Beta Node Handbook

Instructions for Node operators on how to deploy a Node.

xx network

xx.network

August 19, 2020

Version 1.5

Table of Contents

Changes to This Document

v. 1.0

Released 6/15/2020

v. 1.1

Released 6/17/2020

- Fix IOPS ratings in the Node + Gateway section.
- Add additional information to Elixxir/GPUMaths about a separate repository `gpumathsnative`.
- Correct instructions for compiling the Server binary to include getting the `gpumathsnative` repository.

v. 1.2

Released 6/18/2020

- Add the `-O` flag to the `curl` commands in step 2 in the Node and Gateway Software section.
- Add the Testing section, which contains instructions for how to set up a Go environment and perform tests.
- Fix all `git clone` commands to use the master branch.

v. 1.3

Released 6/25/2020

- Fix units for bandwidth in the Hardware Requirements section from *Mbit* to *Mbps*.
- Update name of repository from `gpumaths` to `gpumathsgo` in the Core section of Software Overview.
- Update the arguments in the Service File Arguments section of Management Tools.
- Add `pyOpenSSL` to the list of Python dependencies that are installed in step 6 of Updating Software and Installing Dependencies and in step 4 of Initial Set Up in the Abridged Node and Gateway Set Up section.
- Fix method to retrieve public IP in the Configuring Node and Gateway section and make clear which IP address to use.
- Add troubleshooting note to Node and Gateway Software.
- Add details to the IP address format in Configuring Node and Gateway.
- Add more detail on how to input the registration code in the Configuring Node and Gateway section and change order of the steps to be more logical.
- Add `-y` flag to `apt install` in the Database section.
- Fix Node and Gateway configuration examples in the Configuring Node and Gateway section.
- Remove the *Manually Downloading Binaries* section because the feature is not available yet.
- Add instructions for setting up Bandwidth Limiting to the Appendix.
- Change *service script* to *service file* in all uses.
- General capitalization, punctuation, grammar, and formatting improvements.

v. 1.4

Released 6/26/2020

- Add `tmpdir` argument to Service File Arguments for both Node and Gateway.
- Change `logLevel` to 0 in the `node.yaml` example in Configuring Node and Gateway.
- Add flags to the `tar` and `curl` commands to enable verbose output.
- Add information for what to do with certificates if running Gateway on a different machine from Node in Generate TLS Credentials.
- Add link to instructions to move database to alternate location to the Database section of Node and Gateway Set Up and in the Database section of Abridged Node and Gateway Set Up.
- Add instructions for ensuring that the user has correct permissions on the `/opt/xxnetwork` directory in Services.
- Modify the `ps` command to return all xx network processes in the Verify the Service has Started section.

v. 1.5

Released 8/19/2020

- Fix flags used in the `tar` command in Node and Gateway Software.
- Update Configuring Node and Gateway to include instructions for modifying the Gateway's advertised IP address.
- Update Node and Gateway Software under Abridged Node and Gateway Set Up to include instructions for modifying the Gateway's advertised IP address.
- Fix broken links.
- Fix output of the `ps -A | grep xxnetwork` command in Verify the Service has Started.
- Add Backup Important Files section to Node and Gateway Set Up.
- Add Backup Important Files section to Abridged Node and Gateway Set Up.

Background

This manual is designed to provide the necessary information to deploy, debug, and operate a Node and Gateway on the xx network BetaNet. A Node in the network will participate in the two fundamental tasks that comprise the xx network: mixing communication through Elixxir's cMix protocol and executing consensus through the Praxxis xx consensus.

To learn more about the xx network, Elixxir, and Praxxis, refer to the following papers.

- xx network Whitepaper
  https://xx.network/xx-whitepaper-v1.3.pdf
- Praxxis Technical Brief
  https://xx.network/praxxis-technical-paper-v1.pdf
- Elixxir Architectural Brief
  https://xx.network/elixxir-architecture-brief-v1.0.pdf
- Academic Paper – cMix: Mixing with Minimal Real-Time Asymmetric Cryptographic Operations
  https://xx.network/cmix-whitepaper.pdf

For more information, further discussion, and additional help with this guide or general participation in the BetaNet, use the following contacts.

- Contact email
  nodes@xx.network
- BetaNet Forum
  https://forum.xx.network/
- Discord
  https://discord.gg/Y8pCkbK
- Telegram
  https://t.me/xxnetwork

Continue reading below for general information about how the software works. To skip to the instructions to set up a Node, skip to Node and Gateway Set Up.

Architecture of the BetaNet

> **NOTE:** Currently, the Permissioning server is used to define the network and orchestrate its activities. This allows the engineering team to tune the network and bring the highly decentralized variant of cMix protocol to a workable state for MainNet. As the BetaNet software matures, xx consensus will take over roles currently held by the Permissioning server.

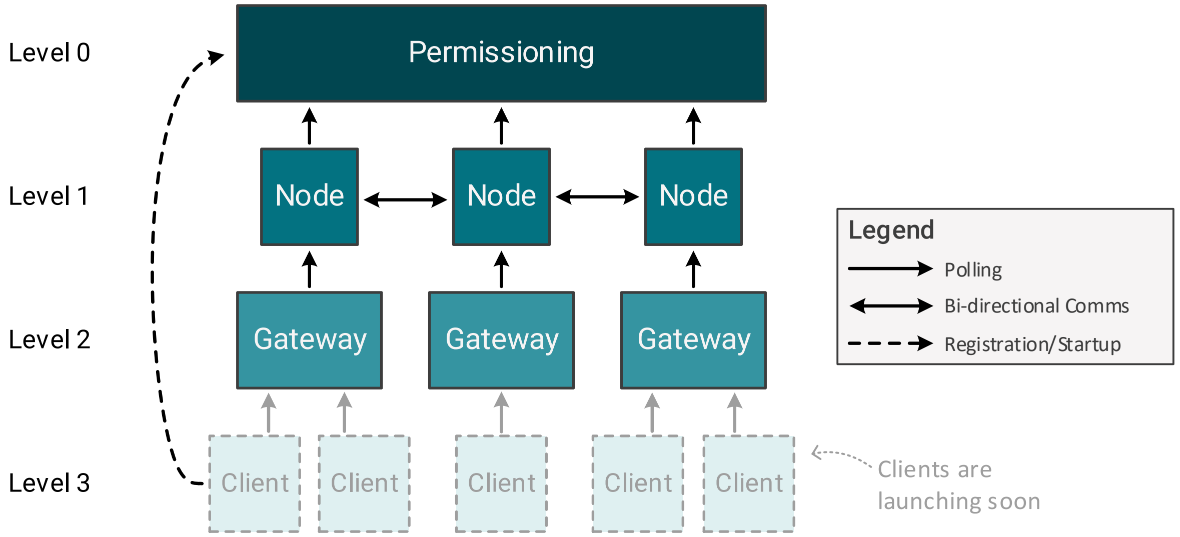
General Architecture

The initial BetaNet is made up of three main entities:

1) ***Nodes:*** The core operators of the network; they execute the cMix protocol.
2) ***Gateways:*** The public facing components of Nodes, one exists per Node. They store received messages and provide public access to data.
3) ***Clients:*** Users use Clients to communicate on the network and are generally deployed on mobile devices.
   ! **NOTE:** Clients will be released within a few months of BetaNet launch.

Hierarchy

BetaNet was built with a tiered Scheduler-Worker design, with all components controlled either directly or recursively by a central *Permissioning* server. Every member polls the entity above them in the hierarchy for information, as shown in Figure 1. Bi-directional communication only exists within the same level.



Clients are launching soon

FIGURE 1: The hierarchy of the BetaNet.

The Network Definition File (NDF) contains all the connection information for the entities in the network and is provided by the Permissioning server through the hierarchy shown. The Nodes provide the hash of their current NDF to Permissioning; if they differ then the updated NDF is provided to the Node. The Gateways poll the Nodes and Clients poll Gateways for the new NDF in the same fashion. However, the NDF provided to the Clients is stripped of Node IP addresses. In the current implementation of Clients, they currently get the NDF from Permissioning. This is expected to change by their release.

Nodes, Gateways, and Clients also receive scheduling instructions from the Permissioning server. These instructions are contained within `RoundInfo` structures, which are both prescriptive and descriptive of changes to rounds, which group a set of Nodes to anonymize communications. A round is created when `RoundInfo` is issued to start a round's precomputation. When the Nodes finish the precomputation, Permissioning issues a new `RoundInfo` that schedules it for real time, which can be delayed depending on the number of queued rounds. A further `RoundInfo` is issued when a round completes or fails.

Network Membership

In the network, all trust is derived from the Permissioning server. Currently, Permissioning is an information collator, but it will be replaced by the xx network consensus mechanism in the near future.

Identities in the network are defined by an asymmetric keypair. Most entities will hold this keypair within a TLS certificate, but some entities will just have a keypair. Nodes and Gateways prove their membership to the network by the inclusion of membership information in a signed NDF.

Initial Run

When first started, a Node will require two TLS certificates and a unique 256-bit randomly generated registration code provided by xx network to join the network. The certificates must be generated using RSA keys and the registration code will be provided in Base64 string format.

On first run, the Node will generate a cryptographic ID and register it with the network, via the Permissioning server. Most of the initial files and generated files are critical and must be preserved by the Node operator. For more information, refer to the Cryptographic and Network Primitives section.

Hardware Requirements

Below are the hardware requirements for running a Node and Gateway. Note that there are additional requirements when running a Node and Gateway on the same machine.

Node

Nodes are high powered machines that use both a CPU and GPU. The software does support full CPU Nodes, but higher core counts are required. As the software becomes more mature and more power is extracted from the GPU, it is likely that the hardware requirements for such Nodes will increase.

CPU	High core count modern CPU Capable of meeting a multithreaded PassMark score of 15,500 or a Cinebench R5 multi score of 1,750. Examples: AMD Ryzen 7 2700x, AMD Ryzen 5 3600x, Intel Core i9-9980HK
GPU	Nvidia Turing, Nvidia GeForce RTX 2070 or greater
RAM	16 GB DDR4 or more ! **Note:** An upgrade path to 32 GB is recommended.
Storage	1 TB High Speed Enterprise NVMe (PCI) SSD Recommended speed: 500,000/500,000 IOPS Recommended reliability: 1.5 million hours MTBF Example: Samsung 970 PRO SSD 1TB – M.2 NVMe ! **Note:** SSD reliability is not as important for the BetaNet when there will not be many transitions. It is likely that lower endurance SSDs will be fine but will need to be replaced when it comes time for MainNet. If an SSD fails, it could take substantial time before a Node is able to get back online. ! **Note:** Full SSD utilization will not occur at launch but will be fully utilized by xx consensus as the BetaNet matures.
Bandwidth	100 Mbps upload / 100 Mbps download

Gateway

Gateways require relatively low powered machines. Every Node must have a Gateway.

CPU	Dual core modern CPU, cloud deployment possible **Example:** AMD Ryzen 3 2200G
GPU	Not required
RAM	2 GB or more
Storage	250 GB Used for database instances. ! **Note:** Full SSD utilization will not occur at launch but will be fully utilized when the Client software is launched (expected in September 2020).
Bandwidth	100 Mbps upload / 100 Mbps download ! **Note:** The Gateway bandwidth requirements are separate from the Node bandwidth requirements.

Node + Gateway

It is possible to run a Gateway off the same machine as a Node; however, it results in a loss of security when they are run off the same IP address and network connection. The Gateway structure makes it more difficult for a Node to be maliciously removed from the network.

The specifications for a machine to run both a Node and Gateway differ slightly and are as follows.

CPU	High core count modern CPU Capable of meeting a multithreaded PassMark score of 15,500 or a Cinebench R5 multi score of 1,750. Examples: AMD Ryzen 7 2700x, AMD Ryzen 5 3600x, Intel Core i9-9980HK
GPU	Nvidia Turing, Nvidia GeForce RTX 2070 or greater
RAM	16 GB DDR4 or more ❗ **NOTE:** An upgrade path to 32 GB is recommended.
Storage 1	1 TB High Speed Enterprise NVMe (PCI) SSD Recommended speed: 500,000/500,000 IOPS Recommended reliability: 1.5 million hours MTBF Example: Samsung 970 PRO SSD 1TB – M.2 NVMe ❗ **NOTE:** SSD reliability is not as important for the BetaNet when there will not be many transitions. It is likely that lower endurance SSDs will be fine but will need to be replaced when it comes time for MainNet. If an SSD fails, it could take substantial time before a Node is able to get back online. ❗ **NOTE:** Full SSD utilization will not occur at launch but will be fully utilized by xx consensus as the BetaNet matures.
Storage 2	240 GB High Speed Enterprise SSD Recommended speed: 100,000/10,000 IOPS Recommended reliability: 1.5 million hours MTBF Example: 883 DCT 240GB ❗ **NOTE:** SSD reliability is not as important for the BetaNet when there will not be many transitions. It is likely that lower endurance SSDs will be fine but will need to be replaced when it comes time for MainNet. ❗ **NOTE:** Full SSD utilization will not occur at launch but will be fully utilized when the Client software is launched (expected in September 2020).
Bandwidth	150 Mbps upload / 200 Mbps download

Turing GPU Requirement

The xx network BetaNet requires a Nvidia Turing GPU due to upgrades to the microarchitecture of the shaders granting an order of magnitude increase in performance for the specific workloads used. The `mul` units have been increased to 32 bits, which greatly increases the speed of the modular exponentiation and modular multiplication which are some of the core cryptographic operations of the cMix protocol.

Software Overview

The xx network BetaNet codebase is combined between xx network's consensus software and Elixxir's cMix protocol. At launch, xx network will be testing and tuning Elixxir software with xx consensus coming online at a later date.

Elixxir software is organized into four groups of repositories as shown in Figure 2: Core, Services, Clients, and Tools. The *Core* libraries handle functionality across Clients and Services, which contain data structures, interfaces, and cryptography common to many of the xx network components. *Services* implements various messenger features, as well as authorizing Clients and Nodes to join the network. *Clients* interact with the network using a shared Client API (Arrow SDK). *Tools* provide several interfaces and utilities to deploy, test, and debug the network.

Most of the code found in the Core and Services is written in the Go programming language. The repositories listed as public below can be found on the public Elixxir GitLab page.

To follow the set up guide, most Node operators will likely elect to download the prepackaged tarballs containing the compiled binaries. To learn more, refer to the Node and Gateway Software section.

