Signpost

An Internship Report
Submitted in Partial Fulfillment for the Award of
M.Tech in Information Technology

By

Bishneet Kaur
MT2011030

To

International Institute of Information Technology
Bangalore – 560100

June 2013

CERTIFICATE

This is to certify that the internship report titled **'Signpost'** submitted by `Bishneet Kaur' (MT2011030) is a bonafide work carried out under my supervision at 'Horizon Digital Economy Research Institute' from January 2013 to June 2013 (6 months), in partial fulfillment of the M.Tech. course of International Institute of Information Technology, Bangalore.

Her performance & conduct during the internship was satisfactory.

Dr. Richard Mortier
Room C13 Computer Science
Jubilee Campus
Wollaton Road
Nottingham
NG8 1BB
UK

Date: June 19th 2013
Place: Nottingham, U.K.

ACKNOWLEDGEMENT

I would like to express my profound gratitude to the entire faculty of **International Institute of Information Technology**, **Bangalore** for providing me with the opportunity to learn and avail the excellent facilities and infrastructure.

My sincere thanks to my supervisor **Dr. Richard Mortier** for his inspiration, constructive suggestion, mastermind analysis and affectionate guidance in my work, without which the completion of this project would have been impossible. My heartfelt thanks to **Charalampos Rotsos** for the assistance and guidance throughout the project.

I would also like to thank **Professor Derek McAuley** for providing me with this great opportunity to work with highly skilled and experienced professionals. Also, my sincere gratitude to the team of Horizon Digital Economy Research Institute especially **Dr. Sophie Dale-Black, Kevin Beales**, **David Morton** for providing me with all the necessary help during whole course of the project.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The architecture of today's internet services restricts user's ability to establish inter-device connectivity only through the visible "Cloud". Most users are unable to access existing mechanisms for establishing secure peer-to-peer connections due to intricacy of their network configuration. They sacrifice security, privacy and low-latency in favor of ease-of-use.

Signpost is a system for distributed, authenticated and secure device-to-device communication. Signpost servers give unique names to the devices within the DNS hierarchy, within their own zone and orchestrate different available techniques for establishing device-to-device connectivity to automatically select the most appropriate one.

1. Organization Overview

Horizon Digital Economy Research is a research institute at **The University of Nottingham** engaged in digital economy research. Funded by **Research Councils UK** (**RCUK**), **The University of Nottingham** and over 100 academic and industrial partners, the institute comprises both a research hub and Doctoral Training Centre.

As technology advances throughout our digital age, society becomes increasingly reliant upon digital technology for everyday life. Researching and understanding how digital technology is designed and used is crucial in ensuring that it delivers both economic and societal benefit to all.

Horizon research focuses on the role of *'**always on, always with you**'* ubiquitous computing technology. It aims to investigate the technical developments needed if electronic information is to be controlled, managed and harnessed to enhance the way we live, work, play & travel in the future.

Horizon is specifically interested in the **"*lifelong contextual footprint*"** by this it means the digital footprints it leaves behind when using mobile, internet and other digital technologies and understanding how these footprints are handled throughout our lives.

Horizon conducts specific research in the wild activities that are informed by its commercial and public sector partners. It operates within a number of sectors:

- **Creative industries**

  The work here includes projects within the leisure and tourism sector and new media to investigate exciting new ways to enhance the experience of visiting.

Working alongside its partners, Horizon's challenge is to move beyond current conceptions of tourism by exploiting digital technology to enrich the entire visiting experience, blending online activities with augmented physical sites to allow creative connections between visitors and residents, and current and past events.

- **Transport**

  Today's society demands greater mobility as families, friends and organizations become more geographically distributed. This raises fundamental questions over how we may travel more sustainably, and how technology may play a role in helping us to coordinate that travel more effectively.

  Horizon's aim within the transport sector is to investigate how digital technologies might influence travel, for example through on demand collaborative travel (e.g. mobile and social-network-enabled car sharing), or how the use of crowd-sourced and GIS (Geographical Information System)-sourced information can play a role in supporting real-time transport operations at a local, national and international level.

- **Energy**

  Horizon aims to tackle a broad range of issues covering digital technologies which cater for the need for more efficient generation, distribution and use of energy.

  We live in a world where sources of energy are becoming increasingly scarce and expensive: for societies that expect water, electricity and gas at the flick of a switch or turn of a tap, the implications of this scarcity for future comfort and well-being are significant. Through its involvement in three TEDDI (Transforming Energy Demand through Digital Innovation) projects; C-Aware, DESIMAX and Wi-be, Horizon's energy

research aims to explore ways to both alter consumer attitudes and behavior, and to improve the efficiency of our energy infrastructure to avoid future energy crises.

2. Project Overview

This section describes about the project and the factors that motivated to build such a system.

2.1 Motivation

The modern internet is broadly divided into two halves: a global core network, “Cloud” and an edge network through which end users access the centralized cloud. Content is published within the core network and served from datacenters via high speed IP routing infrastructure. Devices within the core network route to each other freely but peer-to-peer connectivity is much more complicated in the edge networks. Deployment of middleboxes (e.g public WiFi hotspots) for access controls, moderating bandwidth usage (e.g. 3G mobile links) and information flow (e.g. firewalls) has made end-to-end communication brittle and has introduced asymmetric visibility between the hosts. Also, some devices are hidden behind NAT boxes which prevent incoming connection to being established easily.

When a user wants its devices to communicate securely, the only practical option left is to access the content via a cloud. This causes the inability to operate without an internet connection and lack of bandwidth and high latency in comparison to direct communication on local networks.

These factors motivated the need for a network service which enables user-centric control of connectivity both within a personal set of devices and across to the devices of others and also enables the devices to operate even when disconnected from global internet.

2.2 System architecture

The following figure gives an overview of Signpost architecture.

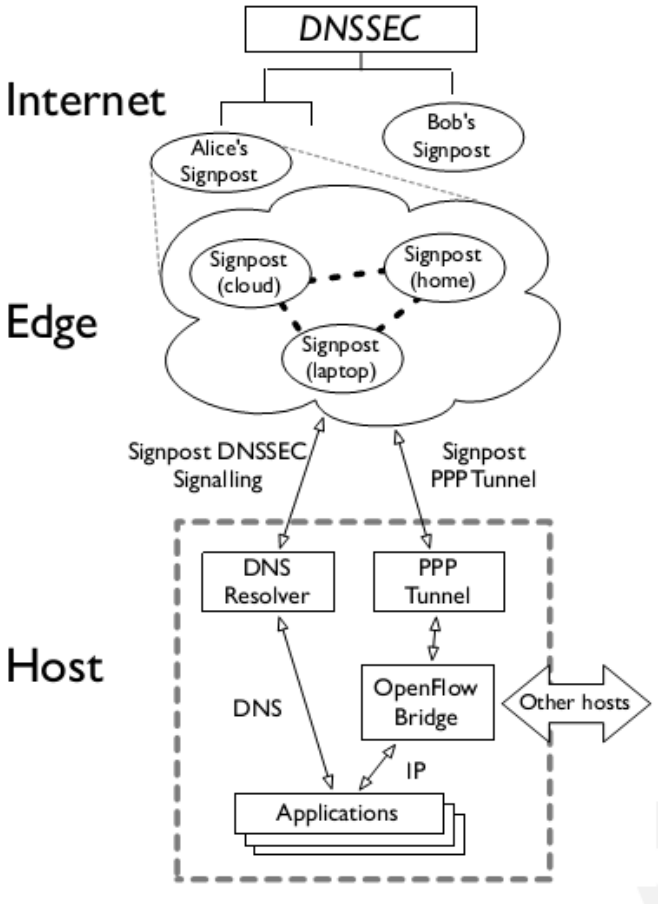

Figure 1: Signpost architecture [1]

Signpost architecture has elements at different levels:

- **Network core (Internet):** At this level, DNSSEC structure delegates domain name for the users with .io extension like alice.io or bob.io
- **Edge**: At this level, the devices and name resolver for a user exists. As shown in Figure1, two devices of Alice (Laptop and HomePC) are present along with the Alice's Signpost cloud which acts a server resolving names for the devices

(laptop.alice.io and home.alice.io) and also maintaining connectivity between them.

- **Host**: The components shown in the Host in Figure1 are present at each device (host) which helps an application running in the host to connect to peer hosts.

2.3 System goals

The goal of the system is to offer following features:

2.3.1 Naming

This feature aims at assigning stable names to devices in the global naming hierarchy and providing an authenticated mechanism to resolve these device names into concrete network addresses using DNS protocol. After authentication, name lookup triggers tactic engine residing in the controller (Signpost server) which probes the network to establish a path between the devices.

2.3.2 Connectivity

This feature aims at establishing a stable and secure channel between devices. This uses tactics and tactic engine. Tactics are the services to establish tunnel between devices and tactic engine execute a range of tactics in parallel and create a communication channel selecting the best available tactic. When Signpost server (having DNS resolver in it) receives a client query, one of the available tunnels is returned in DNS reply.

2.3.3 Control

This feature aims at empowering users with control using two key functions: policy expression and authentication. User policy is defined through simple local configuration,

where users define network path security properties on a per-domain basis. Tactic engine ensures this policy enforcement during path establishment at runtime. User authentication employs a public key cryptography scheme and uses DNSSEC key distribution mechanism.

2.4 My contribution

I worked on the "connectivity" feature that provides connection between devices. The work is described with the help of following figure:

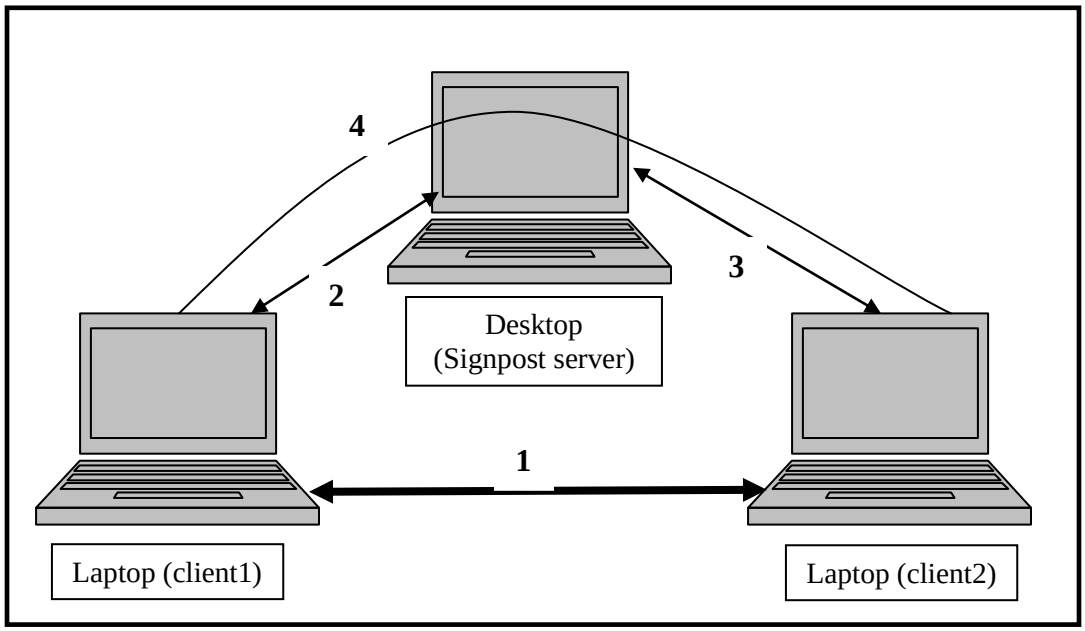

Figure 2: Establishing connectivity between clients

Alice has two laptops Client1 and Client2 which need to connect to each other. They can be behind NAT or in different networks. So, the system goal is to establish a secure and stable connection between them [Link 1 in the above figure] by using:

- **Tactics**: These are the services which provide “tunnels” as mode of connection between two devices while serving the purpose of anonymity, encryption and authentication as well.

  These services have a server part and a client part. Server part will be implemented at Signpost server and client part will be running at each client. A tunnel is established between each client and the server [Link 2 and Link3 in the above figure]. Then each client connects each other using this tunnel going through the server [Link 4 in the above figure]. Most of the services establish tunnels by providing virtual interfaces at each end of the tunnel.

- **Tactic engine**: When client1 wants a connection to client2, it will send a query to Signpost server in which tactic engine is implemented. After receiving this query, tactic engine will look forward to different parameters such as network traffic, bandwidth availability and latency between these two clients. It will execute a range of available tactics and selects the best one which suits the above parameters (maximizing throughput and minimizing latency) and returns it to both the client. Both the clients will execute the tactic and will be able to establish connectivity through server.

I implemented various tactics and built automation scripts to start these tactics at client and server as described in Chapter 3. Also, I did testing to get bandwidth and latency parameters for above tactics in order to compare performance of these tactics as described in Chapter 4.

3. Implementation Details

This section elaborates my contribution in the project giving details about the tools and technologies used and what steps are to be followed to make the tactics function properly.

3.1 Tools and Technologies used

The following table lists various technologies which are used as tactics that can establish connectivity between two devices:

Tactics	Primary Purpose	Authentication	Encryption	Anonymity
Iodine	IP over DNS	No	No	No
OpenVPN	VPN	Yes	Yes	No
ssh_tap	Encryption	Yes	Yes	No
ssh_tun	Encryption	Yes	Yes	No
Tor	Anonymity	No	Yes	Yes
UPnP	NAT Traversal	No	No	No

Table 1: List of network tactics that can establish connectivity between two devices

3.1.1 Iodine

DNS Tunneling

The technique of DNS tunneling is used in the scenario when only DNS queries are allowed through a network. It involves encapsulating binary data within DNS queries and replies, using base32 and base64 encoding, and to use the DNS domain name lookup system itself as a bi-directional carrier for this data. Therefore, as long as domain name

lookups can be done on a network, any kind of data can be tunneled to a remote system, and also the internet.

How it works:

A hostname can only be 255 bytes long. Client encodes a formatted domain name request upto a maximum of 255 bytes and then fake DNS nameserver decodes it back. This fake NS then encodes the response back and delivers it to the client.

