Esercitazione su Networked Control Systems

Riccardo Muradore, Davide Quaglia, Luisa Repele

June 4, 2013

1 Introduzione

Scopo di questa esercitazione è introdurre i concetti di base dei Networked Control Systems (NCS), una innovativa modalità di simulazione di questi e una metodologia innovativa di progetto di NCS basata sulle reti DiffServ. Questa trattazione sarà lo spunto per esaminare nel dettaglio il fenomeno della congestione.

2 Networked Control System

Un Networked Control System (NCS) è un sistema distribuito e controllato in cui il controllore è collegato al sistema da controllare (detto anche impianto) mediante una rete digitale a pacchetto che può essere cablata (ad esempio fieldbus, ethernet) oppure senza fili (ad esempio WiFi, UMTS, onde sonore). Lo schema a blocchi tipico di un NCS è quello mostrato in Figura 1 in cui u è il flusso dei valori di comando da dare all'impianto, y è la grandezza che si vuole controllare in uscita dall'impianto, r è l'andamento desiderato per y, ed e è l'errore tra andamento desiderato e andamento reale. I valori campionati di u (comandi) e y (misure) sono inseriti in pacchetti e trasferiti sulla rete.

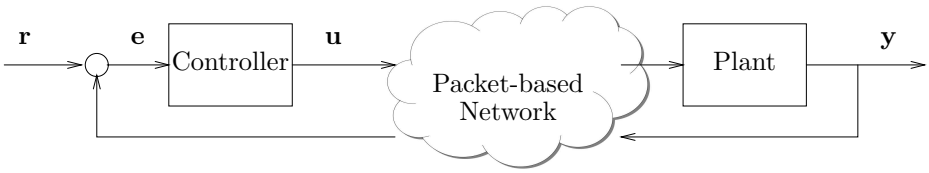


Figure 1: Schema a blocchi di un Networked Control System.

I *vantaggi* di questo approccio sono l'utilizzo di una rete pre-esistente per collegare controllore e impianto e l'eventuale condivisione di tale rete con altre applicazioni (ad esempio, altre coppie controllore-impianto, supervisione dell'impianto mediante trasmissione audio/video).

Gli *svantaggi* derivano dalla condivisione del canale di comunicazione e sono principalmente:

- ritardi non noti a priori e variabili nel tempo;
- perdite di pacchetto di entità non nota a priori e variabile nel tempo;

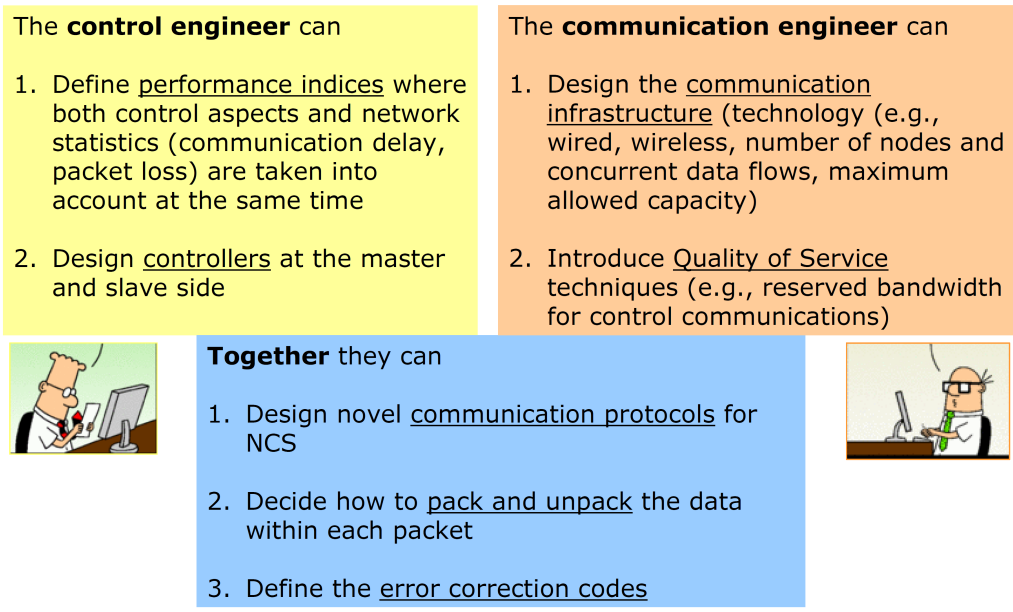


Figure 2: Progettazione congiunta

- nel caso di trasmissione senza fili, errori sul bit di entità non nota a priori e variabile nel tempo.

