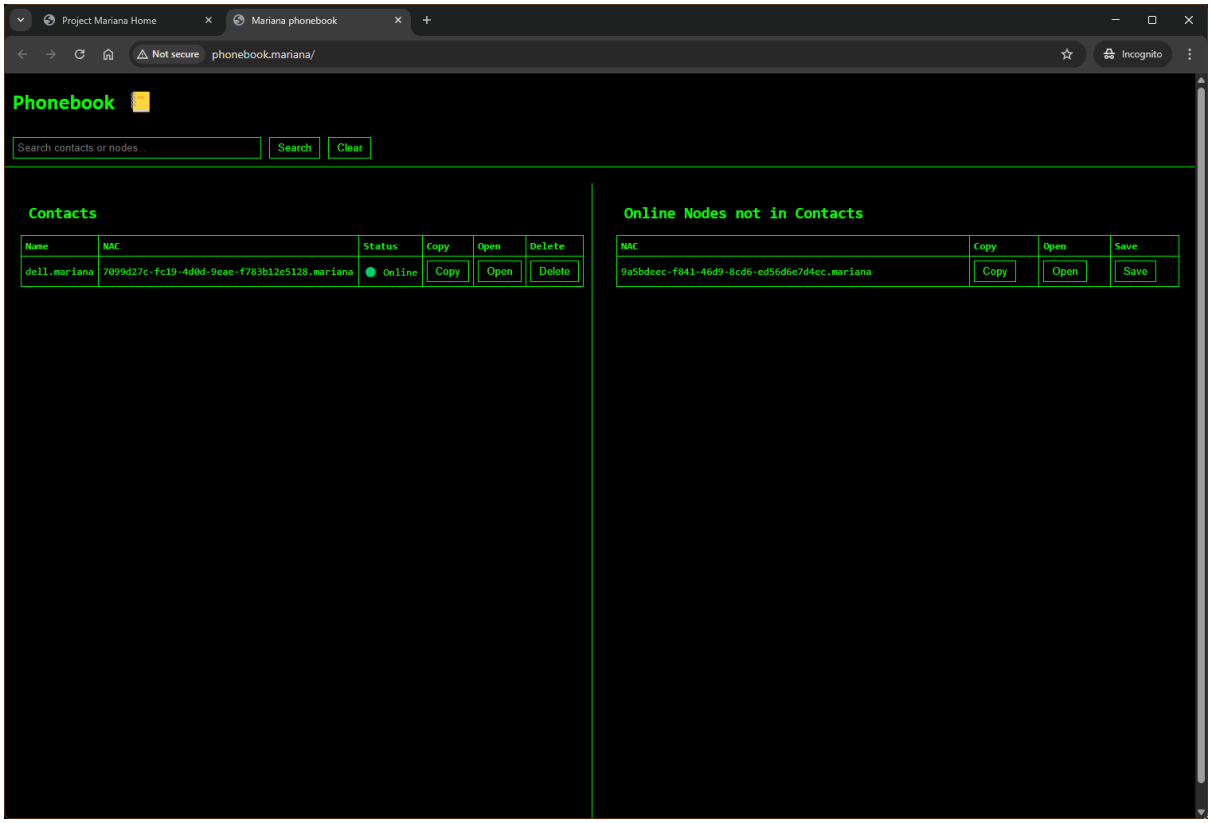

Figure 7: Local phonebook

Mariana Firewall and Security – Trust Nobody

Mariana allows other nodes to access the web servers of other nodes, user server node as proxy to Clearnet, forward ports and sockets, and send files over Cargo Ship. Hence it is imperative that Mariana offers a firewall that a node can use to enable or disable whether they want to serve their local web server over mariana, whether they want other nodes to be able to resolve Clearnet addresses via itself, allow or block ports from being proxied. The security module also features a choice whether a node wishes to receive executable files via Cargo Ship, to prevent the spread of malwares through Cargo Ship.

This firewall sits in between the application layer and presentation layer and silently drops packets that violate the rules.

Figure 8 shows the Mariana Firewall.

