**Progetto SDN:Blocco traffico**

*Scrivere un programma che, da ogni host, generi traffico TCP iperf a intervalli casuali e con durate casuali verso altri host. Scrivere un programma Ryu che osservi i pacchetti SYN. Se una destinazione riceve più di X nuove connessioni nell’ultimo intervallo T la nuova connessione è bloccata. Siano X e T valori arbitrari.*

Obiettivi del progetto:

Lo scopo è osservare i SYN (solo primo pacchetto per ogni connessione TCP). Anche le connessioni non established sono conteggiate, a prescindere dall’esito positivo. È indipendente dalla topologia di rete (anelli, maglie, lineari, a stella …) e dal numero di host, per questo viene usato l’hop by hop Switched Eth (codice di riferimento lab switch5.py). Per la prova usiamo n host, su ciascuno verrà eseguito codice client e server, entrambi funzionanti sul medesimo con uno script per generare traffico casuale. La funzione è eseguita a scopo dimostrativo.

Per semplicità vengono gestiti i pacchetti facendo passare solamente i SYN in IPv4.

Il traffico gestito è unicamente IPv4 unicast per evitare loop sulla rete, viene evitato in questo modo il broadcast storm. Dato che gli host hanno bisogno di conoscere l’indirizzo mac degli altri host e associare l’ipv4 tramite arp (funzionamento in broadcast) è necessario implementare il proxy arp sul controllore.

Se un dato host riceve più di X Syn negli ultimi T secondi il nuovo SYN non verrà inoltrato nella rete, si può verificare la correttezza vedendo le richieste iperf in errore lato client o con TCP dump.

Dato che il progetto funziona anche su connessioni non estabilshed è possibile filtrare un eventuale attacco TCP SYN Flood (un host riceve tanti syn senza che siano vere connessioni, conseguentemente permane nello stato di timeout).

Topologia usata:

La topologia scelta per l’esperimento teorico a scopo di riferimento è una full mesh (maglia completa), ovvero ognuno dei 4 switch è interconnesso direttamente a tutti gli altri. Il numero di collegamenti in una topologia di questo genere è pari a n\*(n-1)/2. Siccome il fattore di dipendenza della complessità della rete (numero di link) varia come n^2/2 al crescere del numero di switch il numero aumenta quadraticamente. Nella nostra topologia sono presenti 4 host connessi ai rispettivi switch diversi. Gli switch implementano OpenFlow mentre il controllore implementa ryu. In vista dell’esperimento in Laboratorio i mac degli host non sono assegnati progressivamente (funzione - - mac), ma tramite proxy arp come specificato sopra.

Funzionamento:

* Il controllore apprende la topologia della rete osservando con il protocollo LLDP (Link Layer Discovery Protocol), apprende i mac address degli host e dove sono connessi, in questo modo ricava il primo tra i cammini minimi. Il passaggio dei dati avviene sui cavi control plane (gialli in lab), i quali hanno lo scopo di controllare la rete e gestire le packet out
* inizialmente tutti i pacchetti vengono inviati direttamente al controllore. I syn possiedono una priorità più alta di tutte le regole
* ricevuti i primi pacchetti si installa la Flowmod per instradare direttamente sul dataplane col percorso minimo calcolato prima
* Quando un host vuole parlare con un altro inizialmente il pacchetto verrà inoltrato al controllore il quale saprà lo switch sul quale inoltrare il pacchetto, lo inoltra (tramite packetout) e comunicherà allo switch il camminino verso quel determinato host tramite una flow mod. In questo modo la prossima richiesta di comunicazione verrà direttamente inoltrata senza passare dal controllore con l’accortezza che i pacchetti syn verranno sempre inoltrati al controllore per la priorità che possiedono.

Struttura Dati:

Si utilizza un dizionario che ha come chiavi i mac di dst, univoci in una rete switch eth (riempiti mano mano dai syn). Un dizionario è una struttura con chiave e valore ed è implementato come una tabella di hesh. Il mac destinazione è usato come indice. (Es. se devo trovare indirizzo mac 2 entro con ind mac senza scorrere l’array ritorna subito le informazioni.) La chiave è dest mac, il valore è la lista dei times temp dei syn, ad ogni syn viene inserito un valore nella lista che costituisce l’orario a cui è stato ricevuto per il dato destination mac.