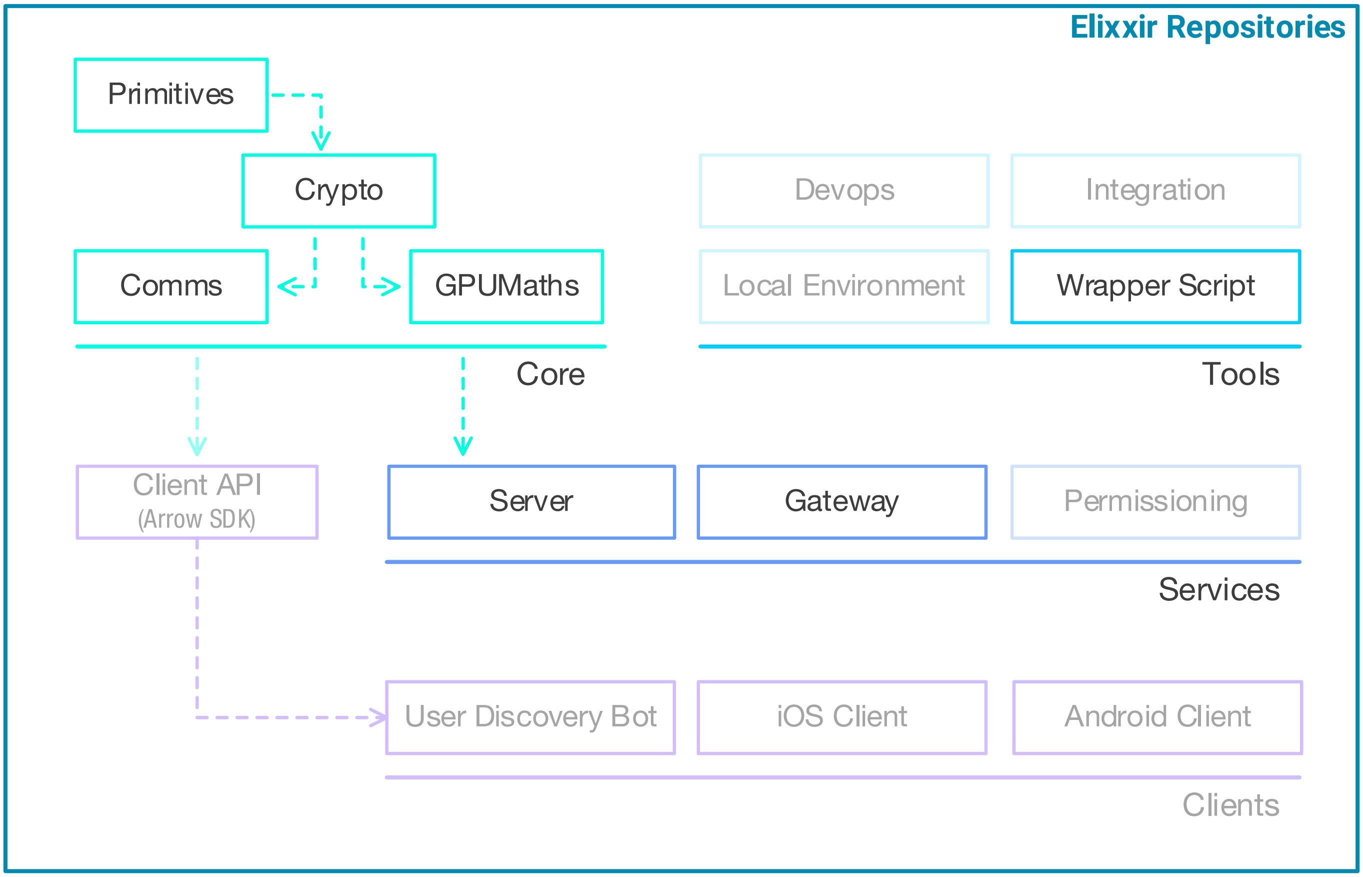


FIGURE 2: Overview of the codebase showing relationships and dependencies. Arrows indicate dependencies. The source code for Primitives, Crypto, Comms, GPUMaths, Server, Gateway, and the Wrapper Script are available on the Elixxir git.

Elixxir Codebase

Core

Elixxir/Primitives (`primitives`) **[PUBLIC]**

The Primitives repository contains the basic data structures and utilities that are used by all repositories. This includes the ID structure for Nodes, Gateways, Users, and other entities on the network, the NDF, and the message payload format. Additional generic structures and utilities are also contained in Primitives, such as file access and rate limiting utilities.

Elixxir/Crypto (`crypto`) [PUBLIC]

The Crypto repository encompasses all of the base cryptographic functionality found in the code base. Relying heavily on Go's *big integer* implementation, Crypto features a cryptographically secure random number generator implementation, libraries for working with large integers in modulo cyclic groups for cMix operations, and basic encrypt/decrypt functionality. Like Primitives, Crypto only contains code that is generic to the larger system. A core approach to this repository is to supply wrappers for operations that may require migration to other implementations in the future.

Elixxir/Comms (`comms`) [PUBLIC]

Comms builds on the generic utilities of xx network Comms to provide specific functionality for Elixxir. It holds a gRPC protocol file along with a thin Client/Server implementation. The repository currently uses TLS certificates with RSA keys for encryption and identification, which will migrate to xx consensus-based quantum secure authenticated channels as development progresses.

Elixxir/GPUMaths (`gpumathsgo`) [PUBLIC]

The GPUMaths repository accelerates the math used by Server, especially for precomputations. It provides a subset of the math implemented in the Crypto repository but accelerated on GPUs. Underlying the acceleration is a publicly available CUDA arbitrary-precision math library, CGBN2. Components of this written in CUDA are in a separate repository `gpumathsnative`.

Services

Elixxir/Server (`server`) [PUBLIC]

The Server repository implements the core cMix functionality and is the software that a Node runs. It performs precomputation and real time computation and processes messages. It receives batches of messages from the Gateways as well as performs network team operations.

Elixxir/Gateway (`gateway`) [PUBLIC]

The Gateway repository contains the API for Clients to interact with the network. Every Node runs a Gateway and the Gateways collect and store messages for Clients. Gateway is designed to be a scalable front-end to the xx network.

Elixxir/Permissioning (`registration`)

Permissioning, also referred to as Registration, manages the NDF for Clients and Servers and schedules cMix rounds within the network. Eventually this functionality will be managed by the distributed xx consensus. For now, this code handles admission, manages which Nodes are part of the network, and orchestrates when Nodes operate.

Clients

Elixxir/Client API (Arrow SDK) (`client`)

All Clients use the Client API to interact and send messages with the cMix network. It uses Go mobile to produce a library compatible with iOS and Android. This library will be released publicly in the near future.

Elixxir/Mobile Clients (`client-ios` and `client-android`)

Currently, clients exist for iOS and Android operating systems. These both use the Go mobile libraries produced in the Client API.

Tools

Elixxir/Wrapper Script (`wrapper`) [PUBLIC]

The Wrapper Script is a Node and Gateway management script that simplifies the running of the xx network software. The script automates the management of the xx network software log files. For easy management or in the event of an error, it starts, stops, and restarts the software without the operator having to revisit the command line. Optionally, it can be set to automatically update the Node and Gateway with the latest xx network binaries and configuration files. To learn more, refer to the Wrapper Script Arguments section.

Elixxir/DevOps `(deployment)`

DevOps is a deployment platform for Microsoft Azure, Google Cloud Platform, and Amazon Web Services (AWS) written in Terraform. DevOps allows for the deployment of test networks, management of deployments of individual Nodes, and the deployment of multi-cloud implementations of xx network.

Elixxir/Integration `(integration)`

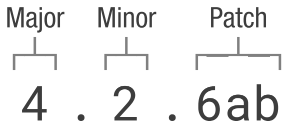
The Integration repository is a series of end-to-end tests designed to test different functionality of the cMix protocol. Several tests focus on different batch sizes with Nodes only. Another test covers all the Client-level interaction within the network.

Elixxir/Local Environment `(LocalEnvironment)`

The Local Environment repository contains a set of scripts designed to allow the testing of the entire platform on a single machine.

Version Scheme

Both Gateway and Node binaries have a version string embedded in them, which consists of a major version, a minor version, and a patch string, separated by a period.



To be able to participate in the network, a Node or Gateway has to have a compatible version with the required version reported by the Permissioning server. For a version to be compatible, the major version must be equal to the required major version and the minor version must be greater than or equal to the required minor version. The patch can be anything, but it will always be present.

Management Tools

Management Scripts Features

The management scripts are designed to make administration of Nodes and Gateways easier and less time consuming as well as provide data back to xx network about the functionality of the Node. Many features that it provides require trusting the xx network. The usage of the management scripts is not required and can be disabled. The following are the primary features of these scripts.

- *Binary restart:* In the event that the binary process stops due to a handled error, the Wrapper Script will restart it. The process can crash because, in the current implementation, some errors are not fully handled when caused by failures of other Nodes on the network. It is highly recommended that this feature is utilized.
- *Binary update:* The Wrapper Script has the ability to accept automatic binary and Wrapper Script updates provided by the xx network. In theory, the xx network can push any code it wants if this feature is activated, but it greatly simplifies the task of running a Node. Included in this feature is the ability to send stop and start commands to the Node. All commands are signed and can be proved to have come from the xx network. Running without this feature is fully supported.
- *Log upload:* This features uploads logs to xx network for debugging. It can be disabled and is not mandatory but is useful to the xx network for development purposes.

The Management Script is two scripts combined, the service file and the actual Wrapper Script. The service file is a systemd script that starts the Wrapper Script and ensures it stays running. The wrapper script is written in Python and does most of the heavy lifting. Both can be found in the Elixxir wrapper repository.

Service File Arguments

The service files maintain the Node and Gateway processes running the background. There is a different service for each. By default, these files (`xxnetwork-node.service` and `xxnetwork-gateway.service`) are located in `/opt/xxnetwork/` and soft linked to `/etc/systemd/system/`. They are the same script just configured differently for Node and Gateway. These scripts call the Wrapper Script with different options. To see the full details of the options, refer to the Wrapper Script Arguments section below.

Node

The following are the available options in the service file configured for Node. The sections in red should never be modified and should maintain the original values as provided. The items in blue can be modified.

```
ExecStart=/bin/bash -c '/opt/xxnetwork/xxnetwork-wrapper.py
--logpath /opt/xxnetwork/node-logs/node.log
--binary /opt/xxnetwork/bin/xxnetwork-node
--s3path server
--s3logbucket alphanet-logs-prod
--s3managementbucket alphanet-management-prod
--s3accesskey KEY
--s3secret SECRET
--s3region us-west-1
--configdir /opt/xxnetwork/
--erroutputpath /opt/xxnetwork/node-logs/node-err.log
--tmpdir /tmp/xxnetwork/node
--idpath /opt/xxnetwork/node-logs/nodeIDF.json >>
/opt/xxnetwork/node-logs/xxnetwork-wrapper.log 2>&1'
```

Gateway

The following are the available options in the service file configured for Gateway. The sections in red should never be modified and should maintain the original values as provided. The items in blue can be modified.

```
ExecStart=/bin/bash -c '/opt/xxnetwork/xxnetwork-wrapper.py
--logpath /opt/xxnetwork/gateway-logs/gateway.log
--binary /opt/xxnetwork/bin/xxnetwork-gateway
--s3path gateway
--s3logbucket alphanet-logs-prod
--s3managementbucket alphanet-management-prod
--s3accesskey KEY
--s3secret SECRET
--s3region us-west-1
--configdir /opt/xxnetwork/
--erroutputpath /opt/xxnetwork/gateway-logs/gateway-err.log
--tmpdir /tmp/xxnetwork/gateway
--idpath /opt/xxnetwork/gateway-logs/gatewayIDF.json >>
/opt/xxnetwork/gateway-logs/xxnetwork-wrapper.log 2>&1'
```

Wrapper Script Arguments

The following arguments should be modified inside the service file described above.

! **NOTE:** The `s3*` options are set per the deployment package sent to you and should not be changed. The other options can be changed per your installation, but it is highly recommended that you go with the defaults sent with the install package.

```
usage: wrapper.py [-h] [-d] [-l LOGPATH] [-i IDPATH] -b BINARY [-c CONFIGDIR]
                  -s S3PATH [-m S3MANAGEMENTBUCKET] --s3logbucket S3LOGBUCKET
                  --s3accesskey S3ACCESSKEY --s3secret S3SECRET --s3region S3REGION
                  [--tmpdir TMPDIR] [--erroutputpath ERROUTPUTPATH]
```

Flag	Default	Required
`-h, --help` Show this help message and exit.	Unset	False
`-d, --disableupdates` Disables automatic updates Used to disable the download and automatic installation of new binaries, the default configuration file, the wrapper script itself, and the security certificate used for verifying wrapper script commands.	Unset	False
`-l LOGPATH, --logpath LOGPATH` The path to store logs, e.g. `/var/log/xxnetwork/server.log`	`/opt/xxnetwork/server-log/node.log`	False
`-i IDPATH, --idpath IDPATH` Node ID path, e.g. `/var/log/xxnetwork/nodeIDF.json` The Node stores the ID file at this path.	`/opt/xxnetwork/server-log/nodeIDF.json`	False
`-b BINARY, --binary BINARY` Path to the binary		True

Flag	Default	Required
`-c CONFIGDIR, --configdir CONFIGDIR` Path to the config directory, e.g. `~/.xxnetwork/` The script searches for the configuration file named *`binary-name`*`.yaml` in the directory passed in through the `--configdir` flag. If it is there, then the path to the file is passed into the binary. If it is not, then no path is passed in and the binary searches for the configuration file in the default directories. For more information, refer to Appendix A: Paths and Files.	`os.path.expanduser("~/.xxnetwork")`	False
`--configoverride CONFIGOVERRIDE` Override for config file path If this flag is set, then the script ignores the `--configdir` flag and uses this file path instead.		False
`-s S3PATH, --s3path S3PATH` Path to the S3 management directory Do not modify.		True
`-m S3MANAGEMENTBUCKET, --s3managementbucket S3MANAGEMENTBUCKET` Path to the S3 management bucket name Do not modify.		False
`--s3logbucket S3LOGBUCKET` S3 log bucket name Do not modify.		True
`--s3accesskey S3ACCESSKEY` S3 access key Do not modify.		True
`--s3secret S3SECRET` S3 access key secret Do not modify.		True
`--s3region S3REGION` S3 region Do not modify.		
`--tmpdir TMPDIR` Directory for temporary files	`/tmp`	False
`--erroutputpath ERROUTPUTPATH` Path to recovered error path The Node stores recoverable errors at this path.	`None`	False

Deployment Schedule

While BetaNet is a functioning network, it is still in development and requires consistent updates. If the Wrapper Script is being used in its default configuration, then Node and Gateway will automatically update.

However, for Node operators with automatic updates turned off, xx network is committed to the following update schedule to allow them ample time to deploy updates.

Scheduled Updates

Every week, there is a single window where major releases and breaking minor releases can be downloaded and installed. The source code will be released on Tuesday before the update goes live on Thursday. Once available on Thursday, any Nodes not up to date cannot participate in the network but will be considered online for the purpose of compensation. Node operators will have until Monday to install the release, after which time they will be considered offline for the purpose of compensation. Refer to the following timeline for specific times and descriptions.

All updates will be posted to the Elixxir GitLab and Node operators will be informed by email. The xx network will strive to post notice of updates before the Tuesday release, if possible.

Tuesday	1:00 pm PT – Release of source code.
Wednesday	
Thursday	1:00 PM PT – Push to Permissioning server, Nodes with out of date code stop functioning. 1:00 PM PT – Push to Nodes with automatic updates enabled. 1:00 PM PT – Grace period for Nodes to update begins.
Friday	
Saturday	
Sunday	
Monday	1:00 PM PT – Grace period for Nodes to update ends, Nodes that have not updated will be considered offline for the purpose of compensation.

Critical Unscheduled Updates

Critical updates may be released with no notice in the event of catastrophic failure or immediate danger to the network.

Nodes will be given 5 business days to update their Node for the purpose of compensation in the event of such an update.

