Progetto di Reti di Calcolatori

Progetto 03 : Reti di Calcolatori A.A. 2024-2025

Habasescu Alin Marian
Matricola: 0733952

October 29, 2025

Contents

1 Introduzione

Questa relazione documenta lo sviluppo del Progetto 03 per il corso di Reti di Calcolatori. L'obiettivo era simulare una rete utilizzando l'emulatore Mininet e gestirla tramite un controller SDN esterno (Ryu) che implementa routing L3 statico. Parte importante del progetto è stata la creazione di un'API REST (Flask) sull'host H1 per avviare test di performance `iperf` tra i nodi della rete, con salvataggio automatico dei risultati su file CSV.

2 Analisi Progettuale

2.1 Obiettivi e Requisiti

Gli obiettivi principali sono stati:

- **Emulazione Topologia:** Realizzare in Mininet la rete specificata (5 host, 4 router L3, 4 subnet). Si noti che il link tra SW3 e SW4 era originariamente etichettato con una subnet mask /32 e dato l'uso non standard di una /32 per un link punto-punto tra router, si è scelto di interpretare tale specifica come un refuso e di implementare il link utilizzando una subnet /30, in modo simile agli altri collegamenti router-router.
- **Configurazione Link:** Applicare le specifiche di banda (Rate) e latenza (Delay) ai collegamenti.
- **Routing L3 Statico:** Usare Ryu per trasformare gli switch Mininet in router con regole statiche.
- **Connettività Completa:** Garantire raggiungibilità tra tutti gli host.
- **Servizio API iPerf:** Implementare un server API REST su H1 (Flask) con endpoint `/start_iperf` (parametri: 'IP_DEST', 'L4_proto', 'src_rate') e `/stop_iperf`.
- **Server iPerf:** Avviare server `iperf` TCP (porta 5001) e UDP (porta 5002) su tutti gli host all'avvio.
- **Logging:** Salvare risultati `iperf` in `.csv` sugli host destinatari e loggare le richieste API su H1.

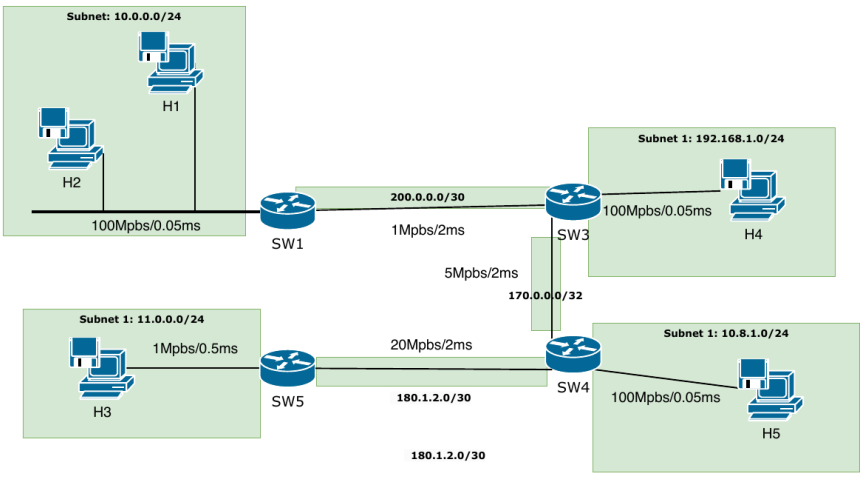

Figure 1: Topologia di rete da implementare in Mininet

2.2 Tecnologie Utilizzate

Le tecnologie impiegate sono:

- Mininet (con TCLink), Ryu SDN Framework, OpenFlow 1.3
- Python 3, Flask
- iperf (versione 2), cURL (per i test API)

3 Architettura del Sistema

Il sistema è composto da tre componenti software principali che interagiscono all'interno dell'ambiente di emulazione Mininet.

3.1 Schema Logico

La topologia di rete implementata cerca di ricreare al meglio quella definita nella Traccia 03, con 5 host (h1-h5) e 4 switch Open vSwitch (s1-s4) configurati dal controller Ryu per agire come router L3. La Figura 2 mostra lo schema logico della rete.

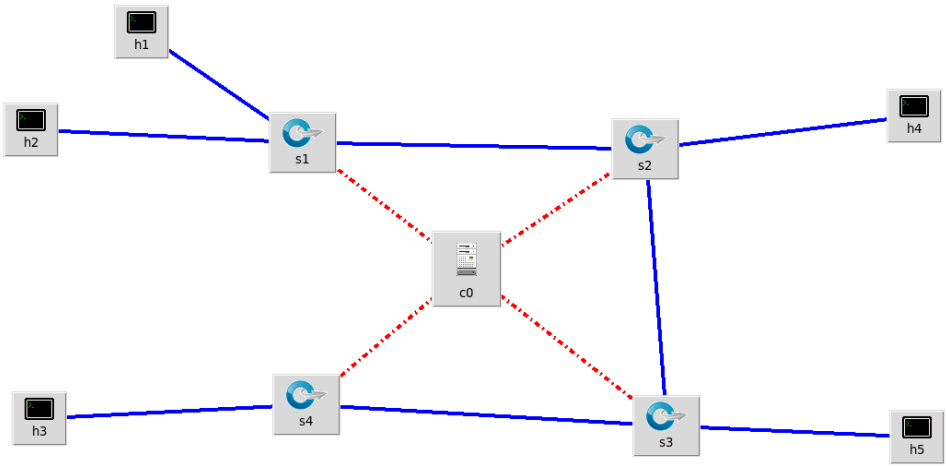

Figure 2: Schema logico della rete emulata in Mininet.

