{ComNetsEmu}

SDN SLICING SU RICHIESTA

Alessandro Brognara, Konstantinos Zefkilis, Luca Pio Pierno

COS'E' IL NETWORK SLICING

Il Network Slicing è una tecnologia che permette di suddividere una rete fisica in più slice, ognuna delle quali funziona come una rete virtuale indipendente.

Topologia

Una topologia di rete è come sono collegati i dispositivi (computer, server) in una rete. Nel nostro caso, abbiamo uno switch centrale che gestisce vari switch di distribuzione, ciascuno collegato a gruppi di host associati a una "slice" specifica.

Gestione delle slice

Abbiamo script che ci permettono di gestire le slice, attivandole o disattivandole a seconda delle necessità. Ad esempio, possiamo isolare una slice spegnendola, oppure riattivarla per ristabilire la comunicazione con il resto della rete.

QoS (Qualità del servizio)

Abbiamo impostato regole che garantiscono che ogni slice abbia la giusta banda e latenza, a seconda delle esigenze. Per esempio, una slice per la simulazione medica avrà bisogno di alta velocità e bassa latenza.

PERCHE’ ?

ISOLAMENTO DEL TRAFFICO: Le diverse applicazioni non interferiscono tra loro. Ad esempio, una video conferenza (che richiede alta qualità) non verrà rallentata da un’altra applicazione meno importante.

FLESSIBILITA’: La rete può essere facilmente adattata a diverse esigenze. Se cambia la domanda, possiamo creare o modificare le slice senza dover cambiare l'intera rete fisica.

SICUREZZA: Ogni slice è separata dalle altre, quindi eventuali problemi o attacchi in una slice non influenzano le altre.

INTRODUZIONE AL PROGETTO

Obiettivo del Progetto: Implementare un sistema di Network Slicing utilizzando Mininet per la simulazione della rete e Ryu come controller SDN (Software-Defined Networking). Configurare la topologia di rete e gestire dinamicamente il traffico tra le diverse slice, garantendo qualità del servizio (QoS) differenziata.

STRUMENTI UTILIZZATI:

Mininet:

Piattaforma di simulazione di rete leggera che permette di creare e testare topologie di rete virtuali.

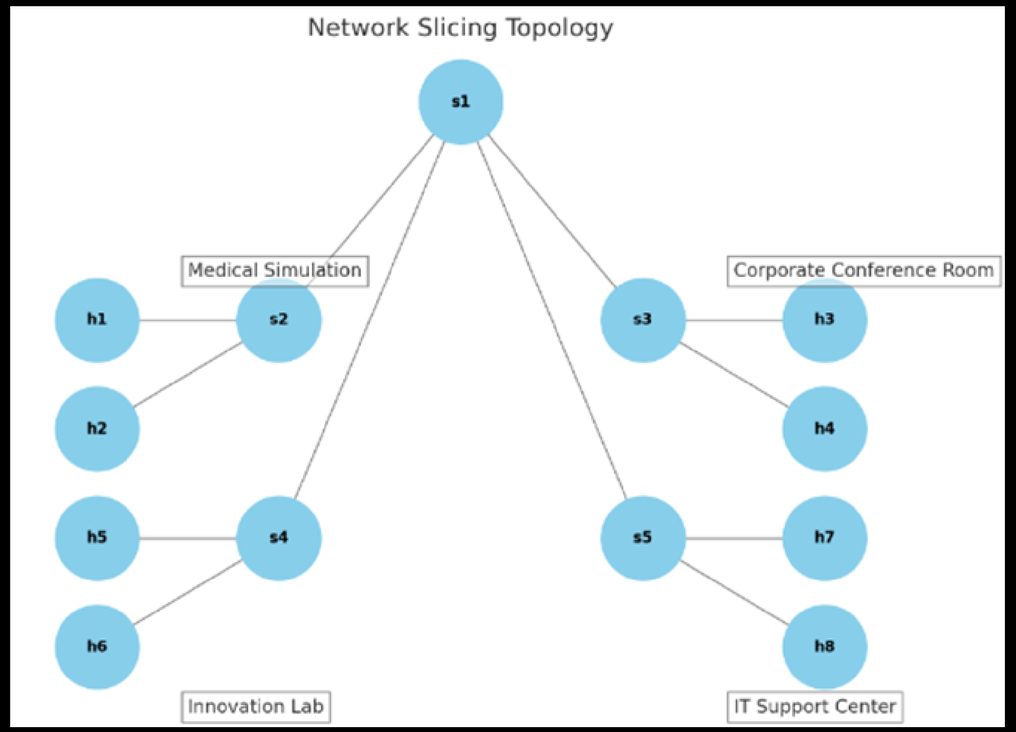
Ryu Controller:

Un framework di controllo SDN basato su OpenFlow, utilizzato per gestire e controllare dinamicamente la rete.