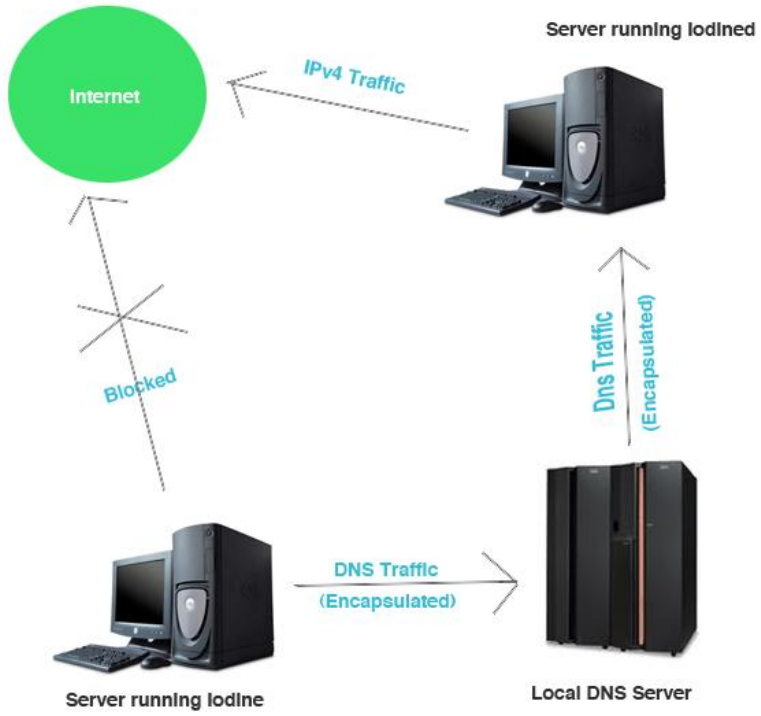

Figure 3: DNS tunneling

Iodine [2] is software that tunnels IPv4 data through a DNS server. To use this tunnel, control over a real domain (like mooo.com), and a server with a public IP address is needed. This server acts as FakeNS which will respond to the requests of the domain name (subd.mooo.com). Client will send IP packets encapsulated by DNS request

packets with this domain name (subd.mooo.com) and server will respond to them as follows[3]:

For instance:

- Iodine client wants to send data to IP 1.2.3.4.
- It encodes packet as base32 and encapsulates it in a DNS query and sends DNS request as: base32encodedpacket.subd.mooo.com. to local relaying DNS server as shown in figure 4.

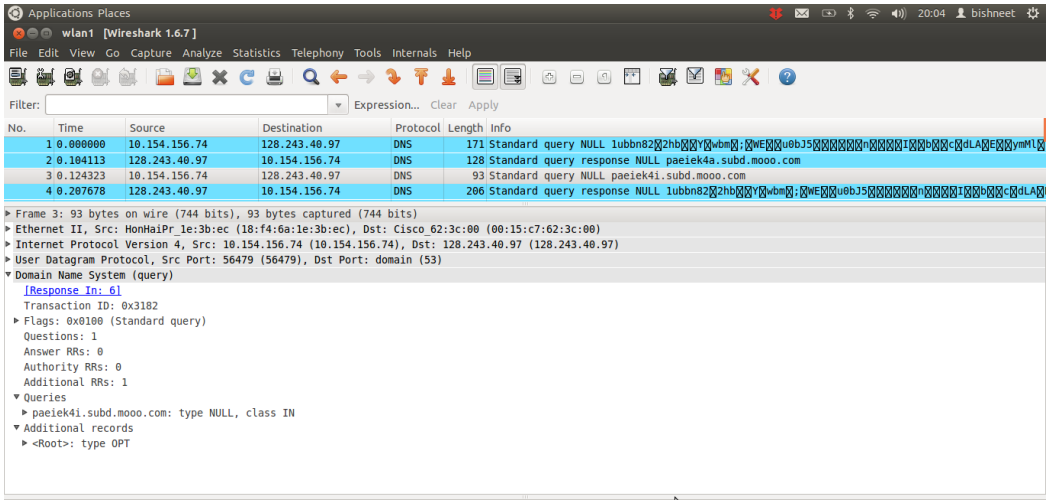

Figure 4: DNS query packets from iodine client

- The request after passing through relaying DNS servers (like servers for .com etc) goes to fake DNS server (fakens.mooo.com) as this is responsible to answer requests for domain subd.mooo.com.
- Fake NS eventually gets the request, decodes base32, reconstructs the packet and sends it off to the internet.
- Iodine server running in fake NS, in response for received DNS query, sends base64 encoded (NULL/TXT) record, with encoded IP packets destined for client as shown in figure 5.

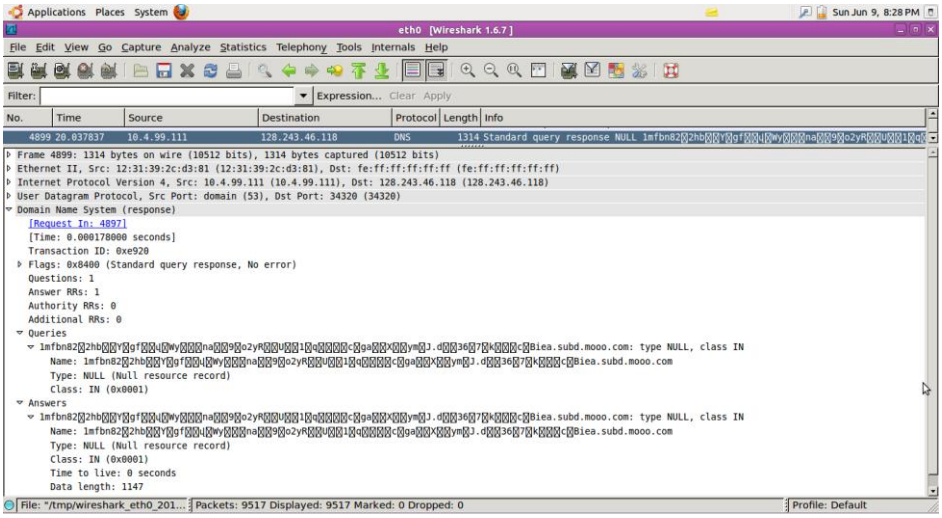

Figure 5: DNS reply packets from iodine server

- Iodine client will receive DNS response, decode DNS record and reconstruct packets as shown in figure 6.

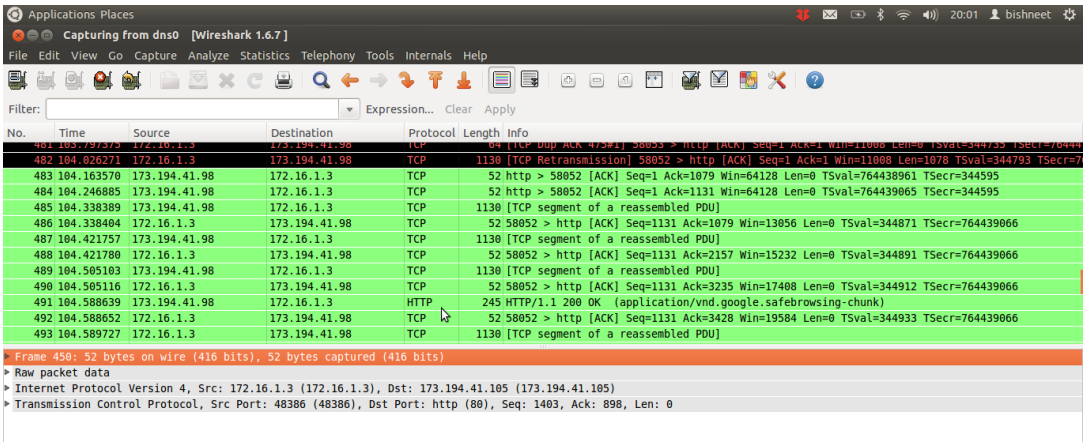

Figure 6: Decapsulated TCP packets at dns interface of client

A tunnel is established between iodine client and iodine server for this purpose. This tunnel is used to establish connectivity between two clients through server.

3.1.2 OpenVPN

Virtual Private Network

A virtual private network (VPN) extends a private network across public networks like the Internet. It enables a host computer to send and receive data across shared or public

networks as if they were an integral part of the private network with all the functionality, security and management policies of the private network. This is done by establishing a virtual point-to-point connection through the use of dedicated connections, encryption, or a combination of the two.

How it works:

For instance: a remote host (with VPN client running in it) with public IP address 1.2.3.4 wishes to connect to a system found inside a company network. The system has internal address 192.168.1.10 and is not reachable publicly. Before the client can reach this server, it needs to go through a VPN server / firewall device that has public IP address 5.6.7.8 and an internal address of 192.168.1.1. All data between the client and the server will need to be kept confidential; hence a secure VPN is used.

- The VPN client connects to a VPN server via an external network interface.
- The VPN server assigns an IP address to the VPN client from the VPN server's subnet. The client gets internal IP address 192.168.1.50, for example, and creates a virtual network interface through which it will send encrypted packets to the other tunnel endpoint (the device at the other end of the tunnel).
- When the VPN client wishes to communicate with the company server, it prepares a packet addressed to 192.168.1.10, encrypts it and encapsulates it in an outer VPN packet, say an IPSec packet. This packet is then sent to the VPN server at IP address 5.6.7.8 over the public Internet. The inner packet is encrypted so that even if someone intercepts the packet over the Internet, they cannot get any information from it. They can see that the remote host is communicating with a server/firewall, but none of the contents of the communication. The inner

encrypted packet has source address 192.168.1.50 and destination address 192.168.1.10. The outer packet has source address 1.2.3.4 and destination address 5.6.7.8.

- When the packet reaches the VPN server from the Internet, the VPN server decapsulates the inner packet, decrypts it, finds the destination address to be 192.168.1.10, and forwards it to the intended server at 192.168.1.10.
- After some time, the VPN server receives a reply packet from 192.168.1.10, intended for 192.168.1.50. The VPN server consults its routing table, and sees this packet is intended for a remote host that must go through VPN.
- The VPN server encrypts this reply packet, encapsulates it in a VPN packet and sends it out over the Internet. The inner encrypted packet has source address 192.168.1.10 and destination address 192.168.1.50. The outer VPN packet has source address 5.6.7.8 and destination address 1.2.3.4.
- The remote host receives the packet. The VPN client decapsulates the inner packet, decrypts it, and passes it to the appropriate software at upper layers.

Overall, it appears as if the remote host (VPN client) and internal system are on the same 192.168.1.0/24 network.

This is used in Signpost where two VPN clients from different networks are given IP which is in the same subnet as VPN server. It will appear that VPN clients and server are in the same network. A tunnel is established between client and server and they both will communicate to each other through server.

For using this technique**, OpenVPN**[6] is used which is an open source software application that implements virtual private network (VPN) techniques for creating secure

point-to-point connections. It uses a custom security protocol that utilizes SSL/TLS for key exchange. It is capable of traversing network address translators (NATs) and firewalls. OpenVPN allows peers to authenticate each other using a pre-shared secret key, certificates, or username/password.

In Signpost, it is used in a multiclient-server configuration, in which the server releases an authentication certificate for every client, using signature and Certificate authority. It uses the OpenSSL encryption library extensively, as well as the SSLv3/TLSv1 protocol, and contains many security and control features.

3.1.3 ssh_tun/ssh_tap

Tun/Tap interfaces:

Tun/Tap interfaces[7] are virtual-network kernel devices. TAP simulates a link layer device and it operates with layer 2 packets such as Ethernet frames. TUN simulates a network layer device and it operates with layer 3 packets such as IP packets. TAP is used to create a network bridge, while TUN is used with routing.

How it works:

Tun/tap interfaces are *software-only interfaces*, ie they exist only in the kernel and, unlike regular network interfaces, they have no physical hardware component (and so there's no physical "wire" connected to them). They can be thought of as a regular network interface that, when the kernel decides that the moment has come to send data "on the wire", instead sends data to some userspace program that is attached to the interface. When the program attaches to the tun/tap interface, it gets a special file descriptor, reading from which, it gives the data that the interface is sending out. In a similar fashion, the program can write to this special descriptor and the data will appear as input to the tun/tap

interface. To the kernel, it would look like the tun/tap interface is receiving the data “from wire". The difference between a tap interface and a tun interface is that a tap interface outputs (and must be given) full ethernet frames, while a tun interface outputs (and must be given) raw IP packets (and no ethernet headers are added by the kernel).

The interface can be **transient**, meaning that it's created, used and destroyed by the same program; when the program terminates, even if it doesn't explicitly destroy the interface, the interfaces ceases to exist. Another option is to make the interface **persistent**; in this case, it is created using a dedicated utility (like **tunctl**), and then normal programs can attach to it but when they do so, they must connect using the same type (tun or tap) used to originally create the interface, otherwise they will not be able to attach.

Once a tun/tap interface is in place, it can be used just like any other interface, meaning that IP addresses can be assigned, its traffic can be analyzed, firewall rules can be created, routes pointing to it can be established, etc.

ssh_tun/ssh_tap uses OpenSSH which is a free version of the SSH connectivity tools and provides secure tunneling capabilities and several authentication methods, and supports all SSH protocol versions. Ssh_tap/ssh_tun allows to connect two tun/tap interfaces together, to create a layer-2 or layer-3 network between remote machines respectively.

3.1.4 Tor

Tor (The Onion Router) protects its users against a common form of Internet surveillance known as "traffic analysis" which can be used to infer who is talking to whom over a public network. Knowing the source and destination of one’s Internet traffic allows others to track his behavior and interests.

Traffic analysis works as:

Internet data packets have two parts: a **data payload** and a **header** used for routing. The data payload is whatever is being sent, whether that's an email message, a web page, or an audio file. Even if the data payload of the communication is encrypted, traffic analysis still reveals a great deal about what a person is doing and, possibly, what he is saying because it focuses on the header, which discloses source, destination, size, timing, and so on. A very simple form of traffic analysis might involve sitting somewhere between sender and recipient on the network, looking at headers. Some attackers spy on multiple parts of the Internet and use sophisticated statistical techniques to track the communications patterns of many different organizations and individuals. Encryption does not help against these attackers, since it only hides the content of Internet traffic, not the headers.

"**Onion Routing**" refers to the layers of the encryption used. The original data, including its destination, is encrypted and re-encrypted multiple times, and sent through a virtual circuit comprising successive, randomly selected Tor relays. Each relay decrypts a "layer" of encryption to reveal only the next relay in the circuit in order to pass the remaining encrypted data on to it. The final relay decrypts the last layer of encryption and sends the original data, without revealing or even knowing its sender, to the destination. This method reduces the chance of the original data being understood in transit and, more notably, conceals the routing of it.

How it works:

Tor[9] helps to reduce the risks of both simple and sophisticated traffic analysis by distributing the transactions over several places on the Internet, so no single point can

link a user to its destination. This is same as using a twisty, hard-to-follow route in order to throw off somebody who is tailing you — and then periodically erasing your footprints. Instead of taking a direct route from source to destination, data packets on the Tor network take a random pathway through several relays that cover its tracks so no observer at any single point can tell where the data came from or where it's going.