Cryptographic and Network Primitives

! **WARNING:** Two TLS credentials (a certificate and private key pair) and an identification file define the identity of a Node. These files are extremely important to back up and store securely. Loss of these files can make it impossible to reconnect a Node to the network. If following the default guide, these files are named `node_cert.crt`, `node_key.key`, `gateway_cert.crt`, `gateway_key.key`, and `nodeIDF.json`.

There are various cryptographic primitives used in the xx network; some are provided by the xx network and some are generated by the Node operator.

Primitives Provided by xx network

The following primitives are provided to you by xx network, either emailed directly or they are downloaded from public repositories.

Registration Code (sent via email)

The registration code is a 256-bit random number encoded in Base64. It is a single-use code used only on initial registration after which, the Node's ID and TLS certificate can be used to identify it within the network. This will be provided to each Node operator prior to the network going live.

Permissioning TLS Certificate (provided inside the tarball)

A TLS certificate is provided to be used to identify the Permissioning server.

Permissioning Server Address (provided inside the tarball)

The DNS address of the Permissioning server is pre-populated inside the YAML files for Node and Gateway.

Wrapper Script Bucket Address (provided inside the tarball)

The IP address the Wrapper Script uses to update the binaries. It is pre-populated in the Wrapper Script.

Log Bucket Address (provided inside the tarball)

The IP address the Wrapper Script uses to upload logs to. It is pre-populated in the Wrapper Script.

cMix Cyclic Group (provided as part of the NDF on initial registration)

The cMix cyclic group is the modulo cyclic group in which cMix operations are conducted. At launch, this will be defined by a 2048-bit strong and safe prime with a generator of 2.

E2E Cyclic Group (provided as part of the NDF on initial registration)

The E2E cyclic group is the modulo cyclic group in which end to end encryption operations by clients will be conducted. At launch, this will be defined by a 3192-bit strong and safe prime with a generator of 2.

Primitives Created by the Node Operator

The following primitives are generated prior to and during the initial registration of the Node. More details about generation can be found in Generate TLS Credentials.

TLS Credentials

The TLS credentials are generated as X.509, SHA-256, RSA 4096-bits, and are recommended to last 730 days (2 years). The generated credentials include:

- Node TLS certificate
- Node private key
- Gateway TLS certificate
- Gateway private key

ID File (IDF)

IDs are 264-bits and are generated from the hash (BLAKE2b) of a salt and the RSA public key from the Node's TLS certificate.

```
ID = Hash(Node_PubKey, Salt) | TYPE
```

Due to the construction of the ID, the ownership of the ID can be proved under RSA's cryptographic assumptions. Due to the hash, IDs are unpredictably generated in a large sparse space so it is overwhelmingly improbable that independently generated IDs will ever collide, so no central checking of ID generation is necessary.

The last byte of the ID is a type byte which describes what type of entity it belongs to. The options for type are:

Type	Hex	Note
`Generic`	0x00	Used for one-off entities such as the Permissioning server, the notifications server, and others
`Gateway`	0x01	The Gateways in the network
`Node`	0x02	The Nodes in the network
`User`	0x03	Unique client ID

On first run, the Node automatically generates the ID and saves it to a JSON file with the following structure.

```
{
  "id": "1Mo9im6HHpoTDt/lTYjSkWV7dAD0Eh+18xUFfxlm4I4C",

  "type": "node",

  "salt": [119, 234, 246, 209, 166, 166, 72, 17, 253, 196,
           172, 187, 230, 2, 132, 137, 49, 219, 142, 58,
           82, 169, 60, 230, 112, 17, 30, 112, 30, 68, 217,
           92],
  "idBytes": [212, 202, 61, 138, 110, 135, 30, 154, 19,
              14, 223, 229, 77, 136, 210, 145, 101, 123,
              116, 0, 244, 18, 31, 181, 243, 21, 5, 127,
              25, 102, 224, 142, 2],
}
```

The ID of the Node or Gateway, presented as a Base64 string.

The type of ID is shown here for readability.

The salt used to generate the ID.

The ID of the Node or Gateway, presented as a byte slice.

Node and Gateway Set Up

Note: What follows is a verbose description of how to deploy a Node and Gateway. The xx network has Node operators with a wide variety of knowledge and experience levels and it is important to cater to all potential Node operators. An abridged version for more experienced operators can be found in the Abridged Node and Gateway Set Up section.

Below you will find the instructions for setting up a Node and Gateway on the xx network for the first time. These instructions detail how to install both a Node and Gateway on separate machines as well as installing them together on the same machine.

The installation instructions assume that your machine has an active internet connection and an empty storage drive or a drive that can be written over with enough space as outlined in the Hardware Requirements section.

Overview

Setting up an xx Node is not particularly difficult, but due to the number of components, can take up to 2 hours. The steps for setting up a Node and Gateway are almost identical and the instructions can be followed for both. The general steps for setting up a Node are:

1. Build the machine
2. Install the operating system
3. Synchronize the system clock (NTP)
4. Install the GPU drivers (if setting up a Node with GPU)
5. Download and extract the Node and/or Gateway software
6. Generate TLS credentials
7. Install and configure the database (if setting up a Node)
8. Set up and start the services

Some Tips for Inexperienced Users

If this is your first time using a command line interface or you do not remember how to use it, the following are some tips to make using the interface a little easier.

- In this document, anytime code is presented in a `black box with monospaced font` it means that it is command line input or output. Commands prefixed by a `$` are commands that you should enter into your command prompt (do not include `$` in the command). Any lines without that prefix are output from the system.

- The `sudo` command is often prepended to commands found in these instructions. This enables commands to be run with elevated privileges. When used, the system will ask for your password to continue running the command.

```
$ sudo nano gateway.yaml
[sudo] password for user:
```

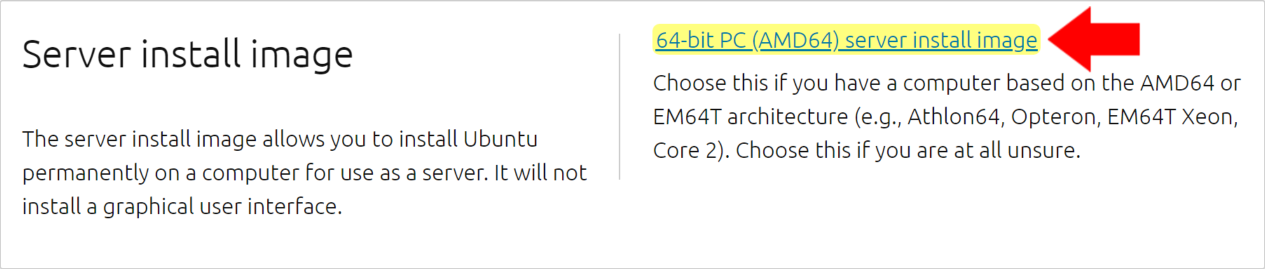
- Whenever the system asks for a password to continue, no characters will appear when typing, but type in your password and press Enter and it will work.

- When typing a command or path, use the Tab key to auto complete a partially written statement.

Installing the Operating System

The xx network software has been tested only on Ubuntu Server 18.04. We cannot guarantee support if a different operating system version is used, although no decisions have been made to specifically preclude any operating systems.

1. Download Ubuntu Server install image from the Official Ubuntu website.



Server install image

The server install image allows you to install Ubuntu permanently on a computer for use as a server. It will not install a graphical user interface.

64-bit PC (AMD64) server install image

Choose this if you have a computer based on the AMD64 or EM64T architecture (e.g., Athlon64, Opteron, EM64T Xeon, Core 2). Choose this if you are at all unsure.

2. Next, a bootable disk with Linux needs to be created. This can be done by writing it to a DVD or more commonly, a flash drive. Follow one of the following tutorials on how to do so depending on your current operating system and chosen media.

   - Create a bootable USB stick on Ubuntu
   - Create a bootable USB stick on Windows
   - Create a bootable USB stick on macOS
   - How to burn a DVD on Ubuntu
   - How to burn a DVD on Windows
   - How to burn a DVD on macOS

3. Once your flash drive or DVD are ready, follow the Tutorial on Installing Ubuntu Server.

   a. In step 6, make sure you select the first option `Install Ubuntu`.

   b. In step 7, make sure to configure your internet connection and get an IP address.

   c. In step 8, ensure that you select `Use an Entire Disk`.

   d. In step 12, pick a server name that does not have any personal identifying information and select a strong password.

   ⚠ **WARNING:** Create a strong but memorable password. It is recommended that it is longer than 12 characters. Store this password in a safe and secure location. Never share this password with anyone.

4. Make sure the machine has turned back on and login using the credentials created in the previous step. Sometimes extra text will be printed and you will not be able to see the login prompt. The prompt should still be there; just type your username and press Enter to continue.

Check Internet Connection

The rest of the instructions require internet access. Follow these steps to ensure the machine is connected.

1. Check your current local connection status and local IP address.

```
$ ip addr
```

This should result in a similar output to below. The machine should have a valid local IP address.

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
default qlen 1000
    link/ether 00:0c:29:f9:b7:83 brd ff:ff:ff:ff:ff:ff
    inet 192.168.127.138/24 brd 192.168.127.255 scope global dynamic enp1s0
       valid_lft 1574sec preferred_lft 1574sec
```

Your local IP address.

```
    inet6 fe80::20c:29ff:fef9:b783/64 scope link
       valid_lft forever preferred_lft forever
```

2. Finally, check that the machine has access to the internet by pinging another server. We chose to use `xx.network`, but you can use any domain.

   ```
   $ ping -c 4 xx.network
   ```

   This should result in similar output to the following example.

   ```
   PING xx.network (104.26.11.97) 56(84) bytes of data.
   64 bytes from 104.26.11.97: icmp_seq=1 ttl=54 time=18.5 ms
   64 bytes from 104.26.11.97: icmp_seq=2 ttl=54 time=16.3 ms
   64 bytes from 104.26.11.97: icmp_seq=3 ttl=54 time=18.8 ms
   64 bytes from 104.26.11.97: icmp_seq=4 ttl=54 time=18.5 ms

   --- xx.network ping statistics ---
   4 packets transmitted, 4 received, 0% packet loss, time 3003ms
   rtt min/avg/max/mdev = 16.323/18.066/18.854/1.018 ms
   ```

Updating Software and Installing Dependencies

To ensure all the software works correctly, the system needs to be updated. In addition, an additional dependency for the Wrapper Script to function.

1. Before continuing, check for updates. This will print many lines about what software is being checked.

   ```
   $ sudo apt -y update
   ```

2. Once the check is done, install the updates.

   ```
   $ sudo apt -y upgrade
   ```

3. Reboot the machine to ensure all updates are installed fully. Once the machine starts up, log back in.

   ```
   $ sudo reboot now
   ```

4. Install the Python package installer.

   ```
   $ sudo apt install -y python3-pip
   ```

5. Update the package installer.

   ```
   $ sudo pip3 install -U pip
   ```

6. Install the `boto3` and `pyOpenSSL` dependencies. The Wrapper Script uses the first package to read commands and send logs to xx network through AWS and the second is used to authenticate them.

   ```
   $ pip3 install --user -U boto3 pyOpenSSL
   ```

Setting Up UFW

Uncomplicated Firewall (UFW) is the default firewall configuration tool for Ubuntu. UFW should already be installed; the following instructions will describe how to enable it and allow the correct ports to be open.

1. First, ensure that UFW is disabled.

   ```
   $ sudo ufw disable
   ```

   This should result in the following output.

   ```
   Firewall stopped and disabled on system startup
   ```

2. Allow the port that the Node will use to communicate on over TCP.

   ```
   $ sudo ufw allow 11420/tcp
   Rules updated
   Rules updated (v6)
   ```

   ! **Note:** Port `11420` is the default port in the provided Node configuration file. A different port may be used, but it must be configured in `node.yaml`, which is downloaded in a future step.

3. Allow the port that the Gateway will use to communicate over TCP.

   ```
   $ sudo ufw allow 22840/tcp
   Rules updated
   Rules updated (v6)
   ```

   ! **Note:** Port `22840` is the default port in the provided Gateway configuration file. A different port may be used, but it must be configured in `gateway.yaml`, which is downloaded in a future step.

4. If you want to manage your node remotely, you may want to allow port `22` over TCP for SSH. SSH should only be enabled with key authentication as. Improper SSH setup can allow unwanted access so take care when setting up remote access. If you do not want to use SSH or do not know how to, then skip this step.

   ```
   $ sudo ufw allow 22/tcp
   Rules updated
   Rules updated (v6)
   ```

5. Finally, enable UFW.

   ```
   $ sudo ufw enable
   ```

   If you are connected over SSH, you may be prompted to continue, press Y. Note that you may be disconnected when doing so.

   ```
   Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
   Firewall is active and enabled on system startup
   ```

Clock Synchronization (NTP)

Commands from the Permissioning server are time-stamped and a synchronized clock is important for a Node to properly interpret them. To do so, NTP (Network Time Protocol) must be set up and synchronized.

In Ubuntu Server 18.04, this is done through `timedatectl`. In most installations, it is already running and Node operators only need to check that it is correctly configured. However, the process for other operating systems may be different and it will need to be enabled.

1. Check if the time synchronization service is running and that the clock is synchronized by entering the following command.

   ```
   $ timedatectl status
   ```

   This will print the following output.

   ```
                         Local time: Sat 2020-06-13 20:43:41 UTC
                     Universal time: Sat 2020-06-13 20:43:41 UTC
                           RTC time: Sat 2020-06-13 20:43:41
                          Time zone: Etc/UTC (UTC, +0000)
          System clock synchronized: yes
   systemd-timesyncd.service active: yes
                    RTC in local TZ: no
   ```

a. If `System clock synchronized` is set to `yes` and `systemd-timesyncd.service active` is set to `yes`, then no further action is needed. Skip to the next section Configuring Local Network (Port Forwarding). Otherwise, continue to the next step.

```
       System clock synchronized: yes
systemd-timesyncd.service active: yes
```

b. If `System clock synchronized` is set to `no` and `systemd-timesyncd.service active` is set to `yes`, then the service has had insufficient time to synchronize the clock. Skip to the next section Configuring Local Network (Port Forwarding) but make sure to check that the clock is synchronized before starting the Node or Gateway software. Otherwise, continue to the next step.

```
       System clock synchronized: no
systemd-timesyncd.service active: yes
```

❗ **NOTE:** Ensure the time is synchronized before starting the Node software in the Services section.

c. If `System clock synchronized` is set to `no` and `systemd-timesyncd.service active` is set to `no`, then the service must be manually started in the next step.

```
       System clock synchronized: no
systemd-timesyncd.service active: no
```

2. Enter in the following command to get a list of time zones.

```
$ timedatectl list-timezones
```

This will print a list of time zones. Use the up key ↑ and down key ↓ to navigate the list and find the time zone that the machine is in. Once found, make a note of the time zone, and press Q to exit.

```
America/Kentucky/Louisville
America/Kentucky/Monticello
America/Kralendijk
America/La_Paz
America/Lima
America/Los_Angeles
America/Lower_Princes
America/Maceio
America/Managua
America/Manaus
lines 86-137
```

Select the time zone and make note of it.

3. Once the correct time zone for the machine is found, use the following command to set it.

