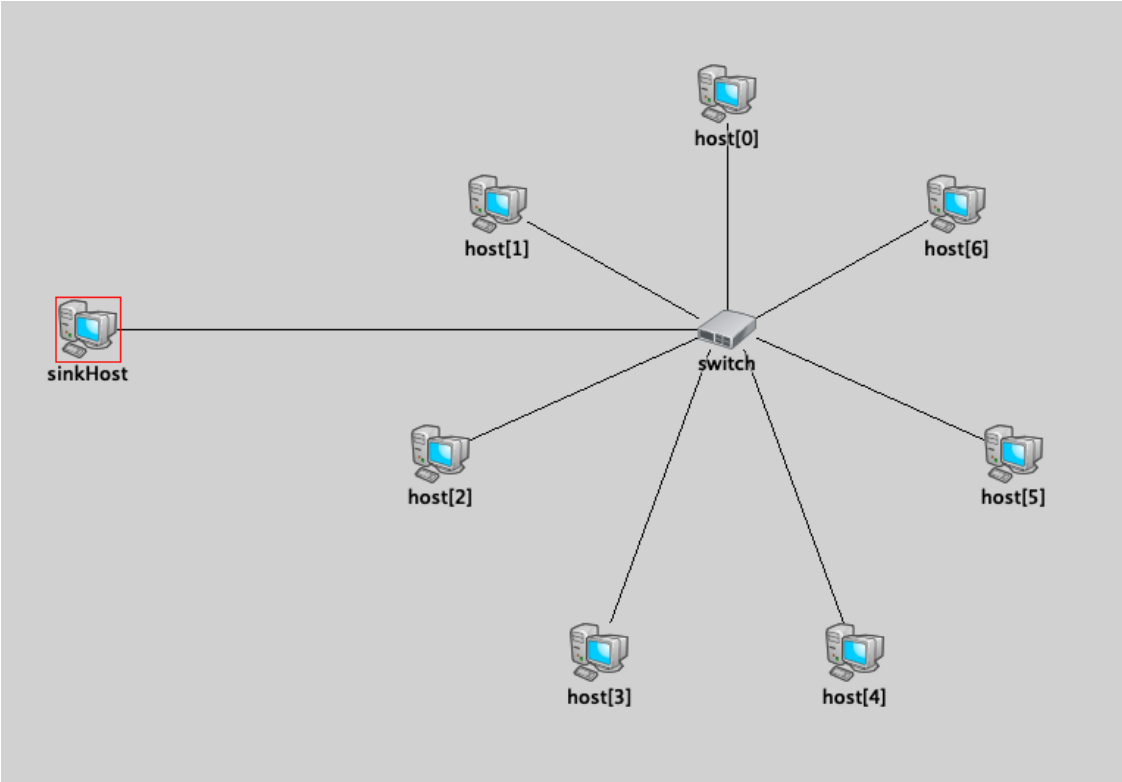
Corso di Reti per l'Automazione Industriale

Elaborato di Omnet++ di Gabriele Costanzo (matr. 1000014221)

Relazione finale – Elaborato E

1. Introduzione



Si vuole simulare lo scenario di una rete Switched Ethernet, a datarate variabile, nella quale 8 host di tipo StandardHost sono collegati fra di loro tramite uno switch centrale: 7 di questi (indicati con ***host[n]***) utilizzano una *UdpBasicApp* a livello applicativo, e in quanto tali invieranno periodicamente pacchetti di lunghezza variabile (fra 100 e 1500 Byte), mentre l'ultimo (indicato con ***sinkHost***) utilizzerà un *UdpSink* e sarà destinatario dei suddetti pacchetti.

Per decidere l'ordine con cui i pacchetti ricevuti dallo switch debbano poi essere inoltrati al *sinkHost*, viene implementato un ***PriorityScheduler*** di *INET*, e si assegnano in maniera statica e parametrizzabile priorità e periodo di invio dei pacchetti per ciascun specifico *host*.

2. Scenario

Gli scenari principali richiesti sono due, dipendenti dalla datarate dei canali utilizzati nella rete: **1Gbps** o **10 Gbps**.

Tuttavia, per condurre un'indagine più approfondita, si è deciso di eliminare la randomicità con cui i messaggi generati possono assumere qualunque lunghezza, considerando anche dei casi *"al limite"* in cui vengano generati o solo messaggi di lunghezza 100 Byte o solo 1500 Byte, portando di fatto a **sei casi** analizzati il totale.

