DONE - Docker Orchestrator for Network Emulation

Studio di fattibilità

CdL in Sicurezza dei Sistemi e delle Reti Informatiche

Samuele Manclossi, Melissa Moioli, Tiziano Radicchi
09882A, 09831A, 12172A

March 29, 2024

"Non quia difficilia sunt non audemos, sed quia non audemos difficilia sunt"

— Seneca

Abstract

Si propone lo studio di fattibilità per la proposta di progetto "DONE": la creazione di uno strumento, ispirato a IMUNES, per l'emulazione di reti mediante l'utilizzo automatizzato di Docker, interfaccia grafica e/o terminale.

Esso si basa sugli stessi principi di virtualizzazione alla base di altri software di emulazione di reti. L'obiettivo è quello di fornire un ambiente di sviluppo e test per reti complesse, in modo da poter testare nuove configurazioni di rete e nuovi servizi senza dover ricorrere a costosi e complessi apparati fisici, garantendo allo stesso momento dipendenze da pochi strumenti quali Docker, OpenVSwitch e la segregazione dei namespaces offerta dai sistemi operativi Linux.

Il nostro studio si concentrerà su tre macroargomenti: la virtualizzazione delle topologie di rete, la logica di emulazione e la logica di interazione, che permetterà all'utente di interagire con il programma sia tramite interfaccia grafica che da terminale.

Contents

1 Introduzione

Alla luce delle funzionalità che gli emulatori di rete oggi esistenti offrono, desideriamo, con l'obiettivo di approfondire le nostre conoscenze in ambito di reti e di sviluppo software, proporre una soluzione alternativa che offra feature extra ed un miglioramento generale della qualità di utilizzo.

1.1 Funzionalità

Le funzionalità che desideriamo offrire attraverso l'utilizzo di `DONE` sono le seguenti:

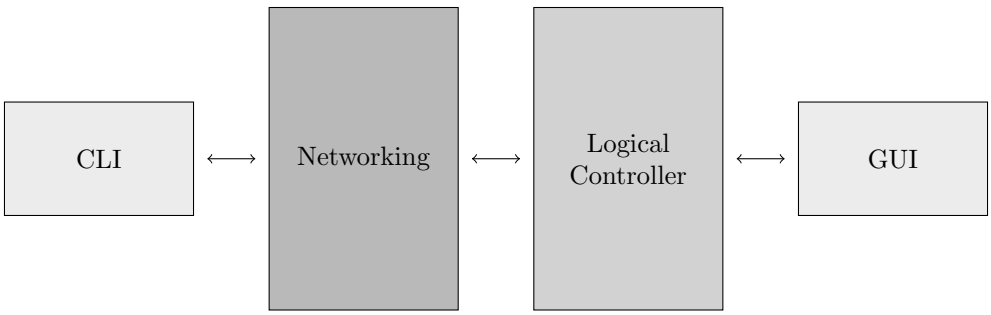
- L'utente può creare, aprire, modificare **progetti di topologie di rete**, aggiungendo apparati, definendo collegamenti tra questi, nonché fare uso di device specifici volti al collegamento della topologia definita a reti esterne alla simulazione. I dispositivi utilizzabili sono hub, switch, nodi, router, connessioni a interfacce esterne, con NAT o meno.
- Una volta realizzata la topologia desiderata, l'utente può **salvare su file il progetto**, con lo scopo di poter riprendere il lavoro in un secondo momento.
- L'utente può assegnare per ogni singolo device delle **configurazioni** che verranno applicate all'avvio della simulazione.
- L'utente può avviare la simulazione, con la possibilità in seguito di **accedere tramite terminale ai singoli nodi** per aggiungere ulteriori configurazioni, per testare la connettività e/o effettuare troubleshooting. Il tutto esclusivamente tramite linea di comando. I nodi possono ospitare vari servizi, come ad esempio server ssh, ftp e web, anch'essi configurabili da linea di comando.

Gli obiettivi che desideriamo perseguire con questo progetto sono:

- **Ridurre le dipendenze** al minimo necessario, in modo da rendere il programma il più portabile possibile.
- Fornire la **capacità di salvare** non solo la topologia fisica, ma anche tutte le configurazioni, senza bisogno di script ulteriori.
- **Maggiore pulizia dell'ambiente**, evitando che Docker rimanga in uno stato intermedio (ossia con i vecchi container ancora presenti) in caso di chiusura della applicazione senza arresto della simulazione, nonché pulizia dalle altre interfacce e apparati virtuali creati.
- Offrire una soluzione open-source che possa essere **estesa** e **modificata**.