LA NOSTRA TOPOLOGIA

L'architettura del progetto è basata su una topologia di rete composta da:

- Uno switch centrale S1 (core switch)
- Quattro switch di distribuzione (s2, s3, s4, s5)
- Otto host (h1-h8), collegati ai rispettivi switch di distribuzione (s1-h1,h2), (s2-h3,h4), (s3-h5,h6), (s4-h7,h8).
- Ogni switch di distribuzione è collegato a due host e gestisce un slice specifico.



FASCE DI RETE E CASI D’USO

Fascia 1: Laboratorio di simulazione mediche

- Bitrate: 150 Mbps
- Latenza: Molto bassa
- Tipo: TCP
- Utilizzo: Simulazioni in tempo reale, alta precisione.

Fascia 2: Sala Conferenze Aziendale (h3, h4)

- Bitrate: 100 Mbps
- Latenza: Moderata
- Tipo: UDP/TCP
- Utilizzo: Videoconferenze e presentazioni dal vivo.

Fascia 3: Laboratorio di Innovazione e Ricerca (h5, h6)

- Bitrate: 120 Mbps
- Latenza: No
- Tipo: TCP
- Utilizzo: Accesso a risorse cloud e calcolo distribuito.

Fascia 4: Centro di Supporto IT (h7, h8)

- Bitrate: 50 Mbps
- Latenza: No
- Tipo: TCP
- Utilizzo: Monitoraggio della rete e supporto continuo.

APPLICAZIONE FASCE DI RETE

QoS (Quality of Service)

tecnologia utilizzata per garantire che la rete distribuisca le risorse in modo efficiente, assegnando priorità al traffico e garantendo che le applicazioni critiche ricevano la larghezza di banda necessaria. In questa implementazione, QoS è stato configurato per diverse fasce di dispositivi (o "slice") all'interno della rete, con specifiche impostazioni di larghezza di banda per ciascuna fascia.

```
mininet> iperf h2 h1
*** Iperf: testing TCP bandwidth between h2 and h1
*** Results: ['112 Mbits/sec', '113 Mbits/sec']
```

```
mininet> iperf h8 h7
*** Iperf: testing TCP bandwidth between h8 and h7
*** Results: ['12.8 Mbits/sec', '14.0 Mbits/sec']
```

UDP (User Datagram Protocol)

protocollo di comunicazione veloce e leggero, utilizzato principalmente per trasmettere dati in situazioni dove la velocità è preferita rispetto all'affidabilità, come nel nostro caso nella seconda slice (Sala Conferenze Aziendale)

```
mininet> h3 iperf -c h4 -p 5050 -u -t 40
------------------------------------------------------------
Client connecting to 10.0.0.4, UDP port 5050
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.3 port 49844 connected with 10.0.0.4 port 5050
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-40.0 sec  5.00 MBytes  1.05 Mbits/sec
[  3] Sent 3567 datagrams
[  3] Server Report:
[  3]  0.0-40.0 sec  5.00 MBytes  1.05 Mbits/sec   0.008 ms    0/ 3567 (0%)
```

PROVA SUL TERMINALE

SLICE 1 DISATTIVATA: La prima slice, che coinvolge gli host h1 e h2, è stata disabilitata, isolandoli completamente dalla rete, quindi tutti i ping verso e da questi due host falliscono, causando la perdita del 46% dei pacchetti.

```
vagrant@comnetsemu:~/comnetsemu/progettoslice$ ./gestione_slice.sh
*** Inserisci comando per gestire le slice (ATTIVA TUTTO, DISATTIVAx, ATTIVAx
) con x numero della slice compreso tra 1 e 4:
DISATTIVA1
*** Spegnimento della slice 1
---------- Disattivazione Slice 1 ----------
Slice 1 disattivata.
```

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X X X X X
h2 -> X X X X X X X
h3 -> X X h4 h5 h6 h7 h8
h4 -> X X h3 h5 h6 h7 h8
h5 -> X X h3 h4 h6 h7 h8
h6 -> X X h3 h4 h5 h7 h8
h7 -> X X h3 h4 h5 h6 h8
h8 -> X X h3 h4 h5 h6 h7
*** Results: 46% dropped (30/56 received)
```

SLICE 1 e 2 DISATTIVATE: Con le slice 1 e 2 disattivate, gli host h1, h2, h3, e h4 sono completamente isolati e non possono comunicare con il resto della rete. Solo gli host delle slice attive (3 e 4: h5, h6, h7, h8) possono ancora comunicare tra loro, con una perdita totale del 78% dei pacchetti.

