UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea triennale in *Ingegneria Informatica e dell'Automazione*

Implementazione di OpenVPN su router 4G per site-to-site vpn in ambiente CG-NAT

Implementation of OpenVPN on a 4G Router for site-to-site VPN in CG-NAT environment

Relatore:
Prof. Ennio Gambi
Correlatore:
Ing. Adelmo De Santis

Laureando:
Simone Viozzi

Anno Accademico 2021/2022

Prefazione work-in-progress

TODO: da riscrivere

Indice

Elenco delle figure

Tutti i diagrammi sono stati realizzati con il tool *diagrams.net* [5], usando le icone fornite da Huawei [3].

Capitolo 1

Introduzione

1.1 TCP/IP e modello a strati

Internet costituisce la più grande rete di comunicazione al mondo, con più di 5 miliardi di dispositivi connessi nello stesso momento [6]. Eppure la sua architettura di base è relativamente semplice, tutto è incentrato su un set di protocolli e i dispositivi che li implementano.

Il suo obbiettivo è quello di collegare utenti attraverso una rete geografica, indipendentemente dalla distanza o da quali mezzi fisici vengano usati per trasferire i dati. Deve quindi essere in grado di gestire una grande varietà di applicazioni, tipi di rete e mezzi di trasmissione. Per garantire la flessibilità necessaria e non aggiungere troppa complessità, l'architettura di Internet è sta suddivisa in alcuni strati, ad esempio il modello TCP/IP è costituito da 4 strati.

1.1.1 Internet Protocol Suite

Per comunicare su Internet, gli Host devono implementare un set di protocolli che costituiscono l'*Internet protocol suite* [1]. I protocolli si suddividono in strati logici che li raggruppano in 4 categorie, ogni Host deve implementare almeno un protocollo per ogni strato.

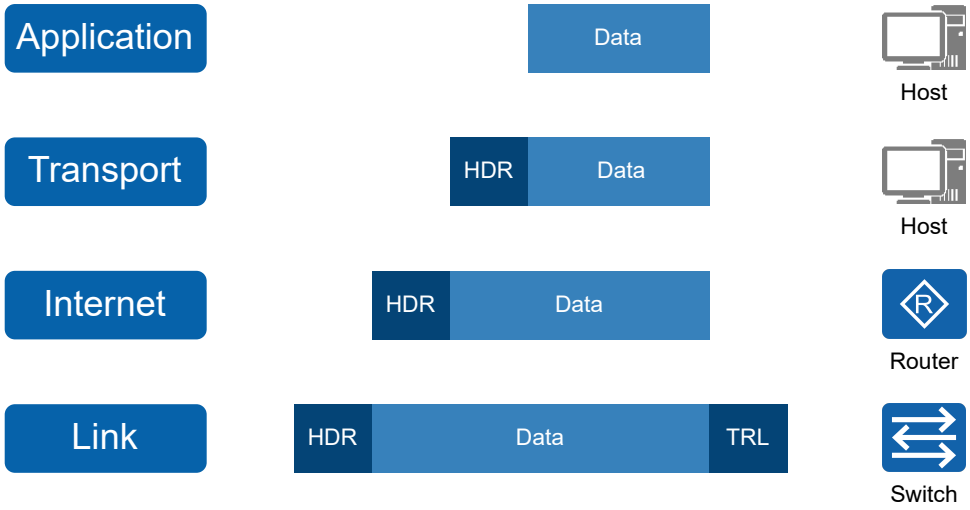


Figura 1.1: Rappresentazione degli strati del modello TCP/Ip, con relativo incapsulamento e dispositivo di dominio

Vediamo una breve descrizione dei layer e delle loro funzioni:

layer 4: **Application**: Il livello applicazione è il layer più alto dell'*Internet protocol suite*, i protocolli di questo livello si suddividono in protocolli utente e di supporto.
I protocolli utente espongono un servizio direttamente all'utente finale, alcuni esempi sono: http, ftp, ssh, etc. Mentre i protocolli di supporto non sono direttamente usati dagli utenti ma sono comunque necessari per il funzionamento della rete, alcuni esempi sono: DNS, SNMP etc.

layer 3: **Transport**: Il livello di trasporto fornisce una comunicazione end-to-end tra per le applicazioni, infatti in generale il campo data del livello di trasporto non viene letto da nessuno se non l'applicazione di sorgente e destinazione. I protocolli principali di questo livello sono TCP e UDP: il TCP è connection-oriented e fornisce alta affidabilità; mentre l'UDP è connection-less, quindi ogni inaffidabilità della rete deve essere gestita a livello applicazione.

layer 2: **Internet**: Tutti i protocolli di trasporto usano il protocollo Internet (IP) per portare i dati dall'host sorgente alla destinazione. Al contrario dei protocolli di livello trasporto il protocollo IP non è end-to-end, quindi è intrinsecamente di tipo connection-less. Non fornisce quindi nessuna garanzia che il pacchetto arrivi a destinazione, o arrivi danneggiato o duplicato. I layer sopra al livello IP sono responsabili di mantenere l'affidabilità dei servizi quando essa è richiesta. Di questo layer fanno parte i protocolli IP, ICMP, etc.

layer 1: **Link**: È il layer più vicino al mezzo fisico su cui viaggiano i dati, ogni host deve implementare il protocollo usato per la specifica interfaccia che usa. Ad esempio un'host con un'interfaccia Ethernet deve implementare i protocolli Ethernet II e IEEE 802.3.

1.1.2 Incapsulamento work-in-progress

Ogni protocollo di ogni layer aggiunge un header e un trailer, incapsulando il prodotto del layer precedente nel suo campo data, possiamo vedere una rappresentazione grafica in fig. 1.1.

Possiamo vedere l'incapsulamento in azione catturando un pacchetto con il programma *Wireshark*, ad esempio in questo caso si tratta di un pacchetto proveniente da una pagina web https:

Wireshark code: 1.1.1

```
1 > Frame 43408: 93 bytes on wire (744 bits), 93 bytes captured (744 bits) on interface wlp5s0, id 0
1 > Ethernet II, Src: IntelCor_eb:91:5f (cc:d9:ac:eb:91:5f), Dst: HuaweiDe_27:a9:24 (0c:e4:a0:27:a9:24)
2 > Internet Protocol Version 4, Src: 192.168.8.119, Dst: 142.250.180.174
3 > Transmission Control Protocol, Src Port: 36354, Dst Port: 443, Seq: 36720, Ack: 13639, Len: 39
4 > Transport Layer Security
> TLSv1.3 Record Layer: Application Data Protocol: http-over-tls
Opaque Type: Application Data (23)
Version: TLS 1.2 (0x0303)
Length: 34
Encrypted Application Data: 389bf9516cce567d0d90ef62ba2a87376091fedb7f66f3b9e60e45a39376b1ae667a
    [Application Data Protocol: http-over-tls]
```

TODO: da perfezionare

Quindi andando ad analizzare lo stack di protocolli si vede:

- **Transport Layer Security**[9]: È il dato effettivo che è stato trasmesso in rete. In questo caso consiste in un pacchetto http cifrato con il protocollo TLS.
- **Transmission Control Protocol**[8]: incapsula il layer applicazione in un layer di trasporto usando il protocollo TCP. Le informazioni principali contenute nell'header sono la porta sorgente e destinazione, ciò permette di individuare a quale applicazione è destinato il pacchetto.
- **Internet Protocol Version 4**[7]: incapsula il layer di trasporto in un pacchetto IP. Nell'header IP sono contenuti l'indirizzo IP di sorgente e destinazione, ciò permette di individuare a quale host è destinato il pacchetto.
- **Ethernet II**[11]: il questo caso il mezzo fisico è una porta Ethernet quindi il pacchetto IP viene incapsulato con il protocollo Ethernet II. Nell'header Ethernet II sono contenuti

l'indirizzo MAC di sorgente e destinazione, ciò permette di individuare a quale host è l'interfaccia fisica di destinazione.

Possiamo vedere graficamente il pacchetto descritto in code:1.1.1 e le informazioni principali di ogni header:

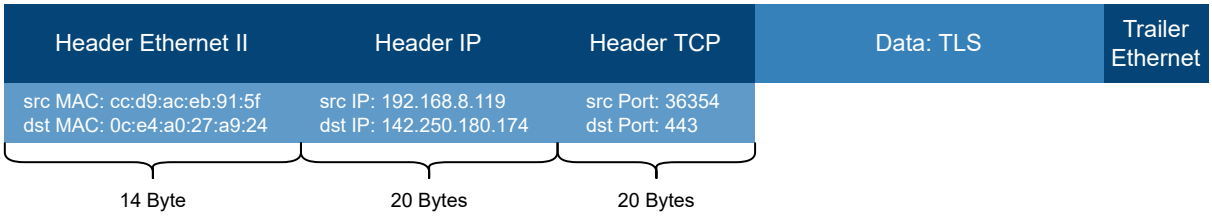


Figura 1.2: Incapsulamento

1.1.3 IP Network Address Translator (NAT)

Per mitigare il problema della saturazione degli indirizzi IPv4 [12], è stato introdotto un dispositivo di rete che consente di riusare spazi di indirizzi privati e ridurre il numero di Ipv4 pubblici necessari.

Il NAT [2] permette di effettuare una mappatura da uno spazio di indirizzi a un altro, modificando così le informazioni di routing nel pacchetto IP. La mappatura può essere sia uno-a-uno, nel caso di *Static NAT* e *Dynamic NAT*; sia molti-a-molti, nel caso di *Network Address and Port Translation* (NAPT), in cui oltre all'IP viene modificata anche la porta del protocollo di trasporto. Ciò consente di mappare un grande numero di indirizzi privati in relativamente pochi indirizzi pubblici.

Una delle applicazioni più comuni è inserirlo nel gateway di una rete privata domestica, ciò permette di mascherare i dispositivi nella rete privata e far si che la rete esterna veda tutto il traffico attraverso un solo dispositivo, il router.

Carrier Grade NAT (CG-NAT)

TODO: cita qualcosa

I CG-NAT sono una tipologia di NAT che lavorano su larga scala e in generale sono implementati all'interno della rete dell'ISP. Sono usati per mappare un grande numero di utenti in relativi pochi indirizzi pubblici. Dato che sono gestiti dall'ISP, gli utenti non hanno nessun controllo sulla sua configurazione.

1.2 Openvpn

OpenVPN è un applicativo open source che ha l' obbiettivo di fornire una VPN che sia semplice da configurare e che funzioni in ogni contesto. Openvpn può incapsulare sia pacchetti IP che

frame Ethernet, in un tunnel sicuro che può viaggiare sia su TCP che UDP. Ha molte opzioni di configurazione, come la possibilità di usare qualsiasi porta, oppure l'uso della compressione. Il tutto è raccolto in un singolo applicativo che può funzionare sia da client che da server, in base alla configurazione fornita.

Possiamo ad esempio vedere una cattura di Wireshark di un pacchetto OpenVPN su UDP e porta 1194:

Wireshark code: 1.2.1

```
1 > Frame 90: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface
  ↪ br-08876ccdf1f5, id 0
2 > Ethernet II, Src: cc:d9:ac:eb:91:5f (cc:d9:ac:eb:91:5f), Dst: 0c:e4:a0:27:a9:24
  ↪ (0c:e4:a0:27:a9:24)
3 > Internet Protocol Version 4, Src: 192.168.1.20, Dst: 51.178.141.119
4 > User Datagram Protocol, Src Port: 47007, Dst Port: 1194
5 > OpenVPN Protocol
6 Type: 0x48 [opcode/key_id]
7 Peer ID: 0
8 Data (36 bytes)
9 Data: 00000019a366196eb2aca181df226faf8514ab73f524f7ef335d55fc57322d032a2095e4
```

1.2.1 Crittografia e autenticazione

Per la cifratura e autenticazione viene usata la libreria OpenSSL, open source e ampiamente usata dalla maggior parte dei servizi su internet, come ad esempio l'https. Ciò fornisce ad OpenVPN la flessibilità di poter usare tutti i cifrari forniti da questa libreria.

L'autenticazione può essere eseguita usando una pre-shared key, un sistema basato sull'utilizzo dei certificati, una semplice password o una combinazione dei precedenti. Il metodo più sicuro è quello basato sui certificati, che sfrutta una *Public key infrastructure* [13] per autenticare che i certificati forniti dai client siano effettivamente autentici. Con questo metodo si crea un certificato per ogni utente che, se opportunamente firmato, permette a quello specifico utente di autenticarsi al server VPN. Questo metodo ha inoltre il vantaggio che un certificato può essere revocato in ogni momento, facendo così perdere l'accesso all'utente che lo stava usando.

Maggiori informazioni possono essere trovate sulla wiki di OpenVPN.

1.2.2 Networking work-in-progress

OpenVPN viene incapsulato dai più comuni protocolli di trasporto (TCP e UDP), ciò lo rende adatto in caso l'ips blocchi VPN di livello più basso, es. ipsec. Può inoltre funzionare attraverso la maggior parte dei server proxy, firewalls e NAT.