```
$ sudo timedatectl set-timezone [your select time zone]
```

4. Using the date command, ensure that the correct time zone was selected. If the printed time is incorrect, return to step 2 to select a new time zone.

```
$ date
  Sat Jun 13 13:45:31 PDT 2020
```

5. Begin the clock synchronization service by entering the following command.

```
$ sudo timedatectl set-ntp on
```

6. Ensure that the service is running by calling `timedatectl` again.

```
$ timedatectl status
```

If `systemd-timesyncd.service active` is set to `yes`, then it has been successful. If `System clock synchronized` is set to `yes`, then this section is done. If it is set to `no`, then it may take up to 30 minutes for the clock to synchronize. You can continue to the following section, but make sure to check that the clock is synchronized before starting the Node or Gateway software.

```
                      Local time: Sat 2020-06-13 20:43:41 UTC
                  Universal time: Sat 2020-06-13 20:43:41 UTC
                        RTC time: Sat 2020-06-13 20:43:41
                       Time zone: Etc/UTC (UTC, +0000)
       System clock synchronized: no
systemd-timesyncd.service active: yes
                 RTC in local TZ: no
```

Configuring Local Network (Port Forwarding)

To ensure that the machine can be accessed from outside the local network, the network's router or gateway must be configured to allow external access to the machine on ports configured above. Three main pieces of information are needed for this part: (1) the port numbers to forward (the defaults are `11420` and `22840`), (2) the protocol to use (TCP), and (3) the local IP address of the machine, which is retrieved below.

1. Get the local IP address of the machine. If Node and Gateway are being run off separate machines on the same network, then run this on both machines.

   ```
   $ hostname -I
   ```

   The local IP address will be printed; it will be in the form of `0.0.0.0`. Make sure to make note of this for the later steps.

   ```
   [your internal IP address]  ←————————  Make note of this.
   ```

   ! **Note:** If the machine has multiple network interfaces or an IPv6 address, they will also appear in this list. Ensure that only the correct internal IPv4 address is used.

! **Note:** The following section describes how to configure the networking equipment on your network. Because of the varying type of equipment configurations, these instructions are generic and may not be accurate for your hardware. Please refer to the manufacturer's instructions for more detailed and accurate information. Configuration of the network will most likely occur from a different machine on the network.

2. To access the router to configure, its IP address is needed.

   ```
   $ ip r | grep default
   ```

   This will output the following line, where the first address printed is the router IP address.

   ```
   default via 192.168.1.1 dev enp1s0 proto dhcp src 192.168.1.37 metric 100
   ```

3. Go to this IP address in a browser (on a different machine) and login using the router credentials. These credentials are either set up by the network administrator or are the default credentials located on the router or found online.

4. Locate the *port forwarding* options (occasionally called *virtual server*). These options are sometimes found under the advanced section.

   ! **Note:** Before forwarding the port found above, you may want to provide a static local IP address to the machine. However, that is outside the scope of these instructions.

5. Add the new ports to forward for the Node and Gateway. For each, create a new entry and enter the IP address found in step 1, set the port to the chosen ports (by default it is `11420` for Node and `22840` for Gateway), and select the TCP protocol. Make sure to save or apply the changes.

GPU Drivers

The Node software requires a Nvidia RTX graphic processor and the installation of the Nvidia CUDA Toolkit (version 10.2). This toolkit installs the GPU drivers and the runtime libraries for CUDA as well as various tools such as a profiler and debugger, which help debug GPU related bugs.

! **NOTE:** If you currently configuring the Gateway or a Node with only CPU usage, then skip this section and proceed to Node and Gateway Software.

To learn more, refer to the CUDA Toolkit download page.

1. Nvidia provides a list of commands to run that install all the necessary software onto the machine. Simply run all the commands in the provided order. Note that this process can be slow.

```
$ wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64/
  cuda-ubuntu1804.pin
$ sudo mv cuda-ubuntu1804.pin /etc/apt/preferences.d/cuda-repository-pin-600
$ wget http://developer.download.nvidia.com/compute/cuda/10.2/Prod/local_installers/
  cuda-repo-ubuntu1804-10-2-local-10.2.89-440.33.01_1.0-1_amd64.deb
$ sudo dpkg -i cuda-repo-ubuntu1804-10-2-local-10.2.89-440.33.01_1.0-1_amd64.deb
$ sudo apt-key add /var/cuda-repo-10-2-local-10.2.89-440.33.01/7fa2af80.pub
$ sudo apt-get update
$ sudo apt-get -y install cuda
```

! **NOTE:** The multi-line commands cannot be directly copied because there will be extra whitespace. Either type the command in full, modify the line prior to pasting in the console, or get the original commands from the CUDA Toolkit download page for Ubuntu.

2. Once the installation is complete, reboot the system.

```
$ sudo reboot now
```

3. Once the system reboots, log back into the computer.

Verifying the Driver Installation

1. Check that the system has claimed the device.

```
$ sudo lshw -c display
```

This should result in a similar output to the following.

```
  *-display
       description: VGA compatible controller
       product: NVIDIA Corporation
       vendor: NVIDIA Corporation
       physical id: 0
       bus info: pci@0000:06:00.0
       version: a1
       width: 64 bits
       clock: 33MHz
       capabilities: pm msi pciexpress vga_controller bus_master cap_list rom
       configuration: driver=nvidia latency=0
       resources: irq:56 memory:f6000000-f6ffffff memory:e0000000-efffffff
       memory:f0000000-f1ffffff ioport:e000(size=128) memory:c0000-dffff
```

2. Next, check the driver and state information.

```
$ nvidia-smi
```

This should result in a similar output to the following.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.82       Driver Version: 440.82       CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce RTX 2070    Off  | 00000000:06:00.0 Off |                  N/A |
| 41%   51C    P0    28W / 175W |      0MiB /  7979MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Node and Gateway Software

1. Change the working directory to `/opt/`.

```
$ cd /opt/
```

2. Download the tarball for Node and/or Gateway using the `curl` command using the following URLs. The newest versions can also be found on the xx network website.

For Node:

```
$ sudo curl -L -O https://xx.network/codebase/node.tar.gz
```

For Gateway:

```
$ sudo curl -L -O https://xx.network/codebase/gateway.tar.gz
```

3. Next, the downloaded files need to be extracted to the `/opt/` directory. Use the `tar` command to extract them with the flags `-xf` so that the directory structure remains intact. This will create a new `/opt/xxnetwork/` directory with the extracted files inside.

For Node:

```
$ sudo tar -xvf node.tar.gz
```

For Gateway:

```
$ sudo tar -xvf gateway.tar.gz
```

! **Note:** Both `node.tar.gz` and `gateway.tar.gz` contain some files that are the same (e.g. `xxnetwork-wrapper.py`). When extracting both files, it is expected that these files will be overwritten. Order of extraction does not matter.

! **Note:** If this command produces an error similar to `not in gzip format`, then that means the file did not download correctly in the previous step. Try downloading the file again and ensure the URL is correct.

Inside the newly created `/opt/xxnetwork/` directory are the following files needed to install the software. Those files are:

Node	Gateway	Note
`bin/xxnetwork-node`	`bin/xxnetwork-gateway`	The binary for Node or Gateway.
`node.yaml`	`gateway.yaml`	The configuration file for the binary.
`xxnetwork-node.service`	`xxnetwork-gateway.service`	systemd service file.
`xxnetwork-wrapper.py`	`xxnetwork-wrapper.py`	A management script for the binary.
`generate_certs.py`	`generate_certs.py`	TLS certificate/key generator script.
`creds/permissioning_cert.crt`	`creds/permissioning_cert.crt`	The TLS certificate for Permissioning.
`creds/network_management.crt`	`creds/network_management.crt`	The TLS certificate used to authenticate commands sent to the Wrapper Script.
`lib/libpowmo75.so`		GPU dependency (for Node only).

Generate TLS Credentials

TLS credentials for the Node and Gateway are critical to its identity in the network. For more information refer to Cryptographic and Network Primitives. If running Node and gateway on different machines, then generate the certificates on the Node machine and copy the necessary credentials to the Gateway server.

! **WARNING:** The two TLS credentials (a certificate and private key pair) generated in the following section are extremely important to back up and store securely. Loss of these files can make it impossible to reconnect a Node to the network. If following the default configuration, these files are named `node_cert.crt`, `node_key.key`, `gateway_cert.crt`, and `gateway_key.key`. In addition, it is important to back up the file `nodeIDF.json`, which is generated on the initial run.

1. Using the provided `generate_certs.py` script, generate the TLS certificates and keys for Node and Gateway. It will provide prompts for all the information required. Start the script using Python 3. Note that this must be run inside the `/opt/xxnetwork/` directory.

```
$ cd /opt/xxnetwork/ && python3 generate_certs.py
```

! **NOTE:** You do not need to use this script to generate the required TLS credentials. Use the requirements in the TLS Credentials section to ensure the generated certificates meet the network requirements.

2. The script will launch and query for various information to create the credentials for Node and Gateway. If you choose not to provide any details, default values will be used. After every answer press Enter.

```
This script will ask you to input information to be used in key generation.
If you do not wish to enter any given field, a default will be provided, attributed to the xx
network.
Country (default: 'KY (Cayman Islands)'): US
State/province (default: ''): CA
Locality (default: 'George Town'): Claremont
Organization (default: 'xxnetwork'):
Organizational unit (default: 'nodes'): testNodes
Email (default: 'admin@xx.network'):
Domain (default: 'xx.network'):
```

Once all the details have been provided, the certificates will be generated and moved into `creds/`.

```
Generating a RSA private key
............... ++++
....++++
writing new private key to 'creds/node_key.key'
-----
~~~~~
Generating a RSA private key
```

```
.................................++++
............++++
writing new private key to 'creds/gateway_key.key'
-----
```

3. To check that all the correct files are in the `creds/` directory, use the `ls` command to list the files.

```
$ ls -1 creds/
gateway_cert.crt
gateway_key.key
network_management.crt
node_cert.crt
node_key.key
permissioning_cert.crt
```

4. If running Gateway on a separate machine from Node, then copy `gateway_cert.crt`, `gateway_key.key`, and `node_cert.crt` to the Gateway machine and delete `gateway_key.key` from the Node machine.

Configuring Node and Gateway

Configuring Node and Gateway requires modifying their respective YAML files retrieved in the previous steps. These instructions go over the basic configuration to get a Node and Gateway working in the default state. Refer to the Server and Gateway README files to learn more.

! **NOTE:** If you currently configuring the Node and not the Gateway, then proceed to step 6.

1. Get the IP address of the Node. If the Node is being run on the same machine as Gateway, then this IP address is `0.0.0.0:`*`Port`* (where the default port is `11420`). If the Node and Gateway are installed on separate machines, then you will need to get the Node's *public* IP address, by running the following command on the Node.

```
$ curl ipinfo.io/ip
[your IP address]
```

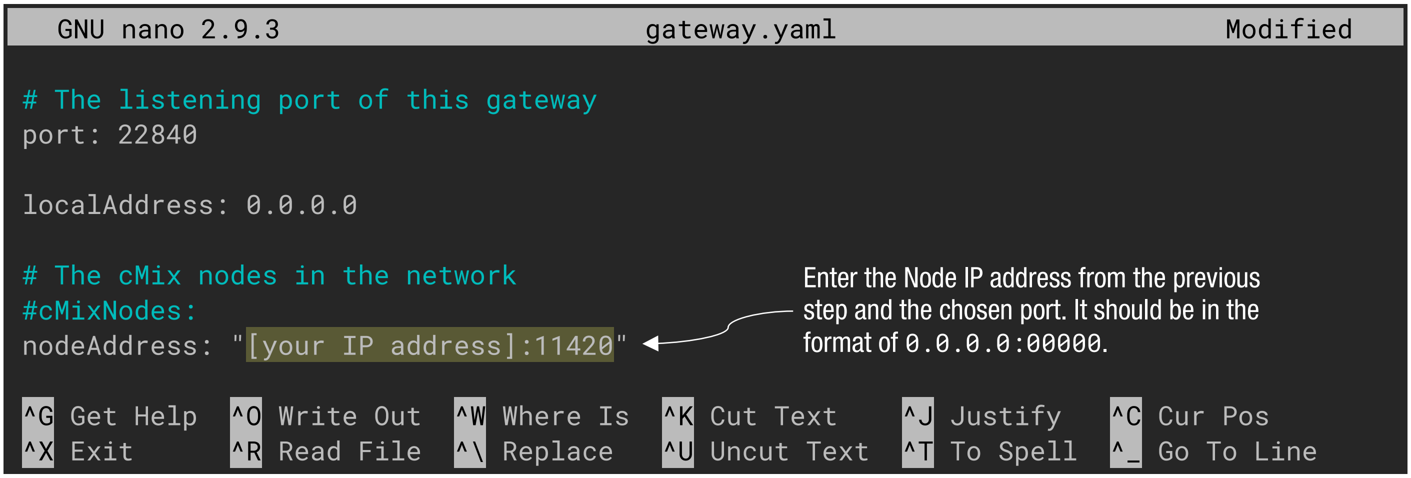
Make note of this IP address for the next steps.

2. The Gateway configuration file has to be modified to include this IP address. To do this, open `gateway.yaml` in nano or your favorite text editor.

