Report Homework 2 - ns-3

Nel documento contenente le nostre direttive, viene specificato come per la creazione del nostro *network* sia richiesto di basarci su un'immagine che ne descrive le specifiche; immediatamente si può notare come esso sia composto da topologie di natura diversa collegate tra loro. Descriviamo alcune delle scelte progettuali fatte.

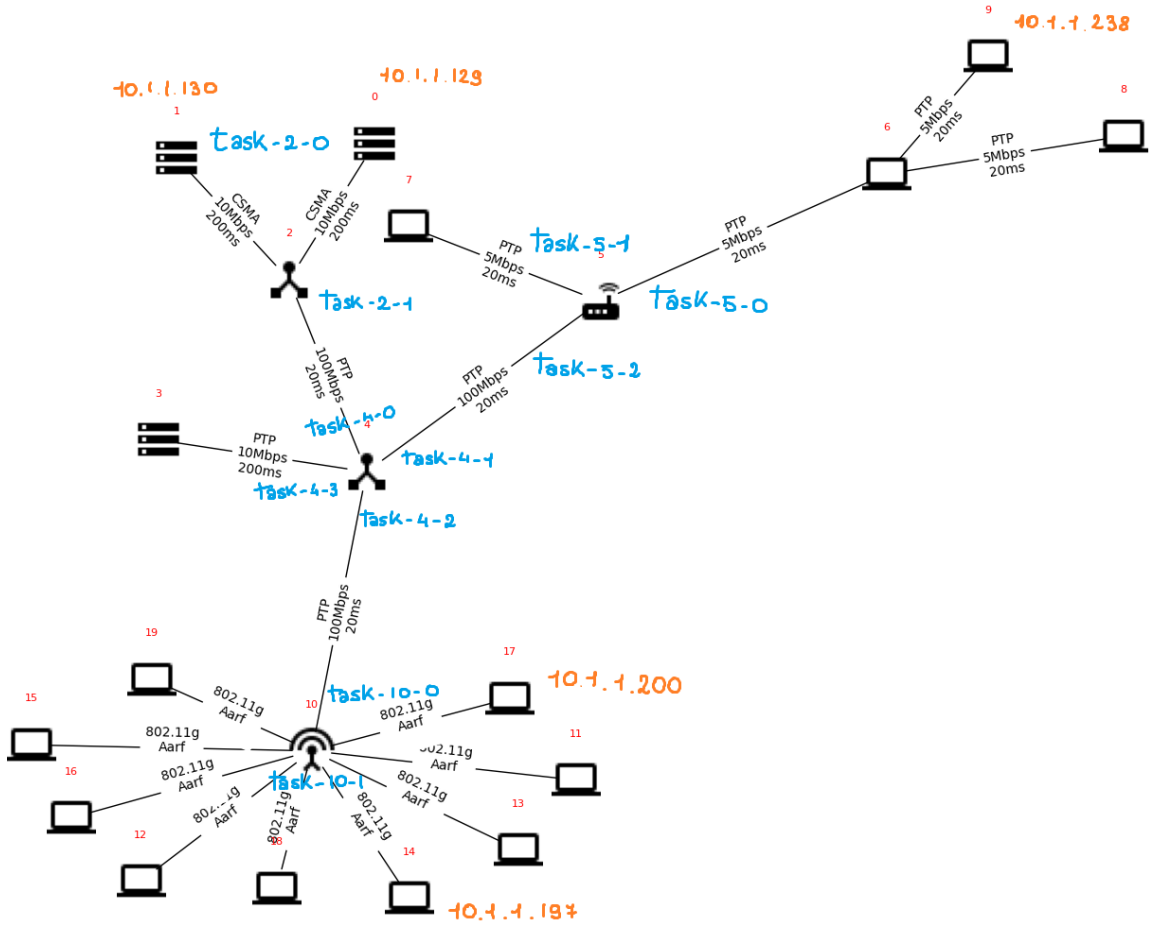

Immagine della topologia, corredata di indirizzi IP di nodi d'interesse e nomi generati per i file .pcap