1.2 Architettura

Alla luce delle funzionalità desiderate, l'architettura che abbiamo ideato per raggiungere gli obiettivi preposti è così organizzata:



Pertanto, le componenti da realizzare sono:

- **Networking**: Utilizzo di container e apparati virtualizzati. Si tratta del modo in cui vengono realizzati, come in IMUNES, i componenti veri e propri. Essi sono poi connessi tra di loro mediante gli appositi comandi e possono simulare una rete. Sarà gestito dal relativo controller attraverso librerie apposite sviluppate ad hoc.
- **Logical Controller**: Automatizzazione, a fronte di una topologia creata, della creazione della configurazione di rete e realizzazione di logica di controllo, a più alto livello, che possa gestire:
  - Strutture relative alla topologia creata;

  - Salvataggio della topologia;
  - Gestione e salvataggio delle configurazioni;
  - Invio delle informazioni necessarie al livello di Networking per generare la topologia creata.

- **GUI**: si tratta dell'interfaccia su cui l'utente può disegnare la topologia logica della rete, posizionando quindi nodi e link tra nodi, rilocandoli, modificandoli e interagendoci. Permetterà di accedere a CLI dedicate ai singoli nodi per inserire configurazioni.
- **CLI**: offre un'alternativa all'utilizzo combinato di GUI e logical controller, permettendo di interagire direttamente col livello di networking[1]. Essa consiste in un'istanza interattiva di Python arricchita da funzioni ad hoc che consentono di modificare il sistema live fin dalla fase di progettazione.

È inoltre prevista la possibilità, durante la simulazione, indipendentemente dalla scelta della modalità di utilizzo (sia essa GUI o CLI), di connettersi tramite **shell** ai singoli nodi.

[1] I due approcci sono mutualmente esclusivi: scegliere di utilizzare la CLI in fase di progettazione significa rinunciare alla possibilità di salvare la configurazione e la disposizione dei nodi che sono invece parte integrante dell'approccio predefinito GUI - Logical Controller.

2 Analisi

2.1 Componenti

Di seguito si analizzeranno in maniera più dettagliata le singole componenti, descrivendo come si intende realizzarle.

Tecnologie utilizzate Complessivamente, il progetto sarà realizzato utilizzando le seguenti tecnologie:

- Linguaggio **C**, con l'ausilio della libreria `raylib` per quanto concerne l'interfaccia grafica;
- **Docker**, per la virtualizzazione di nodi e router;
- **OpenVSwitch** per la virtualizzazione di switch;
- Linguaggio **Python 3** per la realizzazione della CLI.

L'assenza di ulteriori dipendenze[2] assicura un'**elevata portabilità** dell'applicazione, basando tutte le configurazioni di rete necessarie su strumenti già presenti in un ambiente Linux, come ad esempio i namespaces e la segregazione dello stack di rete tramite questi ultimi.

Come introdotto nella sezione precedente, il progetto è strutturato in **quattro layer distinti**:

Networking La componente di networking è interamente gestita tramite `openvswitch-switch`, `Docker` e l'interazione con i namespace dei container. La **realizzazione concreta della topologia** prevede la generazione dei nodi/switch richiesti, linkati grazie ad interfacce virtuali collegate da cavi virtuali, lavorando direttamente sul namespace del container.

Logical controller Rappresenta il fulcro della logica di controllo. Riceve informazioni sulla topologia creata (dunque nome, tipo e collegamenti di ciascun apparato) dal livello di GUI, le formatta, parsa ed invia al livello di Networking all'**avvio della simulazione**. Gestisce inoltre il salvataggio dei progetti (topologia, collegamenti ed eventuali configurazioni dei nodi) e l'apertura di quelli salvati in precedenza.

GUI Realizza la componente grafica con cui l'utente può interagire tramite l'applicazione, e **raccoglie i dati** generati dall'utente. Questi ultimi verranno adeguatamente incapsulati e trasmessi al Logical Controller. Si occupa inoltre di gestire i salvataggi della disposizione a livello grafico di nodi e link, appoggiandosi alle funzionalità fornite dal Logical Controller.