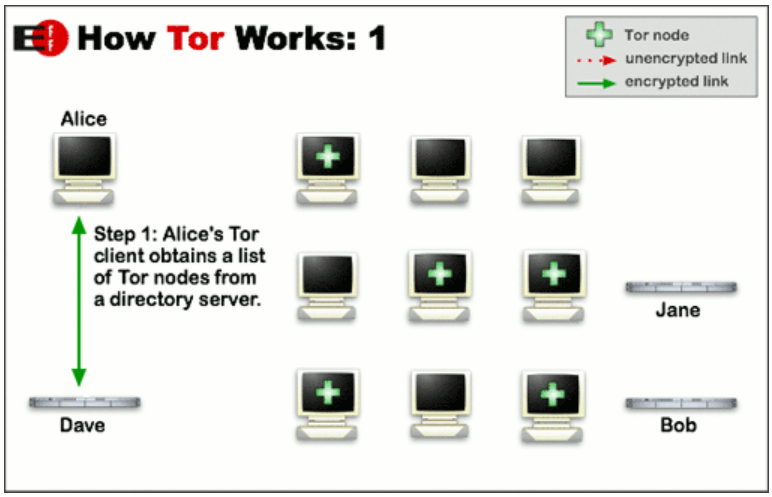

Figure 7: Working of Tor (i)

To create a private network pathway with Tor, the user's software or client incrementally builds a circuit of encrypted connections through relays on the network. The circuit is extended one hop at a time, and each relay along the way knows only which relay gave it data and which relay it is giving data to. No individual relay ever knows the complete path that a data packet has taken. The client negotiates a separate set of encryption keys for each hop along the circuit to ensure that each hop can't trace these connections as they pass through.

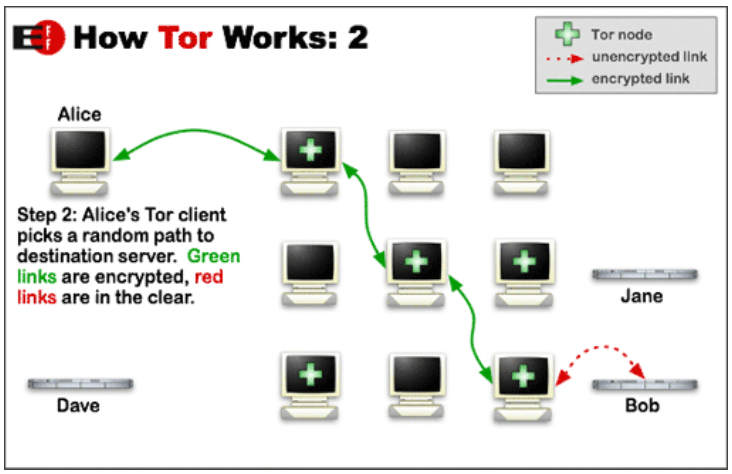

Figure 8: Working of Tor (ii)

Once a circuit has been established, many kinds of data can be exchanged and several different sorts of software applications can be deployed over the Tor network. Because each relay sees no more than one hop in the circuit, neither an eavesdropper nor a compromised relay can use traffic analysis to link the connection's source and destination.

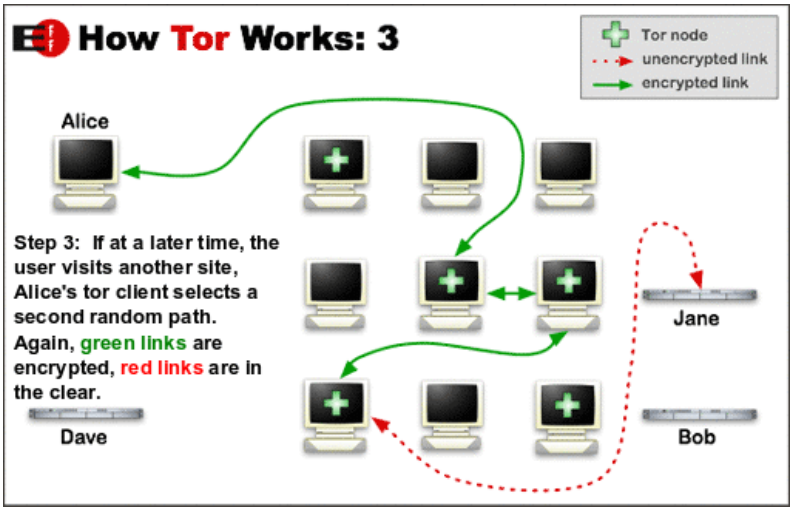

Figure 9: Working of Tor (iii)

Hidden Services

Servers configured to receive inbound connections through Tor are called hidden services. Rather than revealing a server's IP address, a hidden service is accessed through

its onion address. The Tor network understands these addresses and can route data to and from hidden services, even those hosted behind firewalls or network address translators (NAT), while preserving the anonymity of both parties. Tor is necessary to access hidden services.

Hidden service protocol [11]

Step 1: A hidden service needs to advertise its existence in the Tor network before clients will be able to contact it. Therefore, the service randomly picks some relays, builds circuits to them, and asks them to act as introduction points by telling them its public key. The introduction points and others are told the hidden service's identity (public key), but not about the hidden server's location (IP address).

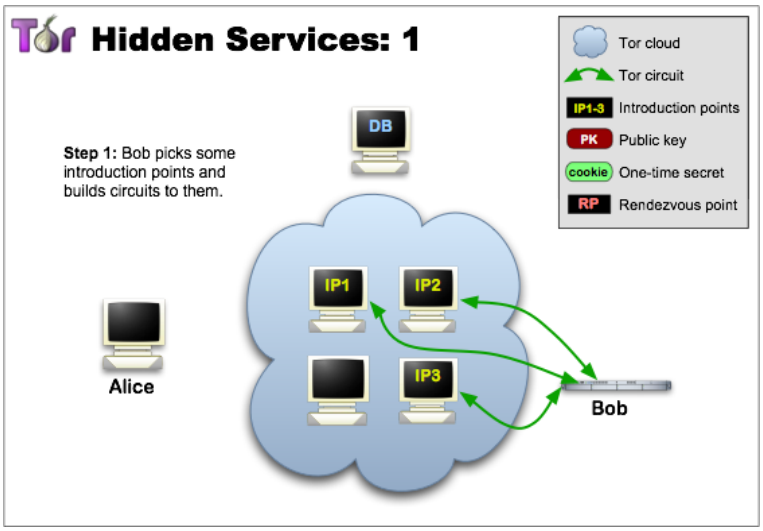

Figure 10: Working of hidden services (i)

Step 2: The hidden service assembles a hidden service descriptor, containing its public key and a summary of each introduction point, and signs this descriptor with its private key. It uploads that descriptor to a distributed hash table. The descriptor will be found by clients requesting XYZ.onion where XYZ is a 16 character name derived from the service's public key. After this step, the hidden service is set up.

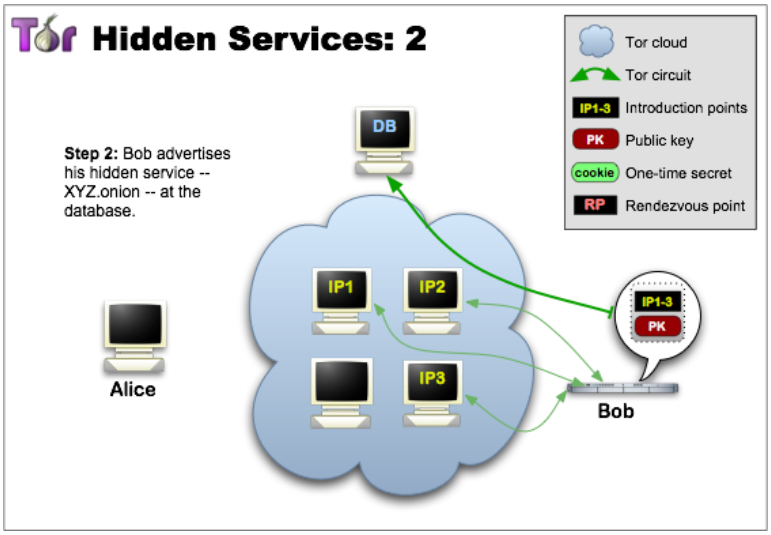

Figure 11: Working of hidden services (ii)

Step 3: A client that wants to contact a hidden service needs to learn about its onion address first. After that, the client can initiate connection establishment by downloading the descriptor from the distributed hash table. If there is a descriptor for XYZ.onion, the client now knows the set of introduction points and the right public key to use. Around this time, the client also creates a circuit to another randomly picked relay and asks it to act as rendezvous point by telling it a one-time secret.

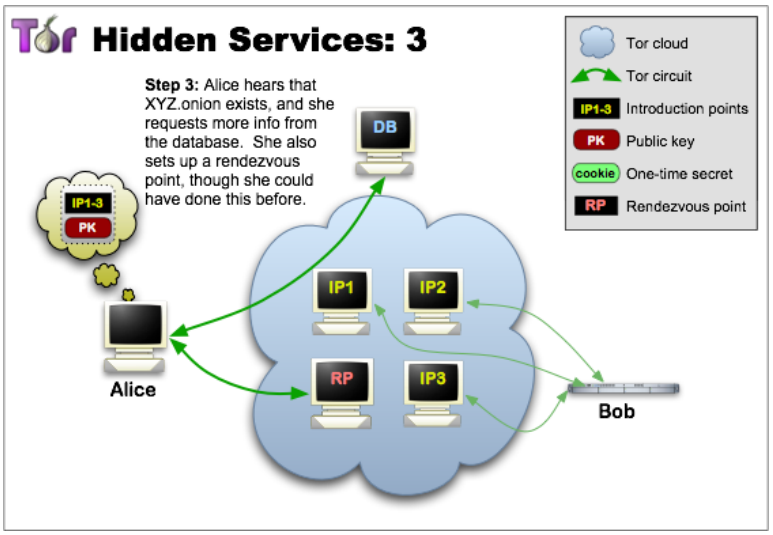

Figure 12: Working of hidden services (iii)

Step 4: When the descriptor is present and the rendezvous point is ready, the client assembles an introduce message (encrypted to the hidden service's public key) including the address of the rendezvous point and the one-time secret. The client sends this message to one of the introduction points, requesting it be delivered to the hidden service. Again, communication takes place via a Tor circuit: nobody can relate sending the introduce message to the client's IP address, so the client remains anonymous.

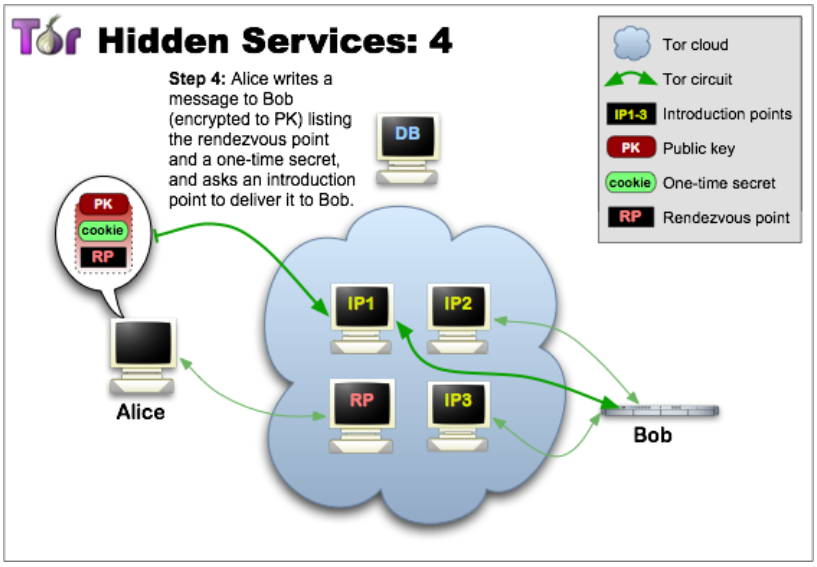

Figure 13: Working of hidden services (iv)

Step 5: The hidden service decrypts the client's introduce message and finds the address of the rendezvous point and the one-time secret in it. The service creates a circuit to the rendezvous point and sends the one-time secret to it in a rendezvous message.

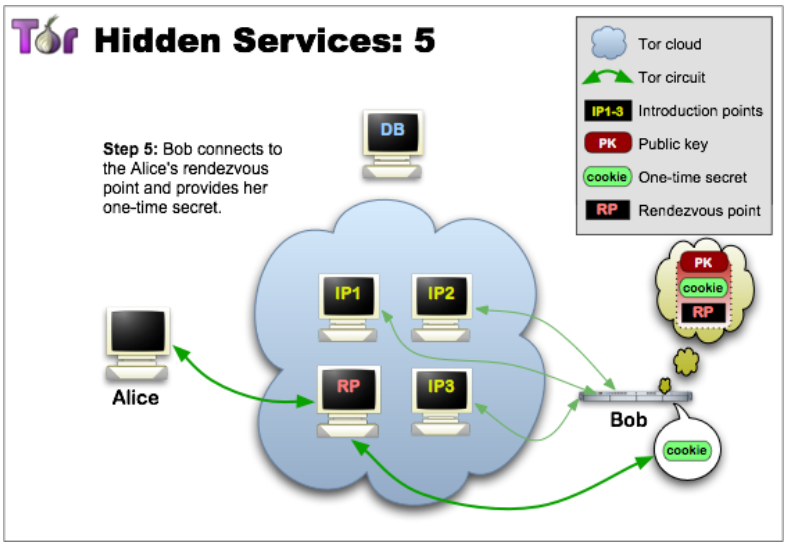

Figure 14: Working of hidden services (v)

Step 6: The rendezvous point notifies the client about successful connection establishment. After that, both client and hidden service can use their circuits to the rendezvous point for communicating with each other. The rendezvous point simply relays (end-to-end encrypted) messages from client to service and vice versa.

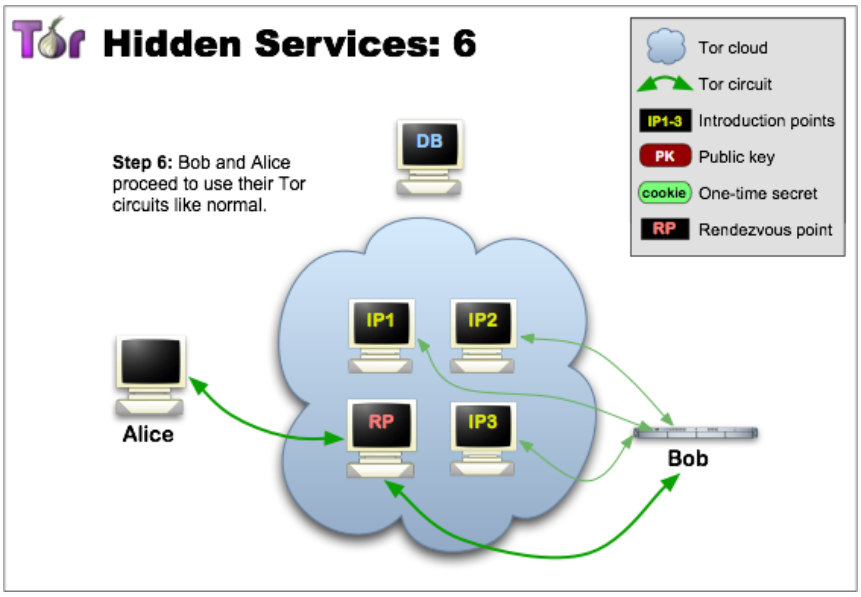

Figure 15: Working of hidden services (vi)

The Hidden Services are used in Signpost as a tactic to provide anonymity to the Tor client. An http server with python is started at a specific port in the same system where tor server is running. Tor client will connect to this server using tor networks.

