4. Modify the default queuing discipline to be fq; this is required to use BBR.

```
$ sudo sysctl -w net.core.default_qdisc=fq
```

5. Apply these two options to `sysctl.conf` so they persist on reboot.

```
$ sudo /bin/bash -c 'echo "net.ipv4.tcp_congestion_control=bbr" >> /etc/sysctl.conf'

$ sudo /bin/bash -c 'echo "net.core.default_qdisc=fq" >> /etc/sysctl.conf'
```

Optional Configuration to Make Initial Connection Not Slow

While disabling `tcp_slow_start_after_idle` will keep the congestion windows large after connections are used a few times, you can optionally modify the initial congestion window size using the following instructions to make the minimum congestion window size large enough for cMix batches.

WARNING: Please note that these changes can have negative effects in other areas of network performance, so it is generally not recommended to use these settings and you should only change these options if and only if you are certain that the rest of your network will work with these new settings.

1. First, set congestion windows size on sending and receiving to be 700 times the maximum segment size.

```
$ sudo ip route change `ip route | grep "^default" | head -1` initcwnd 700 initrwnd 700
```

2. Store the above command in `/etc/rc.local`, to make it run on boot.

```
$ sudo /bin/bash -c 'echo -e "#!/bin/bash\nip route change \`ip route | grep
  \"^default\" | head -1\` initcwnd 700 initrwnd 700\nexit 0" > /etc/rc.local' &&
  sudo chmod +x /etc/rc.local
```

Note, if this fails to work or causes problems, such as increased timeouts or instability, remove the file and reboot.

```
$ sudo rm /etc/rc.local && sudo reboot
```

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 its drivers.

1. Install the Nvidia driver.

```
$ sudo apt install -y nvidia-driver-460
```

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
```

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 1.16 is required for both Server and Gateway.

1. Download and install the latest version of Go version 1.16. 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
   ```

Node

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 the CUDA Toolkit 11.2. 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 https://developer.download.nvidia.com/compute/cuda/11.2.2/local_installers/\
  cuda-repo-ubuntu1804-11-2-local_11.2.2-460.32.03-1_amd64.deb
$ sudo dpkg -i cuda-repo-ubuntu1804-11-2-local_11.2.2-460.32.03-1_amd64.deb
$ sudo apt-key add /var/cuda-repo-ubuntu1804-11-2-local/7fa2af80.pub
$ sudo apt-get update
$ sudo apt-get -y install cuda
```

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

```
$ sudo reboot now
```

3. Next, install `libgmp-dev`, which is a dependency the GPU code needs to compile.

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

4. 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 https://gitlab.com/elixxir/gpumathsnative.git
$ pushd gpumathsnative/cgbnBindings/powm
$ make fatbin
$ sudo make install
$ popd
```

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

5. Download the latest master branch of Server.

6. 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
```

7. 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 1.16 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 the latest version of Go version 1.16. Refer to the Go install page for more information.

   ```
   $ curl -L -O https://golang.org/dl/go1.16.4.linux-amd64.tar.gz
   ```

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

   ```
   $ sudo tar -C /usr/local -xzf go1.16.4.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 go1.16.4 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 or xx network GitLab page and select a repository to download. Alternatively, select one from the following list of public repositories:
   - `primitives`
   - `crypto`
   - `comms`
   - `gpumathsgo`
   - `server`
   - `gateway`
   - `client`
   - `user-discovery-bot`
   - `localenvironment`
   - `integration`
   - `wrapper`

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

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

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

```
$ git clone -b master https://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 `-run` flag.

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

Setting Up Node for Testing

To run some of the tests on Node, 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. First, install the CUDA Toolkit 11.2. 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 https://developer.download.nvidia.com/compute/cuda/11.2.2/local_installers/\
  cuda-repo-ubuntu1804-11-2-local_11.2.2-460.32.03-1_amd64.deb
$ sudo dpkg -i cuda-repo-ubuntu1804-11-2-local_11.2.2-460.32.03-1_amd64.deb
$ sudo apt-key add /var/cuda-repo-ubuntu1804-11-2-local/7fa2af80.pub
$ sudo apt-get update
$ sudo apt-get -y install cuda
```

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

```
$ sudo reboot now
```

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

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

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

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

6. 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
```

7. When running tests with GPU on Node, 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, which must be explicitly specified when running the software.

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 (†).

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

permissioning:
  paths:
    # Path to the self-signed TLS certificate for the Permissioning server. Expects PEM
    # format. (Required)
    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. (Required)
    errOutput: "/opt/xxnetwork/node-logs/node-err.log"
    # Path to where the identity file (IDF) is saved. The IDF stores the Node's network
    # identity. This is used by the wrapper management script. (Required)
    idf: "/opt/xxnetwork/node-logs/nodeIDF.json"
    # Path where log file will be saved.
    log: "/opt/xxnetwork/node-logs/node.log"
⋮
metrics:
  # Path to store metrics logs.
  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.

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

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

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

# Path to the self-signed TLS certificate for the Permissioning server. Expects PEM format.
# (Required)
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.

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

# Path to where the identity file (IDF) is saved. The IDF stores the Gateway's Node's
# network identity. This is used by the wrapper management script. (Required)
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": "dGVzdDAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAB",
      "Address": "0.0.0.0:80000",
      "Tls_certificate": "-----BEGIN CERTIFICATE-----..."
    },
    ⋮
  [],
  "Nodes": [
    ⋮
    {
      "Id": "dGVzdDMAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAC",
      "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": "dGVzdDYAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAD"
    "Cert": "-----BEGIN CERTIFICATE-----...",
    "Address": "0.0.0.52:18001",
    "DhPubKey": "eyJWYWx1ZSI6NTEwOTQzODAxNTcyOTQ4NjUzN...",
  },
  "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, certificate, address, and Diffie–Hellman key of User Discovery

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                          Your local IP address.
  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
```

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
```