CLI Non prevista come configurazione di default, viene lanciata tramite una flag da linea di comando e si propone come **alternativa** all'approccio grafico fornito dall'interazione tra GUI e Logical Controller. Offre un **minimalismo** che, tuttavia, esclude la possibilità di usufruire di funzionalità nativamente implementate nell'approccio standard quali il salvataggio dei progetti o la traduzione automatica della configurazione visualizzata in sistema *running*. Il principale vantaggio è quello del **testing *on-the-go*** dei sistemi, permesso anche dalla versatilità e semplicità del linguaggio Python.

2.2 Suddivisione del lavoro

Di seguito come intendiamo suddividere il lavoro:

- L'**interfaccia grafica** e la **raccolta dei dati** forniti dall'utente (dove e quali nodi collocare, e come questi vengono collegati tra di loro) viene gestita da Samuele Manclossi tramite la realizzazione di una GUI che si appoggi a strutture dati e funzioni ad hoc. Un altro compito della GUI è quello di salvare e caricare la disposizione grafica dei nodi per utilizzi futuri. Inoltre, Samuele provvederà anche a fornire una **CLI** (wrapper Python) per interagire con le funzioni di `netlib`.
- La riformulazione e **trasmissione** di tali dati alla libreria di rete, la gestione dello **start/stop** della simulazione (anche in caso di uscita forzata, in modo da evitare di lasciare sulla macchina ospitante container *pendenti*), oltre che il **salvataggio e parsing** delle configurazioni passate su file (così da poterle agevolmente caricare in futuro) è compito del Logical Controller implementato da Melissa Moioli.

[2] Diamo per scontato che un dispositivo Unix-based sia dotato di editor di testo e terminale.

- Infine, la realizzazione della **libreria di rete** (`netlib`) a cui vengono trasmessi i dati dai livelli superiori, che comunica direttamente con Docker ed OpenVSwitch consentendo la rappresentazione tramite container di ogni dispositivo, nonché il collegamento diretto a ciascuno di essi tramite **shell**, è compito di Tiziano Radicchi. Il tutto viene permesso da un accurato studio circa le possibilità di collegamento di container tramite la configurazione dei rispettivi namespace.

Nonostante la seguente suddivisione in macroaree, le scelte architetturali ed implementative sono state prese all'unanimità ed il codice prodotto è frutto della **collaborazione di tutti i membri**, talvolta portati a contribuire anche ad aree altre rispetto alla propria.

3 Proof of Concept

Seguono una serie di screenshot della versione corrente realizzata ai fini dello studio di fattibilità del progetto.

Topologia di esempio Si propone la seguente topologia di rete con alcuni test di connettività:

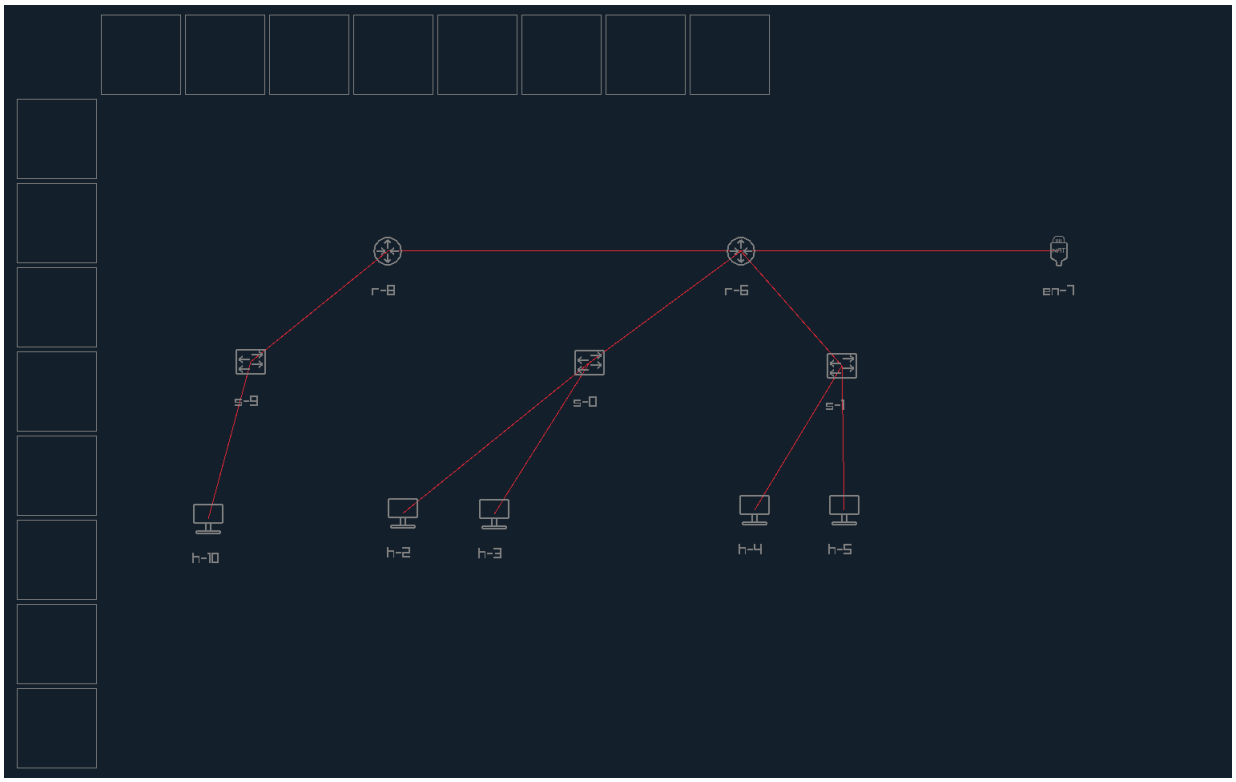


Figure 1: Alcune subnet collegate ad un'interfaccia esterna con NAT.

```
tiz314@Gemini:~$ docker exec -it h-10 bash
root@ab365e522fdc:/# ip addr add 10.0.0.1/24 dev sveth-h-10-s-9
root@ab365e522fdc:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
321: sveth-h-10-s-9@if320: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 100
0
    link/ether 02:f8:21:c1:9c:d2 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.0.1/24 scope global sveth-h-10-s-9
       valid_lft forever preferred_lft forever
    inet6 fe80::f8:21ff:fec1:9cd2/64 scope link
       valid_lft forever preferred_lft forever
root@ab365e522fdc:/#
```

Figure 2: Assegnamento IP ad `h-10`.

```
root@ab365e522fdc:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
321: sveth-h-10-s-9@if320: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 100
0
    link/ether 02:f8:21:c1:9c:d2 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.0.1/24 scope global sveth-h-10-s-9
       valid_lft forever preferred_lft forever
    inet6 fe80::f8:21ff:fec1:9cd2/64 scope link
       valid_lft forever preferred_lft forever
root@ab365e522fdc:/# ip route
default via 10.0.0.2 dev sveth-h-10-s-9
10.0.0.0/24 dev sveth-h-10-s-9 proto kernel scope link src 10.0.0.1
root@ab365e522fdc:/# ping -c 1 10.0.2.1
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.
64 bytes from 10.0.2.1: icmp_seq=1 ttl=62 time=0.657 ms

--- 10.0.2.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.657/0.657/0.657/0.000 ms
root@ab365e522fdc:/#
```

Figure 3: Ping tra `h-10` ed `h-2`.

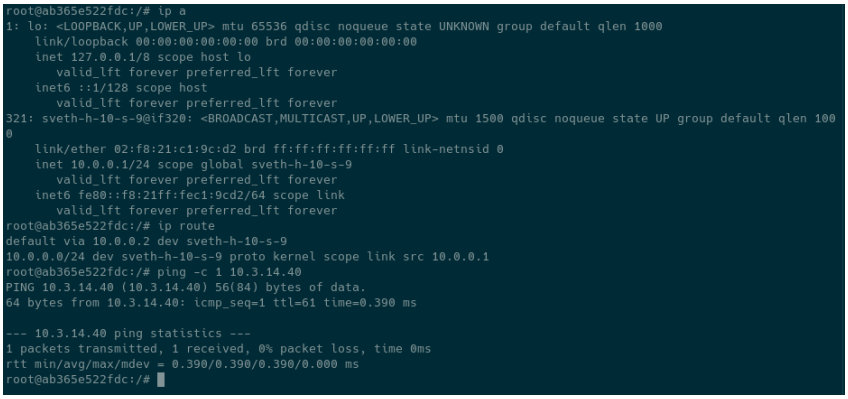


Figure 4: Ping tra `h-10` e host fisico.

Sviluppi futuri Tra gli sviluppi futuri prevediamo:

- Interfaccia grafica migliorata e più *user-friendly*;
- Connessione con CLI direttamente da DONE;
- Salvataggio delle topologie.