Отчёт по проекту

Constella: одноранговая сеть VPS-узлов с VPN-overlay

Авторы: Поповиченко Роман, Василенко Михаил, Охрименко Михаил

16 ноября 2025 г.

Содержание

1 Введение

Цель работы — описать и продемонстрировать работу проекта **Constella** и показать, как существующая децентрализованная P2P-сеть VPS-узлов дополняется простым VPN-overlay на базе **WireGuard**. Отдельно рассматриваются вопросы развёртывания кластера, выбора лидера, работы Telegram-бота и интеграции туннеля для узлов, находящихся за NAT без проброса портов.

В практической части:

- разворачивается сеть из двух публичных узлов: `netherlands-01` и `usa-01`;
- демонстрируется автоматический выбор лидера и работа Telegram-бота;
- показывается, что узел за NAT без VPN не может стать полноценным участником кластера;
- конфигурируется WireGuard-хаб на `netherlands-01` и клиент на `usa-01`;
- на основе команд `ip a show`, `wg show` и `ping` доказывается, что межузловой трафик реально идёт через VPN-туннель;
- обсуждается, как этот же механизм используется для домашних машин без белого IP.

Исходный код и Docker-конфигурация проекта находятся в репозитории: https://github.com/Rmgs123/Constella_Plus. Далее репозиторий будем обозначать просто **Constella**.

2 Обзор архитектуры Constella

2.1 Назначение проекта

Constella — это одноранговая сеть VPS-узлов с автоматическим выбором лидера и Telegram-ботом. Каждый сервер является равноправным участником сети; лидер определяется детерминированно среди “живых” узлов, а Telegram-бот всегда работает только на одной ноде (у владельца лидирующего узла).

Основные цели:

- Простое объединение нескольких серверов в децентрализованный кластер без единого центра.
- Автоматический выбор и переезд лидера при отказах.
- Управление серверами (просмотр метрик, перезагрузка, спидтест) через Telegram-бота.
- Возможность использовать VPN-overlay для соединения узлов за NAT.

2.2 P2P-членство, heartbeat и выбор лидера

Каждый узел хранит локальное состояние сети в файле `state/network_state.json`. В нём содержатся:

- идентификатор сети `network_id`;
- имя владельца `owner_username`;
- общий секрет `network_secret`;
- список узлов (`peers`) с полями `name`, `addr`, `node_id`, `status`, `last_seen`;
- структура `bot_lease`, определяющая, какой узел сейчас является владельцем бота и до какого момента действует "аренда".

Каждый узел периодически отправляет heartbeat остальным участникам и обновляет поле `last_seen`. Узлы, которые долго не отвечают, помечаются как `down`. Лидер выбирается детерминированно по ключу вида `LeaderKey = (priority, node_id)` среди живых узлов. При отказе текущего лидера происходит переизбрание.

2.3 Telegram-бот и bot-lease

Telegram-бот запускается только на лидере. Это обеспечивается механизмом `bot_lease` в `network_state.json`: бот периодически продлевает "аренду" (lease) и, если lease истёк или лидер сменился, следующий узел берёт на себя роль хоста бота.

С точки зрения пользователя, достаточно:

- указать `BOT_TOKEN` и `@owner_username` при `init` или `join`;
- запустить Docker-контейнеры;
- дальше бот автоматически переезжает между узлами при смене лидера.

2.4 VPN-overlay на основе WireGuard

В проект добавлен VPN-overlay, который используется для межузлового трафика Constella. Поведение задаётся переменными окружения:

- `VPN_MODE = none | hub | client`;
- `VPN_INTERFACE_NAME` (по умолчанию `wg0`);
- `VPN_CIDR` (по умолчанию `10.42.0.0/24`);
- `VPN_LISTEN_PORT` — порт WireGuard на хабе;
- `VPN_HUB_ENDPOINT`, `VPN_HUB_PUBLIC_KEY`, `VPN_ADDRESS` — параметры клиентского туннеля.

Логика следующая:

1. Если `VPN_MODE=hub`, `install.sh` генерирует конфигурацию WireGuard, поднимает интерфейс и назначает хабу адрес из `VPN_CIDR`.

2. Если `VPN_MODE=client`, скрипт создаёт клиентский конфиг, назначает IP из той же подсети и поднимает интерфейс `wg0`.

3. Приложение пытается получить IP на `wg0` из подсети `VPN_CIDR`. Если всё успешно, **публикуемый адрес узла** (`PUBLIC_ADDR`) автоматически выставляется в виде `<vpn_ip>:4747`. В противном случае используется обычный публичный адрес.