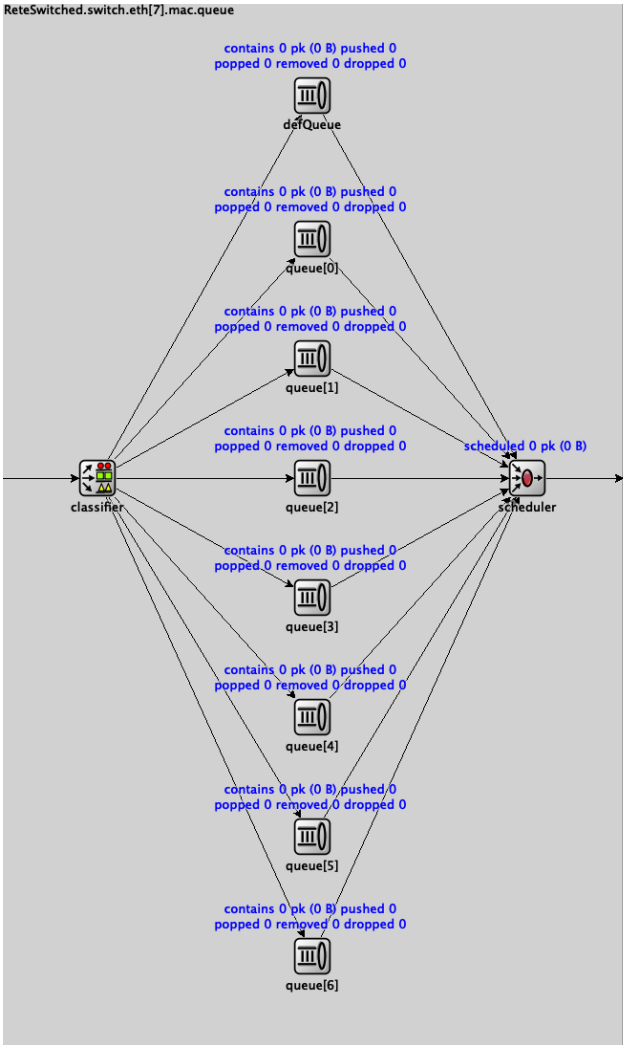
È stato scelto di mantenere statica la priorità assegnata ai messaggi di ciascun host e monotonicamente decrescente, ovvero l'host[0] ha una priorità superiore a quella di host[1], il quale ha una priorità maggiore di host[2], e via dicendo fino a host[6].

I parametri assegnati per designare i periodi di invio di ciascun host sono stati scelti in modo tale da evitare una rapida saturazione delle code pacchetti, tramite due specifici accorgimenti:

• i periodi di invio sono stati scelti per essere inversamente proporzionali rispetto alla priorità, prevedendo tempi di attesa in coda più lunghi per gli host con priorità più bassa

• i periodi di invio sono stati scalati tramite un variabile con funzione di "moltiplicatore" per tenere conto dei diversi datarate e delle diverse lunghezze dei messaggi nei diversi casi.

3. Scelte implementative



Esistono diverse metodologie valide per impostare la priorità dei pacchetti, e in ultima istanza è anche possibile sovrascrivere i comportamenti di code, classificatori e scheduler estendendone le classi in C++.

Dopo un'attenta ricerca nella documentazione di INET, è stato deciso di implementare il meccanismo delle diverse priorità simulando una **rete QoS**, in cui il **DSCP (DiffServ Code Points)** nell'header IPv4 di ciascun pacchetto permetta di definire un diverso servizio.

Sono stati scelti 7 valori di *DSCP*, da 8 (0x08) a 20 (0x14), assegnati rispettivamente dall'host[0] all'host[6].

Per implementare il suddetto meccanismo, si è scelto di creare un nuovo modulo ***"CompoundPriorityQueue"***, che estende la classica *"CompoundPacketQueue"* e prevede un classifier *BehaviorAggregateClassifier*, otto *PacketQueue* e il *PriorityScheduler* richiesto da consegna, e sostituirlo alla classica *EtherQueue* dell'interfaccia Mac nella porta dello switch a cui è collegato il *sinkHost*.

Il *BehaviorAggregateClassifier* controlla il campo *DSCP* e in accordo a quanto detto sopra inoltra il pacchetto ad una di 7 code (indicate come **queue[n]**), mentre una prima coda **defQueue** prende l'output di default del classifier, nel caso in cui non sia presente alcun *DSCP*: è questo il caso dei 7 pacchetti ***ARP*** inviati al *sinkHost* ad inizio simulazione.

L'ordine delle queue rispecchia la priorità dei pacchetti che contengono (con la coda *defQueue* a priorità massima), e lo strict *PriorityScheduler* sfrutta suddetto ordine per decidere l'ordine con cui schedulare i pacchetti.

Per il canale è stato scelto di estendere un *ThruputMeteringChannel* e di imporre il datarate come multipli di 1 Gbps impostando un parametro numerico intero ***"customDatarate"*** a moltiplicare.