   ! **NOTE:** The following process of using the nano text editor to modify a file is used elsewhere in this document. Refer back here for detailed steps on how to use it.

```
$ sudo nano /opt/xxnetwork/gateway.yaml
```

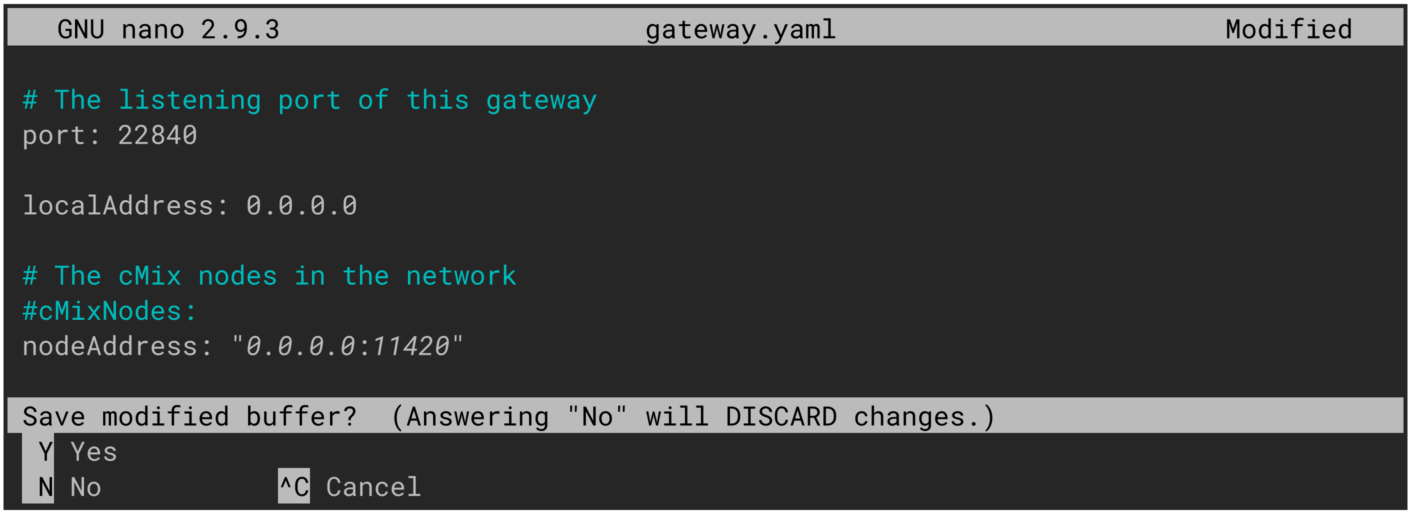
3. Once the file is open, use the down arrow key ↓ to get to the line that says `nodeAddress` and replace the placeholder with the one found in step 1. Make sure to include the chosen port, as shown.



```
  GNU nano 2.9.3                     gateway.yaml                     Modified

# The listening port of this gateway
port: 22840

localAddress: 0.0.0.0

# The cMix nodes in the network
#cMixNodes:
nodeAddress: "[your IP address]:11420"

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line
```

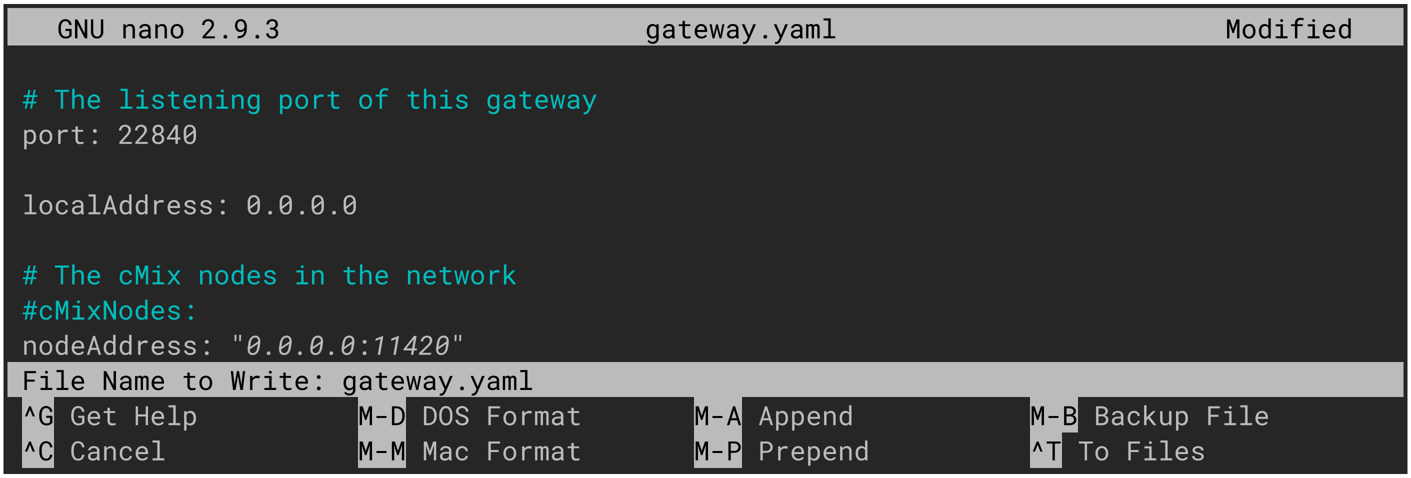
Enter the Node IP address from the previous step and the chosen port. It should be in the format of `0.0.0.0:00000`.

4. To save the file, press Ctrl + X and when prompted to save the buffer, press Y.



```
  GNU nano 2.9.3                     gateway.yaml                     Modified

# The listening port of this gateway
port: 22840

localAddress: 0.0.0.0

# The cMix nodes in the network
#cMixNodes:
nodeAddress: "0.0.0.0:11420"

Save modified buffer?  (Answering "No" will DISCARD changes.)
 Y Yes
 N No             ^C Cancel
```

5. Finally, when prompted with the file name, press Enter to exit.



```
  GNU nano 2.9.3                     gateway.yaml                     Modified

# The listening port of this gateway
port: 22840

localAddress: 0.0.0.0

# The cMix nodes in the network
#cMixNodes:
nodeAddress: "0.0.0.0:11420"
File Name to Write: gateway.yaml
^G Get Help          M-D DOS Format       M-A Append           M-B Backup File
^C Cancel            M-M Mac Format       M-P Prepend          ^T To Files
```

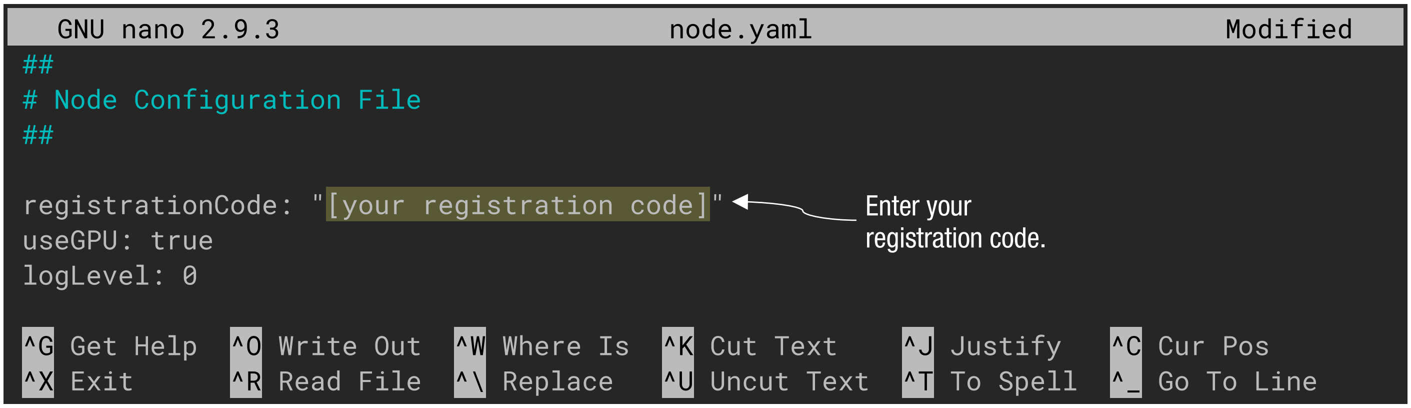
6. Every Node operator will receive a unique registration code from nodes@xx.network in order to join the network. The registration code will look like the following:

Example Registration Code: `UvyGCZU8WP18PmmIdcpVmx00QA3xNe7sEB9Hixkk=`

The Node configuration file has to be modified to include this registration code. To do this, open `node.yaml` in nano or your favorite text editor.

```
$ sudo nano /opt/xxnetwork/node.yaml
```

7. Enter this code into the `registrationCode` field between the two quotes `""`.



```
  GNU nano 2.9.3                       node.yaml                      Modified
##
# Node Configuration File
##

registrationCode: "[your registration code]"
useGPU: true
logLevel: 0

^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text     ^J Justify     ^C Cur Pos
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text   ^T To Spell    ^_ Go To Line
```

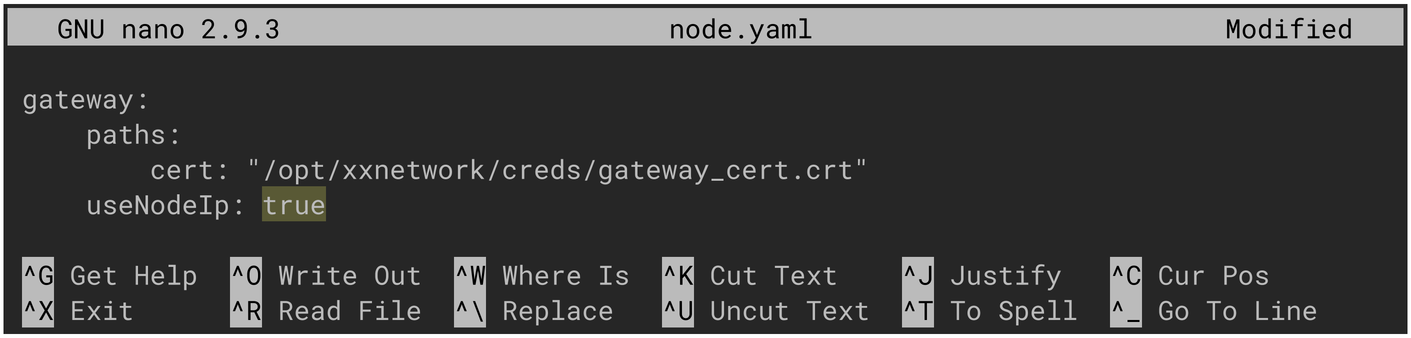
Enter your registration code.

8. If the Node is CPU only (does not have or use the GPU), ensure to set `useGPU` to `false` in `node.yaml`. Most Node configurations will have a GPU and this option should remain `true`.

   ! **NOTE:** Running a Node without a GPU is not a recommended configuration. Only do so if the CPU in the machine is very powerful.

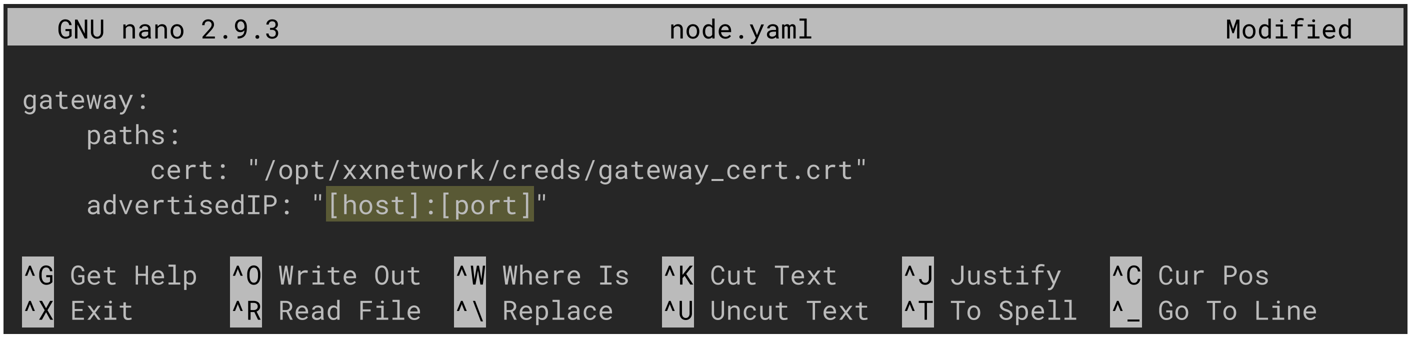
9. If Node and Gateway are going to be set up to communicate via public IP addresses, then skip this step. Otherwise, if Node and Gateway are being run locally (i.e. on the same machine or on the same local network), then Node Operators will need to modify how the Gateway's public IP is advertised via the Permissioning server using one of two optional flags in the Node YAML file. If your YAML file is missing the desired flag, you will need to add one to the locations shown below.

   a. Most Node Operators will want to set the `useNodeIp` flag (found under `gateways`) to `true`, which will use the Node's public IP address as Gateway's advertised IP address (while maintaining the Gateway port).



```
  GNU nano 2.9.3                      node.yaml                       Modified

gateway:
    paths:
        cert: "/opt/xxnetwork/creds/gateway_cert.crt"
    useNodeIp: true

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text    ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text  ^T To Spell   ^_ Go To Line
```

   b. For Node Operators with more complex networks setups, the Gateway's advertised IP address can be explicitly set using the `advertisedIP` flag (found under `gateways`). Make sure to set a host and a port in the form of *`host:port`*.



```
  GNU nano 2.9.3                      node.yaml                       Modified

gateway:
    paths:
        cert: "/opt/xxnetwork/creds/gateway_cert.crt"
    advertisedIP: "[host]:[port]"

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text    ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text  ^T To Spell   ^_ Go To Line
```

   ⚠ **WARNING:** Node will crash if both `useNodeIp` and `advertisedIP` are set at the same time. Only one flag can be used.

10. Once the change is made, save the file by pressing Ctrl + X and when prompted to save buffer, press Y. Finally, when prompted with the file name, press Enter.

11. If you placed the TLS certificate and key files in a different location than the default one in the Generate TLS Credentials section, then the Node and Gateway configuration files have to be modified to point to the new paths. Otherwise, skip this step. For more information, refer to Appendix A: Paths and Files.

    a. In `node.yaml`:

       i. Modify `node` > `paths` > `key` to point to the absolute path of `node_key.key`.

       ii. Modify `node` > `path` > `cert` to point to the absolute path of `node_cert.crt`.

       iii. Modify `gateways` > `paths` > `cert` to point to the absolute path of `gateway_cert.crt`.

   iv. Modify `permissioning` > `paths` > `cert` to point to the absolute path of `permissioning_cert.crt`.

b. In `gateway.yaml`:

   i. Modify `keyPath` to point to the absolute path of `gateway_key.key`.

   ii. Modify `certPath` to point to the absolute path of `gateway_cert.crt`.

   iii. Modify `serverCertPath` to point to the absolute path of `node_cert.crt`.

   iv. Modify `permissioningCertPath` to point to the absolute path of `permissioning_cert.crt`.

Configure Service Files

To ensure that the Node and Gateway services are run as the correct user, they will need to be modified.

1. The following steps require the machine's username. For most users, this will be the username created and installation of the operating system. If you do not remember your username, use `whoami`.

```
$ whoami
  [your username]
```

2. Open the service file for the binary being run. If both Node and Gateway are being run, this modification will need to be made with both files. Open either `xxnetwork-node.service` or `xxnetwork-gateway.service` with nano or your favorite text editor.

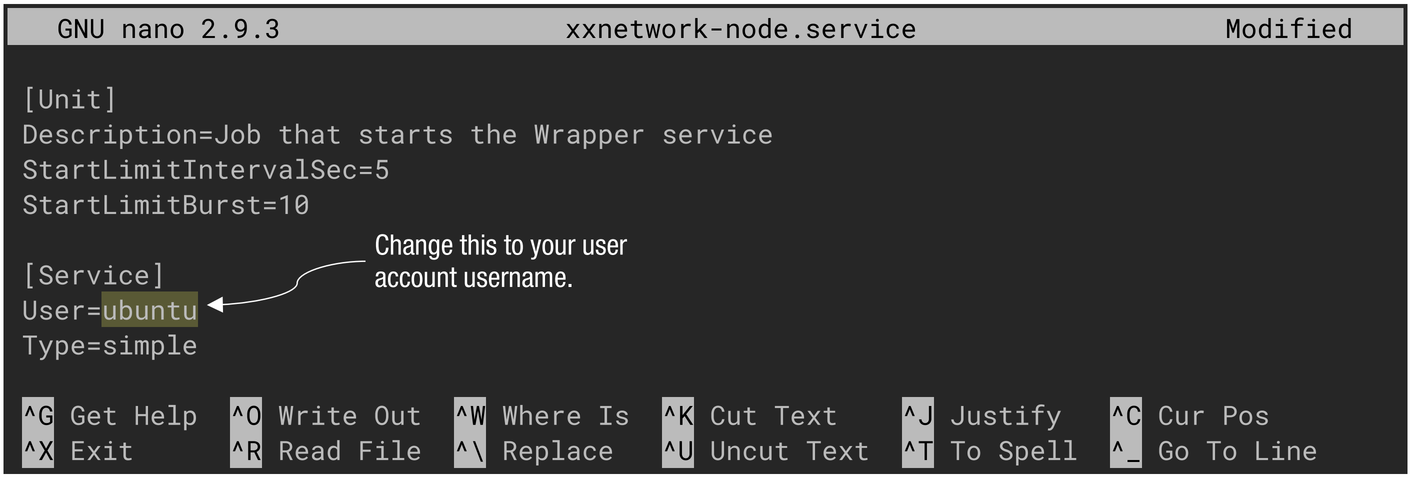
   FOR NODE:

```
$ sudo nano /opt/xxnetwork/xxnetwork-node.service
```

   FOR GATEWAY:

```
$ sudo nano /opt/xxnetwork/xxnetwork-gateway.service
```

3. Under the `[Service]` heading, after the `User=` field, modify the username to the one found in step 1.



```
  GNU nano 2.9.3                 xxnetwork-node.service                 Modified

[Unit]
Description=Job that starts the Wrapper service
StartLimitIntervalSec=5
StartLimitBurst=10

[Service]
User=ubuntu
Type=simple

^G Get Help    ^O Write Out   ^W Where Is   ^K Cut Text     ^J Justify    ^C Cur Pos
^X Exit        ^R Read File   ^\ Replace    ^U Uncut Text   ^T To Spell   ^_ Go To Line
```

Change this to your user account username.

