Progetto 05:

Relazione Tecnica

Progetto 05: Reti di Calcolatori A.A. 2024–2025

Bonì Alessia – Matricola 0698496

1. Introduzione

Questa relazione descrive la progettazione e l'implementazione di una topologia di rete virtuale in Mininet, controllata da un controller SDN esterno basato su Ryu, e l'integrazione di due applicazioni Flask. La topologia originale comprendeva cinque host, quattro switch L3, uno switch L2 e un nodo NAT creato con `net.addNat()`. Successivamente, per risolvere alcune limitazioni, sono stati aggiunti un nodo `nat_router` interno e un server dedicato all'esecuzione di traceroute.

- **H1**: server REST per l'esecuzione di comandi da remoto tramite richieste HTTP.
- **Server traceroute**: nodo dedicato all'esecuzione di traceroute verso tutti gli host della rete.

2. Premessa e criticità

L'uso di `net.addNat()` ha presentato le seguenti difficoltà:

1. **Scatola nera**: il nodo NAT opera al di fuori del controllo del controller SDN, gestito dal sistema operativo.
2. **Interfacciamento automatico**: si connette al primo switch disponibile, sovrapponendo potenzialmente i link esistenti e generando errori `NETLINK: file exists`.

Soluzione adottata: creazione di un nodo interno `nat_router` che riproduce le funzionalità di NAT sotto il pieno controllo del controller SDN, integrando il server Flask per il traceroute.

3. Obiettivi e requisiti

- Garantire connettività completa tra tutti i nodi di Mininet.
- Gestire traceroute dal server verso ciascun host.

- Permettere l'esecuzione di comandi su H1 tramite API REST.
- Simulare accesso a Internet con funzionalità NAT.
- Configurare specifiche di banda e ritardi per i link.
- Eseguire un traceroute partendo da un nodo server e come destinazione i nodi host della rete

4. Tecnologie utilizzate

- **Mininet** con link TCLink (banda, delay)
- **OpenFlow 1.3**
- **Ryu** (ryu-manager)
- **Python**
- **Flask**

5. Architettura di rete

La topologia è definita in `topology.py` e include:

- **Switch L2**: S1
- **Switch L3**: R1, R2, R3, R4
- **Host**: H1, H2, H3, H4, H5
- **Nodo NAT** originale: `net.addNat()`
- **Nodo NAT interno**: `nat_router`
- **Server traceroute**

Lo script si occupa di:

1. Assegnare indirizzi IP e route di default (verso il rispettivo router).
2. Creare i link con specifiche di bandwidth e delay.
3. Avviare il controller Ryu e collegare gli switch.
4. Avviare le applicazioni Flask tramite la CLI di Mininet.

6. Controller SDN (`controller.py`)

Il controller implementa:

- **Routing statico** e longest prefix match.

- Gestione ARP per gli switch e i router virtuali.
- **DNAT** per esporre il server interno alla rete esterna.

Componenti principali:

1. **Mappature statiche**: associazioni DPID ↔ indirizzi IP/MAC, tabelle di routing e NAT.
2. **Handler packet-in**:
   - `_handle_l2_switch()`: forwarding a livello 2.
   - `_handle_nat_router()`: gestione DNAT e inoltro verso `nat_router`.
   - `_handle_l3_router()`: routing IP tra le subnet.
3. **Gestione ARP**: risponde alle richieste ARP degli host.
4. **NAT**: modifica indirizzi di destinazione per esporre il servizio traceroute.

7. Applicazioni Flask

7.1 `h1_server.py`

- **Funzionalità**: API REST per eseguire in background comandi di sistema (ad es. `ping`, `ifconfig`, `netstat`, `hostname`).
- **Whitelist**: comandi autorizzati. Richieste non consentite vengono rifiutate.
- **Logging**: ogni richiesta (eseguita, rifiutata o fallita) viene registrata in formato JSON con timestamp e IP del client.
- **Esempio di richiesta**:

```
host_sorgente> curl -s -X POST -d "command=ping -c <numero_pacchetti> <IP_destinazione>" http://10.0.0.2:8001/execute
```

7.2 `traceroute_server.py`

- **Funzionalità**: esecuzione di traceroute UDP verso una lista di destinazioni (`TARGET_HOSTS`).
- **Tecnica**: invio di pacchetti UDP con TTL crescente e ascolto di ICMP Time Exceeded.
- **Output**: risultati in tempo reale su console e log JSON (`traceroute_log.json`).

- **Requisiti**: privilegi di root per l’uso di socket raw.

Risultati ottenuti:

Si testano i vari componenti relativi al progetto

Esecuzione di ryu: con il comando ryu-manager controller.py

Esecuzione della topologia: con sudo python topology.py

Test della correttezza della larghezza di banda e dei vari ritardi:

```
*** Aggiunta del controller remoto
*** Aggiunta dei dispositivi di rete
*** Aggiunta degli host
*** Aggiunta del Router NAT (gestito da Ryu) e del Server
*** Creazione dei collegamenti interni (rispettando le porte del controller)
(1000.00Mbit 0.05ms delay) (1000.00Mbit 0.05ms delay)
(1.00Mbit 2ms delay) (1.00Mbit 2ms delay) (5.00Mbit 2ms delay)
 (5.00Mbit 2ms delay) (20.00Mbit 2ms delay) (20.00Mbit 2ms delay)
 (100.00Mbit 0.05ms delay) (100.00Mbit 0.05ms delay)
 (1.00Mbit 0.5ms delay) (1.00Mbit 0.5ms delay)
 (100.00Mbit 0.05ms delay) (100.00Mbit 0.05ms delay)
 (54.00Mbit 0.05ms delay) (54.00Mbit 0.05ms delay)
  (54.00Mbit 0.05ms delay) (54.00Mbit 0.05ms delay)
*** Aggiunta e collegamento del nodo NAT di Mininet
*** Avvio della rete
*** Configuring hosts
h1 h2 h3 h4 h5 server nat0
(1000.00Mbit 0.05ms delay) (54.00Mbit 0.05ms delay)
(54.00Mbit 0.05ms delay) (1000.00Mbit 0.05ms delay)
(1.00Mbit 2ms delay) (5.00Mbit 2ms delay) (1.00Mbit 2ms delay)
(100.00Mbit 0.05ms delay) (20.00Mbit 2ms delay)
(1.00Mbit 0.5ms delay) (5.00Mbit 2ms delay) (20.00Mbit 2ms delay)
(100.00Mbit 0.05ms delay)
*** Esecuzione della CLI di Mininet
```

Test di connettività: si usa pingall per vedere se le componenti sono raggiungibili:

```
pingall
*** Ping: testing ping reachability
h1 → h2 h3 h4 h5 server X
h2 → h1 h3 h4 h5 server X
h3 → h1 h2 h4 h5 server X
h4 → h1 h2 h3 h5 server X
h5 → h1 h2 h3 h4 server X
server → h1 h2 h3 h4 h5 X
nat0 → X X X X X X
*** Results: 28% dropped (30/42 received)
```

Tutte le componenti ad eccezione del nodo nat di mininet, si connettonno fra di loro

Test per Il nat se può pingare l'esterno: prova con ping 8.8.8.8

```
nat0 ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=116 time=168 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=85.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=116 time=279 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=116 time=139 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 85.804/167.929/278.970/70.583 ms
```

Avvio del server di traceroute:

mininet> server sudo python traceroute_server.py

> Avvio del comando traceroute...
> Destinazioni: ['10.0.0.2', '10.0.0.3', '11.0.0.2', '192.168.1.2', '10.8.1.2']
> Eseguendo traceroute dal server verso 10.0.0.2...
> Traceroute verso 10.0.0.2 completato.

Si avrà questo risultato nel file traceroute.json


```
{
  "run_timestamp": "2025-06-28T06:22:16.973795",
  "traces": [
    {
      "destination": "10.0.0.2",
      "output": [
        "Hop   Address          Host Name                       Time     ",
        "1     10.0.0.2         10.0.0.2                      38.71 ms  ",
        " Traceroute terminato "
      ]
    },
    {
      "destination": "10.0.0.3",
      "output": [
        "Hop   Address          Host Name                       Time     ",
        "1     10.0.0.3         10.0.0.3                      42.03 ms  ",
        " Traceroute terminato "
      ]
    },
    {
      "destination": "11.0.0.2",
      "output": [
        "Hop   Address          Host Name                       Time     ",
        "1     11.0.0.2         11.0.0.2                      49.47 ms  ",
        " Traceroute terminato "
      ]
    },
    {
      "destination": "192.168.1.2",
      "output": [
        "Hop   Address          Host Name                       Time     ",
        "1     192.168.1.2      192.168.1.2                   16.78 ms  ",
        " Traceroute terminato "
      ]
    },
    {
```

```
        "destination": "10.8.1.2",
        "output": [
          "Hop   Address          Host Name                          Time      ",
          "1     10.8.1.2         10.8.1.2                        41.05 ms  ",
          " Traceroute terminato "
        ]
      }
    ]
}
```


Avvio del server su h1: h1 python h1_server.py &

Esempi di comandi:

Comando Permesso:

`h3 curl -s -X POST -d "command=ifconfig" http://10.0.0.2:8001/execute`


```
{
    "timestamp": "2025-06-28T06:23:58.646780",
    "source_ip": "11.0.0.2",
    "requested_command": "ifconfig",
    "status": "success",
    "result": {
      "stdout": [
        "h1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu
        "        inet 10.0.0.2  netmask 255.255.255.0  broadcast 10.0.0.255",
        "        ether 00:00:00:00:00:01  txqueuelen 1000  (Ethernet)",
        "        RX packets 39  bytes 3200 (3.2 KB)",
        "        RX errors 0  dropped 0  overruns 0  frame 0",
        "        TX packets 33  bytes 3215 (3.2 KB)",
        "        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0",
        "",
        "lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536",
        "        inet 127.0.0.1  netmask 255.0.0.0",
        "        loop  txqueuelen 1000  (Local Loopback)",
        "        RX packets 1  bytes 112 (112.0 B)",
        "        RX errors 0  dropped 0  overruns 0  frame 0",
        "        TX packets 1  bytes 112 (112.0 B)",
```

```
            "       TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0"
        ],
        "stderr": [],
        "return_code": 0
    }
}
```


Comando non permesso:

`h2 curl -s -X POST -d "command=ls -l /" http://10.0.0.2:8001/execute`


```
{
        "destination": "10.8.1.2",
        "output": [
            "Hop   Address           Host Name                          Time      ",
            "1     10.8.1.2          10.8.1.2                        41.05 ms  ",
            " Traceroute terminato "
        ]
    }
]
}
```


Conclusioni:

Il progetto ha raggiunto la maggior parte degli obbiettivi prefissati

Il nodo server inizialmente doveva essere esterno , ma dati i problemi di `mininet.addNat()` non è stato possibile raggiungere l'obbiettivo