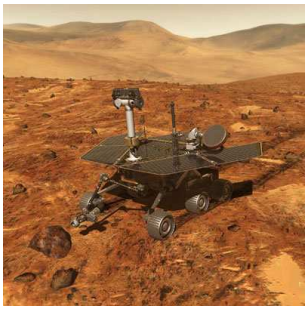
La stabilità e le performance di questi tipi di sistemi sono fortemente influenzati da questi problemi. Essi influenzano sia la trasmissione dei comandi da controllore a impianto sia la trasmissione delle misure da impianto a controllore. Scopo quindi della ricerca in questo campo è il raggiungimento di certe prestazioni di controllabilità dell'impianto nonostante la presenza delle suddette problematiche di trasmissione. Questo scopo finora è stato raggiunto considerando la rete nel caso peggiore e agendo unicamente sul progetto del controllore. In questa esercitazione si vedrà un approccio innovativo in cui si considera anche la rete come dimensione dello spazio di progetto.

Solitamente chi progetta i sistemi di controllo non si occupa della parte di comunicazione che considera come data. Analogamente il progettista di reti non si interessa delle particolari applicazioni che utilizzeranno la rete ma cercherà di progettare un canale ragionevolmente buono per tutti i tipi di processi. Sfortunatamente due "ottimi locali" (lato controllo e lato rete) non sempre raggiungono l'ottimo globale. Per questo motivo è utili che i due tipi di progetto siano integrati e portati avanti contemporaneamente (Figura 2).

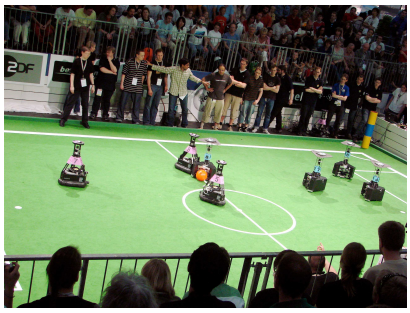
3 Applicazioni dei Networked Control System

Le applicazioni che fanno parte dei NCS sono svariate. Alcuni esempi:

- **Rover**: I rover sono dei veicoli per l'esplorazione della Luna e di Marte. Sono dotati di una certa autonomia decisionale (e.g. collision avoidance) ma per alcuni compiti richiedono comandi inviati dalla Terra (e.g. strumenti di analisi, percorsi). Il round-trip-time per le comunicazioni Marte-Terra varia dai 9 minuti ai 50 minuti.



(a) Rover.



(b) Robocup.

Figure 3: Applicazioni di NCS

- **Robocup**: lo scopo di Robocup è lo sviluppo di multi-robot cooperativi and sistemi multi-agente operanti in ambiente avverso (la squadra avversaria).
- **Unmanned Aircraft**: veicoli aerei comandati a distanza per scopi ludici o meno...
- **Italian UAV display team**: formazione di veicoli che devono comunicare tra di loro per mantenere una certa formazione sequendo un aereo leader (come gli stormi).

(a) Italian UAV display team.

(b) Autonomous underwater vehicle AUV.

Figure 4: Veicoli autonomi.

- **Autonomous underwater vehicle AUV**: Veicoli comandati a distanza in cui il mezzo di trasmissione ha una velocità di propagazione dei segnali relativamente bassa rispetto al caso wireless tradizionale.
- **Teleoperation**: Un sistema di teleoperazione (bilatera) consiste di un haptic device (master robot), di una rete di trasmissione e di un robot slave. L'operatore agisce sul master e lo slave deve rispondere in maniera appropriata. Particolare attenzione va data al caso in cui lo slave interagisce con l'ambiente.