4. Once the change is made, save the file by pressing Ctrl + X and when prompted to save buffer, press Y. Finally, when prompted with the file name, press Enter.

5. Repeat step 1 through step 4 for the other service file if both Node and Gateway are being run.

Database

Node uses a PostgreSQL database to store key pairs with clients. The following instructions detail how to install PostgreSQL and set up the required database.

! **NOTE:** Node operators may wish to install the database in a separate location or drive. It is recommended following DigitalOcean's comprehensive guide on how to do so after finishing step 3.

1. Install PostgreSQL and its dependencies.

```
$ sudo apt install -y postgresql-client postgresql postgresql-contrib
```

2. Once the installation is complete, enable the PostgreSQL service.

```
$ sudo update-rc.d postgresql enable
```

3. Next, start the service.

```
$ sudo service postgresql start
```

4. Create a database user with the username `node`, which is used to access the database.

```
$ sudo -u postgres createuser --createdb --pwprompt node
```

5. During this process, you will be asked to set a password for the `node` user. Create a long and secure password but note that it will be only used in the following few steps and you will not need to remember it once everything is configured.

```
Enter password for new role:
Enter it again:
```

Save this password for the next steps.

⚠ **WARNING:** Never store this password digitally unless directed to do so by xx network. Never provide this password to anyone.

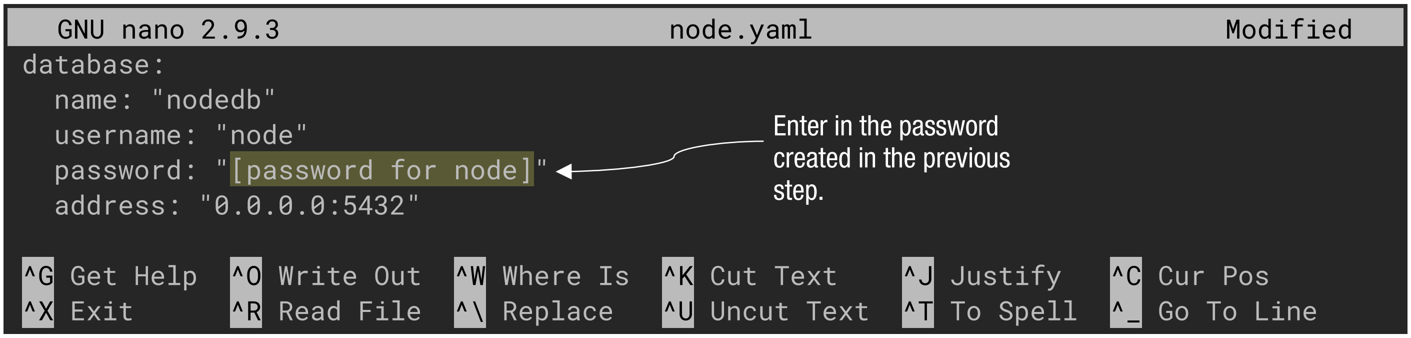
6. Create the required database with the name `nodedb`.

```
$ sudo -u postgres createdb -O node nodedb
```

7. Next, modify the Node configuration file to use the newly generated password in step 5. To do this, open `node.yaml` in nano or your favorite text editor.

```
$ sudo nano /opt/xxnetwork/node.yaml
```

8. Under the field `database` > `password`, enter in the new password. If the database was created with a different username or database name, then update those values too.



```
  GNU nano 2.9.3                    node.yaml                          Modified
database:
  name: "nodedb"
  username: "node"
  password: "[password for node]"
  address: "0.0.0.0:5432"

^G Get Help   ^O Write Out   ^W Where Is   ^K Cut Text     ^J Justify    ^C Cur Pos
^X Exit       ^R Read File   ^\ Replace    ^U Uncut Text   ^T To Spell   ^_ Go To Line
```

Enter in the password created in the previous step.

9. Once the change is made, save the file by pressing Ctrl + X and when prompted to save buffer, press Y. Finally, when prompted with the file name, press Enter.

Verifying the Database

The following steps will describe how to ensure the database was created correctly. This section is optional but highly recommended.

1. Login to the user `postgres`, which was automatically created to run the PostgreSQL database server.

```
$ sudo su postgres
```

2. Run the command `psql` to start and interactive PostgreSQL session.

```
$ psql
psql (10.12 (Ubuntu 10.12-0ubuntu0.18.04.1))
Type "help" for help.
```

3. Once at the `postgres` prompt, enter in `\l` to get a list of databases (do not enter the `postgres-#` part)

```
postgres-# \l
```

4. the database has been set up correctly If the newly created database `nodedb` with the owner `node` exists.

```
                                   List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 nodedb    | node     | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
(4 rows)
```

5. Enter `\q` to exit the command line.

```
postgres-# \q
```

6. Enter `exit` to return back to your user.

```
$ exit
```

Services

This section describes how to install the service files for Node and Gateway. Follow only the instructions related to the type of machine you are deploying. Follow both Node and Gateway instructions if running both off of the same machine.

WARNING: The following steps will start the Node and/or Gateway software and will have it join the xx network. If you do not want to do so currently, then do not do the following steps until you are ready.

1. Before starting the Node or Gateway service, ensure that the clock is synchronized by entering the following command.

```
$ timedatectl status
```

This should result in the following output.

```
                      Local time: Sat 2020-06-13 20:43:41 UTC
                  Universal time: Sat 2020-06-13 20:43:41 UTC
                        RTC time: Sat 2020-06-13 20:43:41
                       Time zone: Etc/UTC (UTC, +0000)
       System clock synchronized: yes
systemd-timesyncd.service active: yes
                 RTC in local TZ: no
```

2. If both `System clock synchronized` and `systemd-timesyncd.service active` are set to `yes`, then proceed to the next step. Otherwise, refer back to the Clock Synchronization (NTP) section.

3. Soft link the systemd service file(s) to the systemd directory `/etc/systemd/system/`.

   For Node:

   ```
   $ sudo ln -s /opt/xxnetwork/xxnetwork-node.service /etc/systemd/system
   ```

   For Gateway:

   ```
   $ sudo ln -s /opt/xxnetwork/xxnetwork-gateway.service /etc/systemd/system
   ```

4. Reload the systemd service files.

   ```
   $ sudo systemctl daemon-reexec
   ```

5. To ensure that your user has the correct permissions, run the following command. Make sure to replace both instances of `[user]` with your username.

   ```
   $ sudo chown [user]:[user] -Rv /opt/xxnetwork
   ```

6. Next, enable the service(s) for the appropriate binary.

   For Node:

   ```
   $ sudo systemctl enable xxnetwork-node.service
   ```

   This should result in the following output.

   ```
   Created symlink /etc/systemd/system/multi-user.target.wants/xxnetwork-node.service
   → /etc/systemd/system/xxnetwork-node.service.
   ```

   For Gateway:

   ```
   $ sudo systemctl enable xxnetwork-gateway.service
   ```

   This should result in the following output.

   ```
   Created symlink /etc/systemd/system/multi-user.target.wants/xxnetwork-
   gateway.service → /etc/systemd/system/xxnetwork-gateway.service.
   ```

7. Then, start the services.

   For Node:

   ```
   $ sudo systemctl start xxnetwork-node.service
   ```

   For Gateway:

   ```
   $ sudo systemctl start xxnetwork-gateway.service
   ```

Verify the Service has Started

1. First, check the status of the service to ensure it is running.

FOR NODE:

```
$ sudo systemctl status xxnetwork-node.service
```

This should result in the following output. Press Q to exit the output.

```
● xxnetwork-node.service - Job that starts the Wrapper service
   Loaded: loaded (/opt/xxnetwork/xxnetwork-node.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2020-06-23 15:36:32 PDT; 10s ago
 Main PID: 6295 (bash)
    Tasks: 8 (limit: 4915)
   CGroup: /system.slice/xxnetwork-node.service
           ├─6295 /bin/bash -c /opt/xxnetwork/xxnetwork-wrapper.py --logpath /opt/xxnetw...
           ├─6296 python3 /opt/xxnetwork/xxnetwork-wrapper.py --logpath /opt/xxnetwork/n...
           └─67181 /opt/xxnetwork/bin/xxnetwork-node
```

FOR GATEWAY:

```
$ sudo systemctl status xxnetwork-gateway.service
```

This should result in the following output. Press Q to exit the output.

```
● xxnetwork-gateway.service - Job that starts the Wrapper service
   Loaded: loaded (/opt/xxnetwork/xxnetwork-gateway.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2020-06-23 15:36:37 PDT; 2min 0s ago
 Main PID: 6343 (bash)
    Tasks: 12 (limit: 4915)
   CGroup: /system.slice/xxnetwork-gateway.service
           ├─6343 /bin/bash -c /opt/xxnetwork/xxnetwork-wrapper.py --logpath /opt/xxnetw...
           ├─6344 python3 /opt/xxnetwork/xxnetwork-wrapper.py --logpath /opt/xxnetwork/g...
           └─6418 /opt/xxnetwork/bin/xxnetwork-gateway
```

2. Next, check that the process is running. If the commands result in the expected output, then the Node or Gateway process is running.

```
$ ps -A | grep xxnetwork
```

This will print a list of running xx network processes. If `xxnetwork-gatew` appears on the list, then the Gateway process is running. If `xxnetwork-node` appears on the list, then the Node process is running.

```
 2433 ?        00:00:17 xxnetwork-gatew
 9915 ?        00:00:22 xxnetwork-node
```

❗ **NOTE:** The PID (the first number printed) and the elapsed time will be different.

⚠ ATTENTION ⚠

Once the Node and Gateway are properly functioning, Node Operators must backup some important files or risk losing access to the network. Follow the instructions on the next page detailing how to do so.

Go to Next Page

Backup Important Files

WARNING: The following instructions are extremely important to follow to ensure that your Node and Gateway can stay operating on the network. Failure to follow these instructions can lead to your Node or Gateway being permanently unable to operate on the network.

WARNING: The files described below are sensitive as they identify your Node and Gateway uniquely. Anybody who gains access to these files can appear on the network as your Node or Gateway. It is extremely important to keep these files secure as any third party access can lead to the compromising of your Node and Gateway's identity.

There are six files that your Node/Gateway requires to authenticate itself on the network. Loss of these files will lead to your Node or Gateway being unable to participate in the network and these files cannot be regenerated. Therefore, every Node Operator *must* backup these files to a secure location not on the same machine. It is recommended that these files be backed up to multiple secure locations. For example, an flash drive or secure cloud storage.

1. Locate the six files that must be backed up. If you used the default paths in the configuration instructions, then these files can be found according to the table below.

	File	Default Name	Default Location
1.	Node's public certificate	`node_cert.crt`	`/opt/xxnetwork/creds/`
2.	Node's private key	`node_key.key`	`/opt/xxnetwork/creds/`
3.	Node's ID file (IDF)	`nodeIDF.json`	`/opt/xxnetwork/node-logs/`
4.	Gateway's public certificate	`gateway_cert.crt`	`/opt/xxnetwork/creds/`
5.	Gateway's private key	`gateway_key.key`	`/opt/xxnetwork/creds/`
6.	Gateway's ID file (IDF)	`gatewayIDF.json`	`/opt/xxnetwork/gateway-logs/`

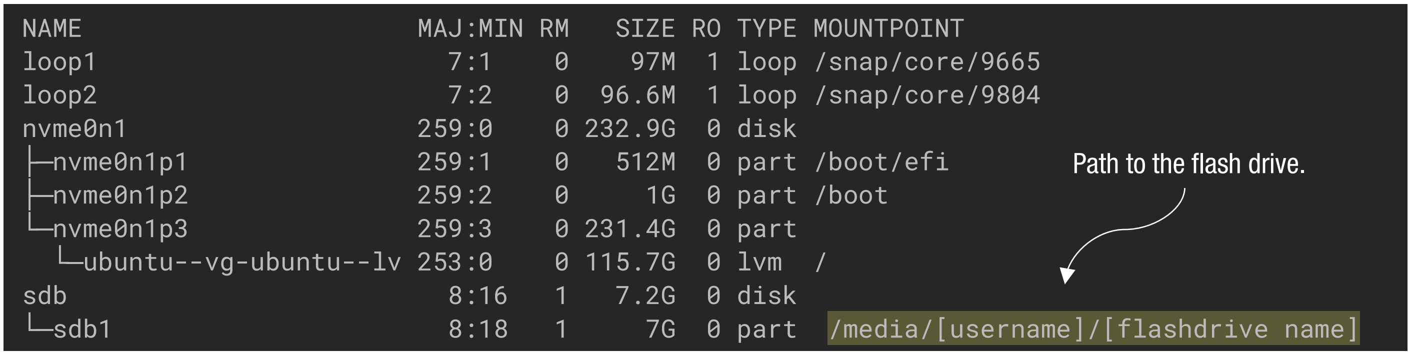
If you cannot locate the files, then their paths can be found in the Node and Gateway YAML files as shown in the table below.