3.1.5 UPnP

Suppose some service is running in the system which is behind NAT and a user who is in different network wants to use this service. Since, he does not know IP of the system running this service (only external IP is known), he can't access this service. So, he needs to do some Port Forwarding. To do this port forwarding automatically without user's intervention, UPnP/IGD is used.

Universal Plug and Play (UPnP) is a set of networking protocols that permits networked devices, such as personal computers, Internet gateways, Wi-Fi access points and mobile devices to seamlessly discover each other's presence on the network and

establish functional network services for data sharing, communications, and entertainment.

The UPnP discovery protocol, known as the **Simple Service Discovery Protocol (SSDP)** allows a device to discover UPnP router. UPnP router multicasts discovery messages in its network.

One of the ways by which UPnP provides NAT Traversal problem is by **Internet Gateway Device Protocol (IGD Protocol)** which is used in Signpost.

Internet Gateway Device Protocol (IGD Protocol)

Internet Gateway Device (IGD) protocol, allows software to configure routers for NAT traversal without user-intervention. The UPnP router configured with IGD can allow to seek for available devices on the network via SSDP A seek request is sent via HTTP and port 1900 to the multicast address 239.255.255.250.

With UPnP/IGD, the long and error-prone manual configuration for port-forwarding can be done automatically. It creates iptables rule for port forwarding automatically for a UPnP client. The client needs to mention external port where outside traffic is coming at UPnP router and an internal port to which it wants to forward this traffic. By adding this rule, a UPnP router configured with IGD can enable traversal of the internet traffic from an external address to an internal client.

How it works:

For instance, UPnP router has following configurations:

eth1 (External port): 128.243.35.178/24

eth0 (Internal port): 172.16.1.1/16

Client behind NAT has following configuration:

eth0: 172.16.1.2/16

If the client wants the traffic coming on some external port (say 55555) at UPnP router, to be forwarded to its internal port (say 22), it can add port forwarding at its end. This will automatically be added in iptables of UPnP router configured with IGD. You need not manually configure UPnP router's iptables. This is convenient as adding/deleting iptables rule is router's vendor specific.

3.2 Implementing tactics

3.2.1 Iodine

Installations: iodine

Steps performed:

1. For setting up iodine tunnel, control over a real domain (like mooo.com), and a server with a public IP address is needed. A domain name can be set in zone files of BIND or at sites such as http://freedns.afraid.org/

2. Get a domain name (the one used here is ‘mooo.com’). Delegate a subdomain (like "subd.mooo.com") to the Iodine server at freedns.afraid.org/subdomain/edit.php

3. Create a subdomain of type NS which will act as fake DNS Server [4]:

Type	NS
Subdomain	subd
Domain	mooo.com (public)
Destination	fakens.subd.mooo.com

After saving, create another subdomain of type A giving IP to the above created fake DNS server. (This IP will be the IP of the system on which Iodine server will be running)

Type	A
Subdomain	fakens.subd
Domain	mooo.com (public)
Destination	IP_of_system_where_iodine_server_is_running

This looks like:

subd	IN	NS	fakens.mooo.com
fakens	IN	A	IP_of_system_where_iodine_server_is_running

So any query for the domain 'subd.mooo.com' is responded by fake nameserver fakens.mooo.com whose IP is given in the second line of the above record. Iodine server will run at this IP and will decode the encapsulated DNS packets (with DNS request for subd.mooo.com) sent by iodine client.

4. Start iodine at server first followed by client.

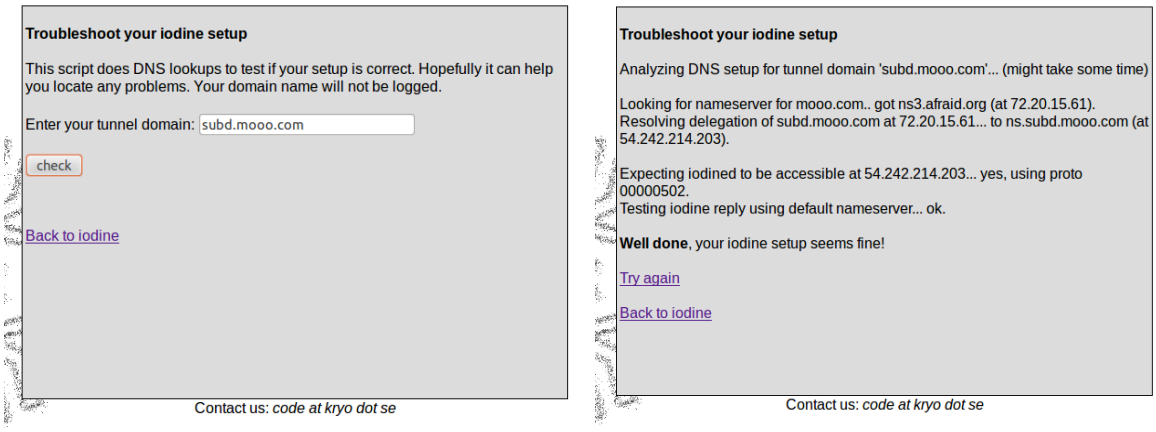

Figure16: checking iodine tunnel is set up successfully

Output:

You can see only DNS requests and replies packets for domain subd.mooo.com at eth/wlan interfaces of client and server. A tunnel is created between client and server with dns interfaces at both ends. At dns interfaces, you can see TCP packets coming from outside world. These packets were encapsulated as DNS packets and sent to client as DNS replies by the server. The client then decapsulate DNS packets and get TCP packets at its dns interface.

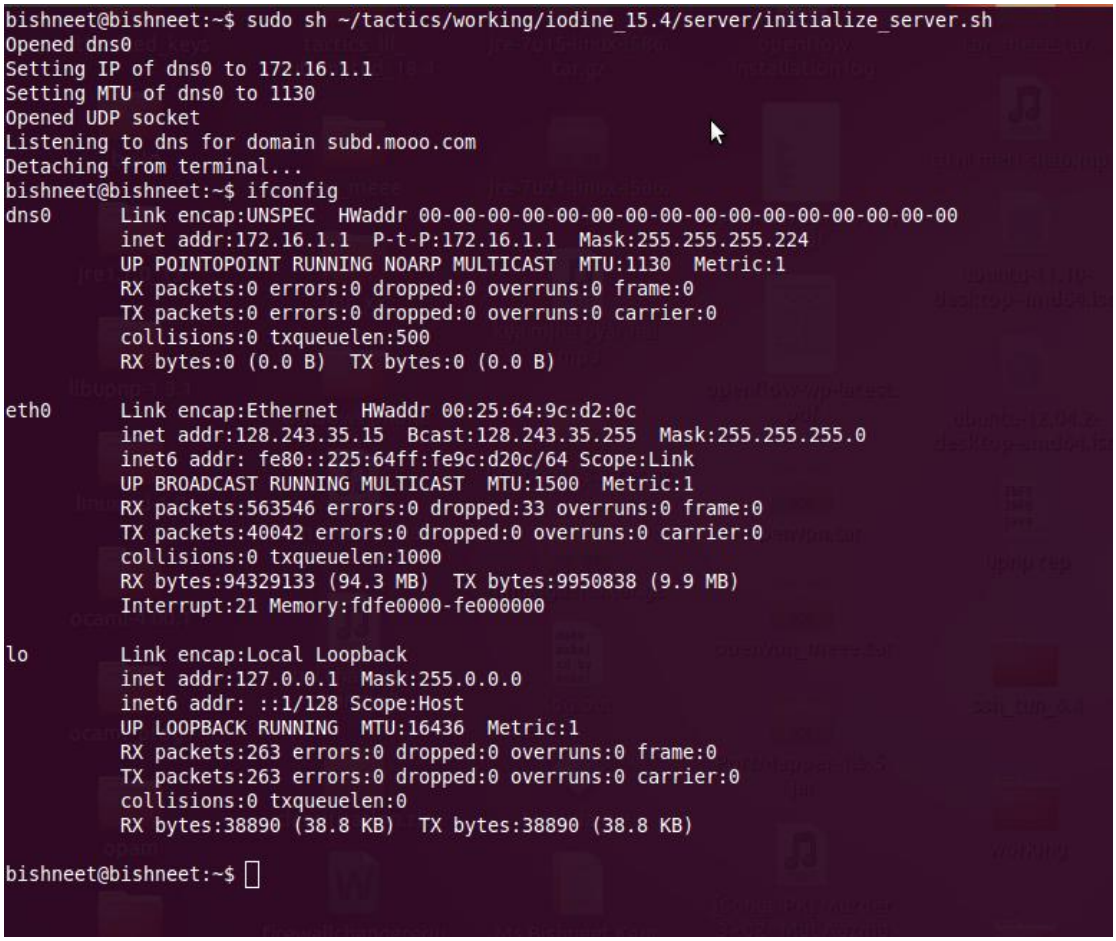

Figure17: iodine at server

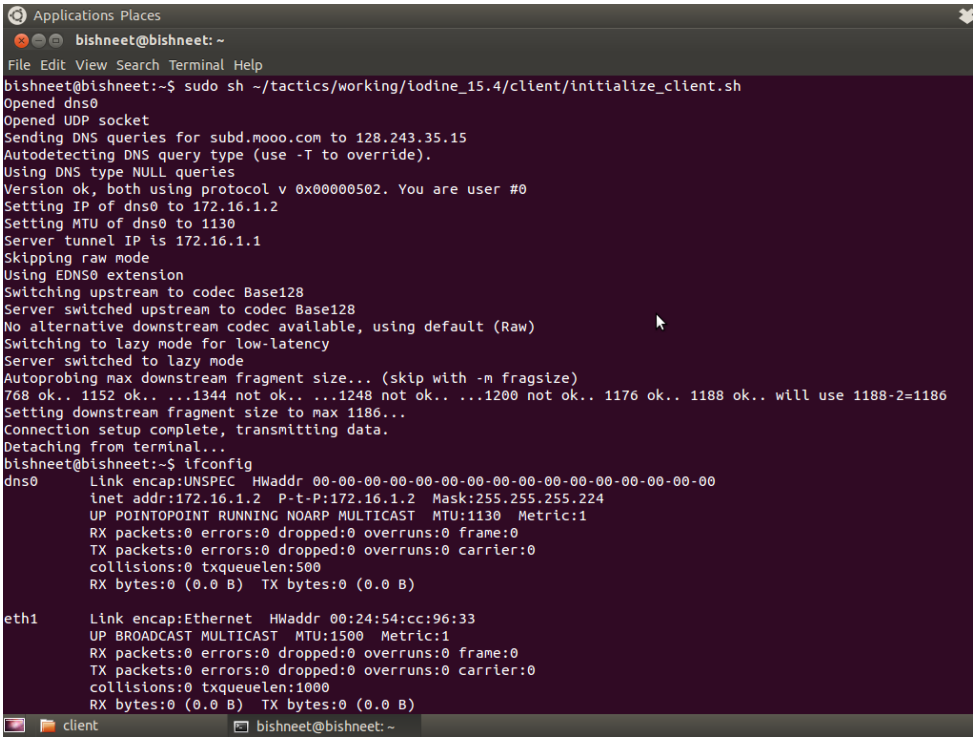

Figure18: iodine at client1

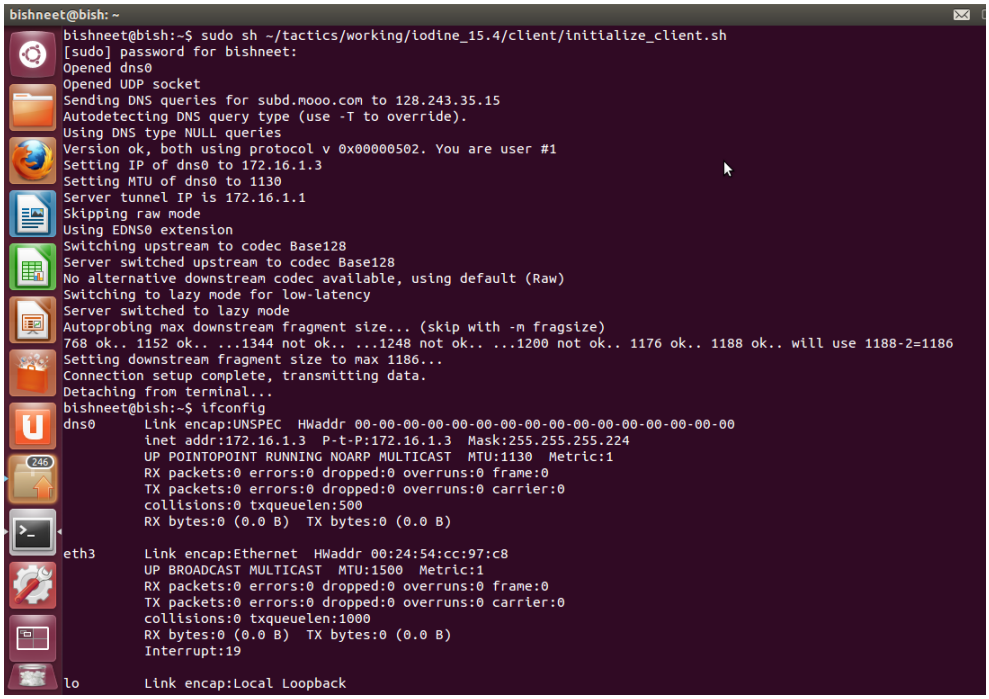

Figure19: iodine at client2

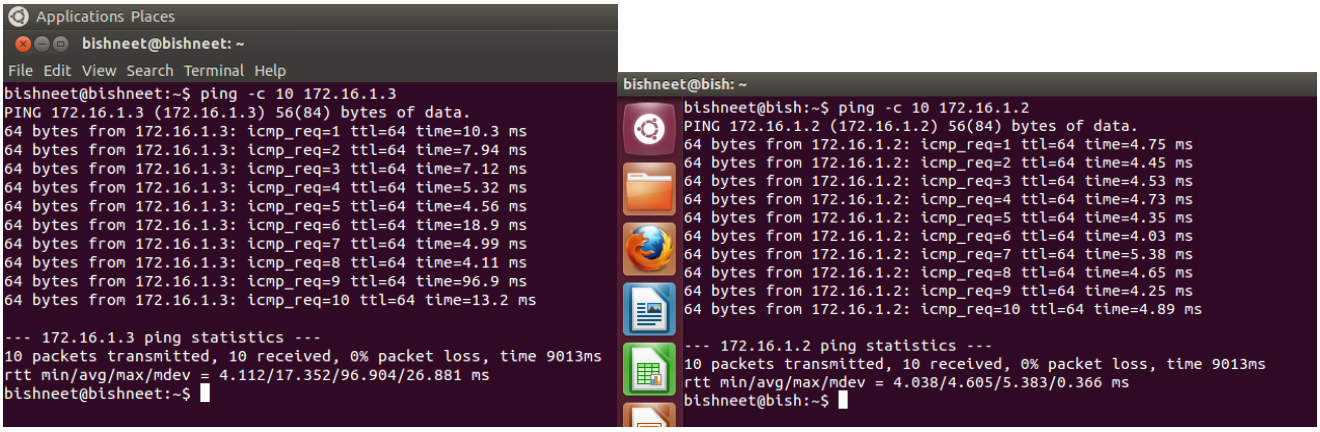

Figure20: pinging between both the clients

Problems faced:

- It can be the case that some relaying NS does not forward query and iodine client won't be able to reach iodine server. In that case, you need to manually find out the IP of the relaying NS which can forward your query bypassing the faulty NS and give it in the command used to start iodine at client. In this way, the query will directly go through another relaying NS bypassing or not directing to the faulty NS.
- Make sure no other process is using port 53 at both client and server. Either stop that process or change the port to which iodine should start at both client and server.