I nodi sono l'elemento centrale in ns-3: si è dunque deciso di descrivere quattro diverse istanze di NodeContainer, di cui la prima contenente tutti i nodi -in modo da poter mantenere facilmente un'indicizzazione fedele alle istruzioni- e le seguenti i nodi divisi per appartenenza alle diverse topologie. Helper che aiutino a stabilire ruoli interni ai collegamenti tra nodi, come PointToPointStarHelper, sono stati ritenuti tuttavia inadatti, in quanto non sono state trovate situazioni in cui il loro uso non sarebbe risultato forzato. Infatti, come suggerisce la documentazione, le topologie a stella sono generalmente pensate per fornire supporto a nodi centrali collegati a link dalle stesse caratteristiche con molteplici "punte".

Lo strato di rete, ossia il protocollo IP, necessita di interfacce associate ai dispositivi di rete di cui sopra. Si è proceduto con l'assegnazione manuale di indirizzi IP statici, minimizzando la mask e riducendo al minimo il numero di potenziali host della sottorete. A questo punto, ci occupiamo di implementare una categorizzazione dei nodi in Laptop (ie Clients), Server e Router. Nella nostra simulazione, avviene comunicazione fra nodi secondo le seguenti specifiche:

TCP delivery of a file of 1173 MB starting at 0.27 s: Sender: Node 17 - Receiver: Server 0
TCP delivery of a file of 1201 MB starting at 3.55 s: Sender: Node 9 - Receiver: Server 1
TCP delivery of a file of 1837 MB starting at 3.40 s: Sender: Node 14 - Receiver: Server 0
Wi-Fi operating in Ad Hoc mode, stationary devices, no random walk
UDP Echo application with Client 17 and Server 3
Size of packet: 2032 Bytes Periodicity:20ms MaxPackets: 250

La scelta di BulkSenderApplication per la rappresentazione dei mittenti TCP piuttosto che OnOffApplication nasce dalla considerazione che la simulazione voglia porre l'accento su trasmissioni continue e sull'effetto che queste possono avere sullo stato di congestione di reti. Si è scelto di usare porte non note casuali, avendo cura di evitare porte con ruoli specifici, sia per la comunicazione UDP che TCP.

Topologia e ritardi di rete

1. Riprendendo quanto scritto sopra, notiamo all'interno della rete descritta dall'immagine la presenza di diverse topologie:
   i. I collegamenti PointToPoint sono realizzati con l'aiuto di un Helper, che si occupa della creazione del canale e dei dispositivi di rete (PointToPointNetDevice). Il protocollo simulato è un protocollo PPP; Link come questi trovano ampia applicazione in collegamenti semplici fra due entità: "*a common solution for easy connection of a wide variety of hosts"*.
   ii. Abbiamo poi una topologia CSMA. La documentazione sottolinea che la simulazione opera sulla base di un protocollo CSMA/CD che abbia istantaneo *carrier-sense*, mentre i più comuni modelli Ethernet (IEEE 802.3) comporterebbero CSMA/CD con *exponential backoff*.
   iii. L'implementazione della rete Wi-Fi è realizzata in modalità *adHoc*. Questa rete simula un collegamento in cui non esiste un punto centrale (*Access Point*), ma dove ogni nodo comunica con gli altri. Siamo anche in questo caso assistiti da ns-3: d'interesse è WifiHelper per impostare lo standard di rete IEEE 802.11g-2003 (anche detto Wi-Fi 3) per la gestione di WLAN, mentre viene adoperato *Adaptive Auto-Rate Fallback* (AARF) come algoritmo di *rate control*, ie Collision Avoidance.
2. Esaminando le informazioni contenute nel file .txt, possiamo individuare tre diversi flussi di comunicazione TCP fra coppie di nodi. Per tracciare la comunicazione nella sua interezza, ci siamo avvalsi dei file .pcap da noi generati e faremo riferimento a metodologia generale che può essere applicata a tutte le comunicazioni, in quanto la natura delle stesse è pressoché invariata. Filtriamo per ottenere le singole conversazioni con il filtro:

   ```
   ip.src == [IP sorgente] && ip.dst == [IP destinatario]
   ```

   Isolando il traffico possiamo identificare tutte le parti della conversazione e in questo modo tracciare il percorso nella rete dei pacchetti, in particolare il *three-way handshake* e il successivo scambio di pacchetti regolato dagli ACK (che osserviamo essere non cumulativi). Riportiamo il percorso dei pacchetti nella rete:
   i. **Nodo 17** - Router 10 - Router 4 - Router 2 - **Server 0**
   ii. **Nodo 9** - Nodo 6 - Router 5 - Router 4 - Router 2 - **Server 1**
   iii. **Nodo 14** - Router 10 - Router 4 - Router 2 - **Server 0**

   A questo punto, ci siamo resi conto sia dalle catture di Wireshark sia verificando il numero di byte effettivamente arrivati alle PacketSinkApplications, che la comunicazione ii. non avviene laddove contemporaneamente stiano procedendo i. e iii. Abbiamo dunque indagato le cause, e le riportiamo al punto successivo.
3. La comunicazione tra il nodo 9 e il server 1 non avviene. Per verificarne il perché, l'abbiamo isolata e svolta autonomamente, e il problema sembra avvenire solamente quando almeno una delle altre due comunicazioni TCP sta trasmettendo dati. In questo caso, attraverso l'uso del filtro

   ```
   ip.src == 10.1.1.238
   ```

   possiamo isolare solo due tipologie di pacchetti relativi alla comunicazione ii.: un SYN proveniente dal client e diretto al server, e la sua ritrasmissione che avviene dopo un periodo di tempo pari al RTT riportato da Wireshark (3s). Sospettiamo quindi che avvenga un *buffer overflow* relativo al NetDevice montato sul router 2 in corrispondenza del collegamento PointToPoint con il router 4, in quanto non troviamo più traccia del SYN proprio in questo punto, che è dove le comunicazioni TCP iniziano a condividere il percorso. È nostra convinzione che questo sia dovuto alla velocità ridotta dei collegamenti fra i nodi 9-6 e 6-5. Possibili soluzioni del problema potrebbero essere alternare le comunicazioni (tuttavia ciò potrebbe avvenire solo se conosciute a priori) oppure implementare un algoritmo più intelligente per la gestione del buffer del router. ns-3 infatti simula una semplice modalità *drop-tail*, in cui è perso l'ultimo pacchetto ad arrivare al buffer: ciò non permette al Nodo 9 di poter stabilire una connessione con il Server 1, e quindi dopo tre tentativi la comunicazione non avviene.

4. Tutte le valutazioni che sono fatte sul throughput sono relative ai file .pcap generati dal router montato sul nodo 4, in quanto lo si è ritenuto centrale nella comunicazione TCP. Con l'ausilio del pannello "*Packet Lengths*" degli strumenti per la statistica di Wireshark, siamo stati in grado di analizzare il flusso dei pacchetti TCP/IP per tutti i file .pcap ricavati. Ci siamo concentrati sullo studio del collegamento fra il Router 2 e il Router 4, in quanto qui avviene la sovrapposizione dei percorsi di tutte le connessioni TCP. Usando un filtro che ci permettesse di escludere pacchetti arrivati dopo il secondo 2 della simulazione, abbiamo ottenuto che la quantità trasmessa sul collegamento fino a quel momento fosse di 7,304 byte. In quanto la formula del throughput medio è **bit trasmessi / tempo**, abbiamo calcolato un valore di 29,216 bit/s.
5. In maniera analoga a quanto fatto in precedenza, abbiamo ottenuto il numero di byte trasmessi sullo stesso link al secondo 5 (11,968 byte) e lo abbiamo elaborato nella stessa maniera per ottenere il throughput medio a questo istante di tempo, pari a 19,148.8 bit/s. Notiamo che questo valore sia molto minore di quanto abbiamo ricavato al punto 4; ciò è sicuramente dovuto al fatto che, secondo quanto stipulato dalle indicazioni ricevute, abbiamo stabilito che la seconda e la terza trasmissione TCP partissero rispettivamente al secondo 3.55s e 3.40s. Questo porta a una capacità di trasmissione minore e quindi motiva la diminuzione del throughput.