Si calcola delta t, nel codice implementato tramite while; si scorre la lista per contare il numero di SYN nell’intervallo di tempo delta t fintanto che è inferiore di T e il numero di syn minore uguale ad X. Si fa la differenza fra l’ultimo e il precedente, si rimuovono i times temp tramite un altro ciclo while. Si cancellano se #syn maggiore di X oppure se deta t maggiore di X.

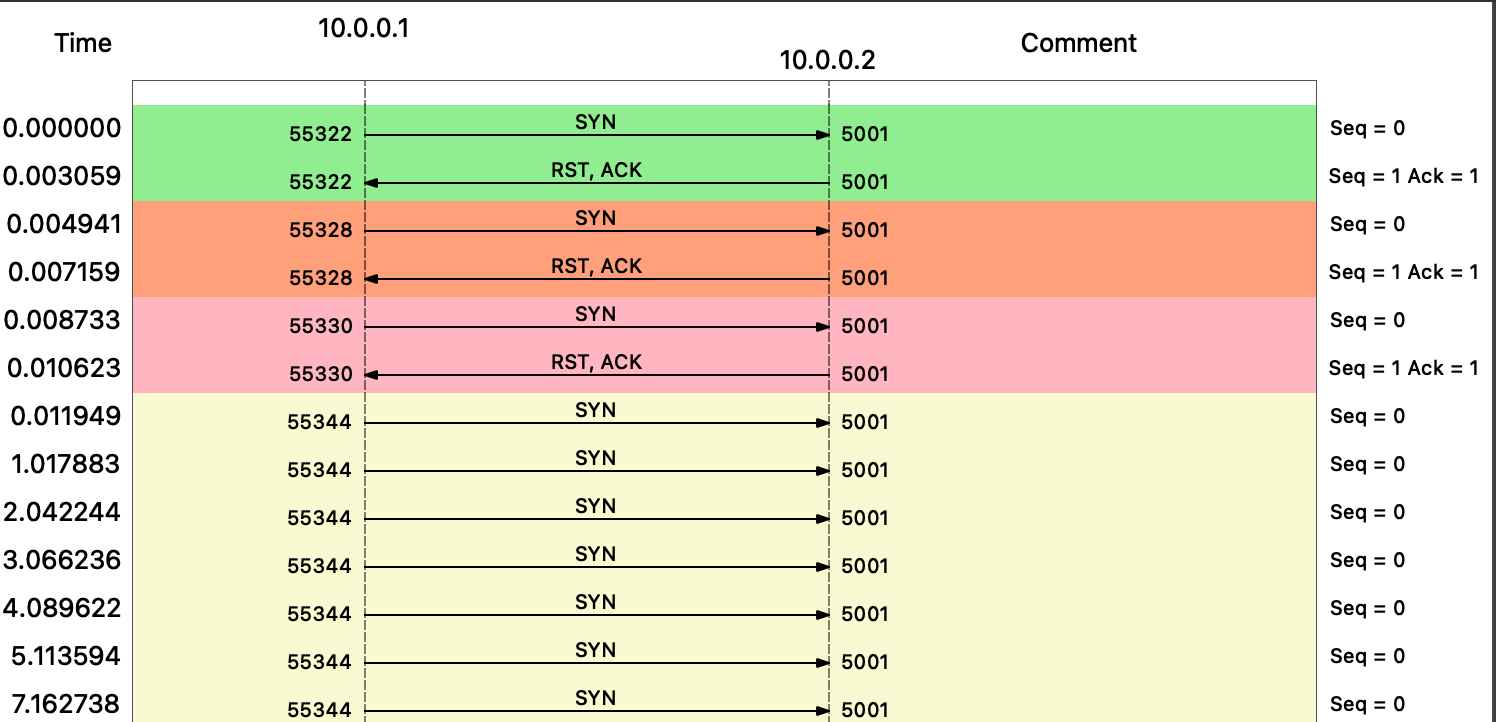
**Script per il traffico:**

Bisogna generare traffico TCP iperf casuale, partono 4 server in contemporanea perché per selezionare la porta comunico gli host di tutta la rete. Per capire su quale mac sono faccio ip -f show, con myip creo la variabile con l’ip su cui sono.