3.2.2 OpenVPN

Installations: openvpn

Steps performed:

1. Establish a PKI (public key infrastructure) which consists of:

- a separate certificate (also known as a public key) and private key for the server and each client, and

- a master Certificate Authority (CA) certificate and key which is used to sign each of the server and client certificates.

OpenVPN supports bidirectional authentication based on certificates, meaning that the client must authenticate the server certificate and the server must authenticate the client certificate before mutual trust is established.

Both server and client will authenticate the other by first verifying that the presented certificate was signed by the master certificate authority (CA), and then by testing information in the now-authenticated certificate header, such as the certificate common name or certificate type (client or server).

2. **At server**, for PKI management, easy-rsa is used, a set of scripts which is bundled with OpenVPN. Copy the scripts from /usr/share/doc/openvpn/examples/easy-rsa/2.0/* These scripts are needed to generate a master CA certificate/key, a server certificate/key, and certificates/keys for each client[5].

3. Server generates certificates and keys for each client. At each run, a new certificate and new keys are generated. So, clients need to copy this new certificate and key everytime.

4. A 'server.conf' file is generated automatically with all the parameters input by user. OpenVPN server is started at the port mentioned using 'server.conf' file.

5. **At client**, copy the client specific keys and certificates from the server at each client.

6. A 'client.conf' file is generated automatically with all the parameters input by user in 'parameters' file. OpenVPN client is then started at the same port on which server is listening using 'client.conf' file.

Output:

- Server will create a VPN using a virtual TUN network interface (for routing) and will listen for client connections on UDP port x (mentioned in parameters file), and distribute virtual addresses to connecting clients from the 10.8.0.0/24 subnet.
- Client will also have a virtual TUN network interface with virtual address being assigned by the server.
- Both clients can also ping each other using IP at their TUN interfaces.