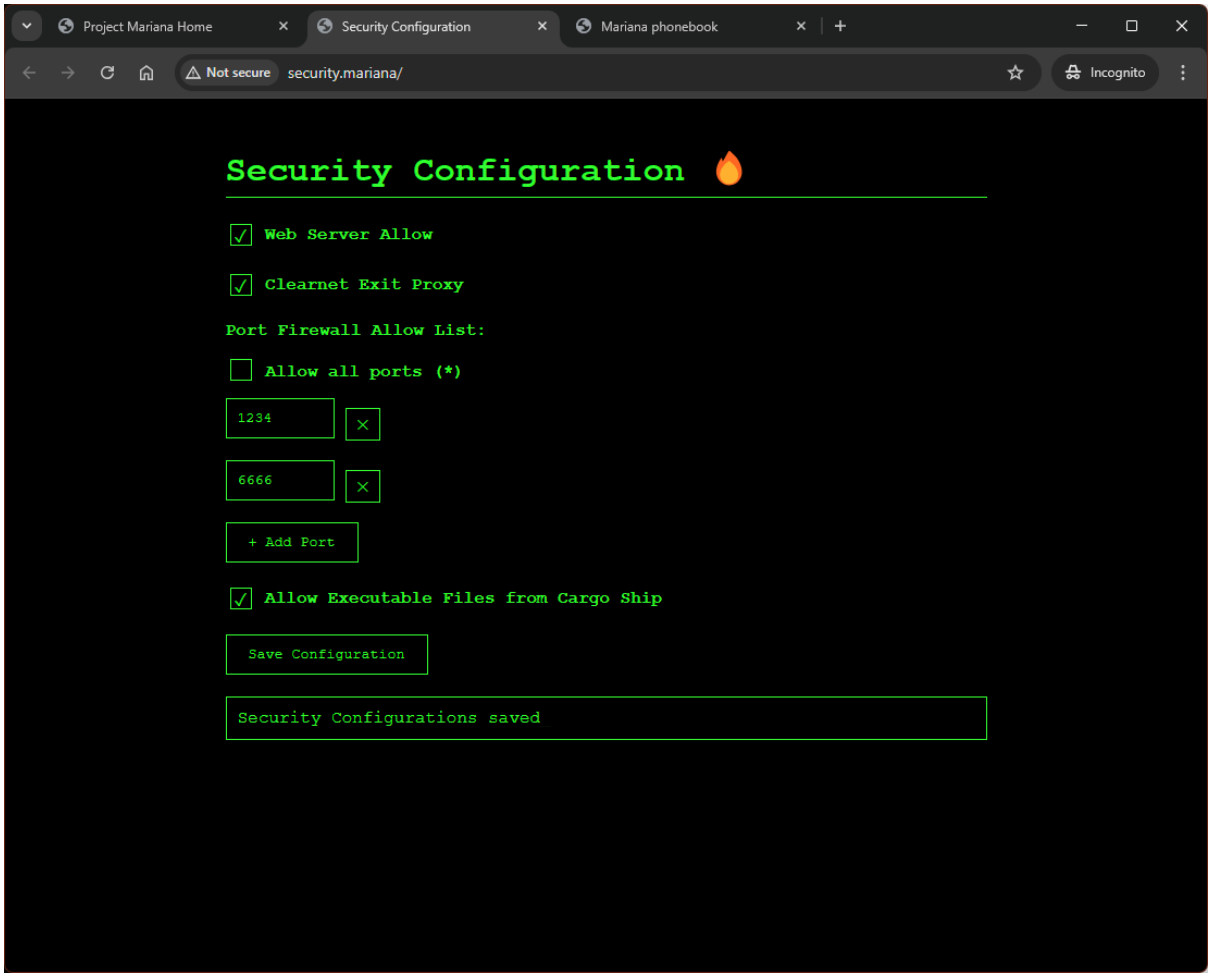

Figure 8: Mariana Firewall

The Bigger Picture – No gatekeepers, no surrender

Mariana's Qubit stands as the gateway to a more secure, fault-tolerant, resilient, decentralized internet. It has probably been clear from the Node Discovery section that a Mariana node does not rely on any centralized bootstrap list, thus allowing any node in any part of the world coming online and be able to join other nodes in local networks, or the internet.