La configurazione del server permette di impostare opzioni che modificano la configurazione di rete del server o dei client, ciò permette ad esempio di aggiungere una rotta alla tabella di routing dei client nel momento in cui si connettono alla VPN.

Per depositare il traffico nella network stack dei client, OpenVPN usa i driver universali TUN/TAP. Può quindi creare un tunnel IP di livello 3 (TUN), o Ethernet livello 2 (TAP).

TODO: da rivedere!

In figura 1.3 possiamo vedere l'incapsulamento del pacchetto OpenVPN descritto in code: 1.2.1. Si vede inoltre come il contenuto del campo data del pacchetto OpenVPN contenga al suo interno un pacchetto IP, ciò non è visibile dall'esterno perchè il tutto è opportunamente cifrato.

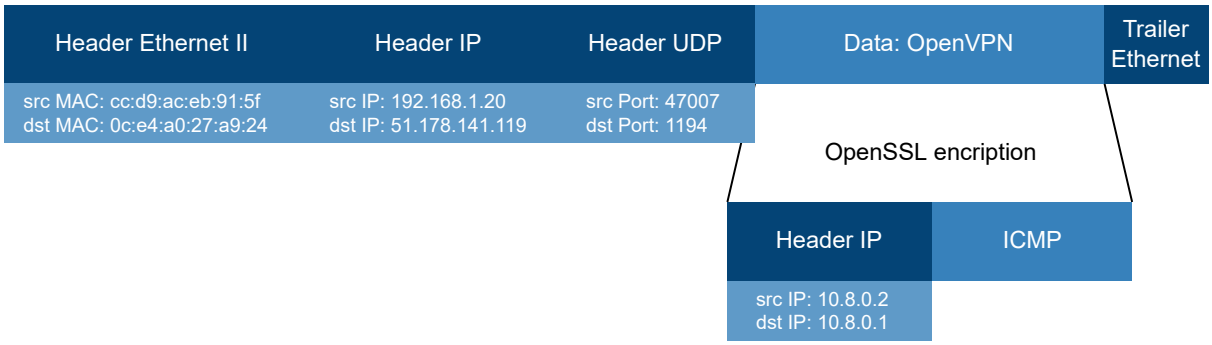


Figura 1.3: Incapsulamento

1.3 OpenWrt OK

Login banner di Openwrt code: 1.3.1

```
BusyBox v1.35.0 (2022-04-24 21:09:51 UTC) built-in shell (ash)
_______                     ________        __
|       |.-----.-----.-----.|  |  |  |.----.|  |_
|   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
|_______||   __|_____|__|__||________||__|  |____|
|__| W I R E L E S S   F R E E D O M
-----------------------------------------------------
OpenWrt SNAPSHOT, r19521-46980294f6
-----------------------------------------------------
```

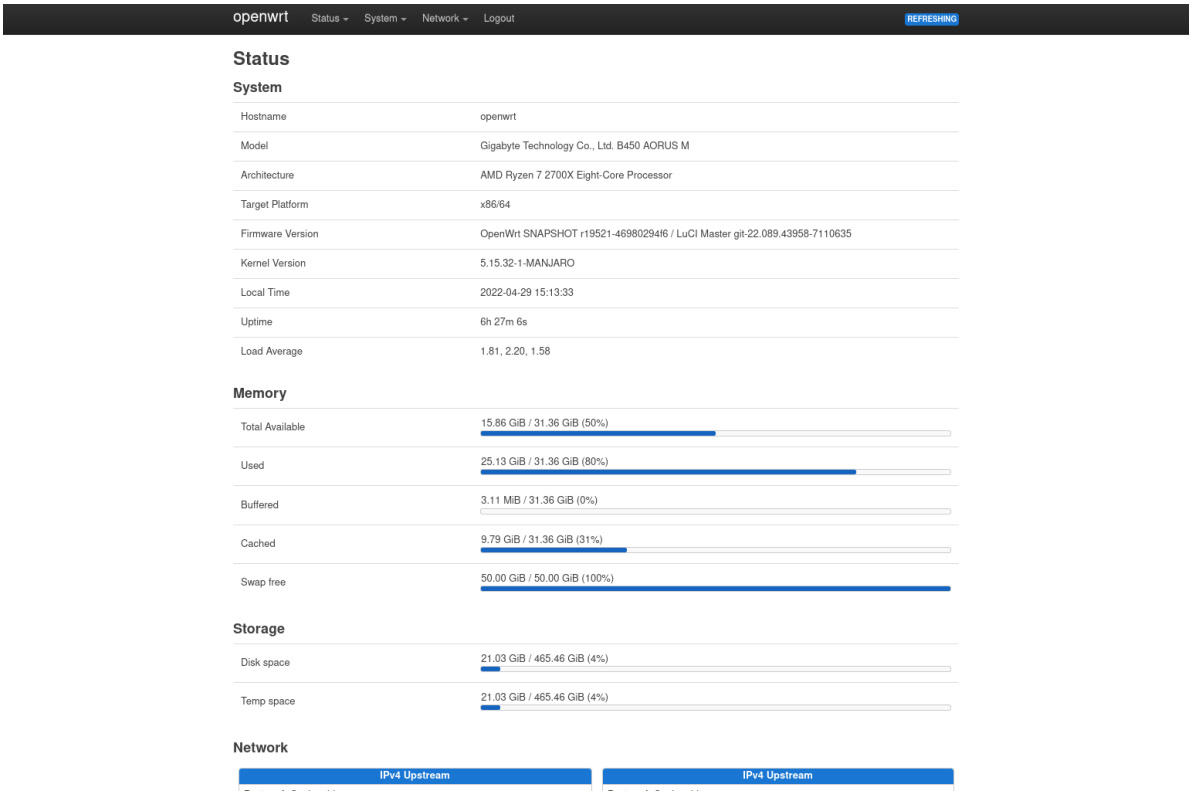
OpenWRT è un sistema operativo open-source per sistemi embedded, principalmente usato come firmware alternativo per router domestici. Si basa su kernel linux, con specifica attenzione all'ottimizzazione per renderlo adatto a dispositivi con risorse estremamente limitate.

Al contrario di altri sistemi operativi per dispositivi embedded, OpenWRT presenta un filesystem con permessi di scrittura. Questo permette di modificare il funzionamento del sistema senza dover reinstallare l'intero firmware a ogni modifica. Per facilitare l'installazione delle funzionalità aggiuntive ha il suo gestore pacchetti (opkg), ciò lo rende estremamente estensibile e configurabile.

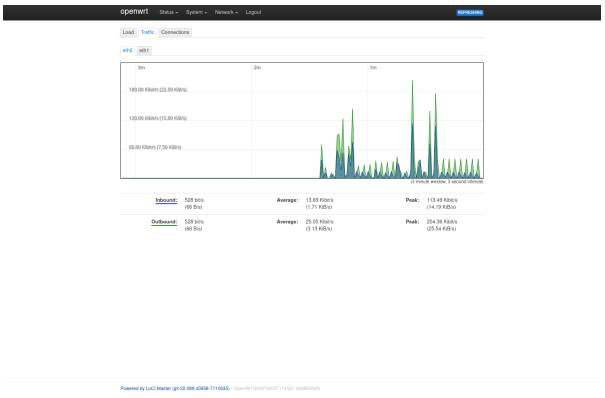
OpenWRT può essere configurato sia tramite shell (ash) che tramite interfaccia web (LuCI).

1.3.1 LuCI web interface

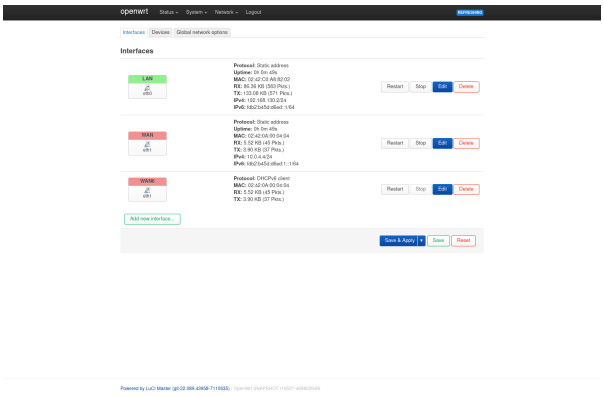
LuCI è l'interfaccia web ufficiale di *OpenWrt*, è un progetto open-source nato dalla necessità di un'interfaccia web per sistemi embedded che sia gratuita, completa ed estensibile. L'installazione e configurazione di LuCI è molto semplice, si può infatti seguire la guida ufficiale.



(a) Status page



(b) Graphs page



(c) Interfaces page

Figura 1.4: Interfaccia web LuCI

L'home page, fig. 1.4a, mostra un riepilogo dello stato del router, ad esempio sono presenti: informazioni sull'hardware, informazioni sulla memoria e storage, sono presenti inoltre informazioni riassuntive sulle interfacce di rete e sul *DHCP*. L'interfaccia LuCI è estensiva e permette di configurare quasi ogni aspetto del funzionamento del router, compreso il firewall, il *DHCP*, i processi in esecuzione, etc. Presenta inoltre la possibilità di installare plugins che modificano e/o aggiungono funzionalità non presenti di default nell'interfaccia.

Capitolo 2

Overview dell'architettura e delle componenti utilizzate

2.1 Obbiettivo da ottenere work-in-progress

TODO: la sezione è da espandere! in qualche modo

In una collaborazione tra il Dipartimento di Ingegneria dell'Informazione e l'azienda **Esse-ti S.R.L.** ci è stato esposto un progetto che consiste nel:

- Fornire a dei clienti un router 4G, su cui possono essere connessi vari dispositivi, ad es. di tipo domotico.
- Rendere questi dispositivi accessibili ai clienti attraverso internet

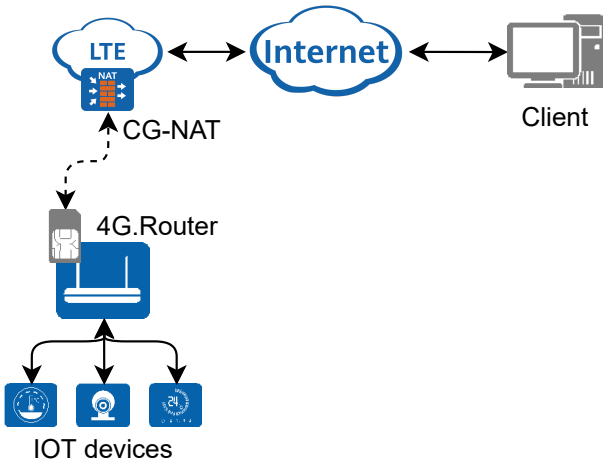


Figura 2.1: Schema concettuale dell'obbiettivo da raggiungere.

Data la presenza del NAT si vede subito che non è realizzabile a meno che il cliente non abbia un'IP pubblico e la sua macchina venga configurata opportunamente. Questo però non è possibile nel caso generale, quindi per risolvere efficacemente questa topologia si deve necessariamente

introdurre una terza macchina provvista di IP pubblico e che funga da ponte tra il *4G.Router* e il cliente.

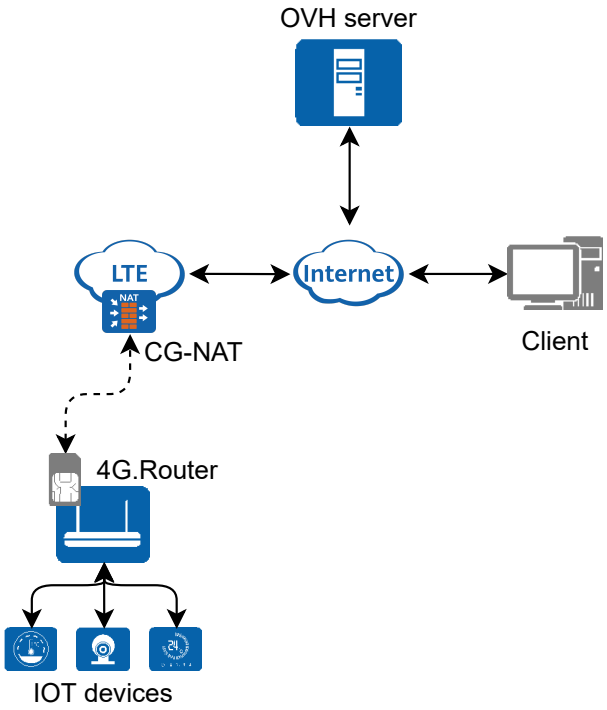


Figura 2.2: Schema concettuale dell'architettura che si dovrà implementare.

In questo modo si può usare il *Server* per configurare una *Virtual Private Network* (VPN), a cui saranno connessi sia il *4G.Router* che il cliente. Ciò consente la creazione di una topologia virtuale in cui tutti i dispositivi connessi alla VPN sono nella stessa rete locale, quindi possono comunicare tra loro.

Inoltre in questo modo viene minimizzata la configurazione da effettuare sulle macchine dei clienti, infatti sarà sufficiente avere un client OpenVPN.