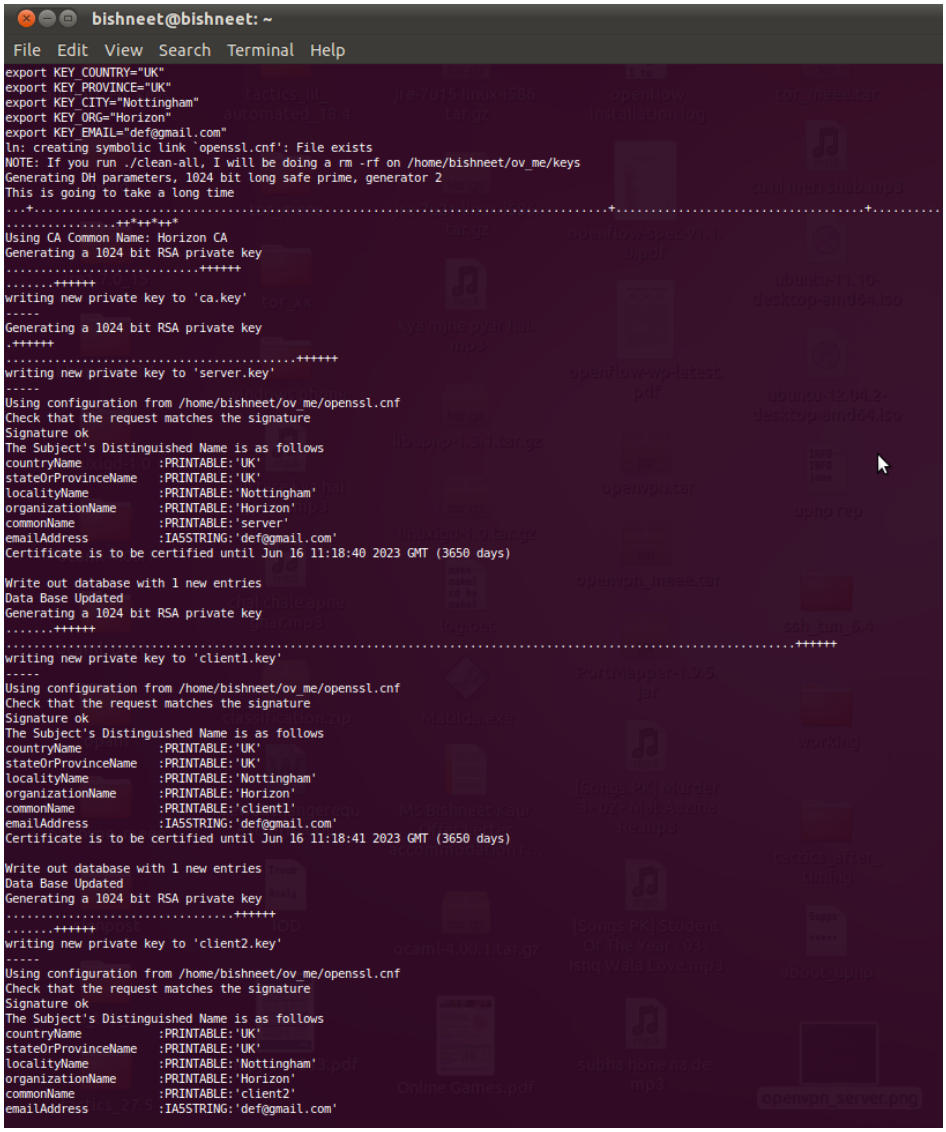

Figure21: key generation at server

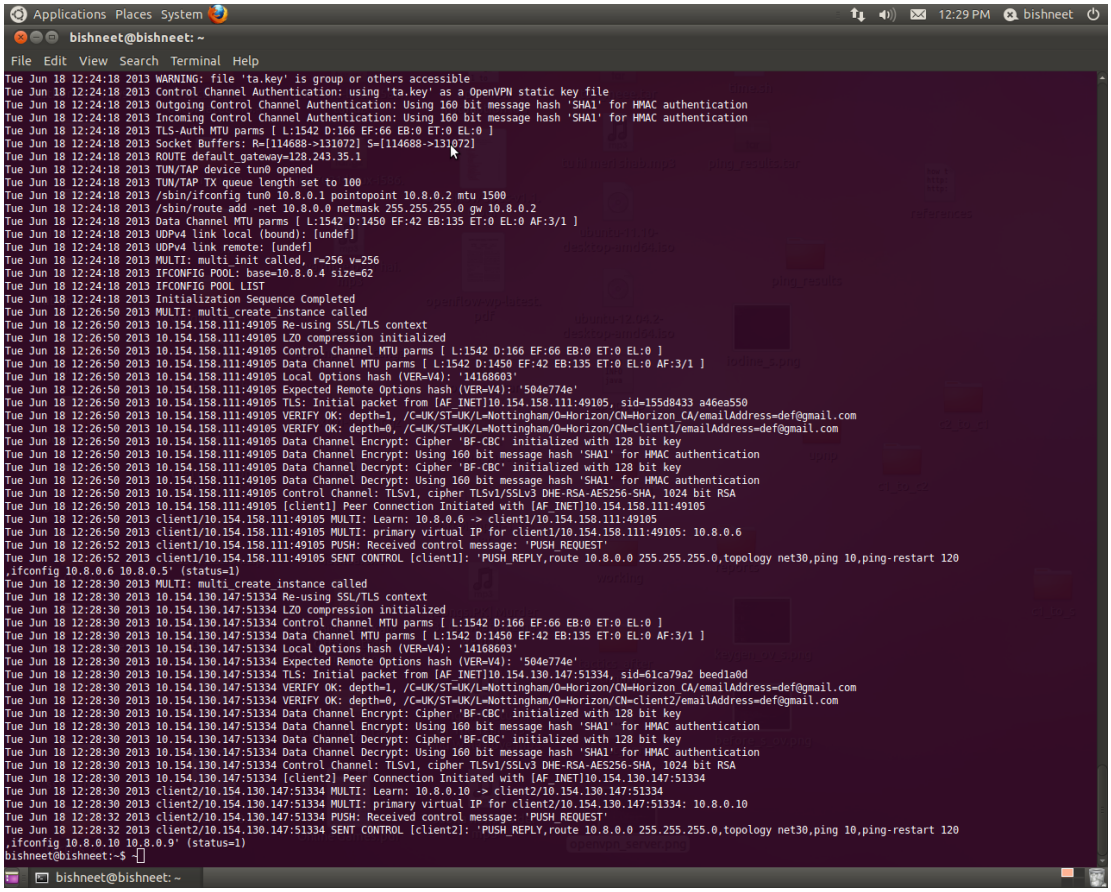

Figure22: openVPN at server

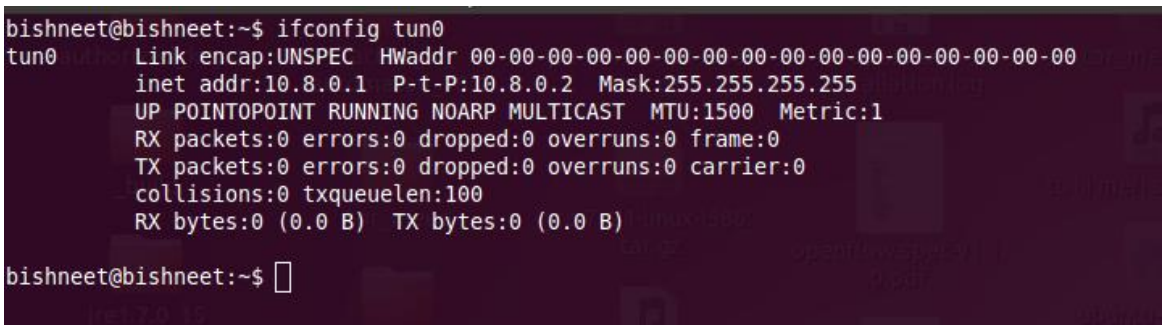

Figure23: tun interface created at server

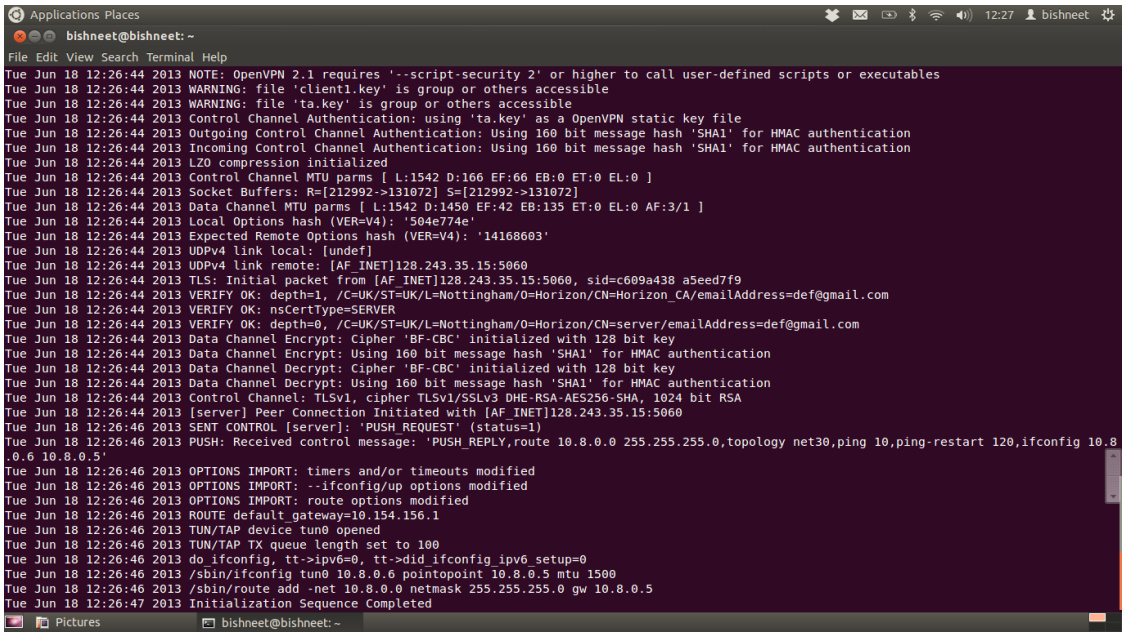

Figure24: openVPN at Client1

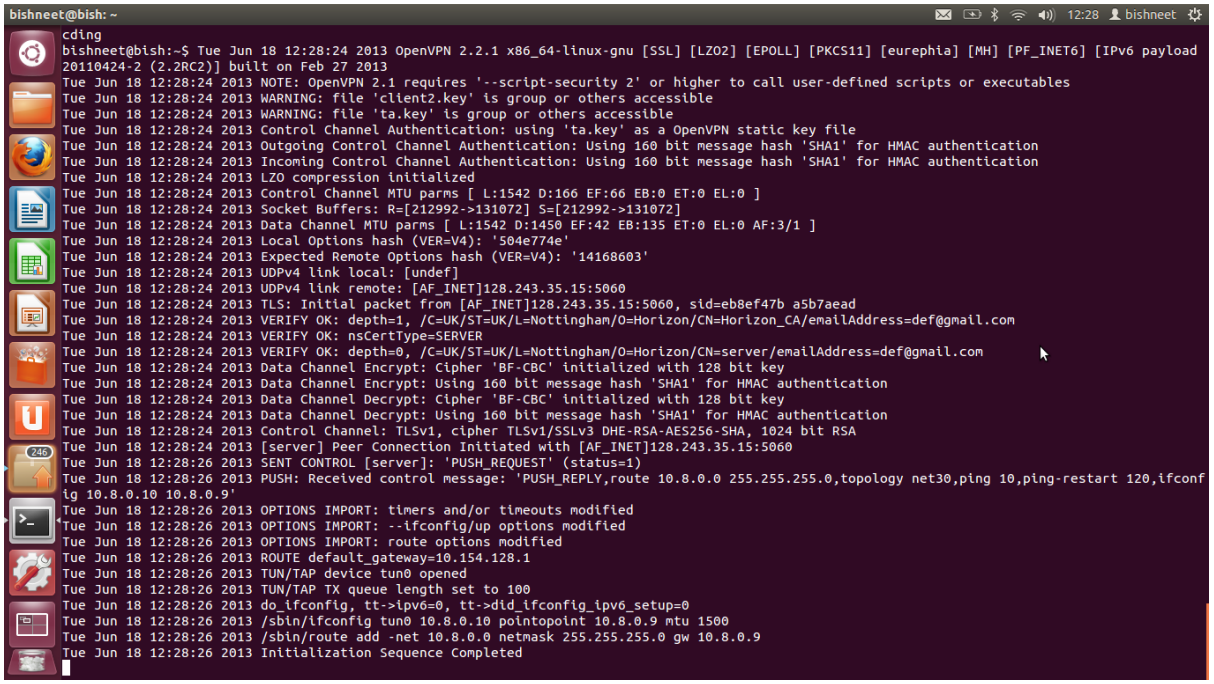

Figure25: openVPN at Client2

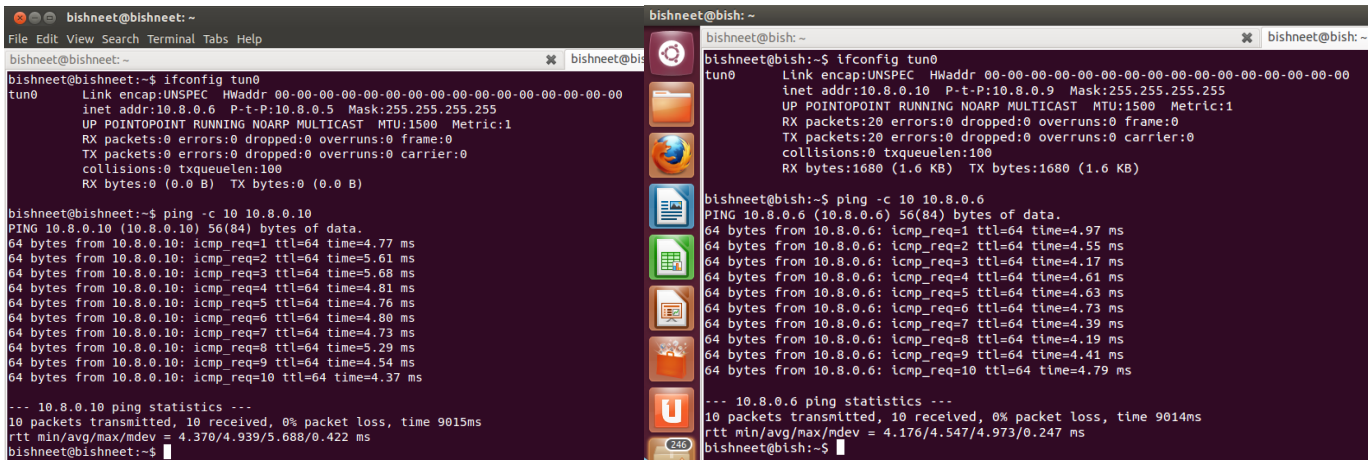

Figure26: pinging between both the clients

Problems faced:

- The dev (tun or tap) and proto (udp or tcp) directives should be consistent at both client and server config files. Also make sure that comp-lzo and fragment, if used, are present in both client and server config files.
- Different client names for different clients should be used while generating keys for them at server.
- A symbolic link is created because openssl.cnf file is not present in OpenVPN software and its absence does not allow OpenVPN to generate keys in Ubuntu 12.04 or higher versions. Symbolic link is created using: "ln -s openssl-1.0.0.cnf openssl.cnf" in 'server/aprior.sh'
- Make sure the port on which server and client contact is not firewalled/busy.

3.2.3 Ssh_tun

Installations: ssh, uml-utilities

Steps performed:

1. Configure the ssh server to permit tunnel. For this, add the directive

   PermitTunnel yes

   to '/etc/sshd_config' file and restart ssh server.

2. ssh to server from client giving Tunnel=point-to-point.

3. Configure the tun interfaces, ie, assign IP to them at client as well as server.

4. At both the clients, add routes in their routing tables to reach each other through the server. At the same time, enable IP forwarding at server.[8]

Output:

After successful setup, network connectivity is established between both the clients through server. They can ping each other's tun interfaces.

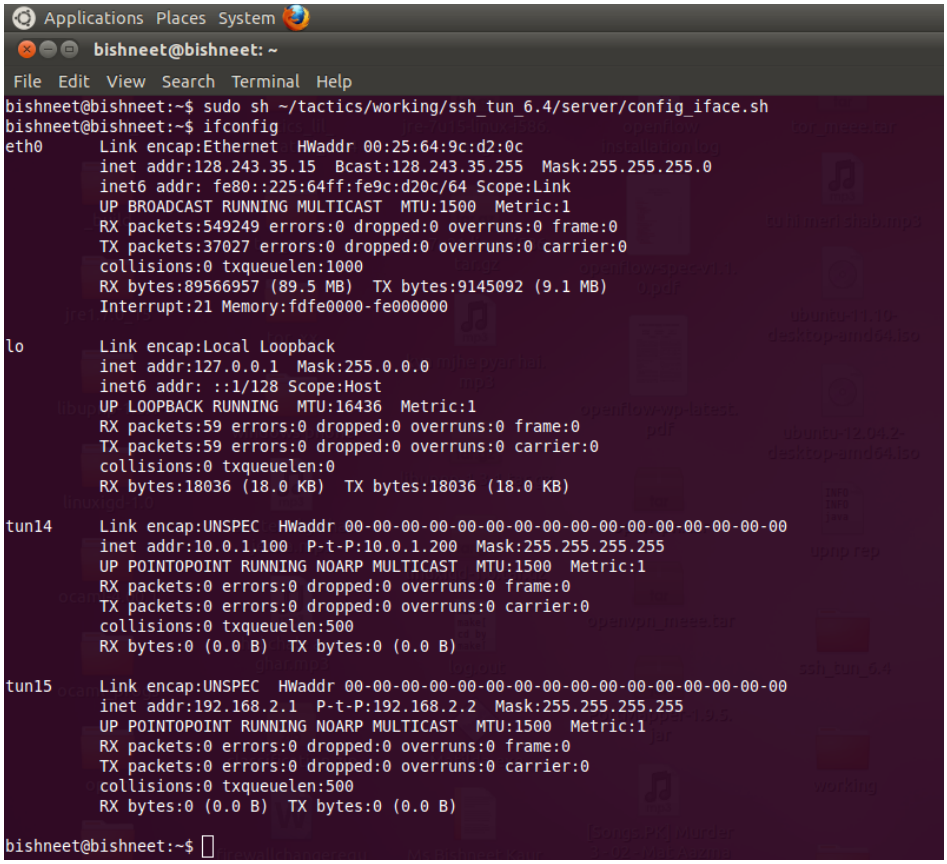

Figure27: ssh_tun at server

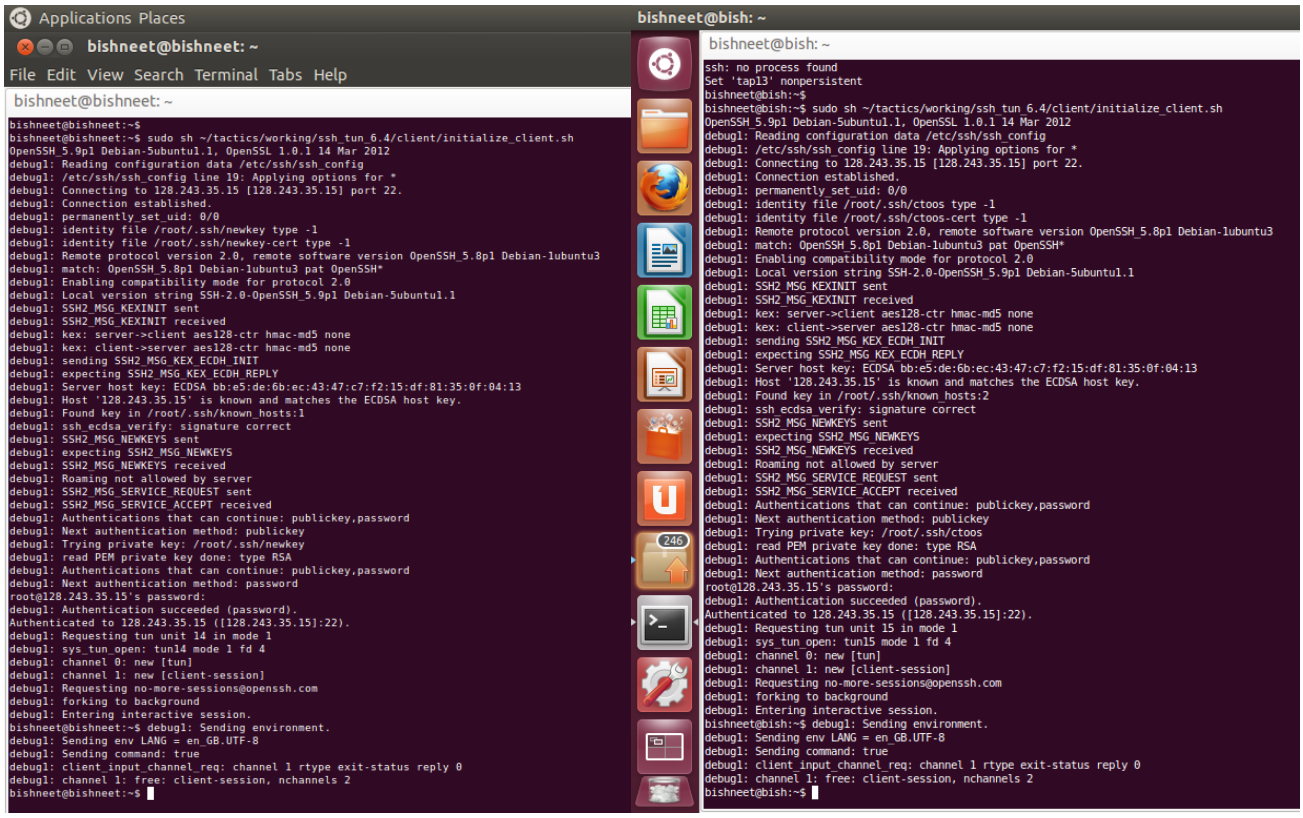

Figure28: pinging between both the clients

3.2.4 Ssh_tap

Installations: ssh, uml-utilities

Steps performed:

1. Configure the ssh server to permit tunnel. For this, add the directive

   PermitTunnel yes

   to '/etc/sshd_config' file and restart ssh server.

2. Make tap interfaces persistent at both server and client. A server needs to have different tap interfaces for different clients. When tap interfaces are persistent client need not login to server as root.

3. Configure the tap interfaces, ie, assign IP to them at client as well as server.

4. ssh to server from client giving Tunnel=Ethernet.

5. At both the clients, add routes in their routing tables to reach each other through the server. At the same time, enable IP forwarding at server.[8]

Output:

After successful setup, network connectivity is established between both the clients through server. They can ping each other's tap interfaces.

