# `dnstun` (IP over DNS) Documentation


### Authors: Shan Lu, Xueyang Hu


This documentation is permanently hosted at link

## Overview

This project tunnels one restricted user's networking traffic into DNS queries, and our manipulated DNS name server will handle all requests, forward decoded IP packets and send back data through DNS response correspondingly.

## Installation

Our DNS server has static public IP address 69.254.169.254 assoicated with hostname "bb.jannotti.com". This is backed by an Amazon EC2 instance(DNSServer). In order to mimic the DNS tunneling scenario, we need another EC2 instance(called DNSClient). Please note that both instances need root permission

At DNSServer, run the `setup_server.sh` script.

At DNSClient, run the `setup_client.sh` script.

Both scripts will set virtual network interface at each side called "tun66". Server will route all traffic within CIDR `192.168.3.0/24` with its virtual IP `192.168.3.1` . Client will route the same traffic with its virtual IP `192.168.3.2`

At both sides, issue the `make` command.

At server side, run the following(please open DNSServer before DNSClient)

```
[ec2-user@ip-172-30-0-181 new_dnstun]$ sudo ./bin/server
Connected
```

At client side, open a client like this:

```
[ec2-user@ip-172-31-63-41 new_dnstun]$ sudo ./bin/client
connection established. server vip: 192.168.3.1, client vip: 192.168.3.2
```

## Creating the tunnel

To test whether bidirectional `ping` works, you should open another terminal at each side.

At DNSClient side, you could ping DNSServer's virtual address like following:

```
[ec2-user@ip-172-31-63-41 ~]$ ping 192.168.3.1
PING 192.168.3.1 (192.168.3.1) 56(84) bytes of data.
64 bytes from 192.168.3.1: icmp_seq=2 ttl=64 time=276 ms
64 bytes from 192.168.3.1: icmp_seq=3 ttl=64 time=74.8 ms
64 bytes from 192.168.3.1: icmp_seq=3 ttl=64 time=875 ms (DUP!)
64 bytes from 192.168.3.1: icmp_seq=4 ttl=64 time=675 ms
64 bytes from 192.168.3.1: icmp_seq=4 ttl=64 time=1465 ms (DUP!)
64 bytes from 192.168.3.1: icmp_seq=6 ttl=64 time=275 ms
64 bytes from 192.168.3.1: icmp_seq=7 ttl=64 time=78.1 ms
```

At DNSServer side, do the same thing vice versa:

```
[ec2-user@ip-172-30-0-181 new_dnstun]$ ping 192.168.3.2
PING 192.168.3.2 (192.168.3.2) 56(84) bytes of data.
64 bytes from 192.168.3.2: icmp_seq=2 ttl=64 time=235 ms
64 bytes from 192.168.3.2: icmp_seq=3 ttl=64 time=36.6 ms
64 bytes from 192.168.3.2: icmp_seq=6 ttl=64 time=225 ms
64 bytes from 192.168.3.2: icmp_seq=10 ttl=64 time=197 ms
64 bytes from 192.168.3.2: icmp_seq=14 ttl=64 time=173 ms
64 bytes from 192.168.3.2: icmp_seq=18 ttl=64 time=151 ms
64 bytes from 192.168.3.2: icmp_seq=22 ttl=64 time=128 ms
64 bytes from 192.168.3.2: icmp_seq=26 ttl=64 time=106 ms
64 bytes from 192.168.3.2: icmp_seq=30 ttl=64 time=83.7 ms
64 bytes from 192.168.3.2: icmp_seq=31 ttl=64 time=683 ms
64 bytes from 192.168.3.2: icmp_seq=32 ttl=64 time=481 ms
64 bytes from 192.168.3.2: icmp_seq=33 ttl=64 time=280 ms
64 bytes from 192.168.3.2: icmp_seq=34 ttl=64 time=80.2 ms
64 bytes from 192.168.3.2: icmp_seq=34 ttl=64 time=880 ms (DUP!)
64 bytes from 192.168.3.2: icmp_seq=34 ttl=64 time=1675 ms (DUP!)
```

# Design Challenges

1. Connection Establishment Since our DNSServer side is designed to support multiple DNSClient instances, each new DNSClient will send a `CONNECT` DNS request to DNSServer. Our DNSServer will allocate an virtual IP address and unique `userId` for this DNSClient, all information sent through a DNS response `RESPONSE`. This one-way handshake will establish a connection between client and server. Below are all commands we defined and their purposes.
    - `CONNECT` DNSClient sends an request asking connection with DNSServer
    - `RESPONSE` DNSServer sends back virtual address, userId assigned to this specific DNSClient
    - `DATA` birectional, indicating this DNS packet contains encoded data.

- `ACK` acknowledgement(fake response), avoiding intermediate DNS server's retransmission.
- `EMPTY` DNSClient periodically sends empty DNS query to DNSServer, whose response will be filled with downstram encoded data.

2. Data Communication
   Due to DNS query's limitation, we have to handle bidirectional data transferring differently. In following context, we define **upstream** client -> server traffic, and **downstream** server -> client traffic.

   - Upstream traffic:
     - Encoding: `base32` encoding, with character `_` as placeholder if necessary
     - Where to put encoded data: Question's domain name

   - Downstream traffic:
     - Encoding: `base64` encoding, with character `_` as placeholder if necessary
     - Where to put data: Answer's TXT record

## `SendString` Utility (Bidirectional Data Communication)

At DNSClient side, you can type `send <message>` in driver like this

```
send abcdeabcde
```

At DNSServer side, you can type `send <userId> <message>` in driver like this

```
send 1 aaaaabbbb
```

Basically client-side SendString will pack an `DATA` DNS packet with a negative IP identification number, so that server will cast binary into string and print it out, without pushing into tun interface. Server-side SendString is a little tricky, as server cannot initiate any DNS query/response to specific DNSClient. It will transform this message into a base64-encoded string and put it into a channel. Whenever there is an `EMPTY` DNS query coming, DNSServer will response it with Answer's TXT fields filled with this string.

# SOCK5 Proxy

Due to limited time, we are unable to finish this part before deadline. We will finish it up voluntarily after the deadline.

TODO

# References

Special thanks to following tutorials and libraries:

- TUN virtual interface configuration: Tun/Tap interface tutorial by waldner
- DNS library written in Go: golang-dns by tonnerre
- TUN library written in Go: water by songgao