3.2 Componenti Software

3.2.1 Script di Topologia Mininet (`topology.py`)

Questo script Python utilizza l'API di Mininet per costruire dinamicamente la rete virtuale. Le sue responsabilità principali sono:

- Creare gli host (H1-H5), assegnando loro indirizzi IP, MAC statici e route di default verso il rispettivo gateway.
- Creare gli switch Open vSwitch (S1-S4), assegnando DPID univoci per l'identificazione da parte del controller Ryu.
- Definire il controller Ryu come controller remoto esterno.
- Stabilire i collegamenti tra host e switch e tra switch, utilizzando 'TCLink' per applicare le specifiche di banda e latenza richieste.
- Avviare automaticamente, dopo l'avvio della rete, i due server `iperf` (TCP e UDP) su ciascun host, con salvataggio dei log su file csv.
- Avviare in background il server API Flask sull'host H1.

3.2.2 Controller SDN Ryu (`controller.py`)

Questo script Python implementa un'applicazione Ryu che agisce da piano di controllo centralizzato per la rete. Realizza un router L3 statico con funzionalità L2 ibride, ispirandosi all'approccio visto nel Progetto 05 ma adattato a questa specifica topologia. Le sue funzioni chiave sono:

- **Configurazione Statica:** Mantiene in memoria le tabelle ARP, le mappature porta/-subnet degli switch e le tabelle di routing statico definite nell'`__init__`.
- **Gestione ARP:** Risponde autonomamente alle richieste ARP indirizzate agli IP dei gateway dei router ('_handle_arp').
- **Switching L2 Locale (S1):** Gestisce il traffico tra H1 e H2 (collegati alle porte 1 e 2 di S1) tramite apprendimento MAC e inoltro locale, senza coinvolgere la logica L3 ('_handle_l2_switch').
- **Routing L3:** Per tutti gli altri pacchetti IP, determina il next-hop sulla tabella di routing, trova il MAC e la porta di uscita corretti, decrementa il TTL, aggiorna gli header MAC e installa una flow rule OpenFlow sullo switch per instradare il traffico ('_handle_ipv4').
- **Regola Table-Miss:** Installa una regola di default a bassa priorità su ogni switch per inviare al controller i pacchetti sconosciuti ('_switch_features_handler').

3.2.3 API Server Flask (`h1_server.py`)

Un semplice server web basato su Flask, in esecuzione sull'host H1, che espone un'API REST per controllare i test `iperf`.

- **Endpoint `/start_iperf` (POST):** Riceve 'IP_DEST', 'L4_proto', 'src_rate'. Costruisce il comando `iperf -c` appropriato (selezionando la porta 5001 per TCP o 5002 per UDP) e lo esegue in background su H1.
- **Endpoint `/stop_iperf` (POST):** Esegue `killall iperf` su H1 per terminare eventuali test client attivi.
- **Logging API:** Ogni richiesta ricevuta viene registrata con timestamp, IP sorgente, comando richiesto e risposta inviata nel file `h1_command_log.json`.

4 Implementazione e Validazione

Questa sezione descrive l'ambiente utilizzato, la metodologia di test e i risultati ottenuti per verificare il corretto funzionamento del sistema implementato.

4.1 Ambiente di Sviluppo e Esecuzione

Il progetto è stato sviluppato e testato sulla VM ufficiale Mininet versione 2.3.0 (basata su Ubuntu 20.04), utilizzando Python 3.8.5, Ryu 4.30 e Flask 3.0.3 L'interazione con i processi è stata gestita tramite tmux.

4.2 Metodologia di Test

La validazione del sistema è avvenuta in più fasi:

1. **Test di Connettività Base:** Dopo l'avvio della topologia e del controller L3, è stato eseguito il comando `pingall` di Mininet per verificare la raggiungibilità L3 tra tutte le coppie di host.
2. **Test Funzionale API e Routing:** È stato creato uno script Bash (`test_L3.sh`) per automatizzare l'invio di richieste `curl` all'API Flask su H1 da parte di diversi host (H2, H3, H4, H5). Lo script ha richiesto l'avvio di test `iperf` (sia TCP che UDP, con diversi rate) tra host situati in subnet differenti, testando così tutti i requisiti del progetto.
3. **Verifica dei Log:** Al termine dei test, sono stati ispezionati i file di log generati:
   - `h1_command_log.json`: Per confermare la corretta ricezione ed esecuzione dei comandi da parte dell'API.
   - `hX_tcp_log.csv` e `hX_udp_log.csv`: Per verificare che i server `iperf` sugli host destinatari (HX) avessero ricevuto il traffico e registrato correttamente i risultati nel formato CSV atteso.