Таким образом, для Constella все узлы оказываются в одной приватной сети `10.42.0.0/24`, независимо от того, есть ли у них белый IP.

3 Развёртывание кластера без VPN

3.1 Инициализация первого узла `netherlands-01`

Первый узел — VPS с белым IP (Нидерланды). Для него выполняются команды:

Листинг 1: Клонирование репозитория и инициализация сети на netherlands-01

```
git clone https://github.com/Rmgs123/Constella.git
cd Constella
chmod +x install.sh

./install.sh init \
  --server-name netherlands-01 \
  --public-addr 178.255.222.XXX:4747 \
  --owner @Rmgss \
  --bot-token 8221870569:XXXXXXXXXXXXXXX
```

Скрипт инициализации создаёт `.env` и начальное состояние сети в `state/network_state.json`. В выводе видим финальную строку:

Листинг 2: Фрагмент вывода init-скрипта

```
Init ready. Run: docker compose up -d --build
```

После этого запускаем контейнер:

Листинг 3: Запуск Docker-контейнеров на netherlands-01

```
docker compose up -d --build
docker compose logs -f
```

На этом этапе кластер состоит из одной ноды, Telegram-бот уже доступен и работает на лидере (им является `netherlands-01`).

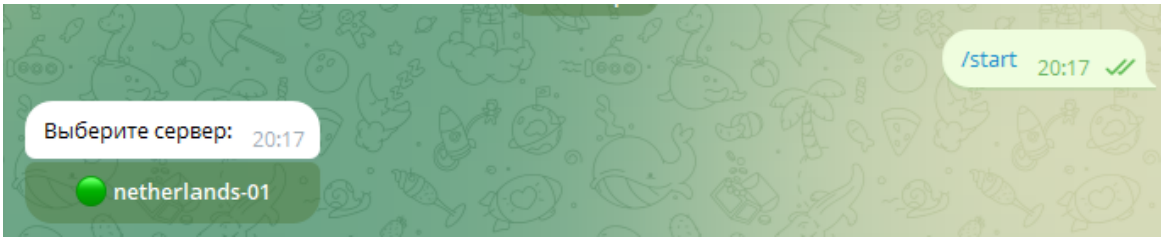

Рис. 1: Интерфейс Telegram-бота после инициализации единственного узла `netherlands-01`

3.2 Подключение второго узла `usa-01`

На втором сервере (также VPS с белым IP) выполняются аналогичные шаги: клонирование репозитория, выдача прав и `join` по пригласительной ссылке.

Листинг 4: Подключение узла usa-01 к сети

```
git clone https://github.com/Rmgs123/Constella.git
cd Constella
chmod +x install.sh

./install.sh join \
  --server-name usa-01 \
  --join "join://178.255.222.176:4747?net=ebf440ec2e29db307bf2fbbe9326d2ce&token=
     B9jMNOG3xkdyILjxokW9qQ&ttl=900s"
```

Скрипт спрашивает подтверждение имени и токен бота, после чего выводит приглашение запустить Docker:

Листинг 5: Фрагмент вывода join-скрипта на usa-01

```
Owner (@username): @Rmgss
Bot token (optional, Enter to skip): 8221870569:XXXXXXXXXXXXXXX

Join ready. Run: docker compose up -d --build
```

Далее:

Листинг 6: Запуск контейнеров на usa-01

```
docker compose up -d --build
docker compose logs -f
```

После запуска оба узла появляются в Telegram-боте, один из них становится лидером (хостом). Можно протестировать команды вроде просмотра статистики, спидтеста и перезагрузки узла.

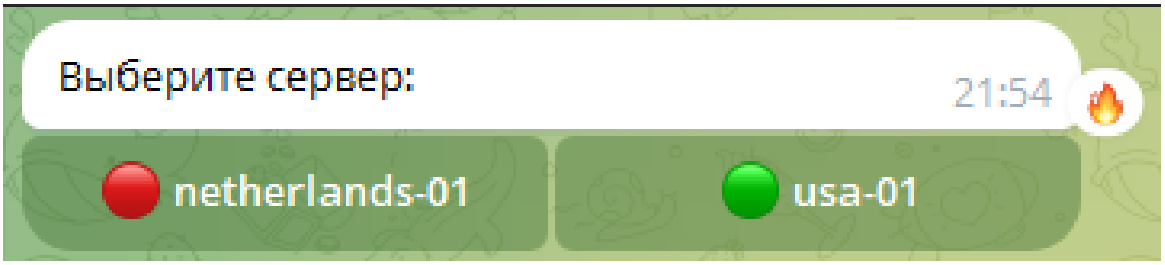

Рис. 2: Список узлов в боте: `netherlands-01` и `usa-01`

Перезагрузка узла запускается через бот, а на сервере виден соответствующий лог (`reboot` и отключение контейнера).