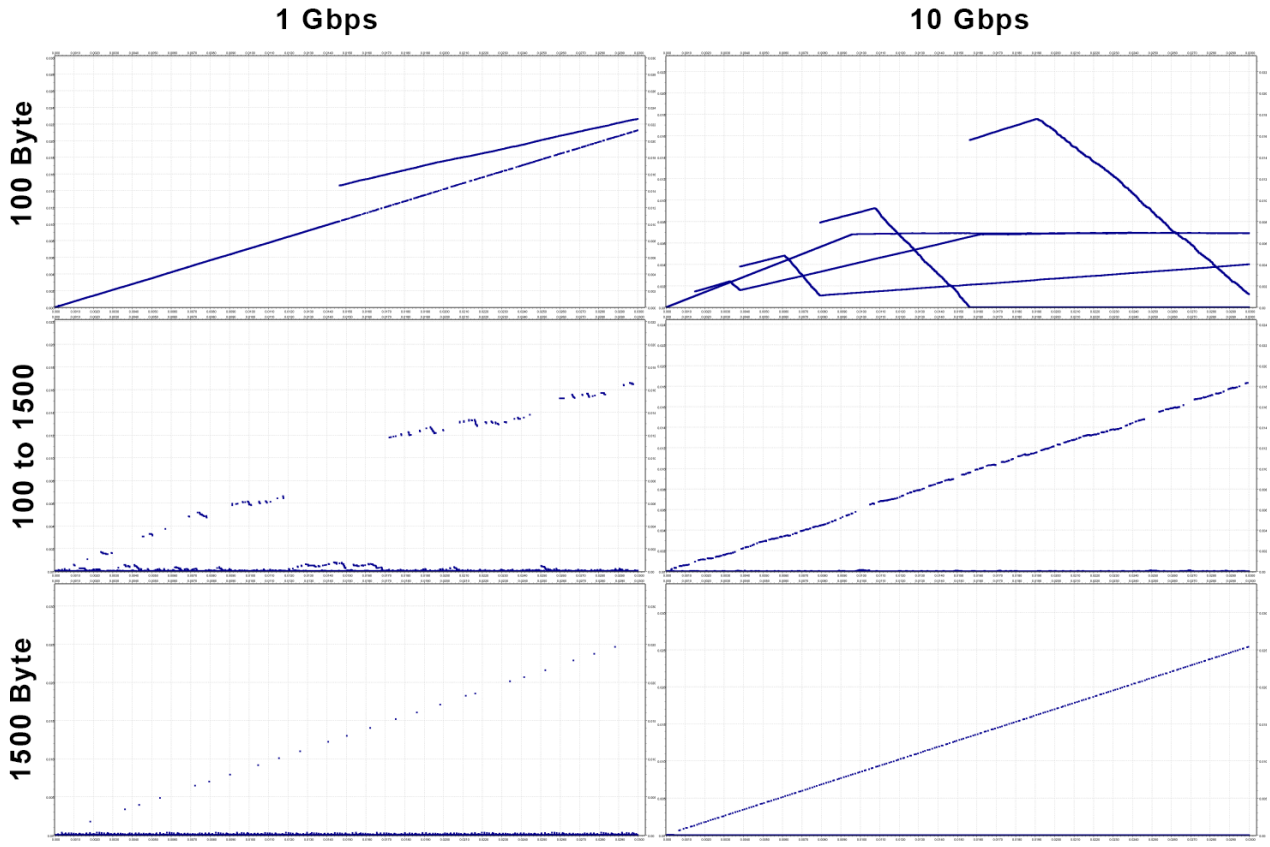
Un altro parametro, ***"multiplier"***, viene utilizzato per scalare i *send interval* dei 7 host pur mantenendo costanti i rapporti fra ciascuno di essi, ovvero host[0] parte da un minimo di 100 ns e host[6] da 700 ns, e tutti essi vengono moltiplicati per lo stesso parametro costante.
Il parametro è assegnato in maniera diversa in ciascuna configurazione, in accordo a lunghezza dei pacchetti e datarate del canale.

4. Statistiche e Risultati

Le statistiche richieste dalla consegna sono *l'End-to-End Delay* e il *Throughput* a livello applicativo, e sono stati valutati sul *sinkHost.*

Per ciascuna delle due statistiche sono stati eseguite delle simulazioni della durata di 30ms, per ciascuna delle sei configurazioni individuati, e quindi sono stati prodotti sei grafici, ordinati per datarate e per dimensione dei messaggi.