	File	YAML File	Flag
7.	Node's public certificate	`node.yaml`	`node.paths.cert`
8.	Node's private key	`node.yaml`	`node.paths.key`
9.	Node's ID file (IDF)	`node.yaml`	`node.paths.idf`
10.	Gateway's public certificate	`gateway.yaml`	`certPath`
11.	Gateway's private key	`gateway.yaml`	`keyPath`
12.	Gateway's ID file (IDF)	`gateway.yaml`	`idfPath`

2. The easiest method to copy these files off of the machine is to use a USB flash drive; you can use the same one that you may have used to install the operating system. First, insert the flash drive and then run `lsblk` to find mount point of the flash drive.

```
$ lsblk
```

This should output a list of mount points on your computer. Locate the name of the flash drive. This can be done by looking at the size and finding which mount point has the same size as your flash drive.



```
NAME                          MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop1                           7:1    0    97M  1 loop /snap/core/9665
loop2                           7:2    0  96.6M  1 loop /snap/core/9804
nvme0n1                       259:0    0 232.9G  0 disk
├─nvme0n1p1                   259:1    0   512M  0 part /boot/efi
├─nvme0n1p2                   259:2    0     1G  0 part /boot
└─nvme0n1p3                   259:3    0 231.4G  0 part
  └─ubuntu--vg-ubuntu--lv     253:0    0 115.7G  0 lvm  /
sdb                             8:16   1   7.2G  0 disk
└─sdb1                          8:18   1     7G  0 part  /media/[username]/[flashdrive name]
```

3. Copy all six files to the flash drive path found in the previous step. If your files are in the default locations, it will be similar to the following command.

```
$ cp /opt/xxnetwork/creds/{node_cert.crt, node_key.key, gateway_cert.crt,
  gateway_key.key} /opt/xxnetwork/node-logs/nodeIDF.json /opt/xxnetwork/gateway-
  logs/gatewayIDF.json /media/[username]/[flashdrive name]
```

   To copy one file at a time, use the following command.

```
$ cp [path of file to copy] [path of flash drive]
```

4. Once the files are copied, remove the flash drive, and save it in a secure location. We highly recommend that you back up these files in at least one other location.

   Ensure that these files only exist in secure locations. If anyone gains access to these files, your Node and Gateway’s identity on the network can be compromised.

Abridged Node and Gateway Set Up

The Abridged Node set up instructions contains all the necessary information for setting up a Node and Gateway in a concise manner. This is an alternative instruction set to the Node and Gateway Set Up that more experienced Node operators may find easier to use.

Initial Set Up

The Node and Gateway software has only been tested on Ubuntu Server 18.04. Node operators are welcome to use alternative operating systems, but support cannot be guaranteed and manual compilation of the binaries may be required. Also, an alternate operating system may cause issues with the Wrapper Script.

1. Download Ubuntu Server 18.04, create a bootable disk, and boot the machine from it.
2. Install using the default options, but ensure that you use the entire disk and ensure it has enough room per the Hardware Requirements.

   ⚠ **WARNING:** Create a strong but memorable password. It is recommended that it is longer than 12 characters. Store this password in a safe and secure location. Never share this password with anyone.

3. Update and install the software on the computer.

   ```
   $ sudo apt update
   $ sudo apt upgrade
   ```

4. Install the Python package installer and the dependencies `boto3` and `pyOpenSSL`.

   ```
   $ sudo apt install -y python3-pip
   $ sudo pip3 install -U pip
   $ pip3 install --user -U boto3 pyOpenSSL
   ```

5. Ensure that the internet is functioning. It is highly recommended that a firewall such as UFW is used. Configure it to allow the ports which will be used for the Node and/or Gateway. The ports found in the default configuration files provided are `11420` for Node and `22840` for Gateway.

   ! NOTE: If you are connected over SSH, you may want to add your SSH port to UFW.

   ```
   $ sudo ufw disable
     Firewall stopped and disabled on system startup
   $ sudo ufw allow 11420/tcp
     Rules updated
     Rules updated (v6)
   $ sudo ufw allow 22840/tcp
     Rules updated
     Rules updated (v6)
   $ sudo ufw enable
     Firewall is active and enabled on system startup
   ```

Clock Synchronization (NTP)

Commands from the Permissioning server are time-stamped and a synchronized clock is important for a Node to properly interpret them. To do so, NTP (Network Time Protocol) must be set up, enabled, and synchronized. Ensure that your system has time synchronization enabled and that the clock is synchronized.

Configure Local Network

Ensure that the machine can be accessed from outside the local network via the ports for Node and/or Gateway. This will most likely involve logging into the network gateway and port forwarding these ports for the machine. You may want to also give your machine a static local IP.

GPU Drivers

The Node software requires a Nvidia RTX graphic processor and the installation of the Nvidia CUDA Toolkit (version 10.2). This toolkit installs the GPU drivers and the runtime libraries for CUDA as well as various tools such as a profiler and debugger, which help debug GPU related bugs.

❗ **NOTE:** If you currently configuring the Gateway or a Node with only CPU usage, then skip this section and proceed to Node and Gateway Software.

1. Nvidia provides a list of commands to run that install all the necessary software onto the machine. Simply run all the commands in the provided order.

```
$ wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64/
   cuda-ubuntu1804.pin
$ sudo mv cuda-ubuntu1804.pin /etc/apt/preferences.d/cuda-repository-pin-600
$ wget http://developer.download.nvidia.com/compute/cuda/10.2/Prod/local_installers/
   cuda-repo-ubuntu1804-10-2-local-10.2.89-440.33.01_1.0-1_amd64.deb
$ sudo dpkg -i cuda-repo-ubuntu1804-10-2-local-10.2.89-440.33.01_1.0-1_amd64.deb
$ sudo apt-key add /var/cuda-repo-10-2-local-10.2.89-440.33.01/7fa2af80.pub
$ sudo apt-get update
$ sudo apt-get -y install cuda
```

   ❗ **NOTE:** The multi-line commands cannot be directly copied because there will be extra whitespace.

2. Once the installation is complete, reboot the system.

```
$ sudo reboot now
```

3. Once the system reboots, log back into the computer.

4. It is recommended that the driver installation is verified to have succeeded. Follow the instructions in Verifying the Driver Installation for more information.

Node and Gateway Software

1. Download and extract the tarball for the software you are running to the `/opt/` directory. If both Node and Gateway are being run on the same machine, then download and extract both files. The newest versions of these files can be found using the direct links below or on the xx network website.

```
https://xx.network/codebase/node.tar.gz

https://xx.network/codebase/gateway.tar.gz
```

2. Generate the necessary TLS credentials using the included `generate_certs.py` script. Alternatively, generate the credentials yourself following the requirements in the TLS Credentials section.

3. Update the paths for the Node and Gateway TLS credentials as well as the Permissioning TLS certificate in the Node and Gateway configuration files. Refer to step 6 in the Node and Gateway Set Up section.

4. In the Gateway configuration file, add the IP address and port of the Node for the `nodeAddress`. If the Gateway is being run on the same machine as the Node, the address is `0.0.0.0:`*`Port`*. If the Gateway is on a separate machine, then use the Node's public IP address.

5. Every Node operator will receive a unique registration code from nodes@xx.network in order to join the network. Enter this code into the `registrationCode` field in `node.yaml`.

6. If the Node is CPU only (does not have or use the GPU), ensure to set `useGPU` to `false` in `node.yaml`.

   ❗ **NOTE:** Running a Node without a GPU is not a recommended configuration. Only do so if the CPU in the machine is very powerful.

7. If the Node and Gateway are run locally, then the Node YAML file must be modified to ensure the Gateway's advertised IP address is correct. If the Gateway and Node and behind the same public IP address, then set the flag `useNodeIp` (under gateways) to `true` to use the Node's public IP address as Gateway's address. To explicitly set the Gateway public IP address, use the `advertisedIP` flag. The address must be in the format of `host:port`. These flags are optional but only one can be used at a time.

8. Modify the services files to execute under the intended user.

Database

The Node uses a PostgreSQL database to store key pairs with clients. The following instructions detail how to install PostgreSQL and set up the required database.

! **Note:** Node operators may wish to install the database in a separate location or drive. It is recommended following DigitalOcean's comprehensive guide on how to do so after finishing step 3.

1. Install PostgreSQL and its dependencies.

```
$ sudo apt install -y postgresql-client postgresql postgresql-contrib
```

2. Once the installation is complete, enable the PostgreSQL service.

```
$ sudo update-rc.d postgresql enable
```

3. Next, start the service.

```
$ sudo service postgresql start
```

4. Create a database user. Give it a strong password as these are the credentials for the Node to access the database. The password will be placed in the Node configuration file.

```
$ sudo -u postgres createuser --createdb --pwprompt [username]
```

⚠ **Warning:** Never store this password digitally unless directed to do so by xx network. Never provide this password to anyone.

5. Create the required database and choose a name.

```
$ sudo -u postgres createdb -O [username] [database name]
```

6. Modify `node.yaml` with the database name, username, and password.

```
database:
  name: "[chosen database name]"
  username: "[chosen username]"
  password: "[chosen password]"
```

Services

This section describes how to install the service files for Node and Gateway.

1. Ensure that the system clock is synchronized before proceeding using `timedatectl status`.

2. Soft link the systemd service file(s) (`xxnetwork-node.service` and `xxnetwork-gateway.service`) to the systemd directory `/etc/systemd/system/`.

```
$ sudo ln -s /opt/xxnetwork/xxnetwork-node.service /etc/systemd/system
$ sudo ln -s /opt/xxnetwork/xxnetwork-gateway.service /etc/systemd/system
```

3. Reload the systemd service files.

```
$ sudo systemctl daemon-reexec
```

4. Enable the service(s) for the appropriate binary.

```
$ sudo systemctl enable xxnetwork-node.service
$ sudo systemctl enable xxnetwork-gateway.service
```

5. Finally, start the services.

```
$ sudo systemctl start xxnetwork-node.service
$ sudo systemctl start xxnetwork-gateway.service
```

⚠ ATTENTION ⚠

Once the Node and Gateway are properly functioning, Node Operators must backup some important files or risk losing access to the network. Follow these instructions detailing how to do so.

Backup Important Files

Warning: The following instructions are extremely important to follow to ensure that your Node and Gateway can stay operating on the network. Failure to follow these instructions can lead to your Node or Gateway being permanently unable to operate on the network.

Warning: The files described below are sensitive as they identify your Node and Gateway uniquely. Anybody who gains access to these files can appear on the network as your Node or Gateway. It is extremely important to keep these files secure as any third party access can lead to the compromising of your Node and Gateway's identity.

There are six files that your Node/Gateway requires to authenticate itself on the network. Loss of these files will lead to your Node or Gateway being unable to participate in the network and these files cannot be regenerated. Therefore, every Node Operator *must* backup these files to a secure location not on the same machine. It is recommended that these files be backed up to multiple secure locations. For example, a flash drive or secure cloud storage.

Below are the six files that you need to backup and their default locations.

	File	Default Name	Default Location
1.	Node's public certificate	`node_cert.crt`	`/opt/xxnetwork/creds/`
2.	Node's private key	`node_key.key`	`/opt/xxnetwork/creds/`
3.	Node's ID file (IDF)	`nodeIDF.json`	`/opt/xxnetwork/node-logs/`
4.	Gateway's public certificate	`gateway_cert.crt`	`/opt/xxnetwork/creds/`
5.	Gateway's private key	`gateway_key.key`	`/opt/xxnetwork/creds/`
6.	Gateway's ID file (IDF)	`gatewayIDF.json`	`/opt/xxnetwork/gateway-logs/`

For more detailed information, refer to the Backup Important Files in the Node and Gateway Set Up.

Manually Deploying Binaries

After following the Node and Gateway Set Up, it is possible to use alternate binaries than the ones supplied by the Wrapper Script. The source code to compile the binaries can be downloaded from the Elixxir GitLab page.

1. To disable automatic update via the Wrapper Script, add `--disableupdates` to the call in the service files located in `/opt/xxnetwork/xxnetwork-node.service` for Node and `/opt/xxnetwork/xxnetwork-gateway.service` for Gateway.

2. Replace the binaries in `/opt/xxnetwork/bin/` with the self-compiled ones generated in the following steps. Then restart the service.

   For Node:

   ```
   $ systemctl restart xxnetwork-node.service
   ```

   For Gateway:

   ```
   $ systemctl restart xxnetwork-gateway.service
   ```

Compile Binaries

A second option to manually deploying binaries is to download the source code and compile it yourself. Go version 1.13 is required for both Server and Gateway.

1. Download and install Go version 1.13. Refer to the Environment Set Up section for detailed instructions.

Gateway

Compiling Gateway is a straightforward process. For additional information, refer to the Gateway `.gitlab-ci.yaml` file.

1. Download the latest master branch of Gateway.

2. Compile Gateway using the following command.

   ```
   $ GOOS=linux GOARCH=amd64 CGO_ENABLED=0 go build -ldflags '-w -s' -o
     xxnetwork-gateway main.go
   ```

   It is also possible to compile it for other systems, though support cannot be guaranteed.

   ```
   $ GOOS=windows GOARCH=amd64 CGO_ENABLED=0 go build -ldflags '-w -s' -o
     xxnetwork-gateway.win64 main.go

   $ GOOS=windows GOARCH=386 CGO_ENABLED=0 go build -ldflags '-w -s' -o
     xxnetwork-gateway.win32 main.go

   $ GOOS=darwin GOARCH=amd64 CGO_ENABLED=0 go build -ldflags '-w -s' -o
     xxnetwork-gateway.darwin64 main.go
   ```

3. Rename the file to `xxnetwork-gateway` and move it to `/opt/xxnetwork/bin`. Then restart the service.

   ```
   $ systemctl restart xxnetwork-gateway.service
   ```

Server

If the Node is expected to run with GPU acceleration, compiling Server requires an extra step to compile. This setup assumes a working Nvidia installation. For additional information, refer to the Server `.gitlab-ci.yaml` file.

1. First, install `libgmp-de`, a dependency for the GPU code to run.

```
$ sudo apt update
$ sudo apt install -y libgmp-dev
```

2. Install the latest version of the `gpumathsnative` repository. This is where the underlying CUDA implementations of the mathematical operations live is what allows operations to be done on the GPU.

```
$ git clone -b master git@gitlab.com:elixxir/gpumathsnative.git
$ pushd gpumathsnative/cgbnBindings/powm
$ make turing
$ sudo make install
$ popd
```

❗ **NOTE:** Skip this step if the Node is being run without GPU support.

3. Download the latest master branch of Server.

4. Finally, compile the Server using the following command.

```
$ GOOS=linux GOARCH=amd64 CGO_ENABLED=1 go build -tags gpu -ldflags '-w -s -L
  /opt/xxnetwork/lib' -o xxnetwork-node main.go
```

5. Name the file `xxnetwork-node` and move it to `/opt/xxnetwork/bin`. Then restart the service.

```
$ systemctl restart xxnetwork-node.service
```

Testing

A majority of the code in the Elixxir codebase is tested. Node operators can run the tests on any public repository themselves using the instructions in this section.

Environment Set Up

Prior to running any tests, set up a development environment with Go 1.13 and some additional software.

! **Note:** These instructions describe how to run tests in Ubuntu Server 18.04. However, almost all tests run successfully on Windows, macOS, and other Linux distributions.

1. First, install the `build-essential` package, which contains a number of tools for compiling binaries. This step is only necessary if running tests on Server with the GPU.

   ```
   $ sudo apt-get update
   $ sudo apt-get install build-essential
   ```

   ! **Note:** If the GPU drivers were previously installed, build-essential may be already installed too.

2. Download Go version 1.13. Refer to the Go install page for more information.

   ```
   $ curl -L -O https://dl.google.com/go/go1.13.12.linux-amd64.tar.gz
   ```

3. Extract the archive into `/usr/local`.

   ```
   $ sudo tar -C /usr/local -xzf go1.13.12.linux-amd64.tar.gz
   ```

4. The `.profile` file for the current user must be modified to include `/usr/local/go/bin` in the `PATH` environment variable. This can be done manually by opening the file and modifying it. Alternatively, use the following command to automatically append the correct line to the `.profile` file.

