1 Branch 3 Tags Go to file Go to file Code

ingon use consistent client D/S naming in the Readme 833f1a3 · 11 hours ago

| | | |
|---|---|---|
| certc | more value serialization | 2 weeks ago |
| client | migrate to new org | 4 days ago |
| cmd/connet | fix error checks | 2 days ago |
| control | fix error checks | 2 days ago |
| examples | update README | 3 days ago |
| logc | cleanup control servers | 2 weeks ago |
| model | migrate to new org | 4 days ago |
| netc | fix error checks | 2 days ago |
| notify | specialized kv store; move to s… | 3 weeks ago |
| pb | migrate to new org | 4 days ago |
| pbc | migrate to new org | 4 days ago |
| pbr | migrate to new org | 4 days ago |
| pbs | migrate to new org | 4 days ago |
| relay | fix error checks | 2 days ago |
| selfhosted | migrate to new org | 4 days ago |
| .envrc | add nix package build | 2 months ago |
| .gitignore | release script | 4 days ago |
| LICENSE | Create LICENSE | 2 months ago |
| Makefile | migrate to new org | 4 days ago |
| README.md | use consistent client D/S namin… | 11 hours ago |
| client.go | require at least one destination … | 2 days ago |
| default.nix | use str instead of string for opti… | 2 days ago |
| e2e_test.go | migrate to new org | 4 days ago |
| flake.lock | update nixpkgs | last month |
| flake.nix | move module to top-level | 2 days ago |
| go.mod | migrate to new org | 4 days ago |
| go.sum | fix nix package | 2 weeks ago |
| package.nix | update readme | 2 days ago |
| process-compose.yaml | fix makefiles and process | 2 months ago |

## About

A p2p reverse proxy with NAT traversal. Inspired by frp, rathole and ngrok

#go #tunnel #proxy #udp #firewall #nat #p2p #reverse-proxy #frp #quic #rathole

Readme
Apache-2.0 license
Activity
Custom properties
**146** stars
**2** watching
**2** forks
Report repository

## Releases 3

**v0.3.0** Latest
16 hours ago

+ 2 releases

## Packages

No packages published

## Languages

**Go** 94.9% **Nix** 3.9%
**Makefile** 1.2%

| server.go | fix error checks | 2 days ago |
|---|---|---|

README Apache-2.0 license

# connet

release v0.3.0 reference go report A+ license Apache2.0

`connet` is a peer-to-peer reverse proxy for NAT traversal. It is inspired by ngrok, frp, rathole and others.

`connet` helps expose a service running on a device to another device on the internet. Unlike the others, the `connet` client runs on both the device that exposes the service (called `destination` in connet's terms) and the device that wants to access the service (called `source`). This means that communication between `connet` clients is never public and visible to the rest of the internet, and in many cases peers can communicate directly.

> **Status** `connet` is currently alpha software. We expect some issues and its APIs are subject to change.

## Features

- **Peer-to-peer communication** Because you run the `connet` client on both the `destination` and the `source`, the server is only used for sharing configuration. In many cases clients can communicate directly, which enables better privacy and performance.
- **Relay support** There are cases when clients are unable to find a path to communicate directly. In such cases, they can use a relay server to maintain connectivity.
- **Security** Everything is private, encrypted with TLS. Public server and client certificates are exchanges between peers and are required and verified to establish connectivity. Clients and relays need to present a mandatory token when communicating with the control server, allowing tight control over who can use `connet`.
- **Embeddable** In case you want `connet` running as part of another (golang) program (as opposed to a separate executable), `connet` has a well defined api for running both the client and the server.

## Architecture

For all communication `connet` uses the QUIC protocol, which is build on top of UDP.

## Quickstart

Latest builds of `connet` can be acquired from our releases page. If you are using NixOS, check also the NixOS section.

To get started with `connet`, you'll need 3 devices:

- Server which your clients can communicate with. In most cases, this server will have a public IP and be directly accessible by clients. A VPS instance at one of the many cloud providers goes a long way here.
- Device `D` that has the `destination` service you want to connect to, running at port `3000`.
- Device `S` (aka `source`) which you want to connect to the service, at port `8000`.

### Server

In the setup above, start `connet server --config server.toml` with the following `server.toml`:

```
[server]
tokens = ["client-d-token", "client-s-token"]
```

```
cert-file = "cert.pem"
key-file = "key.pem"
```

#### TLS Certificates

To run a `connet` server, you'll need a TLS certificate. You have a few options to create such certificate:

- **Recommended** use an ACME client to provision one for you. We've had good experiences running lego.
- Buy a TLS certificate from a Certificate Authority like verisign, namecheap, etc.
- Use a self-signed TLS certificate, an option most appropriate for testing.

To create a self-signed certificate, you can use openssl. Alternatively, you can use a tool like minica. When using self-signed certificate, you'll need your clients (and relays) trusting the server's certificate. Copying the certificate (or CA) public key to the clients and using `server-cas` configuration option is the easiest way to achieve this.

### Client D (aka the `destination`)

Then, on device `D` run `connet --config client-d.toml` with the following `client-d.toml`:

```
[client]
token = "client-d-token"
server-addr = "SERVER_IP:19190"
server-cas = "cert.pem"

[client.destinations.serviceA]
addr = ":3000"
```

### Client S (aka the `source`)

On device `S` run `connet --config client-s.toml` with the following `client-s.toml`:

```
[client]
token = "client-s-token"
server-addr = "SERVER_IP:19190"
server-cas = "cert.pem"

[client.sources.serviceA]
addr = ":8000"
```

## Configuration

You can use both a toml config file as well as command line when running `connet`. If you use both a config file and command line options, the latter takes precence, overriding any config file options. For simplicity, command line options only support a single `destination` or `source` configuration.