File su cui salvare l’output, aspetta 10 secondi per far partire tutto il traffico. Gli host ip contengono tutte le macchine. Siccome non si vuole mandare traffico iper alla mia macchina tolgo il mio indirizzo host, scorro array se my\_iphost!= iphost\_dst allora si inserisce nel nuovo array, e si elimina l’array creato. SI fa TCP Dump cioè analizzo syn e lo salvo in un file wireshark per vedere cosa accade. Estraggo dal mio ip estraggo l’ultimo byte perché host sequenziali quindi so su quale host sono e posso calcolare la porta target(5200+host) un server iperf può gestire una sola connessione. Nel caso pessimo ad uno stesso host possono arrivare al massimo 3 connessioni da qui la necessita di 4 server (uno per il calcolo delle porte) ma dato che è a singolo thread necessito di 3 porte diverse assegnate per ogni host.

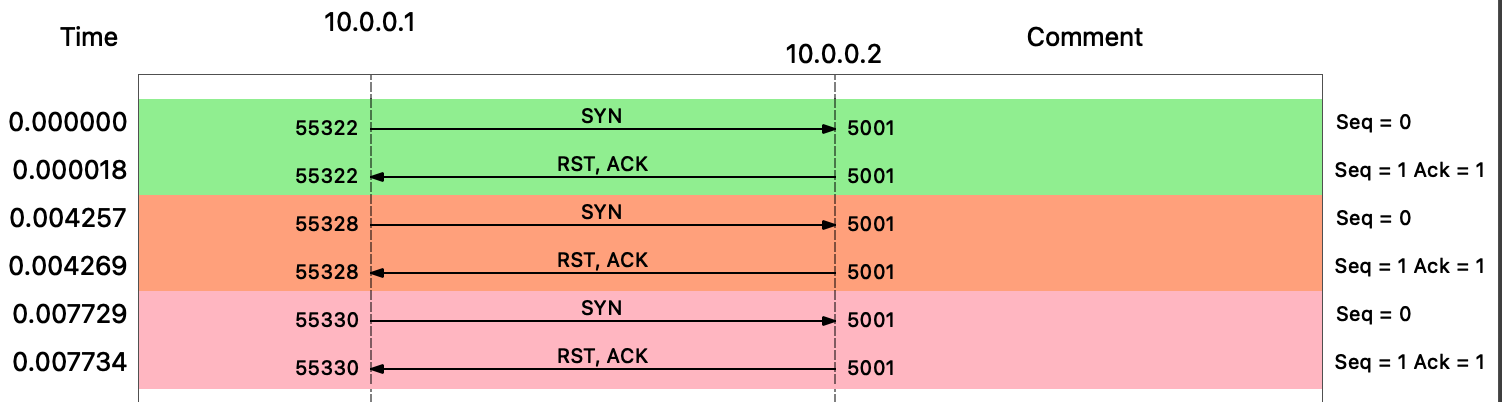
C’è un loop while, per ogni host attesa casuale, durata di connessione casuale con iperf. & permette di far si che prima che termini la connessione, passa avanti ad un’altra. ù

Rule of thumb:

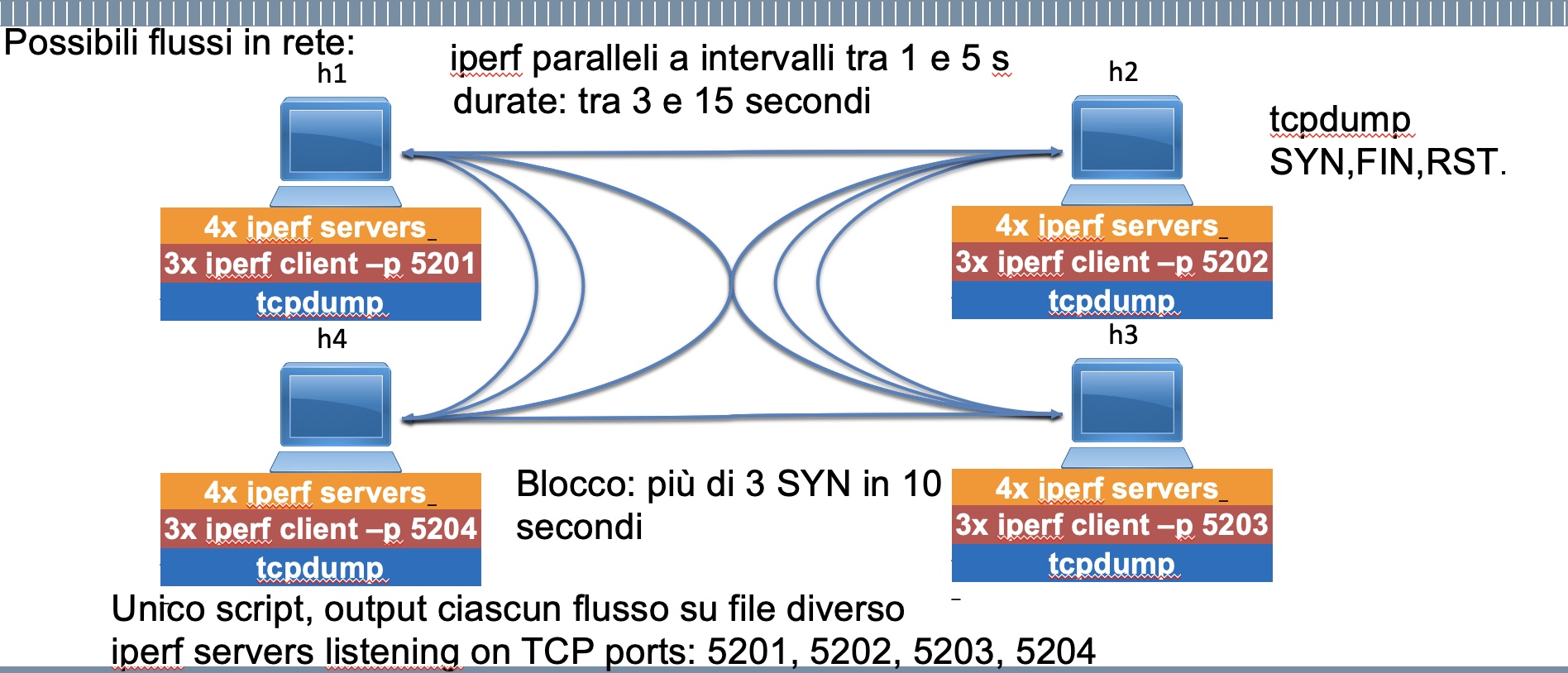
Come dimostrazione iniziale usiamo quello che abbiamo chiamato rule of thumb ossia scegliamo della topologia completa solamente due host. Su h1 poniamo un client che manderà delle richieste di connessione a h2. Su h2 non è presente alcun server, conseguentemente h2 manda un reset di risposta ad ogni syn.

Si aprono 3 SYN in 10 secondi. Quetso è ben visibile in wireshark in ambe le immagini sullo schermo.

h2 riceve 3 syn e risponde con 3 reset, il quarto syn che viene mandato non riceve risposta perché abbiamo impostato X=3, non ricevendo risposta h1 pensa che il pacchetto sia andato perso. Ritrasmette il pacchetto per 6 volte per una implementazione Linux.



Questa immagine rappresenta la risposta di h2, con syn e reset

Dimostrazione:

Considerando la topologia completa

Ogni host genera una connessione con ogni altro host fuorché se stesso. Si generano quindi 3 connessioni per ogni host. Il primo host genera 3 connessioni verso h2-h3-h4. In totale saranno attive al massimo 12 connessioni in parallelo.

Ogni server può gestire contemporaneamente una unica connessione per questo sono presenti 4 server su porte diverse su ogni host.

Come nella topologia semplice viene usata la regola di bloccaggio di 3 SYN in 10 secondi.

La quarta connessione verso h3 viene bloccata perché ne ha ricevute più di 3 negli ultimi 10 secondi indipendentemente dalla sorgente