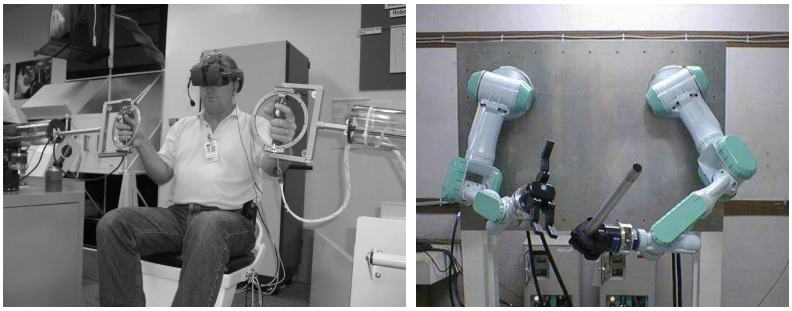


Figure 5: Teleoperazione.

- **Robotic Surgery**: La chirurgia robotica è un caso particolare di teleoperazione in cui l'accuratezza dei movimenti deve essere massima.

Figure 6: Chirurgia robotizzata.

4 Simulazione di NCS

4.1 Co-simulazione

La simulazione è una metodica consolidata per verificare il comportamento di un sistema in cui è stato introdotto un controllore. Uno degli strumenti più utilizzati per la simulazione di sistemi è Matlab/Simulink. Un problema cruciale nella simulazione di NCS è la necessità di riprodurre il funzionamento anche della rete a pacchetto. Matlab/Simulink non dispone di elementi che permettano la modellazione fedele di una rete a pacchetto in cui diverse sorgenti di traffico condividono la stessa capacità trasmissiva. Esistono simulatori di rete dedicati a questo scopo che tuttavia non consentono un'agevole simulazione dei blocchi controllore e impianto e la generazione di grafici. Questo limite ha portato allo studio di una tecnica innovativa per utilizzare in modo combinato e sincronizzato

- Matlab/Simulink per la simulazione di controllore e impianto e la generazione dei vari grafici dei risultati
- un'estensione del SystemC (chiamata SCNSL) per la simulazione della rete in cui vari flussi trasmissivi competono per l'utilizzo del canale.

Questa tecnica, chiamata co-simulazione, permette di utilizzare differenti tool per simulare aspetti diversi di un unico sistema. Lo scopo è di utilizzare il miglior strumento per ogni aspetto del problema. Siccome gli strumenti devono lavorare in maniera sincrona per avere risultati affidabili, il meccanismo di scambio dei dati è l'aspetto più importante.

La Figura 7 mostra come sono utilizzati tali simulatori rispetto allo schema a blocchi di un NCS.

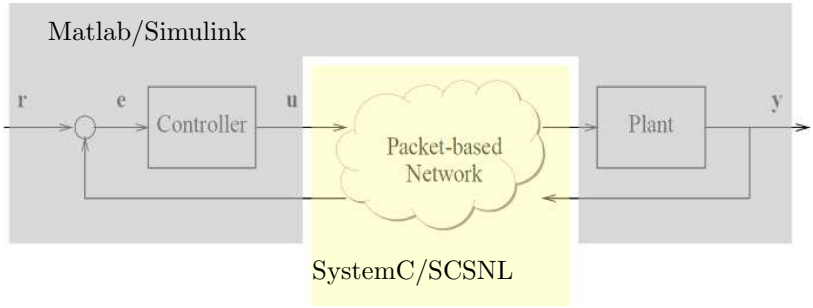


Figure 7: Utilizzo combinato di diversi simulatori per lo studio di un NCS.

4.2 SystemC Wrapper

Per poter far lavorare insieme Matlab/Simulink e SystemC/SCNSL si è dovuto sviluppare un nuovo blocco Simulink chiamato SystemC Wrapper. In questo modo i due tool comunicano via socket bloccanti garantendo il sincronismo tra i tempi di simulazione nei due strumenti. Lo schema concettuale è riportato in figura 8; si può vedere che ingressi e uscite tra Wrapper e altri elementi del modello Simulink sono mappati su aree di memoria nel modello SystemC e quindi nei task modellati in SCNSL.