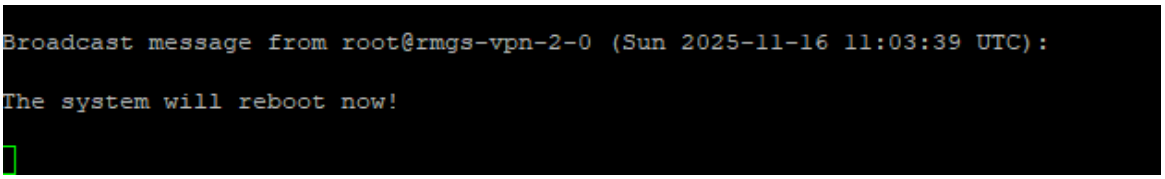

Рис. 3: Пример вывода логов при перезагрузке узла по команде из бота

3.3 Редактирование `network_state.json` при ошибках

Если при экспериментах были добавлены “лишние” ноды или возникли неинициализированные записи (например, с пустым `node_id`), их можно удалить вручную:

Листинг 7: Ручная правка состояния сети

```
docker compose down
nano state/network_state.json

docker compose up -d
docker compose logs -f
```

4 Узел за NAT без VPN: ограничения

Далее предпринимается попытка установить Constella на локальную систему, находящуюся за роутером без проброса портов и без VPN. Процедура похожа: клонирование репозитория и запуск `install.sh join` с join-ссылкой на один из публичных узлов.

Эксперимент показывает, что без доступного снаружи IP/порта и без VPN-overlay такой узел не может стать полноценной частью существующей сети: создаётся отдельная локальная сеть, которая не видна остальным участникам.

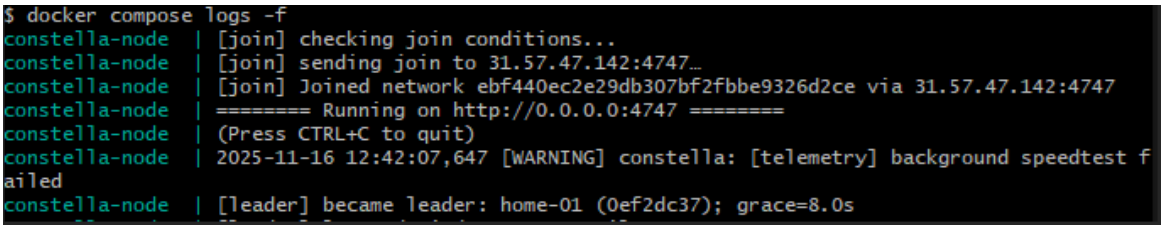

Рис. 4: Попытка подключения домашнего узла без проброса портов — создаётся отдельная сеть

Это ожидаемое поведение: в P2P-сети без дополнительных механизмов для NAT (traversal, relay и т.п.) хотя бы один узел должен иметь публичный адрес, чтобы остальные могли к нему “приклеиться”.

5 Развёртывание VPN-overlay WireGuard

Чтобы соединить узлы, находящиеся за NAT, и при этом не модифицировать основную P2P-логику Constella, используется VPN-overlay на основе WireGuard.

5.1 Очистка старой конфигурации

Перед началом эксперимента с VPN на обоих серверах удаляется старая конфигурация:

Листинг 8: Полная очистка прежнего развёртывания на узле

```
cd ~/Constella
docker compose down -v
cd ~
rm -rf ~/Constella
docker system prune -a --volumes -f
```

5.2 Настройка VPN-хаба на `netherlands-01` (режим hub)

На нидерландском сервере снова клонируем репозиторий и выполняем `init` уже с параметрами VPN-хаба:

Листинг 9: Инициализация сети с VPN-хабом на netherlands-01

```
git clone https://github.com/Rmgs123/Constella.git
cd Constella
chmod +x install.sh

./install.sh init \
  --server-name netherlands-01 \
  --public-addr 178.255.222.176:4747 \
  --owner @Rmgss \
  --bot-token 8221870569:XXXXXXXXXXXXXXX \
  --vpn-mode hub \
  --vpn-cidr 10.42.0.0/24 \
  --vpn-listen-port 51820
```