Rete wireless

1. Per quanto riguarda la comunicazione UDP, non è più d'interesse un file di grandezza prestabilita, ma abbiamo invece un numero di pacchetti- 250- che devono muoversi nella rete fino al destinatario. Questi contengono 2032 byte di data ciascuno (costituito dai nomi e dalle matricole dei componenti del Gruppo) e sono inviati ogni 20ms. Non tutti i frame mandati ricevono ACKnowledgment, come abbiamo potuto notare usando il filtro di Wireshark

   ```
   wlan.fc.retry == 1
   ```

   che ci ha permesso di osservare le ritrasmissioni avvenute al link layer della pila protocollare. In questo modo, abbiamo potuto identificare quali fossero i pacchetti che venivano ritrasmessi in quanto non avevano ricevuto ACK.
2. In virtù di quanto scoperto al punto precedente, supponiamo che la causa delle ritrasmissioni siano collisioni all'inizio della sovrapposizione fra la comunicazione TCP e UDP, in quanto i tempi corrispondono. Sulla base della nostra implementazione del modello, non siamo in grado di intuire dalla posizione degli host collegati alla rete wi-fi se potrebbe essersi verificato un *Hidden Node Problem*, ma supponiamo che il modello, in mancanza di specifiche informazioni al riguardo, non lo simuli di default. Nella nostra simulazione, il protocollo RTS/CTS avviene "spontaneamente" solo quando il frame supera i 2347 byte, valore da noi scelto perché è standard, ma notiamo che autonomamente i pacchetti UDP sono sottoposti a fragmentation, il che può aiutare a evitare collisioni in virtù di una minore quantità di dati da trasmettere, al costo però di un maggiore overhead.
3. L'overhead della EchoApplication (sia client che server) è stato calcolato relativamente al file -pcap generato dalla NetDevice wi-fi del router, con la seguente metodologia:
   i. Per stabilire la connessione, il client deve applicare il protocollo ARP per determinare l'indirizzo MAC del router wi-fi. Questo ha un "costo" in bytes pari a 174.
   ii. Consideriamo un totale di 500 pacchetti (250 spediti dall'EchoClient, 250 ritornati al mittente dall'EchoServer). Ognuno di questi pacchetti ha un overhead singolo di 88 byte per gestire le informazioni relative a tutti i layer della pila protocollare, corrispondente quindi a 44,000 byte.
   iii. Trattandosi di una rete wi-fi, ogni pacchetto deve ricevere un ACK di dimensione 36 byte (al netto di eventuali ritrasmissioni, di cui non teniamo conto in questa sede, ma che potrebbero diventare un fattore importante all'aumentare degli attori concorrenti per la banda). Otteniamo dunque 18,000 byte di ACK.

   L'overhead complessivo è dunque di 62,174 byte, che se posto in proporzione con il numero di byte inviati (ricavato attraverso l'operazione 500 pacchetti*2032 byte di data) 1,016,000 può essere letto come 16 byte di data per ogni byte di overhead.
4. Forzando RTS/CTS (diminuendo il numero di byte a cui avviene a 0), notiamo immediatamente che applicando lo stesso filtro prima citato, non otteniamo nessun pacchetto ritrasmesso; questo significa che non sono avvenute collisioni e possiamo dunque concludere che con grande probabilità tutte le perdite osservate sulla rete wi-fi fossero precedentemente causate nella loro interezza da esse.

5. L’overhead complessivo è pari a quello ottenuto precedente con l’aggiunta, per ogni pacchetto inviato o ricevuto e per i frame inviati dal protocollo ARP, di un numero pari a 80 byte (44 di Request-to-send, 36 di Clear-to-send). Questo valore ammonta a 40,040 byte di overhead aggiuntivi, per un totale di 102,234 byte di overhead se RTS/CTS viene forzato. In questa applicazione, riteniamo quindi che in base al numero di frame che vengono ritrasmessi, la forzatura di RTS/CTS porti a un overhead molto maggiore, che rende la comunicazione molto più lenta, con 9 byte di data per byte di overhead trasmessi.