4.1. End-to-End Delay



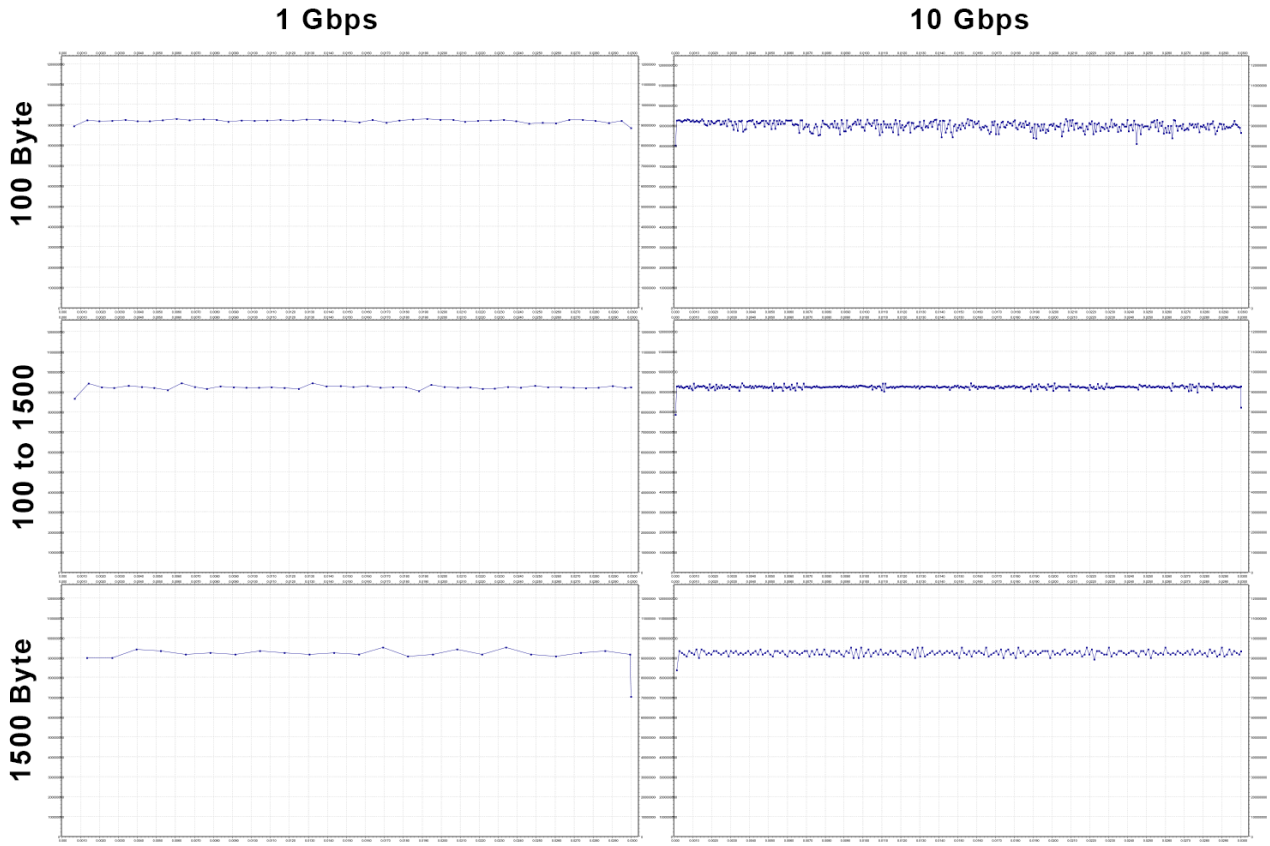
Ad uno dei due estremi, ovvero soli pacchetti di 100 Byte, è stato possibile individuare in maniera chiara ed evidente l'impatto della priorità sul delay end-to-end: nel caso a 10 Gbps, in particolare, si possono individuare almeno 5 curve separate interpolate fra i punti, suggerendo che ciascun host senta effettivamente un diverso delay in base alla sua priorità.

Fatta eccezione per questo grafico – dove i 10 Gbps sono sufficienti per i send interval scelti per evitare un riempimento delle code – in tutti gli altri si può sempre evidenziare lo stesso comportamento: la maggior parte dei punti sono interpolati su una retta a valore approssimatamente costante, mentre altri si distaccano e sono posti su un'ipotetica retta a valore linearmente crescente.

Questo fenomeno è dovuto al fatto che – per i send interval scelti – i pacchetti arrivano ad accumularsi nelle code anziché essere immediatamente inviati, e le code a priorità più bassa attendono un periodo più lungo prima di essere scelti dallo scheduler.

Una prima supposizione vorrebbe che – dato un ben preciso tempo di simulazione – dovrebbe essere possibile individuare 7 rette separate, al pari della prima considerazione del capitolo; tuttavia, senza applicare alcuna modifica alle code nella loro attuale implementazione è probabile incorrere in una loro saturazione molto prima di ottenere l'effetto desiderato.

4.2. Throughput



Il throughput si mantiene tendenzialmente costante lungo tutto il periodo di simulazione, e l'unica considerazione degna di nota da effettuare sui grafici è che in corrispondenza di un maggiore datarate si può riscontrare anche un jitter maggiore, indipendentemente dalla lunghezza dei messaggi.