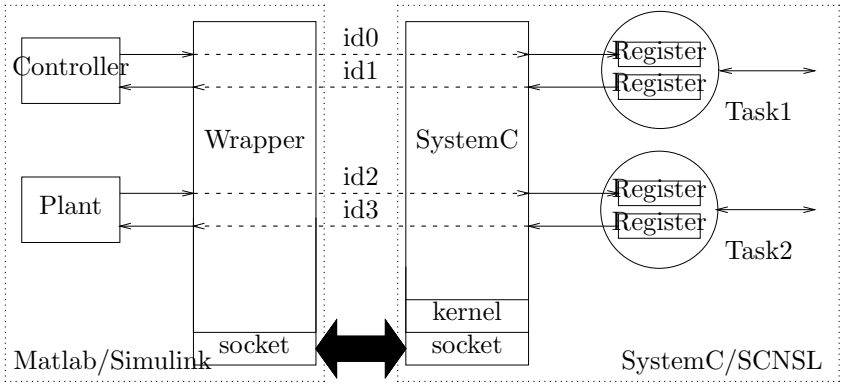


Figure 8: Collegamento tra SystemC Wrapper e SystemC

4.3 SCNSL e Task Wrapper

SCNSL (SystemC Network Simulation Library) è un tool scritto in SystemC che permette di modellare scenari di rete nel quale vi sono diversi tipi di nodi, oppure diversi nodi sono descritti a diversi livelli di astrazione (TLM/RTL), interagendo tra loro. È un tool altamente modulare, infatti gli utenti possono definire il proprio task a seconda delle proprie esigenze. L'uso del linguaggio SystemC porta il vantaggio che Hardware, Software e Network possono essere uniti per progettare e validare sistemi embedded. Il simulatore è disponibile al sito `http://sourceforge.net/projects/scnsl`.

Per collegare SCNSL al tool di Matlab è necessario utilizzare un Task speciale (TaskWrapper) in grado di mettere in comunicazione i due diversi tool. Il task in questione dovrà essere un task TLM con alcune caratteristiche importanti.

Il TaskWrapper ha una prima porta (*_start_send*) d'input di default necessaria al task stesso a capire quando un nuovo dato è arrivato da Matlab. Questa porta non deve essere specificata all'interno del costruttore.

Le altre porte necessarie alla comunicazione (porte d'input e porte d'output) devono essere specificate al momento della costruzione come ad esempio

```
SimulinkTask_t t1( "_SimulinkTask1", 1, n1, ch1,
                   "ISS_ID0", 0x1, 0x1195, 3, 4);
                                           ^  ^
```

Questo esempio mostra la creazione del task dove vengono specificate 3 porte d'input definite dall'utente (+ 1 *_start_send* di default) e 4 porte d'output. Inoltre bisogna anche specificare l'identificatore univoco delle porte usato nella associazione con SystemC Wrapper di Matlab (id0, $\cdots$, in Figura 8).

Il taskWrapper deve implementare le due funzioni per la scrittura (*writingProcess()*)e per la ricezione dei dati (*b_transport*()), come un normale task TLM.

La *writingProcess()* legge i dati dalle porte d'input del wrapper, li mette in un pacchetto e lo invia nella rete. La funzione *to_systemc*() è necessaria per convertire il dato dal formato di Matlab (long long) al formato double.

Per semplicità riportiamo qui un semplice esempio di codice:

```
void SimulinkTask_t::writingProcess() throw(){
   double buffer1;
   double data;
   double label;
   data_packet_t dp;

   while( true ){
       wait( _data_arrived );

       // Read from _start_send port:
       _start_send.read( & data );
       systemc_start_send = to_systemc( data );

       // If _start_send value is changed it
       // means that a new value is coming from Matlab.
       if( ! ( _old_start_send > systemc_start_send ||
              _old_start_send < systemc_start_send ) )
           continue;

       // Storing the old value of the _start_send port.
       _old_start_send = systemc_start_send;

       // Read the data
       _input_data[ 0 ].read( & data );
       buffer1 = to_systemc( data )

       dp.data = static_cast< double * >
               (::operator new[] ( sizeof( double )));
       dp.data[ 0 ] = data;

       Scnsl::Core::byte_t * buff =
         static_cast< Scnsl::Core::byte_t * >( malloc ( PACKET_SIZE ) );
       memcpy( buff, & dp, sizeof( data_packet_t ) );

       //priority of packet
       label = 1;

       // Calling the send method.
       TlmTask_if_t::send( 0,
             reinterpret_cast< Scnsl::Core::byte_t * >( buff ),
             static_cast< size_t >( PACKET_SIZE ),
             static_cast< label_t >( label ) );
   }
}
```