La configurazione virtuale vista dal 4G.Router e dai clienti sarà quindi:

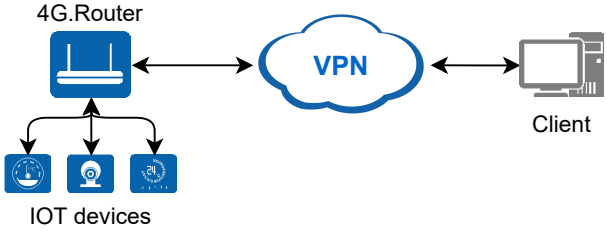


Figura 2.3: Topologia virtuale vista dal cliente.

2.2 Specifiche dei componenti

Per la realizzazione di questa topologia sono necessari i seguenti componenti:

- Esse-ti 4G.Router
- Server
- Host domotico
- Macchina del cliente

Alcuni di questi componenti necessiteranno di delle caratteristiche specifiche.

2.2.1 Esse-ti 4G.Router work-in-progress

Ci è stato fornito dall'azienda Esse-ti, consiste in un gateway 4G con funzionalità di router. Le specifiche complete possono essere trovate sul sito del produttore (link).

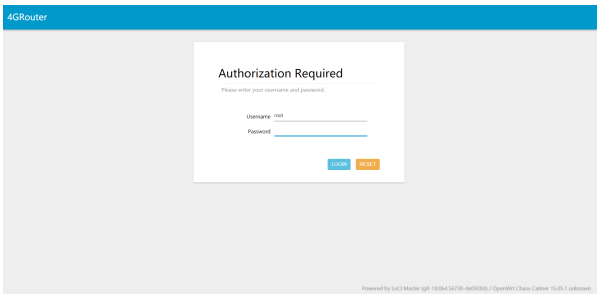
Figura 2.4: Esse-ti 4G.Router

Per l'implementazione di questa architettura sono necessarie solo un sub-set delle specifiche:

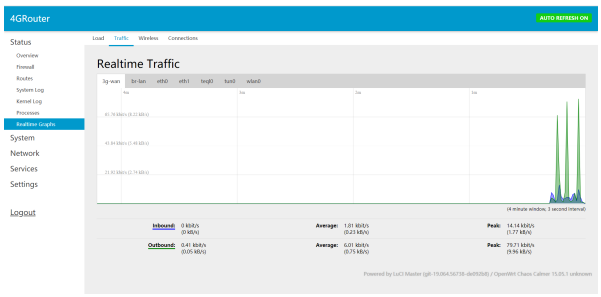
- Access Point wireless per offrire connettività Internet Wi-Fi
- Client Dynamic DNS per consentire all'utente di raggiungere da remoto, tramite Internet, il router stesso e tutti i dispositivi connessi via Wi-Fi o porta LAN
- Gateway telefonico per consentire l'invio e la ricezione di chiamate attraverso la rete 4G LTE/UMTS/GSM a telefoni fissi

Ha il sistema operativo *Draghino*, una versione personalizzata di OpenWRT, in cui è stata modificata l'interfaccia web... mh no

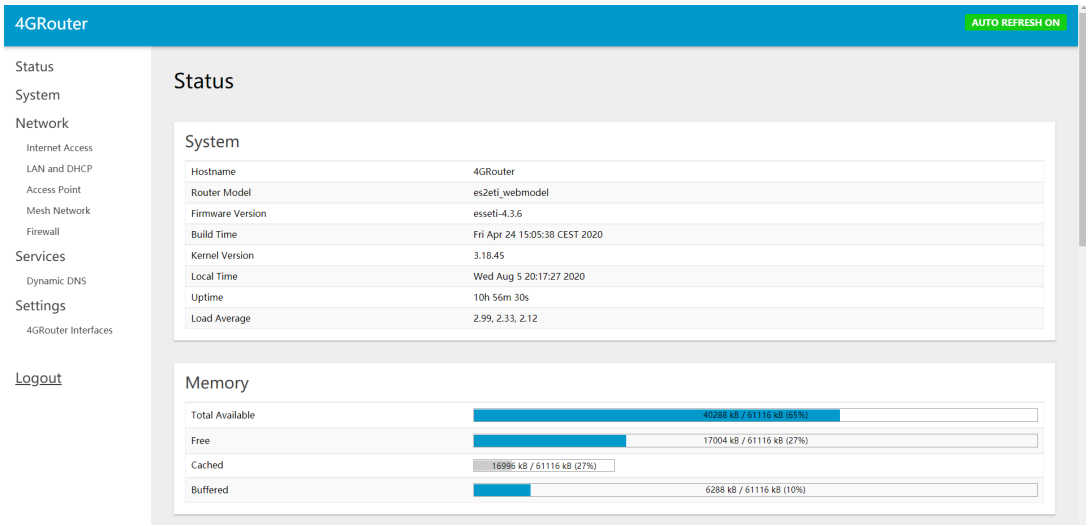
La configurazione del dispositivo può essere fatta sia da terminale, entrando in ssh, sia da interfaccia web:



(a) Schermata di autenticazione



(b) Grafico del traffico



(c) Schermata con stato riassuntivo

Figura 2.5: Interfaccia web Esse-ti 4G.Router

Per semplicità si farà riferimento all'*Esse-ti 4G.Router* chiamandolo semplicemente *Router*.

2.2.2 VPS OVHCloud

La VPS ha il solo vincolo di dover avere un'ip pubblico e una connessione a internet abbastanza veloce. Dovrà infatti sopportare un traffico simmetrico in upload / download.

Per la realizzazione della topologia è stata selezionata una macchina VPS del provider *OVH-Cloud*, con le seguenti caratteristiche:

- 2 core virtuali
- 4Gb di memoria ram
- 80Gb di storage NVMe
- 500Mbps simmetrici di banda

- ipv4 pubblico
- Ubuntu 16.04

Per semplicità si farà riferimento alla *VPS OVHCloud* come *Server*.

2.2.3 Host domotico

Per simulare un'host domotico ed effettuare le varie operazioni di testing è stata usata una *raspberry pi*, con le utility *ping* e *tracepath*.

2.2.4 Macchina del cliente work-in-progress

Per avere la massima flessibilità la macchina del cliente deve essere generica e non deve necessitare di nessuna configurazione specifica. Data la scelta di usare OpenVPN come provider VPN, e dato che OpenVPN è cross-platform, la macchina del cliente non ha specifiche di sistema operativo. L'unica necessità è di avere il client OpenVPN installato sul sistema, ad esempio:

TODO: don't like forse lo tolgo

- con sistema operativo Windows si deve scaricare l'eseguibile dal sito ufficiale
- su linux è sufficiente cercare nei repository ufficiali della distribuzione che si sta usando, es. Ubuntu: `apt-get install openvpn`.

Capitolo 3

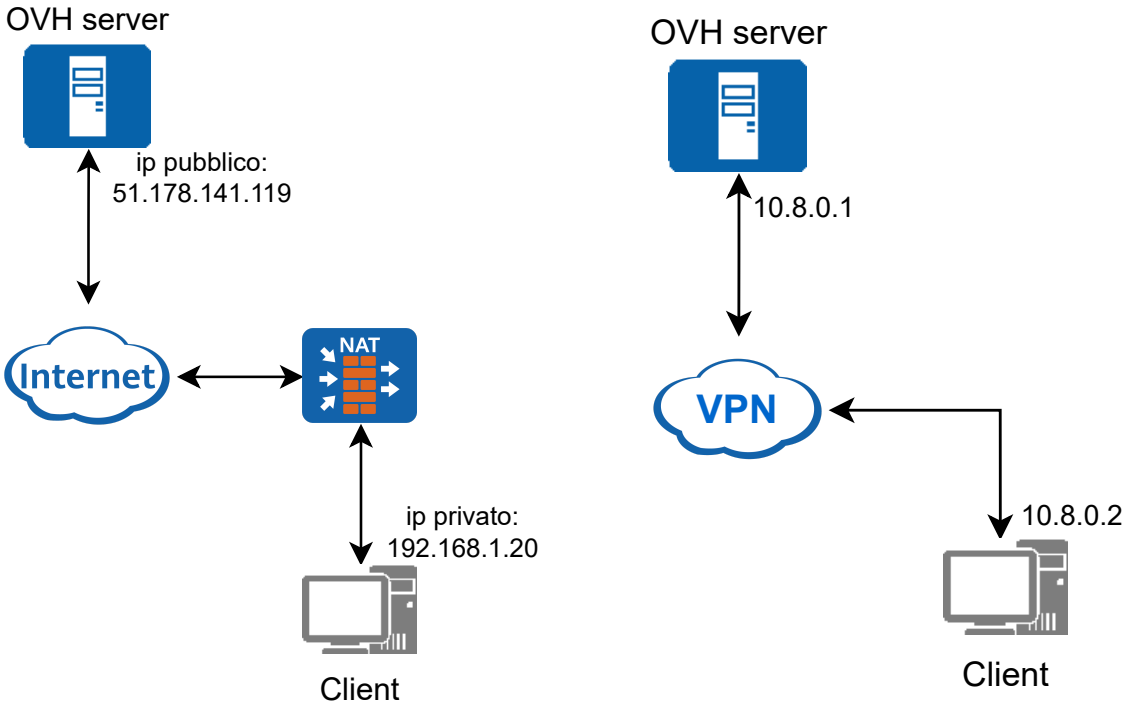
Configurazione del *Server*

3.1 Overview della configurazione e prerequisiti

In questo capitolo andremo a installare e configurare *OpenVPN server* sulla VPS di OVHCloud.

Per facilitare la configurazione e il testing, supponiamo di partire da una topologia che contiene solo il server OpenVPN e un generico client. Come da specifiche il Server deve avere a disposizione un'IP pubblico e il client deve essere il più generico possibile, lo supponiamo quindi sotto a un NAT.

Supponiamo inoltre che l'IP pubblico del Server sia `51.178.141.119`, si avrà quindi una configurazione iniziale come in figura 3.1a.



(a) Configurazione di partenza per questo capitolo.

(b) Configurazione virtuale da raggiungere per questo capitolo.

Figura 3.1: Configurazione di partenza e di obbiettivo per il capitolo 3.

Per instaurare una comunicazione bidirezionale tra il server e il client, si dovrà configurare opportunamente una rete VPN, che risulterà nella configurazione rappresentata in figura 3.1b.

3.2 Creazione della *Public key infrastructure*

Per la gestione dell'autenticazione dei client alla VPN è necessario creare una *Public key infrastructure* (PKI), come descritto nella sezione 1.2.1. Inoltre, per una maggiore sicurezza è indicato separare la *Certificate Authority* dal *Server OpenVPN* [4], supponiamo quindi di usare un secondo server chiamato *Server CA*.

Il *Server CA* verrà usato in fase di configurazione del server e di creazione dei certificati per i client, dopodiché non sarà più necessario.

TODO: va bene senza mettere nessuna spiegazione per lo schema??

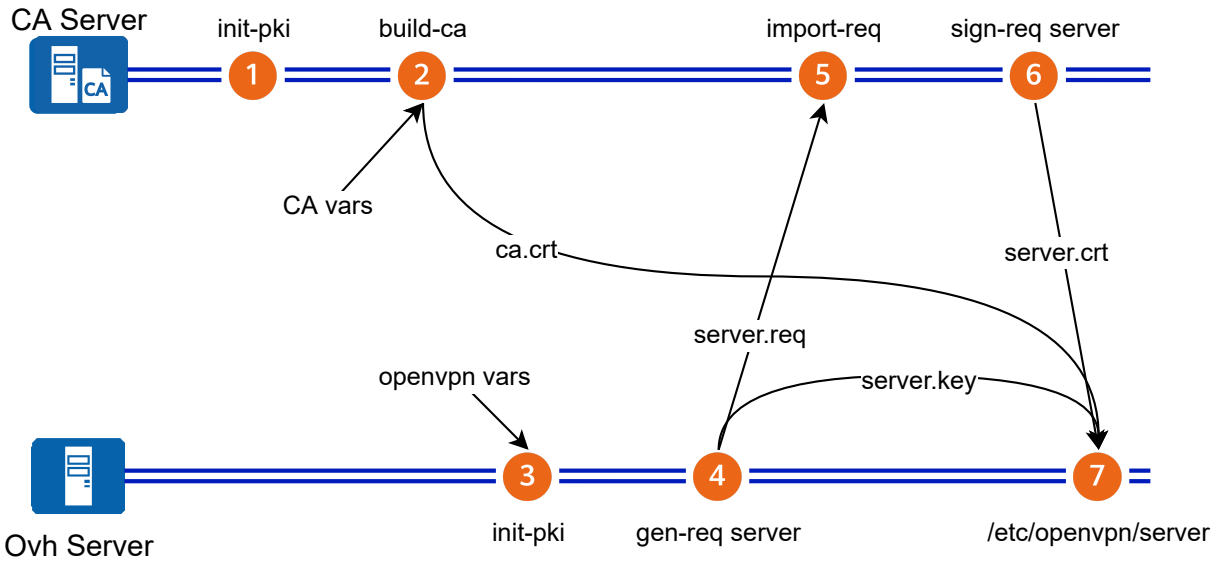


Figura 3.2: Diagramma per schematizzare la procedura di creazione della CA.