В выводе инициализации видно, что создаётся конфигурация WireGuard и поднимается интерфейс `wg0`:

Листинг 10: Фрагмент вывода init-хаба WireGuard

```
WireGuard hub config written to state/wg/wg0.conf
Hub public key: rBSKi3HhkkLoQCqytEUu7zTim+ul0G9sEd6OgurRQ28=
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip -4 address add 10.42.0.1/24 dev wg0
[#] ip link set mtu 1420 up dev wg0
WireGuard interface wg0 ready at 10.42.0.1/24
Share the hub public key and allocate client IPs from 10.42.0.0/24 for NAT clients.

Init ready. Run: docker compose up -d --build
```

Далее хаб поднимается как обычный узел Constella:

Листинг 11: Запуск контейнеров на VPN-хабе

```
docker compose up -d --build
docker compose logs -f
```

5.3 Подготовка клиента `usa-01` (режим client)

На сервере `usa-01` также очищается среда, устанавливается WireGuard, а затем разворачивается Constella в режиме клиента:

Листинг 12: Подготовка узла usa-01 и установка WireGuard

```
cd ~
rm -rf ~/Constella
docker system prune -a --volumes -f

sudo apt update
sudo apt install -y wireguard wireguard-tools

wg --version
# wireguard-tools v1.0.20210914 - https://git.zx2c4.com/wireguard-tools/
```

Далее:

Листинг 13: Подключение узла usa-01 как VPN-клиента

```
git clone https://github.com/Rmgs123/Constella.git
cd Constella
chmod +x install.sh

./install.sh join \
  --server-name usa-01 \
  --join "join://178.255.222.176:4747?net=fe361bffa331477e63553ffe5d465749&token=
      G2BEeXVljMWJxNDGIyVkKg&ttl=900s" \
  --vpn-mode client \
  --vpn-address 10.42.0.2/32 \
  --vpn-hub-endpoint 178.255.222.176:51820 \
  --vpn-hub-public-key rBSKi3HhkkLoQCqytEUu7zTim+ul0G9sEd6OgurRQ28=
```

Вывод:

Листинг 14: Создание клиентского конфига и поднятие wg0 на usa-01

```
Owner (@username): @Rmgss
Bot token (optional, Enter to skip): 8221870569:XXXXXXXXXXXXXXX

WireGuard client config written to state/wg/wg0.conf
Client public key: Mlkg6owvnGklUJnWraDLEJpvsNOHIs9J3teLc0RgaAA=
Provide this public key to the hub and ensure it is added as a peer before bringing
    the tunnel up.
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip -4 address add 10.42.0.2/32 dev wg0
[#] ip link set mtu 1420 up dev wg0
[#] ip -4 route add 10.42.0.0/24 dev wg0
WireGuard interface wg0 ready at 10.42.0.2/32

Join ready. Run: docker compose up -d --build
```

После этого на `usa-01` поднимаем контейнеры:

Листинг 15: Запуск Constella на VPN-клиенте

```
docker compose up -d --build
docker compose logs -f
```

5.4 Добавление клиента в конфиг хаба и перезапуск туннеля

На стороне хаба нужно добавить клиента в файл `state/wg/wg0.conf` в виде нового блока `[Peer]`:

Листинг 16: Добавление клиента в конфиг WireGuard на хабе

```
nano state/wg/wg0.conf


[Interface]
PrivateKey = sPl8TNS...
Address = 10.42.0.1/24
ListenPort = 51820
SaveConfig = false

# Add Peer
[Peer]
PublicKey = Mlkg6owvnGklUJnWraDLEJpvsNOHIs9J3teLc0RgaAA=
AllowedIPs = 10.42.0.2/32
```

Перезапускаем интерфейс на хабе:

Листинг 17: Перезапуск WireGuard-интерфейса на хабе

```
wg-quick down state/wg/wg0.conf 2>/dev/null
wg-quick up state/wg/wg0.conf

wg show
```

5.5 Проверка PUBLIC_ADDR и работы кластера

После успешного поднятия туннеля проверяем, что на `usa-01` окружение настроено корректно:

Листинг 18: Проверка и правка .env на usa-01

```
nano .env

# Example:
SERVER_NAME=usa-01
LISTEN_ADDR=0.0.0.0:4747
PUBLIC_ADDR=10.42.0.2:4747 # VPN-adress is needed

VPN_MODE=client
VPN_INTERFACE_NAME=wg0
VPN_CIDR=10.42.0.0/24
VPN_HUB_ENDPOINT=178.255.222.176:51820
VPN_ADDRESS=10.42.0.2/32
VPN_ALLOWED_IPS=10.42.0.0/24
```

Если `PUBLIC_ADDR` был пуст, нужно его дописать, затем перезапустить контейнеры:

Листинг 19: Перезапуск после правки окружения

```
docker compose down
docker compose up -d --build
docker compose logs -f
```

После этого `usa-01` становится полноценным участником сети, может быть выбран лидером (хостом), а все P2P-запросы Constella идут по адресу `10.42.0.2:4747` поверх VPN.

6 Доказательство работы туннеля WireGuard

Чтобы убедиться, что используется именно VPN-туннель, а не прямое соединение по внешним IP, выполняются команды `ip a show`, `wg show` и `ping`.

6.1 Интерфейс wg0 на обоих узлах

На хабе:

Листинг 20: Интерфейс wg0 на netherlands-01

```
root@rmgs-vpn-2-0:~/Constella# ip a show wg0
23: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state UNKNOWN group
    default qlen 1000
    link/none
    inet 10.42.0.1/24 scope global wg0
       valid_lft forever preferred_lft forever
```

На клиенте:

Листинг 21: Интерфейс wg0 на usa-01

```
root@usa:~/Constella# ip a show wg0
32: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state UNKNOWN group
    default qlen 1000
    link/none
    inet 10.42.0.2/32 scope global wg0
       valid_lft forever preferred_lft forever
```

6.2 Состояние туннеля по `wg show`

На хабе:

Листинг 22: Состояние туннеля wg0 со стороны хаба

```
root@rmgs-vpn-2-0:~/Constella# wg show

interface: wg0
  public key: rBSKi3HhkkLoQCqytEUu7zTim+ul0G9sEd6OgurRQ28=
  private key: (hidden)
  listening port: 51820

peer: Mlkg6owvnGklUJnWraDLEJpvsNOHIs9J3teLc0RgaAA=
  endpoint: 31.57.47.142:51227
  allowed ips: 10.42.0.2/32
  latest handshake: 3 seconds ago
  transfer: 308.51 KiB received, 395.46 KiB sent
```

На клиенте:

Листинг 23: Состояние туннеля wg0 со стороны клиента

```
root@usa:~/Constella# wg show

interface: wg0
  public key: Mlkg6owvnGklUJnWraDLEJpvsNOHIs9J3teLc0RgaAA=
  private key: (hidden)
  listening port: 51227

peer: rBSKi3HhkkLoQCqytEUu7zTim+ul0G9sEd6OgurRQ28=
  endpoint: 178.255.222.176:51820
  allowed ips: 10.42.0.0/24
  latest handshake: 5 seconds ago
  transfer: 395.71 KiB received, 307.37 KiB sent
  persistent keepalive: every 25 seconds
```

Мы видим:

- на каждом конце интерфейс `wg0` имеет свой IP из подсети `10.42.0.0/24`;
- `allowed ips` настроены так, что только адреса этой подсети маршрутизируются через туннель;
- есть успешные handshake и обмен трафиком (поля `transfer`).

6.3 Пинг по VPN-адресу

Наконец, проверяем связность по VPN-адресам:

Листинг 24: Пинг VPN-адреса клиента с хаба

```
root@rmgs-vpn-2-0:~/Constella# ping -c 4 10.42.0.2
PING 10.42.0.2 (10.42.0.2) 56(84) bytes of data.
64 bytes from 10.42.0.2: icmp_seq=1 ttl=64 time=170 ms
64 bytes from 10.42.0.2: icmp_seq=2 ttl=64 time=164 ms
64 bytes from 10.42.0.2: icmp_seq=3 ttl=64 time=171 ms
64 bytes from 10.42.0.2: icmp_seq=4 ttl=64 time=173 ms

--- 10.42.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 164.220/169.606/173.471/3.408 ms
```

Так как адрес `10.42.0.2` принадлежит туннельной подсети и маршрутизируется через `wg0`, фактически пакеты идут по WireGuard-туннелю. Это и есть доказательство того, что межузловое взаимодействие происходит не непосредственно по публичным IP, а поверх приватного VPN-overlay.