La funzione *b_transport*() una volta ricevuto il pacchetto, lo scompatta e scrive nelle porte Matlab le informazioni necessarie.

```
void SimulinkTask_t::b_transport( tlm::tlm_generic_payload & p, sc_core::sc_time & t ){
   Scnsl::Core::data_extension_t * ext;
   double data;
   double matlab_label;
   unsigned int label = 0;
   data_packet_t * dp;

   if( p.get_command() == Scnsl::Tlm::PACKET_COMMAND )
   {
       p.get_extension( ext );
       label = ext->label;
       dp = reinterpret_cast< data_packet_t * >
                           ( p.get_data_ptr() );

       // data
```

```
        data = to_systemc( dp->data[ 0 ] );
        _output_data[ 0 ].write( & dp->data[ 0 ],
                                 sizeof( double ) );

        // label
        matlab_label =
            to_systemc( static_cast< double >( label ) );
        _output_data[ 1 ].write( & matlab_label,
                                   sizeof( double ) );

    }
    else if( p.get_command() == Scnsl::Tlm::CARRIER_COMMAND )
    {
        // Discarded
    }
}
```

5 Esercitazione

5.1 Installazione

I passi per installare la libreria SCNSL e i test utilizzati in questa esercitazione sono qui sotto riportati:

- Scaricare e scompattare il pacchetto *NCS-diffserv.zip* all'interno della directory /tmp.
- Per la compilazione della libreria SCNSL: prima di tutto eseguire lo script contenuto nella directory /tmp/NCS-diffserv/trunk/scripts/env-setup.sh per l'esportazione delle variabili d'ambiente necessarie alla compilazione

  ```
  >> . env-setup.sh
  ```

- successivamente posizionarsi nella directory /tmp/NCS-diffserv/trunk/obj ed eseguire

  ```
  >> cmake ..
  >> make
  >> make install
  ```

- Per la compilazione dei test: posizionarsi nella directory /tmp/NCS-diffserv/trunk/tests/obj ed eseguire ancora:

  ```
  >> cmake ..
  >> make
  >> make install
  ```

- Spostare i files *go.sh*, *start.sh* e *plr.sh* all'interno della direcotry /tmp/scnsl/bin che conterrà tutti gli eseguibili appena generati con la compilazione dei test.

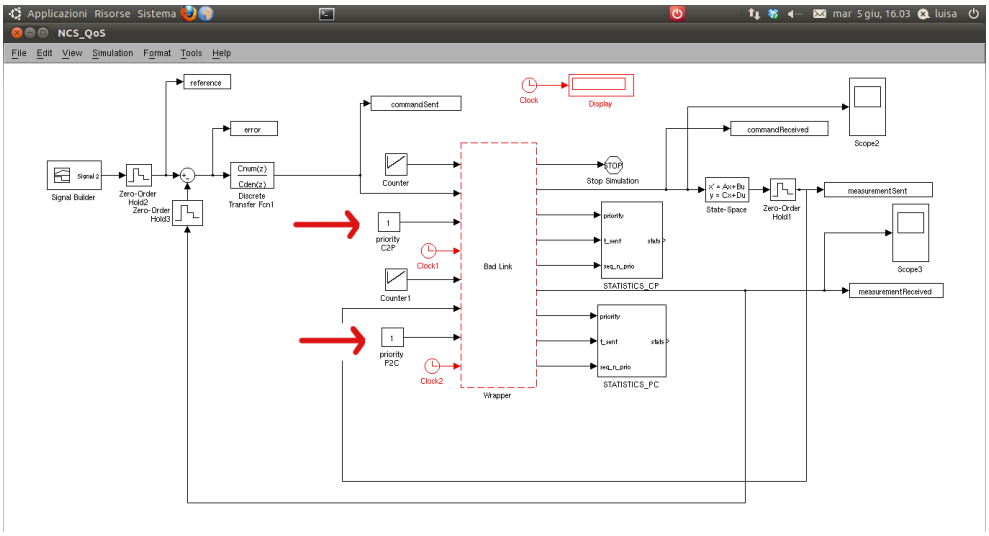


Figure 9: Schematico Matlab

- Per eseguire la simulazione lanciare lo script ./start.sh

```
>> ./start.sh
```

5.2 Descrizione file lato Matlab

Nella directory $/tmp/NCS - diffserv/matlab/NCS\_QoS/$ si trovano i modelli Matlab-Simulink del controllore, impianto e SystemC wrapper mostrati in Figura 8 e i file necessari alla configurazione di tali modelli. In particolare vediamo:

- Il file *NCS_main.m* contiene tutte le variabili necessarie a controllore e impianto per funzionare correttamente. Esso deve essere sempre eseguito all'inizio della simulazione tramite il comando lanciato dalla command-Window di Matlab.

  ```
  >> run('NCS_main')
  ```

  .