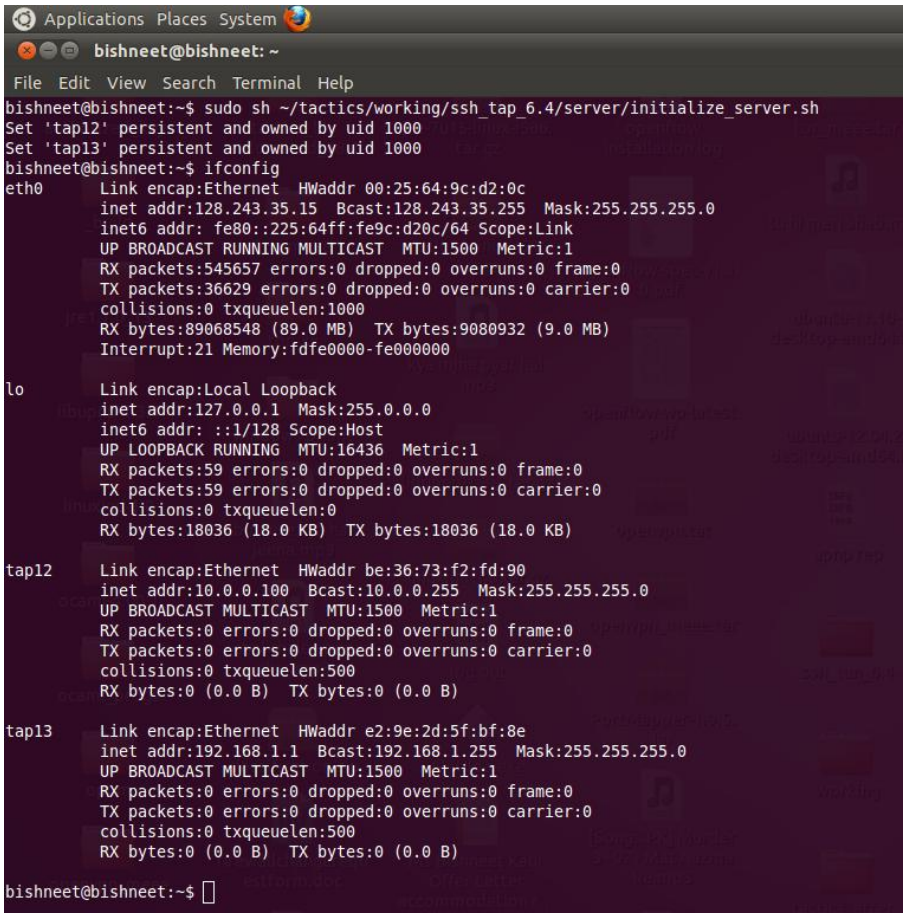

Figure29: ssh_tap at server

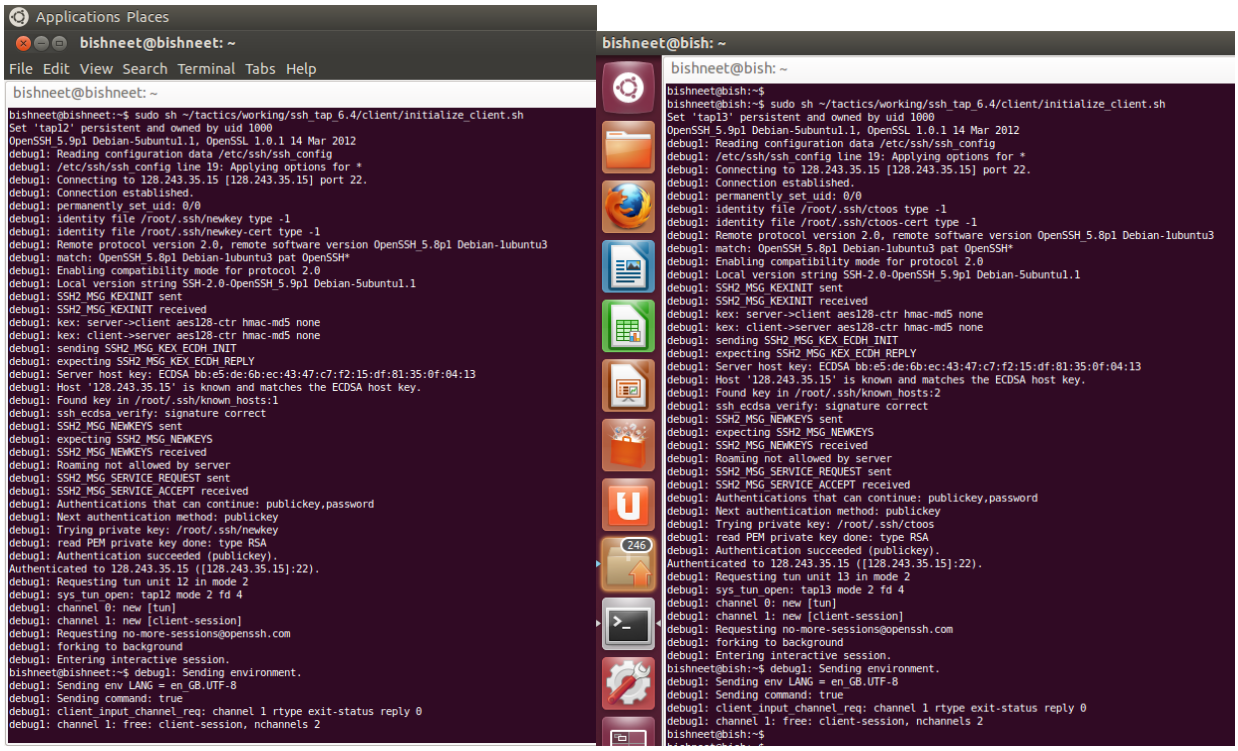

Figure30: ssh_tap at Client1 and Client2

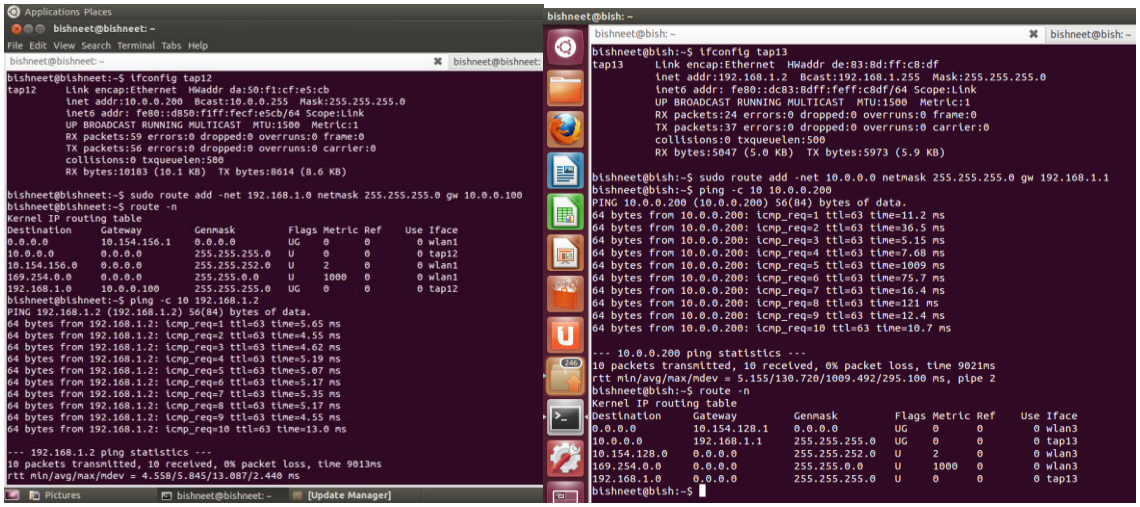

Figure31: pinging between both the clients

3.2.5 Tor

Installations: tor and curl (Currently, Tor version 0.2.2.35 and Tor version 0.2.1.30)

Steps Performed:

1. At server, a web server (http server with python) is set locally for hidden service at the specified port number[12].

2. Configuring hidden service: Edit (/tactics/working/tor_19.3/server/tor.conf)[10]

   - HiddenServiceDir: Give full path to a directory. This is a directory where Tor will store information about that hidden service. In particular, Tor will create a file here named ‘hostname’ which will tell you the onion URL. You don't need to add any files to this directory. This ‘hostname’ file needs to be sent to Tor client so that client can know the onion URL to connect to hidden service.
   - HiddenServicePort: This is the port where hidden service is running.

(This 'tor.conf' file is a copy of '/etc/torrc' file which is provided by tor software.)

In the HiddenServiceDir, two things are added:

   - private_key: Tor will generate a new public/private keypair for your hidden service. It is written into a file called "private_key". Do not share this with client.
   - hostname: This contains a short summary of service's public key -- it will look something like duskgytldkxiuqc6.onion. This is the public name for hidden service (16 characters) and this should be known to client.

3. Start http server for hidden service at the specified port and also Tor Server.

4. At Client, copy 'hostname' which contains onion URL to connect to hiddden service generated at Tor server side.

5. Redirect the reply coming from the hidden service to port 9050 where Tor client is listening.

6. Start Tor client.

Output:

When Client and Server both are running successfully, some html source code will be displayed at client's terminal window. This source code is served by the server running as hidden service to which Tor client has successfully connected.

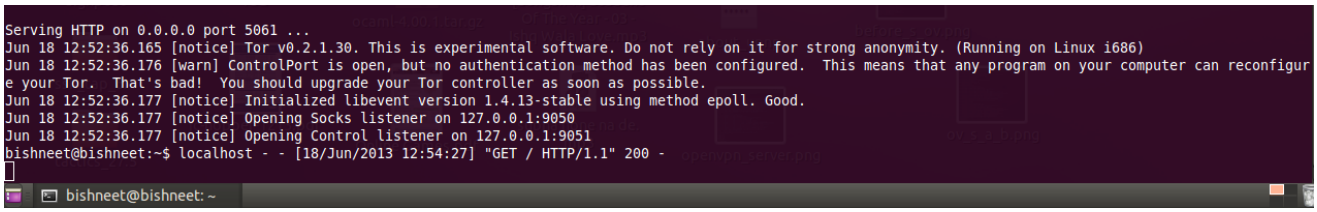

Figure32: Running tor and hidden service at server

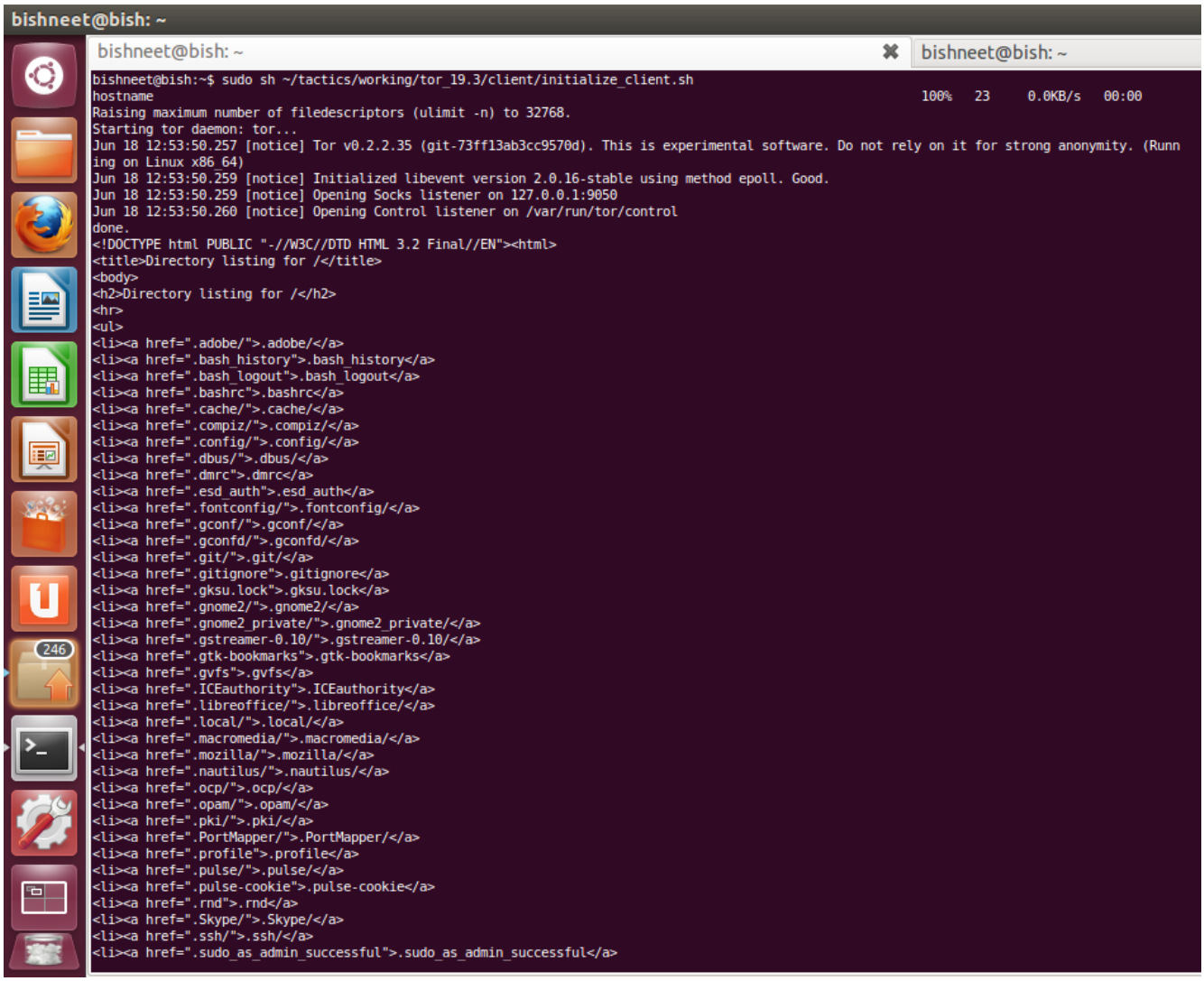

Figure33: Tor at client

Problems faced:

- Sometimes, even though Tor starts successfully at both client and server ends, client may not see html source code at its side, instead it gets a message like this "curl: (7) Can't complete SOCKS4 connection to 0.0.0.0:0. (91), request rejected or failed". This problem occurs due to server being not started immediately or causing delay in serving request. This is solved by stopping Tor at client side and restarting it.
- Everytime initialize_server.sh will start a python server and a new onion URL is generated and new 'hostname' file is added to HiddenServiceDir . Tor client need to have information about this new URL else it won't be able to connect to hidden service.
- Make sure nothing is running at the port where hidden service is being started and also tor is not running before.

3.2.6 UPnP

Steps performed:

At Server,

Libraries used: libupnp-1.3.1 and linuxigd-1.0

Device used: Dreamplug with Debian 6.0

Requirement: C++ compiler

1. Configure dreamplug to behave as UPnP router:
   - Download and install libupnp-1.3.1 [13]
   - Configure libupnp (source: README of libupnp-1.3.1)
   - Download and install linuxigd-1.0.tar.gz [14]

- Configure linuxigd.

At Client,

Software used: PortMapper-1.9.5.jar

Requirement: Java SE Runtime Environment 6.0

1. Install JAVA

2. Download PortMapper-1.9.5.jar [15]

3. Start UPnP client.

Output:

- Whatever the port mapping is specified at client side is shown at router and it is added as an iptable rule at the router.
- Remote host can send packets successfully at the mapped port of the UPnP client. This means NAT punching is done successfully.