Mariana works even in authoritarian regimes if only one relay node or a bridge interface in an entire country is able to access the outside network (and often the government computers have that provision). One node in a whole country may autonomously bridge a whole country to the Mariana network, which is impossible with other decentralized protocols like Tor, I2P, Freenet.

Mariana would work even in the face of Internet shutdowns. It doesn't matter if the government or ISPs have shut down internet connectivity in a region. You can configure your home router to be able to connect with your neighbor's router (which is usually in a close proximity for the Wi-Fi range) and the chain continues, giving you connectivity to everyone that connects to the network. And if there is a border region where atleast one person is able to the internet or an outside network, Mariana would automatically bridge everyone to the network.

Mariana would stand strong in the face of authoritarian regimes and internet shutdowns. It's self-healing and self-discovery features would allow nodes to join and leave the network seamlessly even if some other nodes are taken down by force.

Mariana has End to End Encryption imbibed in its blood, more appropriately in the Transport Layer (Layer 4), ensuring any application developed and deployed on Mariana would inherently get End to End Encryption without having to configure it manually.

With all the privacy preserving and anonymity features described the specifications, Mariana can boldly stand in face of authoritarian regimes, internet shutdowns, mass surveillance and censorship. Data is the currency of the Internet and surveillance is the business model. The only solution is to refuse to allow surveillance. As Julian Assange said "Censorship is always an effort to hide either a lie or a mistake."

Call to Action – A thousand nodes, a million free minds

Similar to all other decentralized networks, Mariana gets stronger with the number of nodes. The more nodes, especially public nodes connect to the system, the more resilient and faster it would get. And Mariana is not just another decentralized network: Mariana stands as a testament to digital freedom. And if you are using a VPS, cloud system or a machine connected to the public internet, you are strongly urged to set up the Mariana daemon to run on your machine.

Privacy is a right, not a luxury. And the number of public nodes increasing in the network would make privacy more and more accessible. Running a node is a small task for today contributes to a larger, unstoppable mesh for tomorrow. No contribution is too small.

And if you are behind a NAT or unable to run a public node, help the network by sharing about it. Inspire other hackers, activists, and normal users to take back the control of their own data. Word of mouth is one of the biggest decentralized non-digital networks that connects billions of people worldwide. Sharing about Mariana would strengthen building a strong digital decentralized network that transcends geographical borders, regions, regimes and connects the world.

One node can be silenced. A thousand nodes can’t. Be the thousand.

Conclusion – In depth of the trench, Mariana grows

Mariana's Qubit is a fully customized, resilient and decentralized network architecture. It is built group-up without relying on existing libraries. The ability to bypass NATs and Packet drops today, grants the ability to bypass censorship and surveillance for tomorrow.

Mariana is engineered with a focus on the autonomy, survivability and adaptability. It's built for a future, not where a central server is a weakness, but where even thousands of servers are surrounded by strong authoritarian firewalls. Mariana is a statement against mass-surveillance and censorship: the very fundamentals of the freedom of speech. On Mariana, no gatekeeper can shutdown free communication. They can shut down a server; they can shut down a node. But they cannot shut down an idea, and that's what Mariana is.

Mariana is not a small crack in the wall of control – it is the light of free communication that shines through it. And with each new public node, it becomes stronger.

Appendix – Online References – Trust yourself

- Source code: The source code is ready to be deployed on various nodes. https://github.com/AdityaMitra5102/Project-Mariana
- First Demo of Mariana: Demo of Web Proxy and Self-Healing network https://youtu.be/7eabW3VF0N4
- Mariana Netcat demo: Demo of Port Proxying https://youtu.be/UAV4fkOJmLA
- Trench Talk Demo: Demo of Trench Talk encrypted chat https://youtu.be/5Qn08vPhnfY
- Cargo Ship Demo: Demo of file share with Cargo Ship https://youtu.be/UyTipWss4IY