   ```
   $ echo 'export PATH=$PATH:/usr/local/go/bin' >>~/.profile
   ```

5. Once the file is saved, to apply the changes immediately, run the following command.

   ```
   $ source ~/.profile
   ```

6. To ensure that everything is working and that Go is the correct version, run the following command.

   ```
   $ go version
   ```

   It should result in a similar output to this:

   ```
   go version go1.13.12 linux/amd64
   ```

7. Finally, create a directory to place the Go source code that will be downloaded in the following sections. Any name can be chosen.

   ```
   $ mkdir ~/dev
   ```

Running Tests

Running tests in Go for most Elixxir repositories is a straightforward process. Server has an extra step that is described below. To run a test, simply download a repository and use `go test`.

1. Go to the Elixxir GitLab page and select a repository to download. Alternatively, select one from the following list of public repositories:

   - `primitives`
   - `crypto`
   - `comms`
   - `gpumathsgo`
   - `server`
   - `gateway`
   - `wrapper`

2. Use `git clone` or any other method to download the repository.

   ```
   $ git clone -b master git@gitlab.com:elixxir/[repository name from step 1].git
   ```

   In the following steps, Primitives will be used as an example.

   ```
   $ git clone -b master git@gitlab.com:elixxir/primitives.git
   ```

3. Change directories to the root of the downloaded repository.

   ```
   $ cd primitives/
   ```

4. To compile and run all the tests for this repository, use the following command.

   ```
   $ go test ./...
   ```

   ! **NOTE:** The first time the code is compiled, all of the dependencies will download first.

   Running the tests can take a while, but once they are finished, a summary of the tests and their elapsed time will be printed out. If any tests fail, those errors will be printed too.

   ```
   ok      gitlab.com/elixxir/primitives/current   0.016s
   ok      gitlab.com/elixxir/primitives/format    0.025s
   ok      gitlab.com/elixxir/primitives/id        0.023s
   ok      gitlab.com/elixxir/primitives/id/idf    0.134s
   ok      gitlab.com/elixxir/primitives/ndf       0.031s
   ok      gitlab.com/elixxir/primitives/rateLimiting      113.118s
   ok      gitlab.com/elixxir/primitives/ring      0.020s
   ok      gitlab.com/elixxir/primitives/states    0.017s
   ok      gitlab.com/elixxir/primitives/switchboard       4.344s
   ok      gitlab.com/elixxir/primitives/utils     0.093s
   ok      gitlab.com/elixxir/primitives/version   0.008s
   ```

5. To run tests located in a specific directory, include the directory name. Also, including the `-v` flag will print more details about which test is being run and the debug logs from it.

   ```
   $ go test -v ./utils
   ```

   The result will be a list of the tests run and their elapsed time. At the bottom is the total time for all the tests.

   ```
   === RUN   TestUnmarshal_DataLengthError
   --- PASS: TestUnmarshal_DataLengthError (0.00s)
   === RUN   TestType_String
   --- PASS: TestType_String (0.00s)
   === RUN   TestType_String_Error
   --- PASS: TestType_String_Error (0.00s)
   PASS
   ok      gitlab.com/elixxir/primitives/id        0.065s
   ```

6. To run only one test or multiple tests matching a regular expression, use the `-ran` flag.

   ```
   $ go test -run [test name or regex]
   ```

Setting Up Server for Testing

To run some of the tests on Server, a GPU is required and the `gpumathsnative` repository must be downloaded. This is what allows operations to be done on the GPU and is where the underlying CUDA implementations of the mathematical operations live.

1. Ensure that the GPU Drivers are set up.

2. Install `libgmp-de`, a dependency for the GPU code to run.

```
$ sudo apt update
$ sudo apt install -y libgmp-dev
```

3. Download the latest master version of the `gpumathsnative` repository.

```
$ git clone -b master git@gitlab.com:elixxir/gpumathsnative.git
```

4. Run the following commands to go to the correct directory and execute the make command.

```
$ pushd gpumathsnative/cgbnBindings/powm
$ make turing
$ sudo make install
$ popd
```

5. When running tests with GPU on Server, include the flag `-tags gpu`.

```
$ go test ./... -tags gpu
```

Appendix

A. Paths and Files

There are a number of file paths that need to be specified for a Node or Gateway to run. Some of these paths point to existing files, others point to where a file should be saved. Many of these are modified through the Node and Gateway configuration files; however, the configuration files themselves must be named `node.yaml` and `gateway.yaml`. The software searches the following directories, in order, and uses the first correctly named configuration file it finds and will fail to load if it cannot be found.

1) `$HOME/.xxnetwork/`
2) `/opt/xxnetwork/`
3) `/etc/xxnetwork/`

Below is an example directory structure if the xx network software was installed following the instructions in this manual. The files that are required at startup are marked with a red asterisk (*) and the files that are generated are marked with a blue dagger (†).

```
/opt/xxnetwork/
├── bin
│     ├── xxnetwork-gateway
│     └── xxnetwork-node
├── cmdlod
│     ├── command_1591809548.jsonl†
│     ├── command_1591856646.jsonl†
│     └── version_1591166323.jsonl†
├── creds
│     ├── gateway_cert.crt*
│     ├── gateway_key.key*
│     ├── network_management.crt*
│     ├── node_cert.crt*
│     ├── node_key.key*
│     └── permissioning_cert.crt*
├── gateway-logs
│     ├── gateway.log†
│     ├── gatewayIDF.json†
│     └── xxnetwork-wrapper.log†
├── gateway.yaml*
├── generate_certs.py
├── lib
│     └── libpowmo75.so*
├── node-logs
│     ├── metrics.log†
│     ├── node-err.log†
│     ├── node.log†
│     ├── nodeIDF.json†
│     └── xxnetwork-wrapper.log†
├── node.yaml*
├── xxnetwork-gateway.service
├── xxnetwork-node.service
└── xxnetwork-wrapper.py
```

FIGURE 3: Default directory tree when running both Node and Gateway in the default configuration. Files with a red asterisk (*) are required at startup. Files marked with a blue dagger (†) are generated.

Node

The Node requires four files to run:

1) The Node private key
2) The Node TLS certificate
3) The Gateway TLS certificate
4) The Permissioning TLS certificate

```
node:
  paths:
    ⋮
    # Path to the self-signed TLS certificate for Node. Expects PEM format.
    # Required field.
    cert: "/opt/xxnetwork/creds/node_cert.crt"
    # Path to the private key for the self signed TLS cert
    # Path to the private key associated with the self-signed TLS certificate.
    # Required field.
    key:  "/opt/xxnetwork/creds/node_key.key"
⋮
gateways:
  paths:
    # Path to the self-signed TLS certificate for Gateway. Expects PEM format.
    # Required field.
    cert: "/opt/xxnetwork/creds/gateway-cert.crt"

permissioning:
  paths:
    # Path to the self-signed TLS certificate for the Permissioning server.
    # Expects PEM format. Required field.
    cert: "/opt/xxnetwork/creds/permissioning_cert.crt"
```

There are three different paths for logs: the normal Node log, the error log, and the metrics logs as well as one for the Node IDF file.

```
node:
  paths:
    # Path where an error file will be placed in the event of a fatal error.
    # This path is used by the Wrapper Script
    errOutput: "/opt/xxnetwork/node-logs/node-err.log"
    # Path where the ID will be stored after the ID is created on first run.
    # This path is used by the Wrapper Script.
    idf:  "/opt/xxnetwork/node-logs/nodeIDF.json"
    # Path where log file will be saved.
    log:  "/opt/xxnetwork/node-logs/node.log"
⋮
metrics:
  # Location of stored metrics data.
  log:  "/opt/xxnetowkr/server-logs/metrics.log"
```

Gateway

To run, Gateway requires four files:

1) The Gateway private key
2) The Gateway TLS certificate
3) The Node TLS certificate
4) The Permissioning TLS certificate

These appear in `gateway.yaml` as follows.

```yaml
# Path to the private key associated with the self-signed TLS certificate.
# Required field.
keyPath: "/opt/xxnetwork/creds/gateway_key.key"

# Path to the self-signed TLS certificate for Gateway. Expects PEM format.
# Required field.
certPath: "/opt/xxnetwork/creds/gateway_cert.crt"

# Path to the self-signed TLS certificate for Server. Expects PEM format.
# Required field.
serverCertPath: "/opt/xxnetwork/creds/node_cert.crt"

# Path to the self-signed TLS certificate for the Permissioning server. Expects
# PEM format. Required field.
permissioningCertPath: "/opt/xxnetwork/creds/permissioning_cert.crt"
```

In addition, Gateway has an optional IDF path field and an optional log field. If no paths are supplied, defaults will be used. When the Gateway runs, it will create files at these paths.

```yaml
# Path where log file will be saved. (default "./gateway-logs/gateway.log")
log: "/opt/xxnetwork/gateway-logs/gateway.log"

# Path to where the IDF is saved. This is used by the wrapper management script.
# (default "./gateway-logs/gatewayIDF.json")
idfPath: "/opt/xxnetwork/gateway-logs/gatewayIDF.json"
```

B. Network Definition File (NDF)

The NDF is a JSON file with a predefined structure that matches the internal `NetworkDefinition` structure.

Some objects on the NDF must have data that matches predefined formats. These are outlined below.

- **`Id`:** must be a byte array 33 bytes long that matches the `id.ID` object. IDs must be generated using Crypto and cannot be created any other way.
- **`Tls_certificate`:** must be a TLS certificate in PEM format. All new lines should be replaced with Unix escape sequence `\n`.

```
{
  "Timestamp": "YYYY-MM-DDTHH:MM:SS.0000000+00:00",
  "Gateways": [
    ⋮
    {
      "Id": "dGVzdDAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAB",
      "Address": "0.0.0.0:80000",
      "Tls_certificate": "-----BEGIN CERTIFICATE-----..."
    },
    ⋮
  [],
  "Nodes": [
    ⋮
    {
      "Id": "dGVzdDMAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAC",
      "Address": "0.0.0.1:40000",
      "Tls_certificate": "-----BEGIN CERTIFICATE-----..."
    },
    ⋮
  ],
  "Registration": {
    "Address": "0.0.0.3:18000",
    "Tls_certificate": "-----BEGIN CERTIFICATE-----..."
  },
  "Notification": {
    "Address": "0.0.0.7",
    "Tls_certificate": "-----BEGIN CERTIFICATE-----..."
  },
  "Udb": {
    "Id": "dGVzdDYAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAD"
  },
  "E2e": {
    "Prime": "E2EE983D031DC1DB6F1A7A67DF0E9A8E5561DB8E...",
    "Generator": "2"
  },
  "Cmix": {
    "Prime": "F6FAC7E480EE519354C0F856AEBDC43AD6014...",
    "Generator": "2"
  }
}
```

Timestamp in RFC3339 format.

Array of `Gateway` objects in the network. Each Gateway has an ID, an address string (containing the IP address and port), and a TLS certificate.

Array of `Node` objects in the network. Each Node has an ID, address, and TLS certificate in the same format as the Gateway object.

The `Registration` field has information about the Permissioning server.

The `Notification` field has information about Notification Bot.

The `Udb` field contains the ID of the User Discovery Bot

Both the `E2e` and `Cmix` fields define the cyclic groups that messaging and end to end encryption operate within. The `E2e` group is based on a 3192-bit strong and safe prime and the `Cmix` group is a 2048 strong and safe prime.

C. Bandwidth Limiting

The cMix protocol uses bandwidth in unique ways. It has short bursts (10~100 milliseconds) of medium-high usage and extended periods (250 milliseconds to 5 seconds) of minimal usage. Overall, at the most taxing configuration testing shows it will saturate 20% to 30% of a 100 Mbps connection.

That 20% to 30% figure is dependent on network conditions and other Nodes it is operating with, but in rare circumstances, it may be 20% to 30% of whatever bandwidth is available. If a Node operator has greater than 100 Mbps available and does not want to allow the network to consume beyond the stated requirements, they can limit bandwidth utilization on the Node.

To do this, Linux traffic control (`tc`) and Hierarchy Token Bucket (HTB), similar to QoS settings on a router, can be used to limit the bandwidth of the Node and/or Gateway to the desired amount. Configuration requires three pieces of information: (1) the port number(s) of the Node and/or Gateway, (2) the network interface name, and (3) the desired bandwidth cap. The following instructions will describe how to get these values and how to configure traffic control.

1. First, get the port number. By default, the port for Node is `11420` and for Gateway it is `22840`. If the port numbers were changed during set up, they can be found in the following locations.

   a. **NODE:** The default location is `/opt/xxnetwork/node.yaml`. The port is under `node` > `port`.

   b. **GATEWAY:** The default location is `/opt/xxnetwork/gateway.yaml`. The port is under `port`.

2. Next, determine the network interface that the machine is operating on. For most machines, the following command should print the currently used network interface.

```
$ ip addr show | awk '/inet.*brd/{print $NF; exit}'
```

   Alternatively, manually find the correct network interface by printing out all available interfaces.

```
$ ip addr
```

   This should result in a similar output to below. Find the name of the network interface that is currently being used. It will have the correct local IP address.

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
qlen 1000
  link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
  inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
  inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
default qlen 1000
  link/ether 00:0c:29:f9:b7:83 brd ff:ff:ff:ff:ff:ff
  inet 192.168.127.138/24 brd 192.168.127.255 scope global dynamic enp1s0
        valid_lft 1574sec preferred_lft 1574sec
  inet6 fe80::20c:29ff:fef9:b783/64 scope link
        valid_lft forever preferred_lft forever
```

Your local IP address.

3. Add the qdisc that will limit the bandwidth for the network interface found.

```
$ sudo tc qdisc add dev [interface name] root handle 1: htb default 30
```

4. Set the max bandwidth for either Node or Gateway. The bandwidth must be set to at least `100Mbit` if running a Node or Gateway. If running them together, Node must be set to a minimum of `100Mbit` and Gateway a minimum of `50Mbit`.

   ⚠ **Warning:** Creating a bandwidth limit lower than specified can result in underperforming and eventual removal from the network.

   For Node:

   ```
   $ sudo tc class add dev [interface name] parent 1:1 classid 1:10 htb rate 100Mbit
   ```

   Make sure to notice this difference.

   For Gateway:

   ```
   $ sudo tc class add dev [interface name] parent 1:1 classid 1:20 htb rate 50Mbit
   ```

5. Set the port number that the bandwidth will be limited on. Make sure to use the correct port found in the previous step.

   For Node:

   ```
   $ sudo tc filter add dev [interface name] parent 1:0 protocol ip prio 1 u32 match
     ip dport [Node port] 0xffff flowid 1:10
   ```

   For Gateway:

   ```
   $ sudo tc filter add dev [interface name] parent 1:0 protocol ip prio 1 u32 match
     ip dport [Gateway port] 0xffff flowid 1:20
   ```

6. To check that the rules were set correctly, list all the classes.

   ```
   $ tc class show dev [interface name]
   ```

   This should print a similar output to below.

   ```
   class htb 1:10 root prio rate 100Mbit ceil 100Mbit burst 15 cburst 15
   class htb 1:20 root prio rate 50Mbit ceil 50Mbit burst 15 cburst 15
   ```

To stop limiting the bandwidth, the rule can be deleted using the following command.

```
$ sudo tc qdisc del dev [interface name] root
```