3.2.1 Creazione della struttura di cartelle necessaria per ospitare la PKI

Per la gestione della PKI verrà usato il tool *easy-rsa*, che fornisce un wrapper intorno alle funzionalità di *OpenSSL* facilitandone l'utilizzo.

Il pacchetto *easy-rsa* è presente nei repository ufficiali di ubuntu e può essere installato con il comando `sudo apt-get install easy-rsa`.

Dopo l'installazione verrà creata una struttura di cartelle pronta a ospitare la PKI nel percorso `/usr/share/easy-rsa/`, ma è sconsigliato usarlo per motivi di sicurezza. Quindi si deve spostare/collegare questo percorso in una cartella di un'utente non privilegiato, andiamo quindi a crearne un collegamento nella home:

Server CA code: 3.2.1

```
$ mkdir ~/openvpn-ca
$ ln -s /usr/share/easy-rsa/* ~/openvpn-ca/ # soft link di easy-rsa nella home
$ chmod 700 /home/ubuntu/openvpn-ca/        # cambio i permessi in modo che solo l'utente
                                            #  corrente possa leggere il contenuto
```

A questo punto possiamo usare lo script `easyrsa` presente nella cartella per creare la PKI.

Server CA code: 3.2.2

```
$ tree
.
|-- easyrsa -> /usr/share/easy-rsa/easyrsa
|-- openssl-easyrsa.cnf -> /usr/share/easy-rsa/openssl-easyrsa.cnf
|-- vars.example -> /usr/share/easy-rsa/vars.example
`-- x509-types -> /usr/share/easy-rsa/x509-types
```

3.2.2 Creazione della *Certificate Authority* CA work-in-progress

Il primo step è la creazione della *Certificate Authority*, passaggio 1 fig.3.2.

Andiamo quindi a usare l'albero di cartelle creato in code: 3.2.1:

Server CA code: 3.2.3

```
$ cd openvpn-ca/
$ ./easyrsa init-pki
init-pki complete; you may now create a CA or requests.
Your newly created PKI dir is: /home/ubuntu/openvpn-ca/pki
```

Ora si devono personalizzare le variabili `vars`, si può sia partire da un file vuoto oppure modificare `vars.example` per poi rinominarlo `vars`.

Andiamo quindi a creare un nuovo file `vars`:

Server CA code: 3.2.4

```
$ vim vars
set_var EASYRSA_REQ_COUNTRY  "IT"
set_var EASYRSA_REQ_PROVINCE "MC"
set_var EASYRSA_REQ_CITY     "Recanati"
set_var EASYRSA_REQ_ORG      "Esse-ti"
set_var EASYRSA_REQ_EMAIL    "s.gasparrini@esse-ti.it"
set_var EASYRSA_REQ_OU       "Esse-ti"
set_var EASYRSA_REQ_CN       "openvpn-ca"

set_var EASYRSA_ALGO         "ec"
set_var EASYRSA_DIGEST       "sha512"
```

Le variabili nel primo blocco determinano i dati che poi verranno registrati nei certificati. Le

ultime 2 sono opzioni di sicurezza. In particolare si setta il tipo di algoritmo di cifratura per usare la crittografia a chiave ellittica (Elliptic-Curve Cryptography); L'ultima opzione setta l'algoritmo di hashing da usare nella firma dei certificati.

A questo punti si deve lanciare il comando `build-ca` per costruire la CA (passaggio 2 fig.3.2):

Server CA code: 3.2.5

```
$ ./easyrsa build-ca

Note: using Easy-RSA configuration from: ./vars

Using SSL: openssl OpenSSL 1.1.1f  31 Mar 2020

Enter New CA Key Passphrase:
Re-Enter New CA Key Passphrase:
read EC key
writing EC key

You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Common Name (eg: your user, host, or server name) [Easy-RSA CA]:

CA creation complete and you may now import and sign cert requests.
Your new CA certificate file for publishing is at:
/home/ubuntu/openvpn-ca/pki/ca.crt
```

Eseguendo il comando verrà chiesto di inserire una passphrase, che verrà usata per criptare la chiave privata appena generata. Se non si vuole criptare la chiave privata, basta lasciare il campo vuoto. Il secondo prompt è relativo al *Common Name* da dare alla certificazione, in questo caso è stato lasciato il valore di default `Easy-RSA CA`.

3.2.3 Configurazione della PKI di OpenVPN

Il procedimento è simile al precedente, ma questa volta va eseguito sul *server*.

Quindi per prima cosa si devono ripetere i passaggi fatti in sezione 3.2.1, questa volta però chiamiamo la cartella `openvpn-pki` in modo da distinguerle.

Andiamo a creare un file `vars`, che questa volta contiene solo le informazioni essenziali, e poi dare il comando `init-pki` (step 3 fig.3.2):

Server code: 3.2.6

```
$ vim vars
set_var EASYRSA_ALGO    "ec"
set_var EASYRSA_DIGEST  "sha512"
$ ./easyrsa init-pki
Note: using Easy-RSA configuration from: ./vars

init-pki complete; you may now create a CA or requests.
Your newly created PKI dir is: /home/ubuntu/openvpn-pki/pki
```

A questo punto il server OpenVPN ha tutti i prerequisiti per creare una sua chiave privata e relativa *Certificate Signing Request* (step 4 fig.3.2).

Come *Common Name* è stato scelto `server`:

Server code: 3.2.7

```
$ ./easyrsa gen-req server nopass

Note: using Easy-RSA configuration from: ./vars

Using SSL: openssl OpenSSL 1.1.1f  31 Mar 2020
Generating an EC private key
writing new private key to '/home/ubuntu/openvpn-pki/pki/private/server.key.438W2xM0g9'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Common Name (eg: your user, host, or server name) [server]:

Keypair and certificate request completed. Your files are:
req: /home/ubuntu/openvpn-pki/pki/reqs/server.req
key: /home/ubuntu/openvpn-pki/pki/private/server.key
```

La chiave `server.key` va copiata nella cartella del server OpenVPN:

Server code: 3.2.8

```
$ sudo cp /home/ubuntu/openvpn-pki/pki/private/server.key /etc/openvpn/server/
```

Il secondo file creato, `server.req`, corrisponde a una *Certificate Signing Request (CSR)* che va firmata e validata dalla CA.

3.2.4 Firma del certificato OpenVPN dalla CA

Per firmare la *Certificate Signing Request* si deve copiare il file `.req` nel *server CA*, supponiamo quindi di averlo copiato nella cartella `/tmp`.

Ora lo si deve importare nella CA e firmarlo, ciò corrisponde ai passaggi 5 e 6 fig.3.2:

Server CA code: 3.2.9

```
$ cd ~/openvpn-ca
$ ./easyrsa import-req /tmp/server.req server
$ ./easyrsa sign-req server server
Using configuration from /home/ubuntu/openvpn-ca/pki/safessl-easyrsa.cnf
Check that the request matches the signature
Signature ok
The Subject\'s Distinguished Name is as follows
commonName            :ASN.1 12:'server'
Certificate is to be certified until Mar 11 15:50:45 2025 GMT (1080 days)

Write out database with 1 new entries
Data Base Updated
```

Verrà creato un file in `/openvpn-ca/pki/issued` chiamato `server.crt`, che contiene il certificato firmato dalla CA.

Per concludere la procedura (step 7 fig.3.2), si devono copiare i file `ca.crt` e `server.crt` dal *server CA* al *server OpnenVPN*, spostandoli nella cartella del server OpenVPN (`/etc/openvpn/server`)

3.3 Generazione della *tls-crypt pre-shared key*

Per aumentare ulteriormente la sicurezza del nostro *server OpenVPN* possiamo creare un'ulteriore chiave, che permette di offuscare il certificato in fase di validazione. In questo caso la chiave è di tipo *preshared* ed è comune per utti gli utenti, serve principalmente per aggiungere un ulteriore livello di sicurezza.

La creazione va fatta sul *server OpenVPN* e il file risultante va copiato nella cartella del server OpenVPN:

Server code: 3.3.1

```
$ cd ~/openvpn-pki/
$ openvpn --genkey --secret ta.key
$ sudo cp ta.key /etc/openvpn/server
```

3.4 Generazione dei certificati per i *clients*

La generazione dei certificati per i client consiste in una procedura molto simile alla creazione del certificato del *Server*, sezione 3.2.3 e 3.2.4.

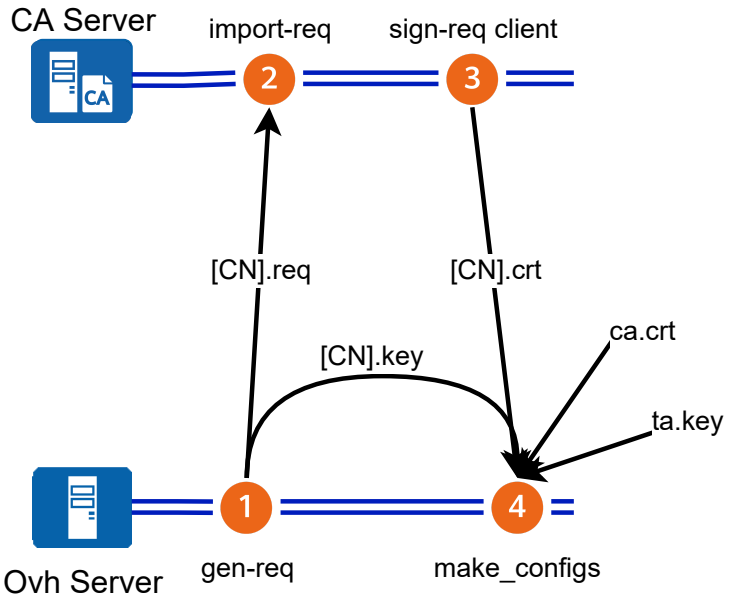


Figura 3.3: Diagramma per schematizzare la procedura di firma di un certificato client.

Per ospitare i certificati dei client e le loro chiavi, creiamo una cartella nella home. Successivamente verrà usata per automatizzare la creazione dei file di configurazione OpenVPN che verranno usati dai client per connettersi alla VPN.

Server code: 3.4.1

```
$ mkdir -p ~/client-configs/keys
$ chmod -R 700 ~/client-configs
```

Creiamo quindi un certificato per un *Client*, è importante che il *Common Name* dato al certificato sia unico ed è consigliabile usare un nome che permetta di collegare il certificato con il cliente che ha richiesto il servizio. Ciò permette di conoscere quale certificato revocare in caso sia necessario.

In questo caso è stato usato il *Common Name* `client1` (step 1 fig.3.3):

Server code: 3.4.2

```
$ cd ~/openvpn-pki/
$ ./easyrsa gen-req client1 nopass
```

Con questo comando verranno creati 2 file, `pki/private/client1.key` costituisce la chiave privata del certificato e va copiata nella directory creata in code: 3.4.1; `pki/reqs/client1.req` è la *Certificate Signing Request* che deve essere copiata nel *server CA* per essere firmata.

Server code: 3.4.3

```
$ cp pki/private/client1.key ~/client-configs/keys/
```

Supponiamo di aver copiato il file `client1.req` nella cartella `tmp` del *Server CA*, possiamo quindi importarla e firmarla (passaggi 2 e 3 fig.3.3):

Server CA code: 3.4.4

```
$ cd ~/openvpn-ca
$ ./easyrsa import-req /tmp/client1.req client1
$ ./easyrsa sign-req client client1
Using configuration from /home/ubuntu/openvpn-ca/pki/safessl-easyrsa.cnf
Check that the request matches the signature
Signature ok
The Subject\'s Distinguished Name is as follows
commonName            :ASN.1 12:'client1'
Certificate is to be certified until Mar 16 13:15:09 2025 GMT (1080 days)

Write out database with 1 new entries
Data Base Updated
```

Ciò produrrà il file di certificato `openvpn-ca/pki/issued/client1.crt` che deve essere copiato nel *Server OpenVpn*, supponiamo di copiarlo nella cartella `/client-configs/keys/`.

3.4.1 Script per la creazione delle configurazioni dei client

Per facilitare la creazione dei file di configurazione dei client, `Common-Name.ovpn`, andremo a creare un apposito script bash che userà la cartella creata in code:3.4.1.