7 Использование туннеля для узлов за NAT

В приведённом эксперименте в качестве клиента выступает VPS `usa-01`, но ровно тот же сценарий применим к **домашнему Linux-пк за NAT**:

- роль хаба сохраняется за публичным сервером (`netherlands-01`), на котором поднят `wg0` с адресом `10.42.0.1/24`;
- вместо `usa-01` используется домашняя машина (например, `home-01`) с установленными `wireguard-tools` и Docker;
- выполняется `install.sh join` с флагами `-vpn-mode client`, `-vpn-address 10.42.0.X/32`, `-vpn-hub-endpoint` и `-vpn-hub-public-key`, как и в случае VPS;
- на хабе добавляется соответствующий `[Peer]` в `wg0.conf` и перезапускается интерфейс;
- в `.env` домашнего узла `PUBLIC_ADDR` выставляется на `10.42.0.X:4747`.

После этого:

1. Домашний узел, находящийся за NAT без проброса портов, становится доступен в приватной сети `10.42.0.0/24`.
2. Constella использует VPN-адрес как `PUBLIC_ADDR`, и все RPC/heartbeat идут через туннель.
3. Указанный узел может быть выбран лидером и полностью участвовать в децентрализованной сети.

Таким образом, VPN-overlay решает главную проблему: узлы без белого IP и без возможности проброса портов получают доступ к общему кластеру, при этом основная логика Constella (P2P-членство, выбор лидера, работа бота) не требует изменений и просто работает поверх новой транспортной “прослойки”.

8 Функциональность Telegram-бота и сбор метрик

Важная часть проекта — Telegram-бот, который служит единым интерфейсом для наблюдения за состоянием узлов и выполнения управляющих действий. Ниже приведены основные экраны и функции бота, а также то, как они связаны с реализованной в коде логикой сбора и агрегации метрик.

Рис. 5: Использование /invite

Бот выводит join-ссылку с ttl (time to live) = 900 секунд (15 минут) по дефолту.

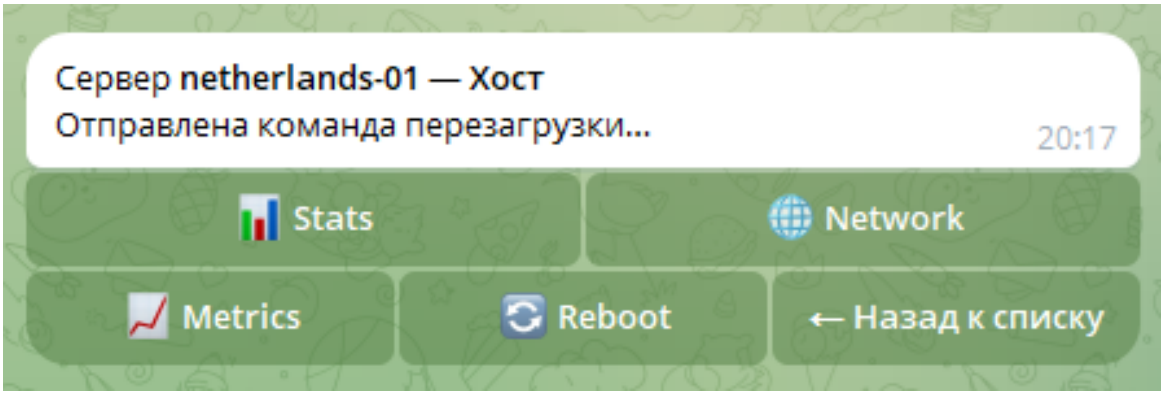

Рис. 6: Выбор конкретного узла и отображение его статуса: имя, роль (лидер/обычный узел) и основное меню действий (после отправки команды на перезагрузку)

После выбора узла бот строит карточку сервера: отображает имя, статус, пометку «Хост» для лидера (определяется функцией `current_leader()` в `app.py`) и инлайн-кнопки для перехода к метрикам, сетевым тестам и перезагрузке.