4.3 Risultati Ottenuti

I test effettuati hanno confermato il corretto funzionamento del sistema.

Connettività L3

L'esecuzione del comando `pingall` ha mostrato una raggiungibilità completa tra tutti gli host.

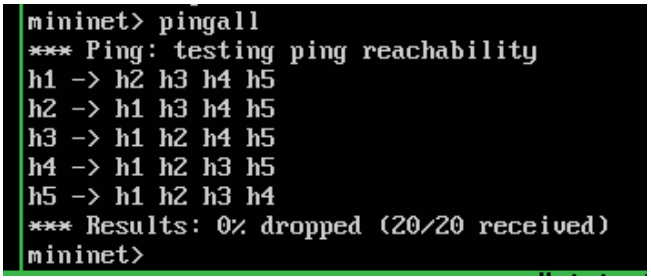

Figure 3: Output del comando 'pingall' dopo l'avvio della rete.

Test API e iPerf

L'esecuzione dello script `test_l3.sh` ha prodotto i risultati attesi. L'API Flask su H1 ha risposto correttamente a tutte le richieste `curl`, come registrato nel file `h1_command_log.json`. Contestualmente, i server `iperf` sugli host destinatari hanno ricevuto il traffico e generato i log CSV attesi. La Figura 4 mostra estratti significativi di questi log.

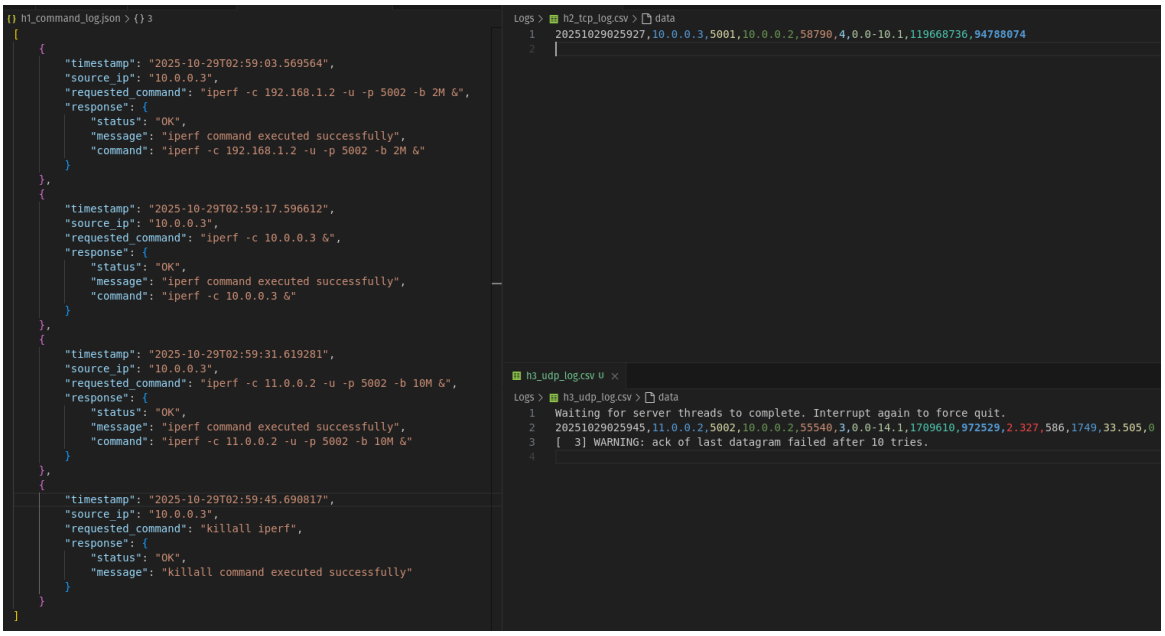

Figure 4: File di log, dopo l'esecuzione dello script 'test_L3.sh'.

5 Conclusioni

Il progetto ha permesso di realizzare con successo l'emulazione di una rete gestita tramite SDN, soddisfacendo tutti i requisiti del Progetto 03. È stata implementata la topologia specificata in Mininet, applicando le corrette caratteristiche ai link. Il controller Ryu sviluppato gestisce efficacemente il routing L3 statico tra le diverse subnet, includendo la gestione ARP e una logica ibrida L2/L3 per lo switch S1. L'API REST su H1, realizzata con Flask, fornisce un'interfaccia funzionale per l'esecuzione remota di test di performance `iperf`, i cui risultati vengono correttamente loggati.