Per concludere la preparazione si devono copiare i file `ta.key` (creato nella sezione 3.3) e `ca.crt` (creato nella sezione 3.2.4), sistemando inoltre l'*owner* in modo che l'utente non privilegiato vi possa accedere:

Server code: 3.4.5

```
$ cp ~/openvpn-pki/ta.key ~/client-configs/keys/
$ sudo cp /etc/openvpn/server/ca.crt ~/client-configs/keys/
$ sudo chown ubuntu:ubuntu ~/client-configs/keys/*
```

Ora andiamo a scaricare e personalizzare la configurazione di esempio per i client:

Server code: 3.4.6

```
$ cd ~/client-configs/
$ wget "https://raw.githubusercontent.com/OpenVPN/openvpn\
            /master/sample/sample-config-files/client.conf" \
                -O base.conf
$ vim base.conf
42   remote 51.178.141.119 1194     # va messo l'IP e la porta del server OpenVPN
88   ;ca ca.crt                     # non useremo i file esterni ma ingloberemo
89   ;cert client.crt               #   questi file direttamente nella
90   ;key client.key                #   configurazione del client
108  ;tls-auth ta.key 1             #
116  cipher AES-256-GCM             # cifratura usata
117  auth SHA256                    # autenticazione usata
118  key-direction 1                # indica che è un client
```

Ora creiamo lo script bash `make_config.sh`:

Server code: 3.4.7

```
$ vim make_config.sh
#!/bin/bash

# usage:
# $ make_config.sh client1
# will use [ca.crt, client1.crt, client1.key, ta.key] to create client1.ovpn

KEY_DIR=~/client-configs/keys
OUTPUT_DIR=~/client-configs/files
BASE_CONFIG=~/client-configs/base.conf

cat ${BASE_CONFIG} \
    <(echo -e '<ca>') \
    ${KEY_DIR}/ca.crt \
    <(echo -e '</ca>\n<cert>') \
    ${KEY_DIR}/${1}.crt \
    <(echo -e '</cert>\n<key>') \
    ${KEY_DIR}/${1}.key \
    <(echo -e '</key>\n<tls-crypt>') \
    ${KEY_DIR}/ta.key \
    <(echo -e '</tls-crypt>') \
    > ${OUTPUT_DIR}/${1}.ovpn
$ chmod 700 make_config.sh
```

Questo script permette di aggiungere in modo *inline* i file necessari all'autenticazione di un client alla VPN, ha il vantaggio di poter inviare al cliente un solo file invece di 5.

Quindi, in base al *Common Name* passato come argomento, lo script userà: il certificato della CA, `ca.crt`; il certificato e chiave relativi al *Client* per cui si sta creando la configurazione, usando il *Common Name* passato come argomento; e la *preshared key*, `ta.key`.

Il tutto viene scritto in un file che ha lo stesso nome del *Common Name* del *Client* per cui si

sta creando la configurazione ma con estensione `.ovpn`.

Quindi per creare la configurazione di *client1* (passaggio 4 fig.3.3):

Server code: 3.4.8

```
$ ./make_config.sh client1
```

Ciò creerà il file di configurazione per *client1* scrivendolo in `client-configs/files/client1.ovpn`, successivamente sarà necessario copiare questo file nella macchina del *client*.

3.5 Creazione del file di configurazione del Server OpenVPN

Il server openvpn viene configurato attraverso il file di configurazione `/etc/openvpn/server/server.conf`, per non partire da una configurazione vuota si può usare la configurazione di esempio offerta da openvpn:

Server code: 3.5.1

```
$ cd /etc/openvpn/server/
$ sudo wget "https://raw.githubusercontent.com/OpenVPN/openvpn/\
                master/sample/sample-config-files/server.conf"
```

A questo punto andiamo a modificare il file di esempio personalizzandolo per le nostre necessità, per facilitare la riproduzione di questa configurazione è stato aggiunto il numero della riga che è stata modificata:

Server code: 3.5.2

```
$ sudo vim server.conf
85  dh none             # non sono stati usati i parametri Diffie-Hellman
92  topology subnet     # topologia raccomandata
244 ;tls-auth ta.key 0  # questa riga va commentata
245 tls-crypt ta.key    # selezione della preshared key
253 cipher AES-256-GCM  # selezione della cifratura scelta
275 user nobody         # utente che eseguirà il server openvpn, in modo da
                        #  restringere i permessi
276 group nogroup       # stassa cosa per il gruppo
318 auth sha256         # selezione del metodo di autenticazione
```

3.6 Configurazioni sulla network stack del Server

OpenVPN per funzionare necessita che il server abbia l'*ip forwarding* nel firewall abilitato. Ciò è necessario per permettere al server OpnenVPN di inoltrare il traffico proveniente dalla VPN nella sua network stack, e viceversa.

Per abilitare l'*ip forwarding* si dovrà modificare il file `/etc/sysctl.conf`, il comando successivo serve a ricaricare le configurazioni dai file:

Server code: 3.6.1

```
$ sudo vim /etc/sysctl.conf
69 net.ipv4.ip_forward = 1
$ sudo sysctl -p
net.ipv4.ip_forward = 1
```

3.7 Configurazione del firewall

Sulla VPS scelta è presente il firewall `firewalld`, ma per una più semplice configurazione è consigliato di disattivarlo e installare `ufw`:

Server code: 3.7.1

```
$ sudo systemctl mask firewalld
$ sudo systemctl stop firewalld
$ sudo apt-get install ufw
$ sudo ufw allow ssh
Rule added
Rule added (v6)
$ sudo ufw enable
```

È importantissimo abilitare l'SSH prima di abilitare il firewall, altrimenti si perderà l'accesso alla VPS.

3.7.1 Configurazione del NAT

Per far si che i pacchetti provenienti dalla *VPN* entrino nella network stack del *server* si deve aggiungere una regola di *NAT* nel firewall. Per farlo si deve conoscere quale è l'interfaccia di rete del *server*, cioè quella che ha come IP il suo IP pubblico:

Server code: 3.7.2

```
$ ip addr
[...]
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen
↪  1000
    link/ether a6:23:5f:48:ba:de brd ff:ff:ff:ff:ff:ff
    inet 51.178.141.119/20 brd 51.178.141.255 scope global dynamic ens3
       valid_lft 1857sec preferred_lft 1857sec
    inet6 fe80::23:bfff:ac24:aace/64 scope link
       valid_lft forever preferred_lft forever
[...]
```

In questo caso il nome dell'interfaccia di rete è *ens3*. Possiamo quindi procedere con la configurazione del NAT:

Server code: 3.7.3

```
$ sudo vim /etc/ufw/before.rules
# ## rules.before
# ## Rules that should be run before the ufw command line added rules. Custom
# rules should be added to one of these chains:
# ufw-before-input
# ufw-before-output
# ufw-before-forward
#

# START OPENVPN RULES
# NAT table rules
*nat
:POSTROUTING ACCEPT [0:0]
# Allow traffic from OpenVPN client to ens3
-A POSTROUTING -s 10.8.0.0/24 -o ens3 -j MASQUERADE
COMMIT
# END OPENVPN RULES


# Don't delete these required lines, otherwise there will be errors
*filter
. . .
```

Nella modifica del file si deve stare attenti a inserire la nuova regola in cima al file e sotto i commenti iniziali, è inoltre importante inserire i commenti nella regola.

3.7.2 Configurazione del packet forwarding

Precedentemente abbiamo abilitato il forwarding nella network stack del server, ora si deve abilitare la corrispondente opzione nel firewall. Si deve quindi cambiare la regola di default per i pacchetti inoltrati da `DROP` ad `ACCEPT`.

Per farlo si deve modificare il file `/etc/default/ufw`:

Server code: 3.7.4

```
$ sudo vim /etc/default/ufw
DEFAULT_FORWARD_POLICY="ACCEPT"
```

3.7.3 Conclusione della configurazione del firewall

Per concludere la configurazione si deve abilitare la porta relativa alla vpn, in questo caso `1194`, e riavviare il firewall:

Server code: 3.7.5

```
$ sudo ufw allow 1194/udp
$ sudo ufw reload
$ sudo ufw status
Status: active
To                         Action      From
--                         ------      ----
22                         ALLOW       Anywhere
1194/udp                   ALLOW       Anywhere
22 (v6)                    ALLOW       Anywhere (v6)
1194/udp (v6)              ALLOW       Anywhere (v6)
```

Se la configurazione è stata effettuata correttamente di avrà un output simile al precedente, in caso contrario si dovranno valutare gli errori dati da ufw.

3.8 Avvio del Server OpenVPN

Ora che la configurazione del server è in una situazione stabile possiamo tentate di avviarlo:

Server code: 3.8.1

```
$ sudo systemctl enable openvpn-server@server.service
$ sudo systemctl start openvpn-server@server.service
$ sudo systemctl status openvpn-server@server.service
● openvpn-server@server.service - OpenVPN service for server
     Loaded: loaded (/usr/lib/systemd/system/openvpn-server@.service; enabled; vendor
     ↪  preset: disabled)
     Active: active (running) since Mon 2022-04-18 13:08:44 CEST; 4h 22min ago
       Docs: man:openvpn(8)
             https://community.openvpn.net/openvpn/wiki/Openvpn24ManPage
             https://community.openvpn.net/openvpn/wiki/HOWTO
   Main PID: 436 (openvpn)
     Status: "Initialization Sequence Completed"
      Tasks: 1 (limit: 9488)
     Memory: 4.8M
        CPU: 199ms
     CGroup: /system.slice/system-openvpn\x2dserver.slice/openvpn-server@server.service
             └─436 /usr/bin/openvpn --status /run/openvpn-server/status-server.log
             ↪  --status-version 2 --suppress-timestamps --config server.conf

Apr 18 13:08:44 server openvpn[436]: TUN/TAP device tun0 opened
Apr 18 13:08:44 server openvpn[436]: Incoming Control Channel Encryption: Cipher
↪  'AES-256-CTR' initialized with 256 bit key
Apr 18 13:08:44 server openvpn[436]: Incoming Control Channel Encryption: Using 256 bit
↪  message hash 'SHA256' for HMAC authentication
Apr 18 13:08:44 server openvpn[436]: net_addr_v4_add: 10.8.0.1/24 dev tun0
Apr 18 13:08:44 server openvpn[436]: UDPv4 link local (bound): [AF_INET][undef]:1194
Apr 18 13:08:44 server openvpn[436]: UDPv4 link remote: [AF_UNSPEC]
Apr 18 13:08:44 server openvpn[436]: MULTI: multi_init called, r=256 v=256
Apr 18 13:08:44 server openvpn[436]: IFCONFIG POOL IPv4: base=10.8.0.2 size=253
Apr 18 13:08:44 server openvpn[436]: IFCONFIG POOL LIST
Apr 18 13:08:44 server openvpn[436]: Initialization Sequence Completed
```

Il comando `systemctl enable` abilita il servizio per essere avviato all'avvio della macchina, mentre `systemctl start` lo avvia immediatamente.

Con il comando `systemctl status` si può verificare lo stato del servizio, si vede che il servizio è *active (running)*.

In caso il servizio fallisse ad avviarsi si avrà un breve log con `systemctl status`, oppure si possono leggere i log completi usando `journalctl -u openvpn-server@server.service`.

3.9 Test della configurazione

Ora che la configurazione per questo capitolo è conclusa possiamo testare che tutto funzioni correttamente.

Per farlo ci spostiamo su una macchina *Client*, con sistema operativo Linux ad esempio. Usando la configurazione creata in code:3.4.8 possiamo connettere il *Client* alla VPN:

Client code: 3.9.1

```
$ sudo openvpn --config client1.ovpn
[...]
Thu Apr 21 12:53:04 2022 Outgoing Data Channel: Cipher 'AES-256-GCM' initialized with 256
↪ bit key
Thu Apr 21 12:53:04 2022 Incoming Data Channel: Cipher 'AES-256-GCM' initialized with 256
↪ bit key
Thu Apr 21 12:53:04 2022 ROUTE_GATEWAY 192.168.1.20/255.255.255.0 IFACE=eth0
↪ HWADDR=02:42:0a:00:04:03
Thu Apr 21 12:53:04 2022 /sbin/ip route add 10.8.0.1/32 via 10.8.0.2
Thu Apr 21 12:53:04 2022 WARNING: this configuration may cache passwords in memory -- use
↪ the auth-nocache option to prevent this
Thu Apr 21 12:53:04 2022 Initialization Sequence Completed
```

Se la configurazione fino a questo punto è corretta si avrà il messaggio `Initialization Sequence Completed`.

Nel *Client* si avrà una nuova interfaccia di rete chiamata `tun0`, questa è l'interfaccia virtuale creata dalla vpn.

Client code: 3.9.2

```
$ ip addr
2: tun0: <MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group
↪ default qlen 500
    link/none
    inet 10.8.0.2/24 scope global tun0
       valid_lft forever preferred_lft forever
```

Si può vedere come l'IP assegnato al *Client* dalla VPN è `10.8.0.2`.

Per testare che la connessione sia instaurata correttamente si può usare la utility `ping`, ad esempio possiamo fare il ping dal *Client* verso l'IP interno alla VPN del *Server*:

Client code: 3.9.3

```
$ ping -c1 10.8.0.1
PING 10.8.0.1 (10.8.0.1) 56(84) bytes of data.
64 bytes from 10.8.0.1: icmp_seq=1 ttl=64 time=0.250 ms
```

Se nel frattempo si esegue la utility `tpcdump` sul server si potranno vedere i pacchetti *echo request* ed *echo reply*:

Server code: 3.9.4

```
$ sudo tcpdump -i tun0
listening on tun0, link-type RAW (Raw IP), capture size 262144 bytes
13:11:12.018615 IP 10.8.0.2 > 10.8.0.1: ICMP echo request, id 11, seq 1, length 64
13:11:12.018640 IP 10.8.0.1 > 10.8.0.2: ICMP echo reply, id 11, seq 1, length 64
```

Si vede quindi che è possibile instaurare una comunicazione bidirezionale tra *Client*, `10.8.0.2`,

e *Server*, `10.8.0.1`.