- Il modello *NCS_QoS.mdl* presenta uno schema dove i pacchetti che attraversano la rete tra controllore e impianto e viceversa hanno sempre la stessa priorità scelta dall'utente all'inizio della simulazione (per far partire la simulazione: menu Simulation → Start). La Figura 9 mostra lo schematico Matlab in questione, le frecce rosse indicano la priorità dei pacchetti in entrambe le direzioni. Il valore 1 indica priorità bassa mentre il valore 0 indica priorità alta. Per modificare tale priorità basterà modificare il blocco *priority_C2P* o *priority_P2C* (doppio click per modificare)

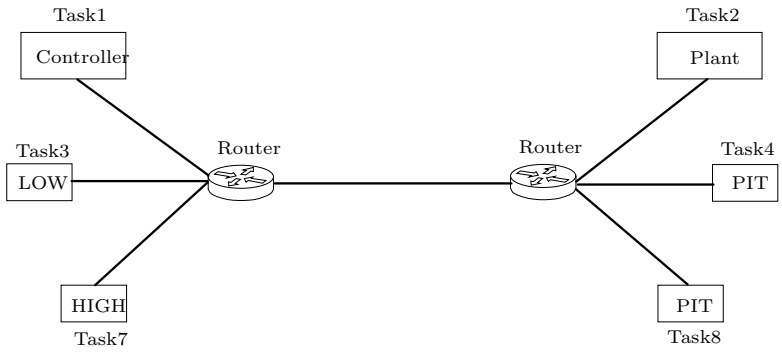


Figure 10: Topologia di rete

- Il modello *NCS_QoS_marker.mdl* invece gestisce la politica di classificazione dei pacchetti tramite un marker che verrà spiegato in dettaglio nella Sezione 6. L'etichetta dei pacchetti non è fissa e decisa a inizio simulazione ma varia a run-time durante la simulazione.

- Il file *plotFigure.m* stampa il tracking error della simulazione, le statistiche di rete tra controllore e impianto (delay e packet loss rate) e nel caso si utilizzi il modello con il marker visualizza la priorità con la quale sono stati inviati i pacchetti durante la simulazione. Anche in questo caso per eseguire tale file basterà eseguire il comando run nel commandWindow di Matlab.

5.3 Scenario di rete

In Figura 10 è mostrata la topologia della rete modellata nel simulatore SCNSL; essa rappresenta una classica topologia bottleneck con link periferici ad alta capacità dedicati per le varie applicazioni che insistono su un link condiviso in cui possono crearsi *fenomeni di congestione*. Tutti i link sono bidirezionali. Si notino i flussi informativi che condividono lo stesso canale. Il traffico concorrente (task 3 e task 7) è di tipo constant bitrate (CBR) con andamento ON/OFF in entrambe le direzioni.

L'idea innovativa che stanno sviluppando gli autori è quella di usare una rete Differentiated Services (DiffServ) sopra la topologia mostrata al fine di dare priorità ai pacchetti relativi al NCS.

Per semplicità di modello sono stati accorpati in un'unica entità sia gli host che generano e ricevono i dati sia gli edge router che effettuano policing e marcatura dei pacchetti. In particolare vedremo che una tecnica di marcatura terrà conto delle politiche di controllo e quindi dovrà essere effettuata direttamente nell'host (sarebbe come dire che l'edge router delega l'host alla marcatura).

I core router gestiscono la capacità del bottleneck mediante uno schema *a due priorità* e quindi con due code.

Il task 3 e il task 7 infatti generano traffico rispettivamente con priorità bassa (etichetta 1) e priorità alta (etichetta 0).

Lo scenario preso in considerazione per questa esercitazione si trova nella directory

$/tmp/NCS - diffserv/trunk/tests/test\_bottleneck\_matlab/$.