### Client

To run in client mode, use `connet --config client-config.toml` command. Here is the full client `client-config.toml` configuration spec:

```
[client]
token = "client-token-1" # the token which the client uses to authenticate against the control server
token-file = "path/to/relay/token" # a file that contains the token, one of token or token-file is required

server-addr = "localhost:19190" # the control server address to connect to
server-cas = "path/to/cert.pem" # the control server certificate
direct-addr = ":19192" # at what address this client listens for direct connections

[client.destinations.serviceX]
addr = "localhost:3000" # where this destination connects to, required
route = "any" # what kind of routes to use, `any` will use both `direct` and `relay`

[client.destinations.serviceY]
addr = "192.168.1.100:8000" # multiple destinations can be defined, they are matched by name at the server
```

```
route = "direct" # force only direct communication between clients

[client.sources.serviceX] # matches destinations.serviceX
addr = ":8000" # the address at which to listen for incoming connections to be forwarded
route = "relay" # the kind of route to use

[client.sources.serviceY] # both sources and destinations can be defined in a single file
addr = ":8001" # again, mulitple sources can be defined
route = "direct" # force only direct communication between clients, even if other end allows any
```

## Server

To run in server mode (e.g. running both control and a relay server), use `connet server --config server-config.toml` command. Here is the full server `server-config.toml` configuration specification:

```
[server]
tokens = ["client-token-1", "client-token-n"] # set of recognized client tokens
tokens-file = "path/to/client/tokens" # a file that contains a list of client tokens
# one of tokens or tokens-file is required

addr = ":19190" # the address at which the control server will listen for connections, default to :19190
cert-file = "path/to/cert.pem" # the server certificate file, in pem format
key-file = "path/to/key.pem" # the server certificate private key file

relay-addr = ":19191" # the address at which the relay will listen for connectsion, defaults to :19191
relay-hostname = "localhost" # the public hostname (e.g. domain, ip address) which will be advertised to clients, def

store-dir = "path/to/server-store" # where does this server persist runtime information, defaults to a /tmp subdirect
```

### Control server

To run in control server mode, use `connet control --config control-config.toml` command. Here is the full control server `control-config.toml` configuration specification:

```
[control]
client-tokens = ["client-token-1", "client-token-n"] # set of recognized client tokens
client-tokens-file = "path/to/client/tokens" # a file that contains a list of client tokens
# one of client-tokens or client-tokens-file is required

relay-tokens = ["relay-token-1", "relay-token-n"] # set of recognized relay tokens
relay-tokens-file = "path/to/relay/token" # a file that contains a list of relay tokens
# one of relay-tokens or relay-tokens-file is necessary when connecting relays

addr = ":19190" # the address at which the control server will listen for connections, default to :19190
cert-file = "path/to/cert.pem" # the server certificate file, in pem format
key-file = "path/to/key.pem" # the server certificate private key file

store-dir = "path/to/control-store" # where does this control server persist runtime information, defaults to a /tmp
```

### Relay server

To run in relay server mode, use `connet relay --config relay-config.toml` command. Here is the full relay server `relay-config.toml` configuration specification:

```
[relay]
token = "relay-token-1" # the token which the relay server uses to authenticate against the control server
token-file = "path/to/relay/token" # a file that contains the token, one of token or token-file is required

addr = ":19191" # the address at which the relay will listen for connectsion, defaults to :19191
hostname = "localhost" # the public hostname (e.g. domain, ip address) which will be advertised to clients, defaults

control-addr = "localhost:19190" # the control server address to connect to, defaults to localhost:19191
control-cas = "path/to/ca/file.pem" # the public certificate root of the control server, no default, required when us

store-dir = "path/to/relay-store" # where does this relay persist runtime information, defaults to a /tmp subdirector
```

### Storage

`connet` servers (both control and relay servers) store runtime state on the file system. If you don't explicitly specify `store-dir`, they will use a new subdirectory in `/tmp` by default, which means that every time they restart they'll loose any state and identity. To prevent this, you can specify an explicit `store-dir` location, which can be reused between runs.

### Logging

At the root of the config file, you can configure logging (`connet` uses slog internally):

```
log-level = "info" # supports debug, info, warn, error, defaults to info
log-format = "text" # supports text and json, defaults to text
```

### Tunning

On some systems, if you might see the following line in the logs:

```
failed to sufficiently increase receive buffer size (was: 208 kiB, wanted: 7168 kiB, got: 416 kiB). See
https://github.com/quic-go/quic-go/wiki/UDP-Buffer-Sizes for details.
```

In which case, we recommend visiting the wiki page and applying the recommended changes.

### NisOS

TBD

#### Flakes

To configure the client as a service:

```
# flake.nix
{
  inputs = {
    # ...
    connet.url = "github.com/connet-dev/connet";
  };
  outputs = { connet, ... }: {
    nixosConfigurations.example = nixpkgs.lib.nixosSystem {
      system = "x86_64-linux";
      modules = [
        # ...
        connet.nixosModules.default
        {
          services.connet = {
            enable = true;
            package = connet.packages."x86_64-linux".default;
            tokenFile = "/run/keys/connet.token";
            serverAddr = "localhost:19190";
            sources.example.addr = ":9998";
          };
        }
      ];
    };
  };
}
```

## Examples

TBD

## Hosting

If you want to use `connet`, but you don't want to run the server yourself, we have also built a hosted service at connet.dev. It is free when clients connect directly, builds upon the open source components by adding account management and it is one of the easiest way to start.