Abbiamo quindi realizzato correttamente la topologia di obbiettivo per questo capitolo, descritta in fig.3.1b.

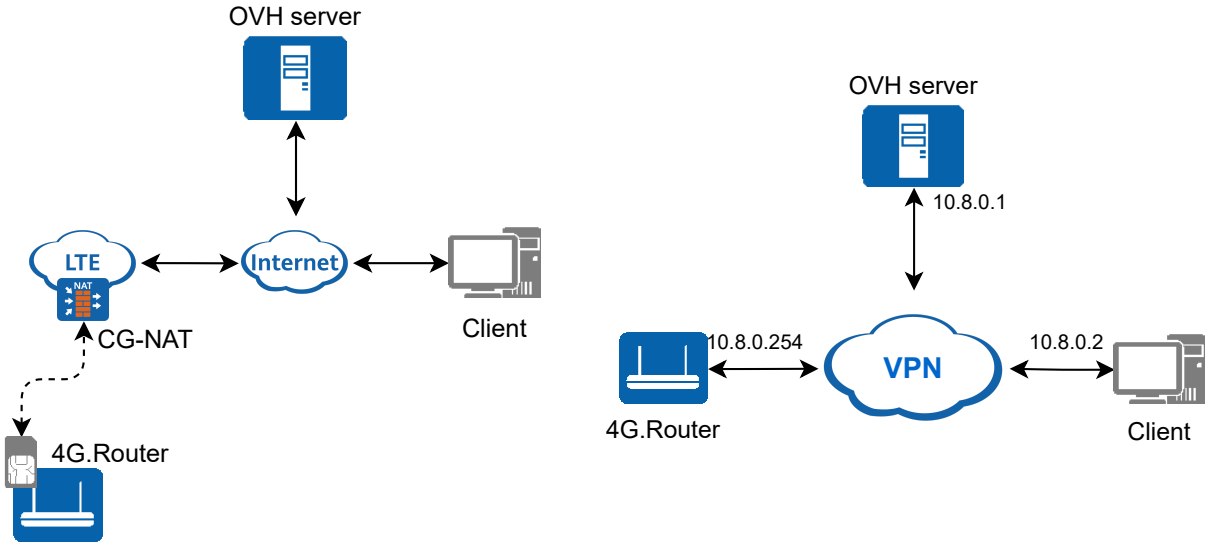
Capitolo 4

Configurazione del *Router*

4.1 Overview della configurazione

In questo capitolo andremo a configurare il *Router 4g* e connetterlo alla VPN.

Continuando a seguire un approccio incrementale, ignoriamo per ora gli *host-domotici* e aggiungiamo alla topologia solo il *Router 4G*. Quindi la configurazione di partenza è descritta in fig.4.1a.



(a) Configurazione iniziale

(b) Configurazione da raggiungere

Figura 4.1: Schemi delle configurazioni iniziali e finali per il capitolo 4

L'obbiettivo di questo capitolo è di configurare il *Router 4G* e il *Server* in modo da poter instaurare un canale di comunicazione tra *Router* e *Client*, raggiungendo una topologia virtuale descritta in fig.4.1b.

Di default OpenVPN non è presente nel *Router 4g*, lo si può installare con:

Router 4g code: 4.1.1

```
$ opkg update
$ opkg install openvpn luci-app-openvpn
```

4.2 Creazione della configurazione OpenVPN per il *Router*

Per creare il file di configurazione OpenVPN per il *Router* si devono seguire gli step descritti in fig. 3.3. Supponiamo di usare come *Common Name* `router`, quindi dopo aver firmato il certificato e ottenuto il file `router.crt` dal *Server CA* possiamo procedere con la creazione della configurazione OpenVPN usando l'apposito script:

Server code: 4.2.1

```
$ ./make_config.sh router
```

Dopodiché si deve copiare il file `router.ovpn` dal *Server* al *Router*, supponiamo di averlo copiato nella cartella `/configs` del *Router*.

4.3 Connessione del *Router* alla VPN

Possiamo quindi spostarci nel *Router* e tentare di connetterlo alla VPN:

Router 4g code: 4.3.1

```
$ openvpn --config /configs/router.ovpn
2022-04-29 17:26:37 OpenVPN 2.5.6 x86_64-openwrt-linux-gnu [SSL (mbed TLS)] [LZ4] [EPOLL]
↪  [MH/PKTINFO] [AEAD]
[...]
2022-04-29 17:26:37 VERIFY EKU OK
2022-04-29 17:26:37 VERIFY OK: depth=0, CN=server
2022-04-29 17:26:37 Control Channel: TLSv1.2, cipher
↪  TLS-ECDHE-RSA-WITH-AES-256-GCM-SHA384, 2048 bit key
2022-04-29 17:26:37 [server] Peer Connection Initiated with [AF_INET]10.0.4.2:1194
2022-04-29 17:26:37 net_addr_ptp_v4_add: 10.8.0.10 peer 10.8.0.9 dev tun0
2022-04-29 17:26:37 Initialization Sequence Completed
```

Se il file di configurazione è stato creato correttamente si vedrà il messaggio `Initialization Sequence Completed`.

Comparirà inoltre l'interfaccia `tun0` a cui è stato assegnato l'indirizzo `10.8.0.3`.

4.4 Auto start del Client VPN nel *Router*

Per abilitare l'auto start di openvpn per il router si deve, per prima cosa, modificare il file `/etc/config/openvpn` in modo che faccia riferimento alla configurazione corretta:

Router 4g code: 4.4.1

```
1 $ vim /etc/config/openvpn
2 20  option config /configs/router.ovpn
```

Ora possiamo abilitarla usando luci:

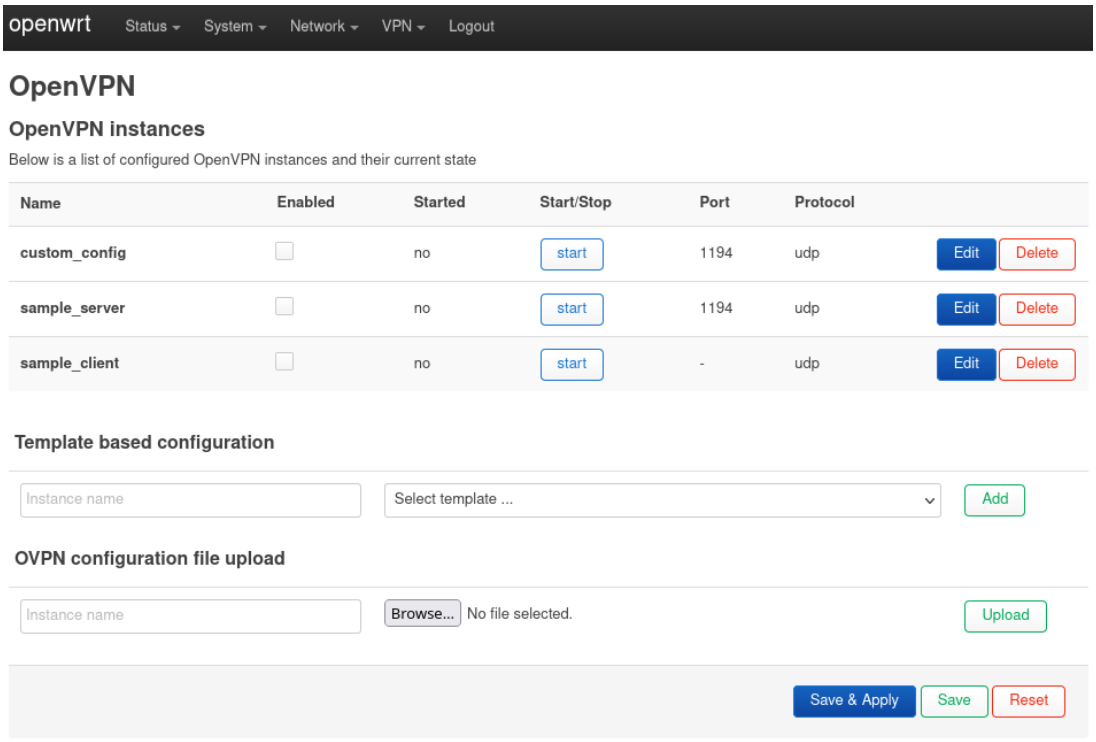


Figura 4.2: Configurazione della VPN tramite LuCI.

Alla riga "custom_config" Si deve mettere il check su *enabled* e premere start, per poi salvare le modifiche. In questo modo il router si connetterà automaticamente alla VPN anche se venisse riavviato.

4.5 Abilitazione del Client-to-Client nel server OpenVPN

In questo momento i client della VPN, `client1` e `router`, possono comunicare tra loro. Ma lo fanno passando per la network stack del *Server*, possiamo verificarlo con le utility `ping` e *tcpdump*:

Router 4g code: 4.5.1

```
$ ping -c2 10.8.0.2                                    # client1
PING 10.8.0.2 (10.8.0.2): 56 data bytes
64 bytes from 10.8.0.2: seq=0 ttl=63 time=0.519 ms
64 bytes from 10.8.0.2: seq=1 ttl=63 time=0.501 ms
```

Dal server possiamo vedere i pacchetti con *tcpdump*:

Server code: 4.5.2

```
$ sudo tcpdump -i tun0
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
16:20:50.791063 IP 10.8.0.3 > 10.8.0.2: ICMP echo request, id 1759, seq 0, length 64
16:20:50.791098 IP 10.8.0.3 > 10.8.0.2: ICMP echo request, id 1759, seq 0, length 64
16:20:50.791273 IP 10.8.0.2 > 10.8.0.3: ICMP echo reply, id 1759, seq 0, length 64
16:20:50.791285 IP 10.8.0.2 > 10.8.0.3: ICMP echo reply, id 1759, seq 0, length 64
16:20:51.791153 IP 10.8.0.3 > 10.8.0.2: ICMP echo request, id 1759, seq 1, length 64
16:20:51.791174 IP 10.8.0.3 > 10.8.0.2: ICMP echo request, id 1759, seq 1, length 64
16:20:51.791365 IP 10.8.0.2 > 10.8.0.3: ICMP echo reply, id 1759, seq 1, length 64
16:20:51.791374 IP 10.8.0.2 > 10.8.0.3: ICMP echo reply, id 1759, seq 1, length 64
```

Si vede che ogni richiesta viene duplicata, la prima è in entrata sulla network stack del *server* e la seconda in uscita. ***TODO:*** forse da mettere lo schema che fa vedere la network stack? Per evitare questo traffico possiamo abilitare l'opzione `client-to-client` nel file di configurazione del server. In questo modo il layer openvpn effettuerà direttamente il forwarding tra i client della vpn [14].

Server code: 4.5.3

```
$ vim /etc/openvpn/server/server.conf
209  client-to-client
$ sudo systemctl restart openvpn-server@server.service
```

Possiamo quindi rieseguire gli stessi test fatti sopra:

Router 4g code: 4.5.4

```
$ ping -c2 10.8.0.2                                    # client1
PING 10.8.0.2 (10.8.0.2): 56 data bytes
64 bytes from 10.8.0.2: seq=0 ttl=64 time=0.351 ms
64 bytes from 10.8.0.2: seq=1 ttl=64 time=0.307 ms
```

Ma questa volta la network stack del *server* non vede nessun pacchetto:

Server code: 4.5.5

```
$ sudo tcpdump -i tun0
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
```

4.6 Assegnazione IP statico al *Router*

Dato che non possiamo sapere a priori quanti client si connetteranno contemporaneamente alla VPN, e l'assegnazione degli IP da parte del server OpenVPN è dinamico, per conoscere sempre quale IP viene assegnato al *Router* è necessario assegnargliene uno statico attraverso un'opportuna configurazione.

Questo viene fatto usando la `cleint-config-dir`, che consiste in una cartella dove l'amministratore può creare dei file di configurazione specifici per determinati utenti. Ciò è utile in caso alcune configurazioni non debbano essere applicate a tutti gli utenti, ma solo a uno. Il nome del file deve essere lo stesso del *Common Name* del certificato del client per cui si sta creando la configurazione.

Nel nostro caso siamo interessati a impostare un'IP statico per il *Router*, quindi si deve creare nella cartella `ccd` un file chiamato `router`.

L'opzione che permette di impostare un IP statico attraverso la `cleint-config-dir` è `ifconfig-push` [10].

Server code: 4.6.1

```
1 $ sudo mkdir /etc/openvpn/server/ccd          # creo la client-config-dir
2 $ sudo vim /etc/openvpn/server/ccd/router
3 ifconfig-push 10.8.0.254 255.255.255.0        # impongo l'ip per questo common name
4 $ sudo vim /etc/openvpn/server/server.conf    # abilito l'opzione nella config del server
5 167  client-config-dir ccd
6 $ sudo systemctl restart openvpn-server@server.service
```

Così facendo al router gli verrà sempre assegnato l'IP `10.8.0.254`, indipendentemente dall'ordine in cui gli host si connettono alla vpn. Ciò ci permette di sapere sempre e a priori qual è l'IP del router.