```
*** Inserisci comando per gestire le slice (ATTIVA TUTTO, DISATTIVAx, ATTIVAx
) con x numero della slice compreso tra 1 e 4:
DISATTIVA2
*** Spegnimento della slice 2
---------- Disattivazione Slice 2 ----------
Slice 2 disattivata.
```

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X X X X X
h2 -> X X X X X X X
h3 -> X X X X X X X
h4 -> X X X X X X X
h5 -> X X X X h6 h7 h8
h6 -> X X X X h5 h7 h8
h7 -> X X X X h5 h6 h8
h8 -> X X X X h5 h6 h7
*** Results: 78% dropped (12/56 received)
```

ATTIVA TUTTO: riattiva tutte le slice della rete che erano state disabilitate. In pratica, riconnette tutti i gruppi di host, permettendo loro di comunicare nuovamente tra di loro e con il resto della rete. Infatti in questo caso ci sono zero pacchetti persi.

```
*** Inserisci comando per gestire le slice (ATTIVA TUTTO, DISATTIVAx, ATTIVAx
) con x numero della slice compreso tra 1 e 4:
ATTIVA TUTTO
*** Attivazione di tutte le slice
---------- Riattivazione di Tutte le Slice ----------
Tutte le slice sono state attivate.
```

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
```

PARTI FONDAMENTALI DEL CODICE

Topologia della rete (topology.py):

Questo codice crea la topologia di base della rete, con uno switch centrale (core_switch) collegato a quattro switch di distribuzione (dist_switch). È fondamentale perché definisce l'architettura fisica su cui si baserà tutto il progetto, permettendo la connessione e il controllo degli host.

```
core_switch = self.addSwitch('s1', dpid="0000000000000001")
for i in range(4):
    dist_switch = self.addSwitch("s%d" % (i + 2), dpid="%016x" % (i + 2))
    self.addLink(core_switch, dist_switch, **link_config)
```

Controller (controller.py):

Questa funzione aggiunge nuove regole di instradamento (flow entries) agli switch della rete. È cruciale perché permette al controller di gestire dinamicamente come i pacchetti di dati vengono instradati all'interno della rete, basandosi sulle esigenze specifiche della topologia e delle slice.

```
def add_flow(self, datapath, priority, match, actions):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
    mod = parser.OFPFlowMod(
        datapath=datapath, priority=priority, match=match, instructions=inst
    )
    datapath.send_msg(mod)
```

Disattivazione della slice 1 (slice1.sh):

Disabilita specifiche porte di rete e blocca il traffico tra determinati switch, isolando la "Slice 1". Questo è essenziale per gestire la suddivisione della rete in diverse slice e permette il controllo delle risorse assegnate a ciascuna slice.

```
sudo ifconfig s2-eth1 down
sudo ifconfig s2-eth2 down
sudo ovs-ofctl add-flow s2 priority=65500,in_port=s2-eth3,actions=drop
sudo ovs-ofctl add-flow s1 priority=65500,in_port=s1-eth1,actions=drop
```

Attivazione delle slice (attiva_slicing.sh):

Configura la qualità del servizio (QoS) per una porta specifica, garantendo una larghezza di banda massima per gli host collegati. Questo è cruciale per assicurare che ogni slice abbia le risorse di rete necessarie per funzionare correttamente, rispettando le priorità e i requisiti di prestazione definiti.

```
sudo ovs-vsctl set port s2-eth1 qos=@newqos -- \
    --id=@newqos create QoS type=linux-htb \
    other-config:max-rate=150000000 \
    queues:1=@1q -- \
    --id=@1q create queue other-config:min-rate=140000000 other-config:max-rate=150000000 >/dev/null
```

Impostazione delle slice (attiva_slicing.sh):

Definizione della larghezza di banda per le slice - attiva_slicing.sh: Questa parte del codice imposta la larghezza di banda massima e minima per le porte specifiche della rete. Ogni slice ha una configurazione di QoS che limita la quantità di larghezza di banda che gli host collegati possono utilizzare. Questo è essenziale per garantire che la rete possa supportare le diverse esigenze delle applicazioni o dei servizi che operano su ciascuna slice, mantenendo prestazioni prevedibili e stabili.

```
sudo ovs-vsctl set port s2-eth1 qos=@newqos -- \
    --id=@newqos create QoS type=linux-htb \
    other-config:max-rate=150000000 \
    queues:1=@1q -- \
    --id=@1q create queue other-config:min-rate=140000000 other-config:max-rate=150000000 >/dev/null
```

```
sudo ovs-vsctl set port s2-eth2 qos=@newqos -- \
    --id=@newqos create QoS type=linux-htb \
    other-config:max-rate=150000000 \
    queues:2=@2q -- \
    --id=@2q create queue other-config:min-rate=140000000 other-config:max-rate=150000000 >/dev/null

# Ripetuto per le altre fasce con valori diversi.
```