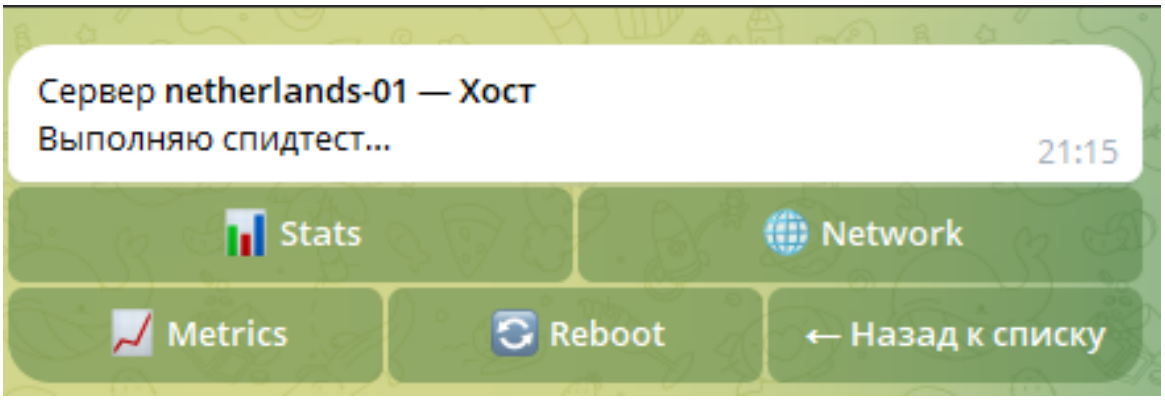

Рис. 7: Экран `Network`: запуск спидтеста

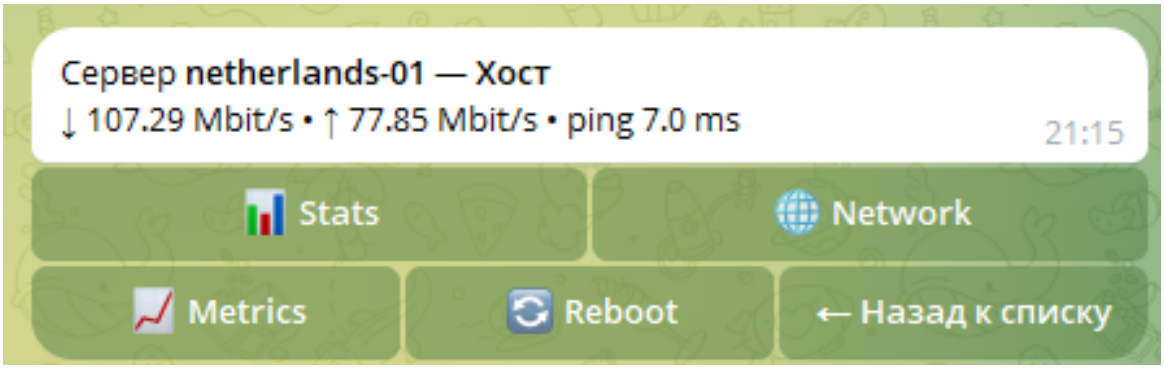

Рис. 8: Экран `Network`: вывод измеренной скорости загрузки/отправки и задержки

На экране `Network` вызывается RPC `RunSpeedtest`. Внутри контейнера запускается `speedtest-cli` (через `run_in_executor`), а результаты (скорость загрузки/отдачи и ping) временно сохраняются в локальные тайм-серии `NET_DOWN_SAMPLES` и `NET_UP_SAMPLES` для дальнейшего анализа.