Capitolo 5

Connessione degli Host domotici alla VPN

***TODO:* è ancora da rileggere**

5.1 Overview della configurazione

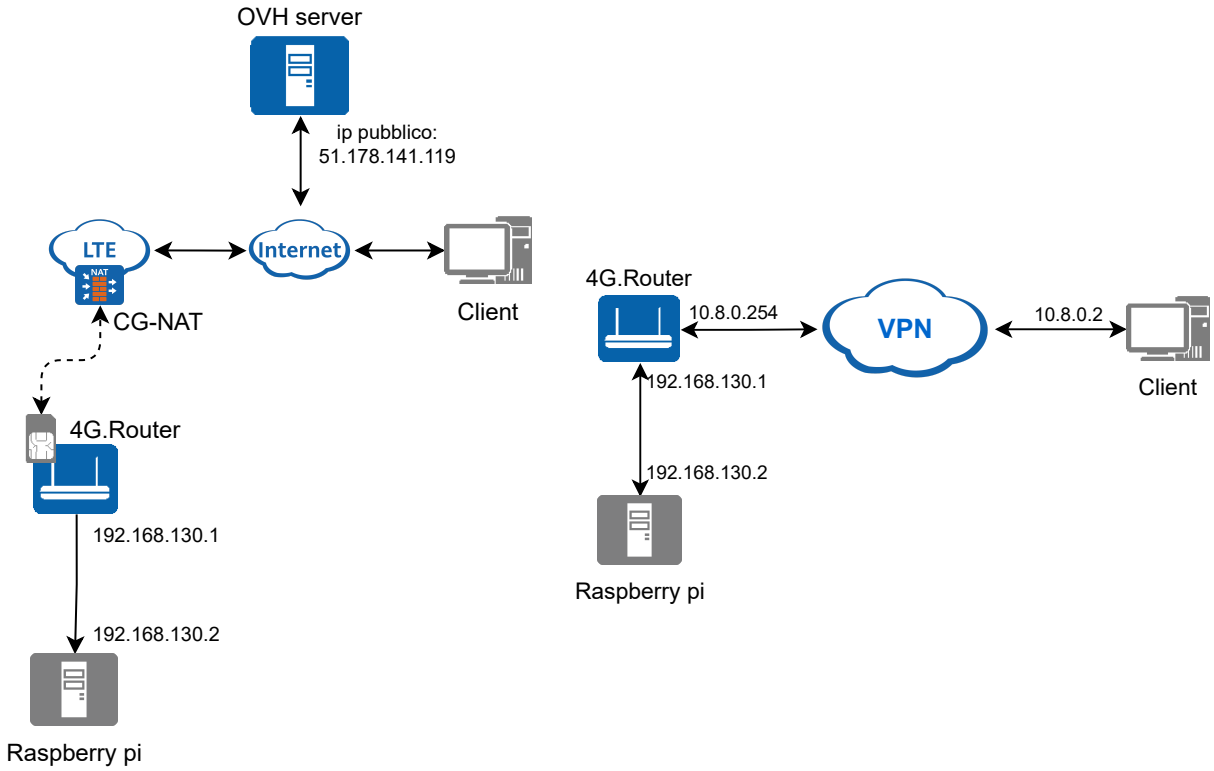
L'ultima parte della configurazione consiste nel rendere disponibile nella VPN la rete locale del *router 4g*, consentendo quindi lo scambio di dati tra gli *host-domotici* e i *Client* della VPN.

Per fare cio' si dovra' configurare il *Router* in modo che effettui il forwarding verso la VPN e il *Server* in modo che annunci la sottorete del *Router* a tutti gli Host della VPN.

Per simulare un host domotico è stata usata una *raspberry pi*, cio' ci consente di poter accedere alla sua shell per effettuare i vari test necessari per confermare che la topologia funzioni correttamente.

In fig. 5.1a possiamo vedere la configurazione iniziale per questo capitolo, cioe' la configurazione finale del capitolo 4 con l'aggiunta dell'*host domotico* connesso al *Router*.

Una volta conclusa la configurazione si arrivera' a una topologia virtuale descritta in fig. 5.1b, che costituisce inoltre la topologia di obbiettivo per questo elaborato (fig. 2.3).



(a) Diagramma della configurazione iniziale (b) Diagramma finale

Figura 5.1: Schemi concettuali della configurazione iniziale e finale per il capitolo 5

5.2 Creazione della firewall zone per la VPN

Per rendere possibile la comunicazione tra la rete *lan* del *Router* e la VPN è necessario configurare opportunamente il firewall.

Ciò viene fatto direttamente da LuCI, nella sezione firewall si avranno preconfigurate 2 zone, lan e wan, come si vede figura 5.2.

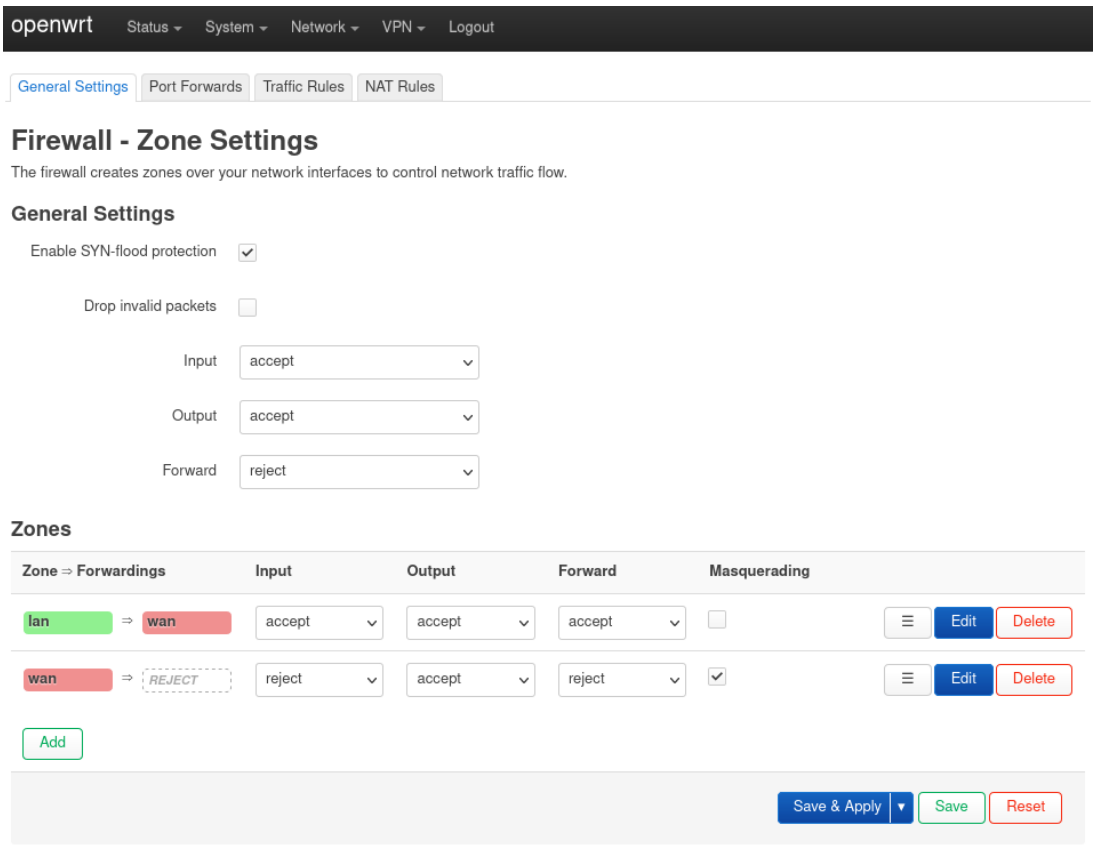


Figura 5.2: Configurazione iniziale del firewall

Si deve quindi aggiungere una nuova zona, chiamata vpn, con le seguenti opzioni:

- Policy di forward: accept
- Forward consentito verso la zona lan
- Forward consentito dalla zona lan

Possiamo vedere la pagina di configurazione:

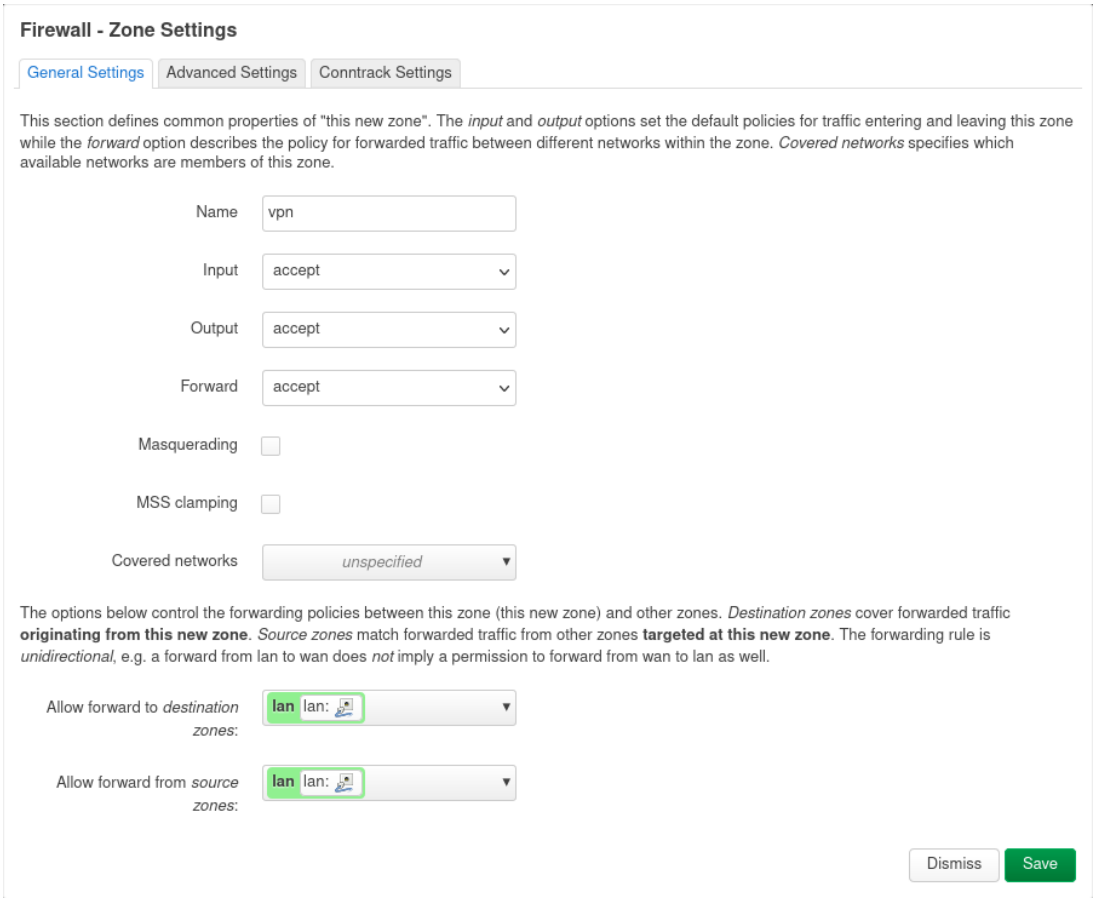


Figura 5.3: Configurazione del firewall

Dopo aver salvato, la pagina del firewall sarà:

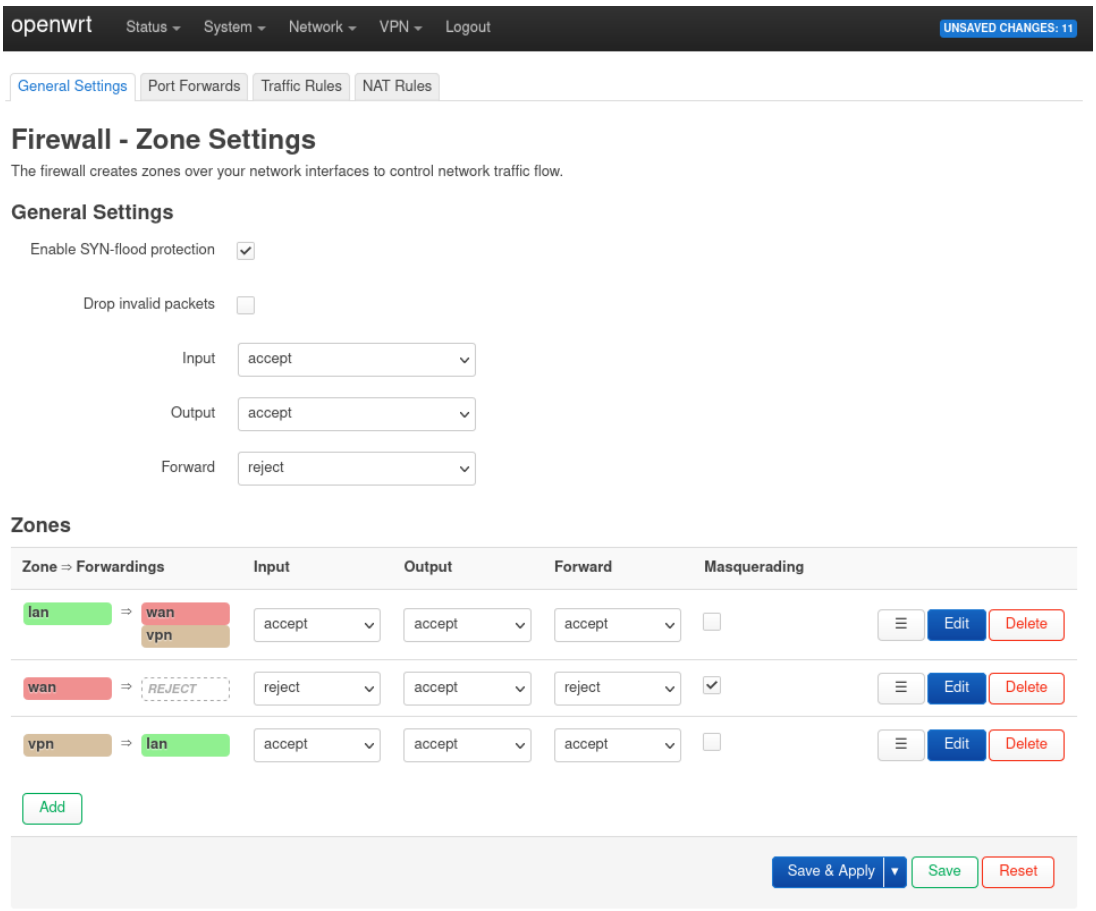


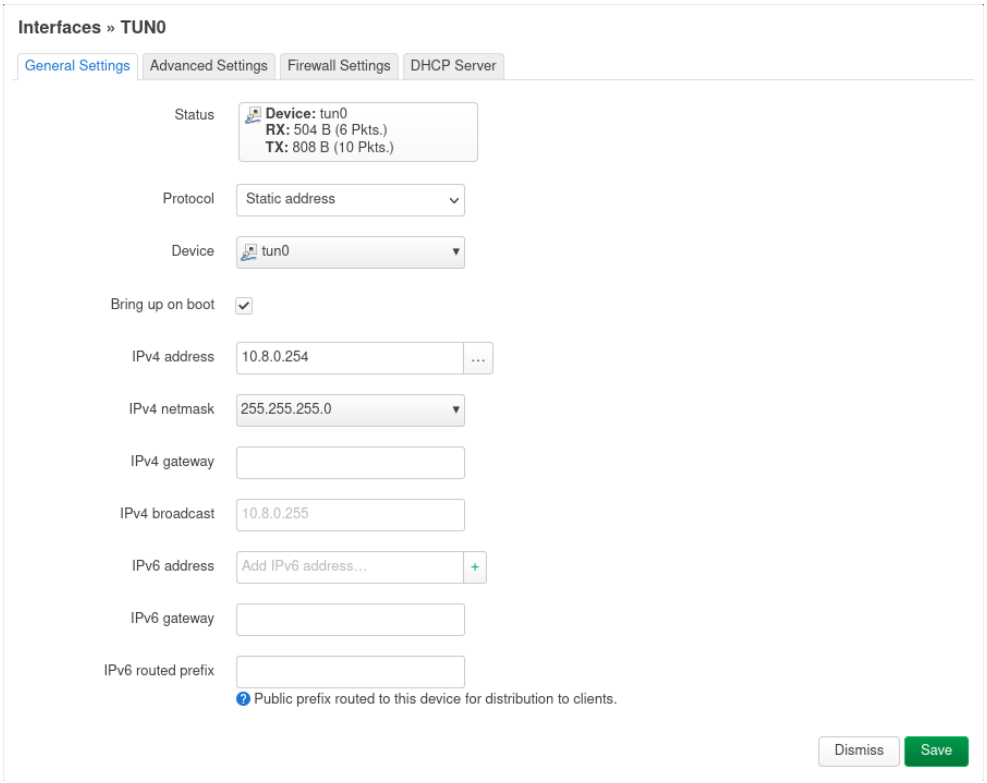
Figura 5.4: Configurazione finale del firewall

5.3 Aggiunta dell'interfaccia tun0 alla zona vpn

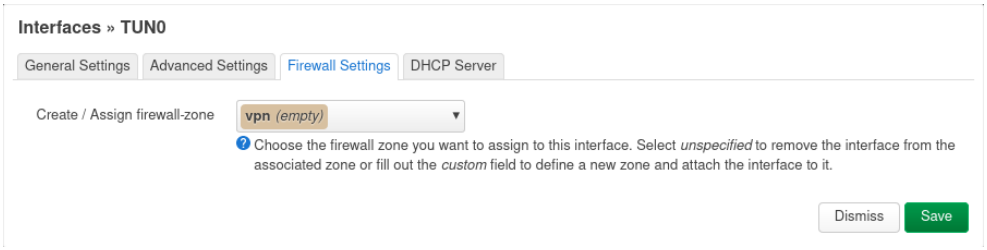
Per rendere attiva la zona firewall appena creata gli si deve assegnare almeno un'interfaccia, in questo caso gli si deve assegnare l'interfaccia tun0.

Ciò va fatto da LuCI, nella sezione *interfaces*. Si deve quindi aggiungere una nuova interfaccia con le seguenti opzioni:

- Name: tun0
- Proto: static
- Device: tun0
- ipv4 address: 10.8.0.254
- ipv4 netmask: 255.255.255.0
- Assign firewall zone: vpn



(a) General settings



(b) Firewall settings

Figura 5.5: Assegnazione interfaccia *tun0* alla zona firewall *vpn* tramite interfaccia LuCI

5.4 Modifiche alla configurazione OpenVPN del *Server*

Per instaurare una comunicazione bidirezionale tra *host-domotico* e *client* è necessario che:

1. il server vpn sia consapevole che il *router* vuole esporre una sua sottorete verso la vpn
2. il client abbia l'opportuna rotta per raggiungere la sottorete dove si trova l'*host-domotico*

Per il punto 1 è necessario aggiungere la seguente riga nel file `/etc/openvpn/server/ccd/router`:

Server code: 5.4.1

```
$ vim /etc/openvpn/server/ccd/router
ifconfig-push 10.8.0.254 255.255.255.0
iroute 192.168.130.0 255.255.255.0
$ vim /etc/openvpn/server/server.conf
169 route 192.168.130.0 255.255.255.0  # aggiungo la rotta nel server
```

Per il punto 2 è necessario aggiungere la seguente riga alla configurazione OpenVPN nel *Server*:

Server code: 5.4.2

```
$ vim /etc/openvpn/server/server.conf
168 push "route 192.168.130.0 255.255.255.0"
```

Dopo aver modificato la configurazione del server va riavviarlo con `systemctl restart`.

In questo modo nella procedura di connessione alla VPN, i client aggiungeranno la rotta verso la sottorete `192.168.130.0/24` nella loro tabella di routing. Possiamo verificarlo con:

Client code: 5.4.3

```
$ ip route
[...]
192.168.130.0/24 via 10.8.0.1 dev tun0
```

5.5 Test della configurazione

TODO: da scrivere di più per i test, sono importanti, forse mettere qualcosa sul troubble shooting... mhhh

Per testare la connessione tra *client* e *host-domotico* usiamo sia `ping` che `tracepath`, possiamo inoltre usare `tcpdump` sul *Server* e *Router* per vedere il percorso dei pacchetti.

Vediamo quindi un ping dal client verso l'*host-domotico*, con server e router in ascolto sull'interfaccia tun0:

Client code: 5.5.1

```
$ ping -c 1 192.168.130.3
PING 192.168.130.3 (192.168.130.3) 56(84) bytes of data.
64 bytes from 192.168.130.3: icmp_seq=1 ttl=63 time=0.643 ms

--- 192.168.130.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.643/0.643/0.643/0.000 ms
```

Server code: 5.5.2

```
$ tcpdump -i tun0
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
```

Il server non ha visto traffico poiché ha l'opzione `client-to-client` abilitata, ciò comporta che il layer vpn effettua il routing senza passare per l'interfaccia di rete del router.

Router code: 5.5.3

```
$ tcpdump -i tun0
listening on tun0, link-type RAW (Raw IP), capture size 262144 bytes
10:44:07.228387 IP 10.8.0.2 > host-domotico: ICMP echo request, id 12, seq 1, length 64
10:44:07.228550 IP host-domotico > 10.8.0.2: ICMP echo reply, id 12, seq 1, length 64
```

Il router vede il pacchetto e lo instrada verso l'host corretto.

Provando nella direzione inversa si ottiene lo stesso risultato

Host-domotico code: 5.5.4

```
$ ping -c 1 10.8.0.2
PING 10.8.0.2 (10.8.0.2) 56(84) bytes of data.
64 bytes from 10.8.0.2: icmp_seq=1 ttl=63 time=0.428 ms

--- 10.8.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.428/0.428/0.428/0.000 ms
```

e il router vede:

Router code: 5.5.5

```
$ tcpdump -i tun0
listening on tun0, link-type RAW (Raw IP), capture size 262144 bytes
11:56:10.469901 IP host-domotico > 10.8.0.2: ICMP echo request, id 14, seq 1, length 64
11:56:10.470230 IP 10.8.0.2 > host-domotico: ICMP echo reply, id 14, seq 1, length 64
```

Possiamo inoltre usare `tracepath` per vedere gli hop:

Client code: 5.5.6

```
$ tracepath 192.168.130.3
1?: [LOCALHOST]                      pmtu 1500
1:  10.8.0.254                       0.471ms
1:  10.8.0.254                       0.473ms
2:  192.168.130.3                    0.567ms reached
    Resume: pmtu 1500 hops 2 back 2
```

host-domotico code: 5.5.7

```
$ tracepath 10.8.0.2
1?: [LOCALHOST]                      pmtu 1500
1:  router-1                         0.062ms
1:  router-1                         0.068ms
2:  10.8.0.2                         0.592ms reached
    Resume: pmtu 1500 hops 2 back 2
```

Capitolo 6

Conclusione workinprogress

L'architettura proposta in questo elaborato e' stata effettivamente implementata su apparati reali durante il tirocinio in collaborazione con l'azienda Esse-ti. Per rendere piu' completo il prodotto che vogliono vendere e' necessario modificare la configurazione per consentire che piu' clienti, ogniuno con il proprio Router 4G, possano usufrire della VPN. Si deve pero' stare attenti a mantenere l'isolamento tra i clienti, per questo motivo e' necessario garantire che un cliente assegnato ad una determinata VPN non possa connettersi a quelle degli altri.

Bibliografia

[1] R. Braden. Requirements for internet hosts - communication layers. STD 3, RFC Editor, October 1989. http://www.rfc-editor.org/rfc/rfc1122.txt.

[2] K. B. Egevang and P. Francis. The ip network address translator (nat). RFC 1631, RFC Editor, May 1994. http://www.rfc-editor.org/rfc/rfc1631.txt.

[3] Huawei. Product image gallery. https://info.support.huawei.com/network/imagelib/getImagePartList?product_family=Router&product_type=Access%20Router%7CIOT%20Gateway&domain=&lang=en.

[4] jesrush (https://serverfault.com/users/480025/jesrush). Openvpn server as it's own ca – security concerns? Server Fault Stack Exchange. URL:https://serverfault.com/q/923049/558773 (version: 2018-05-22).

[5] JGraph. draw.io, 10 2021. https://www.diagrams.net/.

[6] H. R. Max Roser and E. Ortiz-Ospina. Internet. *Our World in Data*, 2015. https://ourworldindata.org/internet.

[7] J. Postel. Internet protocol. STD 5, RFC Editor, September 1981. http://www.rfc-editor.org/rfc/rfc791.txt.

[8] J. Postel. Transmission control protocol. STD 7, RFC Editor, September 1981. http://www.rfc-editor.org/rfc/rfc793.txt.

[9] E. Rescorla. The transport layer security (tls) protocol version 1.3. RFC 8446, RFC Editor, August 2018. http://www.rfc-editor.org/rfc/rfc8446.txt.

[10] L. M. M. (https://serverfault.com/users/176217/lasse-michael-m%c3%b8lgaard). openvpn - ifconfig parameter. Server Fault Stack Exchange. URL:https://serverfault.com/a/1000725/558773 (version: 2020-01-18).

[11] Wikipedia. Ethernet II — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Ethernet%5C%20frame&oldid=1080405638#Ethernet_II, 2022. [Online; accessed 04-June-2022].

[12] Wikipedia. IPv4 address exhaustion — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=IPv4%20address%20exhaustion&oldid=1090374520, 2022. [Online; accessed 04-June-2022].

[13] Wikipedia. Public key infrastructure — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Public%20key%20infrastructure&oldid=1085755772, 2022. [Online; accessed 04-June-2022].

[14] ysdx (https://serverfault.com/users/84729/ysdx). Openvpn client-to-client. Server Fault Stack Exchange. URL:https://serverfault.com/a/738558/558773 (version: 2018-01-17).