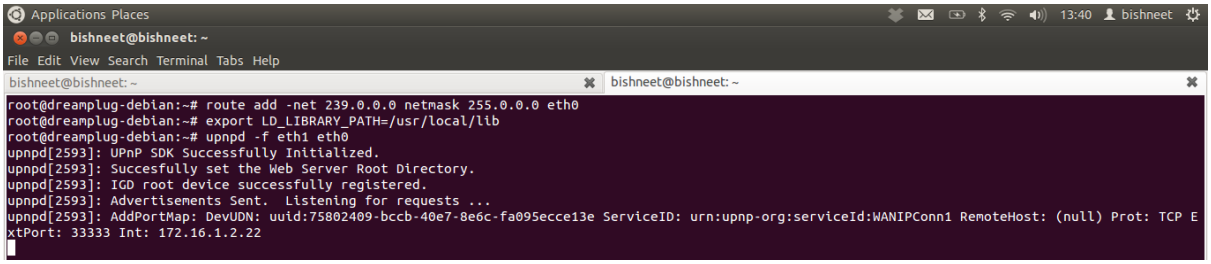

Figure34: UPnP at server

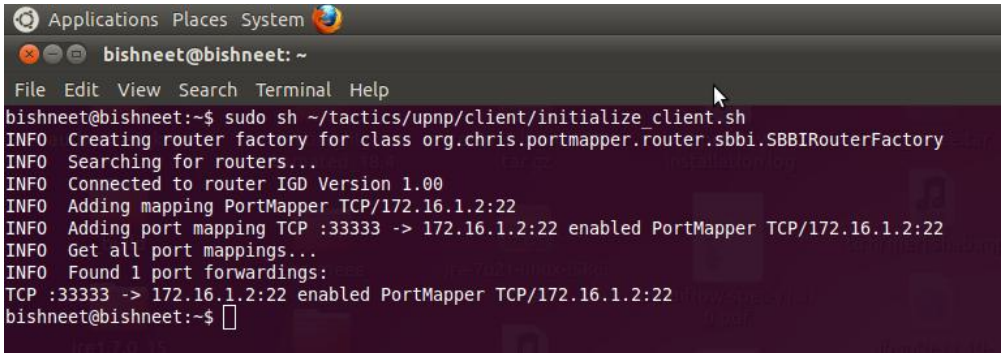

Figure35: UPnP at client

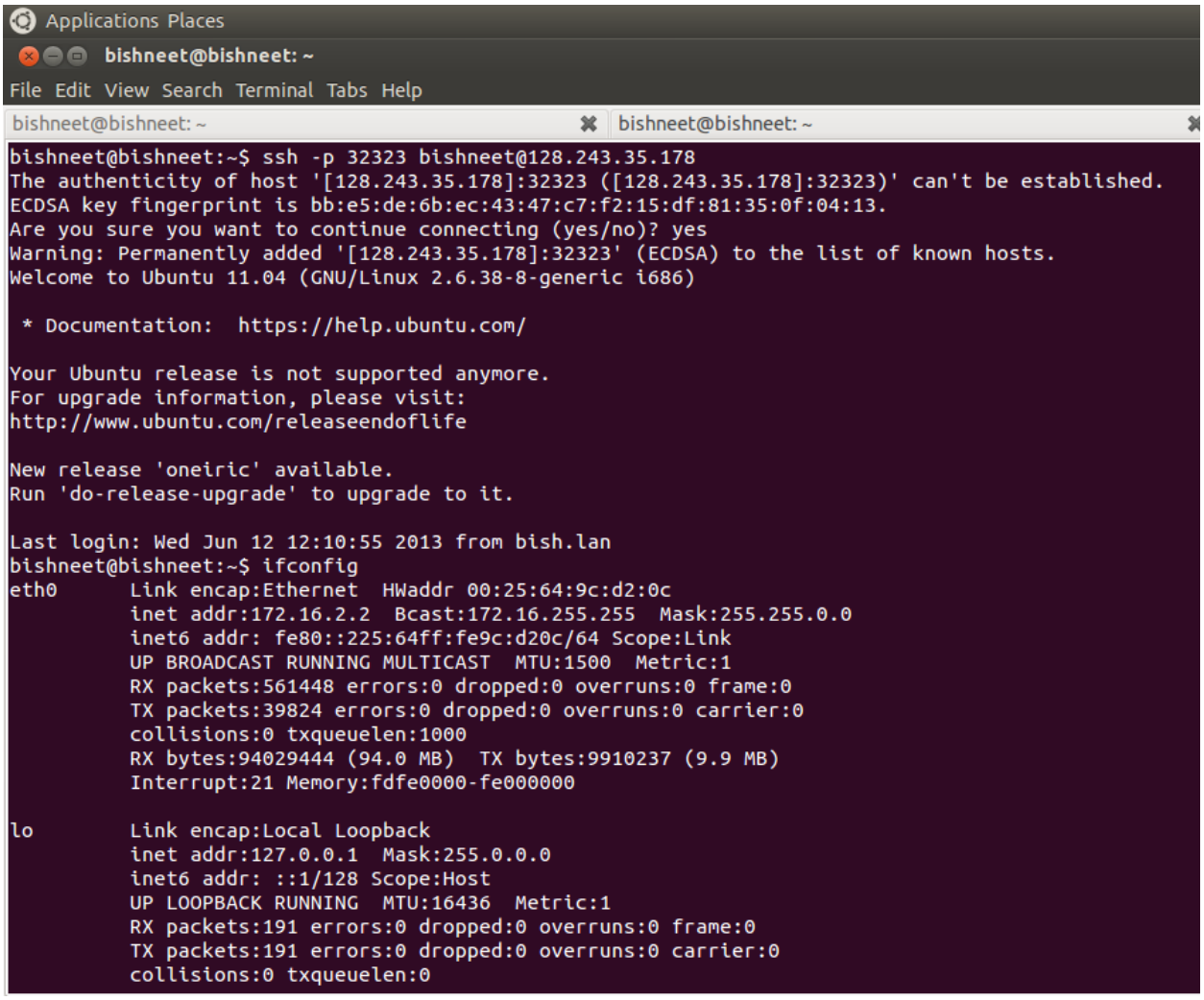

Figure36: Remote host can successfully ssh to UPnP client

Problems faced:

- IGD can't be made to run on virtual interfaces. Therefore, a system with two physical interfaces is a prerequisite.
- Sometimes while installing linuxigd-1.0 and libupnp-1.3.1 at router, 'make install' may not work if not done as sudo. In that case run: sudo make install

3.2.7 Starting all tactics automatically

Shell scripts are developed which can do all the above mentioned task at one go, ie, all the tactics are started one after the other at server and client sides. For this, perform the steps mentioned below:

1. Generate ssh keys between client and server for passwordless login and copying files between them.
2. There is a ‘config’ file which contains general parameters necessary to set up communication between client and server. Provide the parameters.
3. Two scripts (‘server.sh’ and ‘client.sh’) are written which will install necessary packages and start all the tactics one after another at both client and server. Run these scripts after making changes for tactic specific parameters in ‘parameters’ file available in each tactic folder at client and server side.
4. Copy the repository from https://github.com/signposts/tactics.git

4. Results and observations

4.1 System specifications

	Client1	Server	Client2
OS	Ubuntu Release 12.04 (precise) 64-bit Kernel Linux 3.5.0-23-generic GNOME 3.4.2	Ubuntu Release 11.04 (natty) Kernel Linux 2.6.38-8-generic GNOME 2.32.1	Ubuntu Release 12.04 (precise) 64-bit Kernel Linux 3.5.0-23-generic GNOME 3.4.2
Hardware	**Memory**: 2.9 GiB **Processor**: Intel® Core™ i3 CPU M 350 @ 2.27GHz × 4	**Memory**: 3.2 GiB **Processor0**: Intel® Core™ 2 Duo CPU E7500 @ 2.93 GHz **Processor1**: Intel® Core™ 2 Duo CPU E7500 @ 2.93 GHz	**Memory**: 2.9 GiB **Processor**: Intel® Core™ i3 CPU M 350 @ 2.27GHz × 4
Extra info	Support Gigabit Ethernet	Support Gigabit ethernet	Support Gigabit ethernet
OpenVPN	2.0	2.0	2.0
SSH	OpenSSH_5.9p1 Debian-5ubuntu1.1, OpenSSL 1.0.1 14 Mar 2012	OpenSSH_5.8p1 Debian-1ubuntu3, OpenSSL 0.9.8o 01 Jun 2010	OpenSSH_5.9p1 Debian-5ubuntu1.1, OpenSSL 1.0.1 14 Mar 2012
Iodine	version: 0.6.0-rc1 from 2010-02-13	version: 0.6.0-rc1 from 2010-02-13	version: 0.6.0-rc1 from 2010-02-13

Table 2: System specifications for testing

4.2 Results

Key:

C1: Client 1

C2: Client 2

SS: Signpost Server

4.2.1 Measuring Bandwidth

The below table shows the bandwidth offered by each tactic. Bandwidth is measured using **iperf**.

All the values are in Mbits/sec, else otherwise stated

		C1 to SS		C2 to SS		C2 to C1	
		Iperf server(SS)	Iperf client(C1)	Iperf server(SS)	Iperf client(C2)	Iperf server(C1)	Iperf client(C2)
Round1	No tactic	787	789	792	795	744	746
	Iodine	1000Kb/s	1.32	1.04	1.23	2.41	2.84
	OpenVPN	229	229	224	224	140	140
	ssh_tap	164	165	168	169	152	152
	ssh_tun	160	161	166	167	154	155
Round2	No tactic	828	825	829	827	727	725
	Iodine	981 Kb/s	1.15	920 Kb/s	1.12	2.4	2.81
	OpenVPN	230	230	229	229	139	139
	ssh_tap	164	165	163	164	155	155
	ssh_tun	161	162	161	160	153	154
Round3	No tactic	791	793	831	833	753	755
	Iodine	994 Kb/s	1.16	1.03	1.06	2.39	2.81
	OpenVPN	230	230	227	227	139	139
	ssh_tap	162	163	163	164	155	156
	ssh_tun	162	163	161	160	154	155

Round4	**No tactic**	**787**	**790**	**792**	**794**	**735**	**737**
	Iodine	**960 Kb/s**	**1.15**	**963 Kb/s**	**1.16**	**2.4**	**2.84**
	OpenVPN	**233**	**233**	**228**	**228**	**136**	**136**
	ssh_tap	**163**	**164**	**164**	**165**	**154**	**156**
	ssh_tun	**161**	**162**	**162**	**161**	**153**	**153**
		SS to C1		**SS to C2**		**C1 to C2**	
		Iperf server(C1)	**Iperf client(SS)**	**Iperf server(C2)**	**Iperf client(SS)**	**Iperf server(C2)**	**Iperf client(C1)**
Round1	**No tactic**	**818**	**817**	**886**	**886**	**727**	**730**
	Iodine	**4**	**4.11**	**4.02**	**4.12**	**2.39**	**2.76**
	OpenVPN	**202**	**202**	**203**	**203**	**149**	**149**
	ssh_tap	**163**	**163**	**166**	**166**	**151**	**151**
	ssh_tun	**161**	**161**	**160**	**161**	**155**	**155**
Round2	**No tactic**	**902**	**903**	**891**	**892**	**769**	**771**
	Iodine	**4.07**	**4.07**	**4.11**	**4.13**	**2.39**	**2.84**
	OpenVPN	**203**	**204**	**206**	**206**	**140**	**140**
	ssh_tap	**168**	**168**	**164**	**164**	**154**	**155**
	ssh_tun	**165**	**165**	**158**	**158**	**151**	**152**
Round3	**No tactic**	**857**	**858**	**836**	**837**	**737**	**740**
	Iodine	**4.02**	**4.1**	**3.93**	**4.04**	**2.39**	**2.82**
	OpenVPN	**203**	**203**	**204**	**205**	**141**	**141**
	ssh_tap	**168**	**168**	**162**	**163**	**152**	**152**
	ssh_tun	**162**	**162**	**157**	**157**	**151**	**151**

Round4	**No tactic**	**831**	**832**	**844**	**845**	**739**	**741**
	Iodine	**3.94**	**4.05**	**4.05**	**4.08**	**2.38**	**2.78**
	OpenVPN	**204**	**204**	**205**	**205**	**141**	**141**
	ssh_tap	**164**	**164**	**161**	**161**	**153**	**153**
	ssh_tun	**164**	**164**	**157**	**157**	**154**	**154**

Table 3: Bandwidth offered by each tactic

4.2.2 Measuring Latency

The below table gives the latency offered by each tactic. Latency is measured using **ping**.

The table shows the average latency for 10 readings.

All the values are in ms (milli seconds)

		C1 to SS	**C2 to Ss**	**C1 to C2**
Round 1	**No tactic**	**0.411**	**0.409**	**0.685**
	Iodine	**1.002**	**1.006**	**1.439**
	OpenVPN	**0.688**	**0.693**	**1.248**
	ssh_tap	**0.697**	**0.78**	**1.32**
	ssh_tun	**0.606**	**0.77**	**1.253**
Round 2	**No tactic**	**0.412**	**0.413**	**0.635**
	Iodine	**0.985**	**0.999**	**1.463**
	OpenVPN	**0.702**	**0.756**	**1.217**
	ssh_tap	**0.68**	**0.69**	**1.32**
	ssh_tun	**0.707**	**0.668**	**1.285**

Round 3	**No tactic**	**0.407**	**0.4**	**0.643**
	Iodine	**0.937**	**0.97**	**1.403**
	OpenVPN	**0.69**	**0.731**	**1.217**
	ssh_tap	**0.71**	**0.677**	**1.318**
	ssh_tun	**0.676**	**0.709**	**1.29**
		S to C1	**S to C2**	**C2 to C1**
Round 1	**No tactic**	**0.336**	**0.341**	**0.622**
	Iodine	**0.839**	**0.867**	**1.451**
	OpenVPN	**0.610**	**0.649**	**1.234**
	ssh_tap	**0.617**	**0.659**	**1.314**
	ssh_tun	**0.612**	**0.66**	**1.315**
Round 2	**No tactic**	**0.339**	**0.335**	**0.692**
	Iodine	**0.825**	**0.879**	**1.416**
	OpenVPN	**0.583**	**0.647**	**1.216**
	ssh_tap	**0.599**	**0.653**	**1.303**
	ssh_tun	**0.628**	**0.64**	**1.222**
Round 3	**No tactic**	**0.335**	**0.335**	**0.7**
	Iodine	**0.843**	**0.895**	**1.449**
	OpenVPN	**0.588**	**0.621**	**1.266**
	ssh_tap	**0.615**	**0.644**	**1.385**
	ssh_tun	**0.663**	**0.643**	**1.327**

Table 4: Latency by each tactic

4.3 Observations

- Iodine offers least bandwidth and comparatively higher latency. So, it is the least favourable tactic to be used.
- OpenVPN offers higher bandwidth between client and server as compared to ssh_tap and ssh_tun but when two clients connect using OpenVPN bandwidth reduces and becomes worse than ssh_tun and ssh_tap. Ssh_tun and ssh_tap, however, offers relatively same bandwidth throughout.

GLOSSARY

- **NAT** (Network Address Translation): NAT is an Internet standard that enables a local area network (LAN) to use one set of IP addresses for internal traffic and a second set of addresses for external traffic. A *NAT box* located where the LAN meets the Internet makes all necessary IP address translations. NAT devices are commonly used to alleviate IPv4 address exhaustion by allowing the use of private IP addresses on home and corporate networks behind routers with a single public IP address facing the public Internet. The internal network devices communicate with hosts on the external network by changing the source address of outgoing requests to that of the NAT device and relaying replies back to the originating device.
- **NAT Traversal**: NAT traversal is a general term for techniques that establish and maintain Internet protocol connections traversing network address translation (NAT) gateways.
- **SSH** (**Secure Shell):** SSH is a cryptographic network protocol for secure data communication, remote command-line login, remote command execution, and other secure network services between two networked computers that connects, via a secure channel over an insecure network. It uses public key cryptography to authenticate the remote computer and allow it to authenticate the user.
- **DreamPlug**: Dreamplug is a compact and low power plug computer running Debian Linux, based on Marvell's Kirkwood 88F6281 ARM9E SoC. It is intended

to be a device that could act as a web server, a printing server or any other network service. It uses micro-SD internal storage and an external Secure Digital slot, but also offers USB ports and a Serial ATA port to connect external disks.

- **Ping:** Ping is a computer network administration utility used to test the reachability of a host on an Internet Protocol (IP) network and to measure the round-trip time for messages sent from the originating host to a destination computer. It operates by sending Internet Control Message Protocol (ICMP) *echo request* packets to the target host and waiting for an ICMP response. In the process it measures the time from transmission to reception (*round-trip time*) and records any packet loss

- **Iperf**: Iperf is a commonly used network testing tool that can create TCP and UDP data streams and measure the throughput of a network that is carrying them. When one runs TCP tests, what iperf measures is the pace with which it sends data to TCP/IP stack; TPC/IP stack will only accept data from application when buffers are not full. If the buffer is huge, iperf will see high throughput initially, then it will drop. If there's congestion or retransmission going on, iperf will see it as lower throughput.

BIBLIOGRAPHIC REFERENCES

[1] Rotsos C., Howard H., Sheets D., Mortier R., Madhavapeddy A., Chaudhary A., Crowcroft J. “Lost In the Edge: Finding Your Way With DNSSEC Signposts”. ACM FOCI 2013 workshop 2013, in submission.

[2] http://code.kryo.se/iodine/README.html

[3] http://blog.knuples.net/post/setting-up-ip-over-dns-nstx-tunneling-with-iodine/

[4] http://tunnel-ninja.webege.com/article/view/iodine-tunneling-over-dns-request

[6] http://openvpn.net/index.php/open-source/documentation/howto.html

[7] http://backreference.org/2010/03/26/tuntap-interface-tutorial/

[8] http://backreference.org/2009/11/13/openssh-based-vpns/

[9] https://www.torproject.org/about/overview.html.en

[10] https://www.torproject.org/docs/tor-hidden-service.html.en

[11] https://www.torproject.org/docs/hidden-services.html.en

[12] http://www.linuxjournal.com/content/tech-tip-really-simple-http-server-python

[13] http://www.sourceforge.net/projects/upnp

[14] http://linux-igd.sourceforge.net/

[15] http://upnp-portmapper.sourceforge.net/