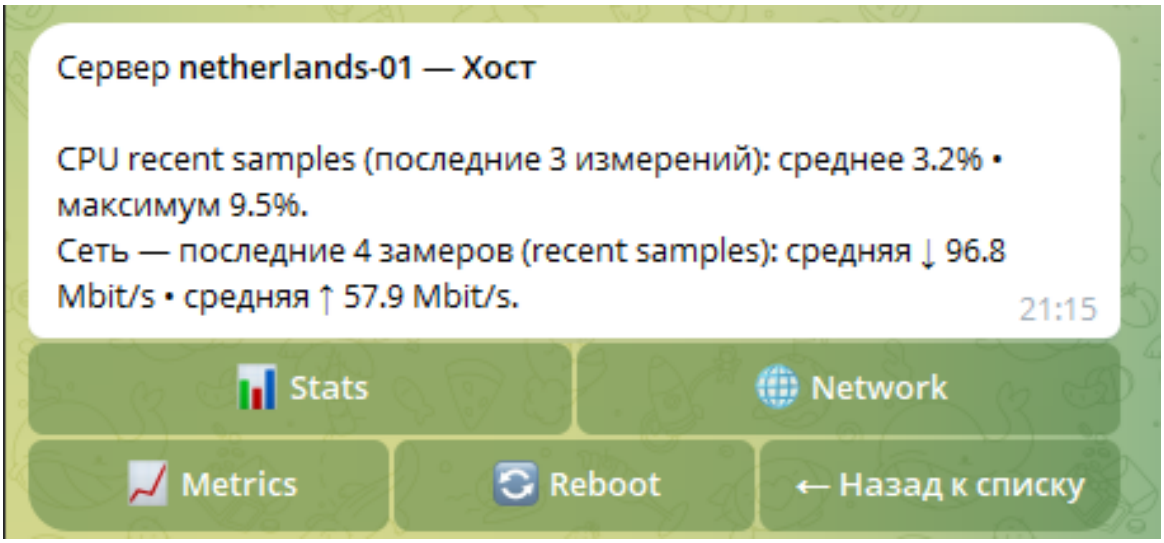

Рис. 9: Экран `Metrics`: компактная текстовая сводка по истории загрузки CPU и сети на основе тайм-серий

Экран `Metrics` использует RPC `GetTS` для запроса тайм-серий по CPU и сети. На стороне узлов циклы телеметрии (`telemetry_loop`) периодически добавляют точки в деки `CPU_SAMPLES`, `NET_DOWN_SAMPLES` и `NET_UP_SAMPLES`. Бот забирает последние значения (функции `summarize_series_points()` и `summarize_net_points()`), вычисляет средние и максимальные значения и показывает их текстом в одном сообщении.

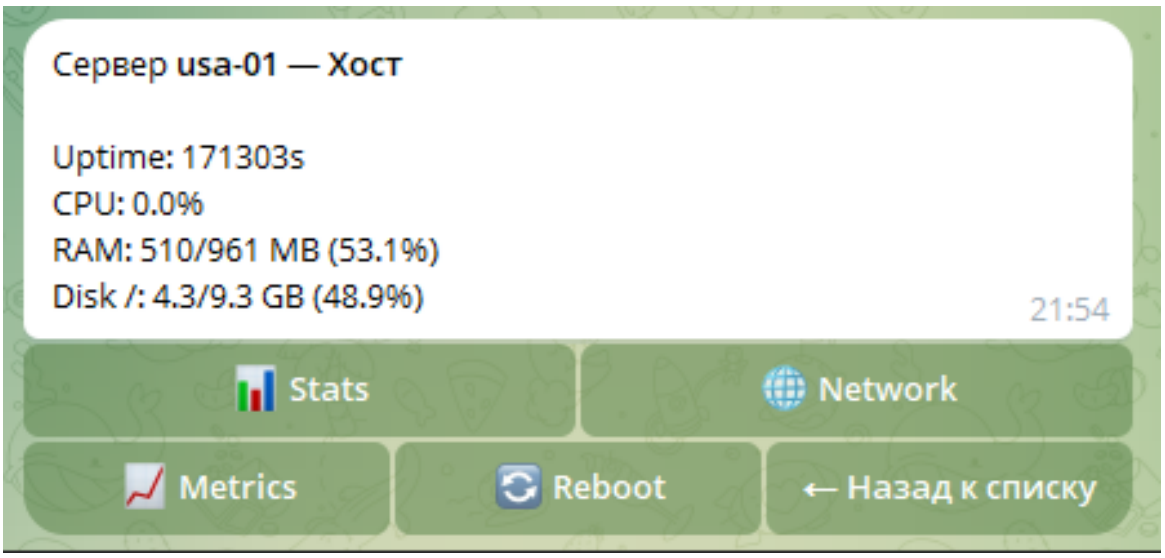

Рис. 10: Использоение /stats

Кнопка `Stats` отправляет RPC-запрос `GetStats` на целевой узел. Обработчик на стороне узла использует библиотеку `psutil` для получения нагрузки по ядрам процессора, состояния памяти и диска, после чего бот форматирует ответ в виде текстового отчёта.

9 Заключение

В отчёте продемонстрировано:

- как развёрнуть базовый кластер Constella из двух публичных VPS;
- как работает P2P-членство, хранение состояния и выбор лидера;
- как через Telegram-бот можно управлять узлами и наблюдать за их состоянием;
- почему узел за NAT без VPN не может подключиться к уже существующей сети;
- как с помощью WireGuard настроить VPN-hub, клиент и перенести межузловой трафик Constella в приватную подсеть;
- каким образом туннель подтверждается командами `ip a show`, `wg show` и `ping`;
- как этот подход позволяет запускать узлы Constella на домашних системах без белого IP и проброса портов.