Nel file *DefineList.hh* nella cartella $/tmp/NCS-diffserv/trunk/test/test\_bottleneck\_matlab$ sono riportati i parametri di simulazione usati per questa topologia:

- $PACKET\_SIZE$ dimensione del pacchetto in byte.
- $QUEUE\_SIZE$ dimensione delle code in byte.
- $DELAY$0.0005 ritardo in secondi di tutti i link
- $CAPACITY$ capacità dei link periferici in in bit/sec
- $CAPACITY\_BOTTLENECK$ capacità del link condiviso (bottleneck) in in bit/sec
- $CONCURRENT\_TRAFFIC\_BITRATE\_H$ bitrate del traffico concorrente generato dal task 7 in bit/sec
- $CONCURRENT\_TRAFFIC\_BITRATE\_L$ bitrate del traffico concorrente generato dal task 3 in bit/sec
- ON_L periodo di ON del task 3 in ms,
- ON_H periodo di ON del task 7 in ms,
- OFF_L periodo di OFF del task 3 in ms,
- OFF_H periodo di OFF del task 7 in ms,

Il file *main.cc* serve per configurare la rete bottleneck come in figura 10. Per attivare i task che generano traffico concorrente per entrambe le priorità basterà chiamare la funzione enable() come mostrato alla riga 474 del file *main.cc*.

6 Architettura NCS basata su DiffServ

Il problema della classificazione dei pacchetti si basa su due aspetti:

1. la stima dell'importanza del pacchetto (comando o misura),
2. l'allocazione delle risorse (uso di alta/bassa priorità).

Per valutare "l'importanza" di un pacchetto abbiamo bisogno della conoscenza di un modello dell'impianto, del controllore e dello stato della rete (ritardo di comunicazione e packet loss rate).

Lo schema in Figura 11 presenta l'architeettura NCS basata su DiffServ.

- **CMarker**: marcatore lato controllore per decidere la politica di invio dei comandi.
- **PMarker**: marcatore lato impianto per decidere la politica di invio delle misure.
- τ_{CP}^H, τ_{CP}^L, p_{CP}^H, p_{CP}^L and τ_{PC}^H, τ_{PC}^L, p_{PC}^H, p_{PC}^L: ritardi di comunicazione e packet loss rates. Questi segnali dipendono dallo stato della rete.
- **CReceiver**: ricevitore lato controllore. Il ricevitore calcola le stime dei parametri di rete $\mu = \{\hat{\tau}_{PC}^H, \hat{\tau}_{PC}^L, \hat{p}_{PC}^H, \hat{p}_{PC}^L\}$ da inviare al PMarker.
- **PReceiver**: ricevitore lato impianto. Il ricevitore calcola le stime dei parametri di rete $\nu = \{\hat{\tau}_{CP}^H, \hat{\tau}_{CP}^L, \hat{p}_{CP}^H, \hat{p}_{CP}^L\}$ da inviare al CMarker.

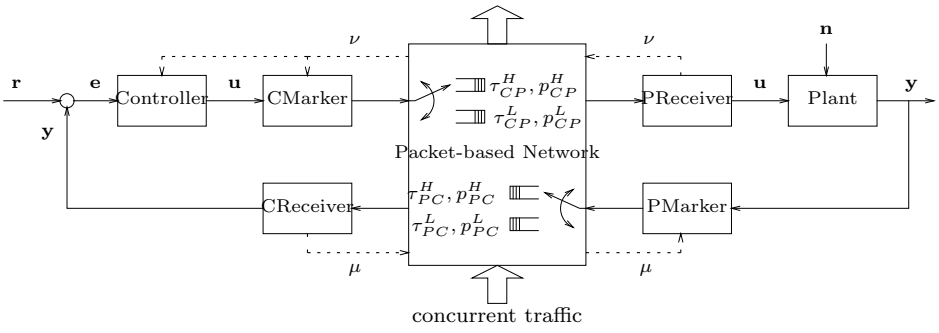


Figure 11: Diagramma a blocchi di un NCS con utilizzo di DiffServ.

Algoritmo per il CMarker

La scelta della politica al passo corrente π_k viene decisa dopo aver eseguito i passi:

1. calcolare la stima dell'uscita assumento la trasmissione del comando usando la priorità H, $\hat{\mathbf{y}}_{get}^H$, (nella stima si usa $\hat{\tau}_{CP}^H$);
2. calcolare la stima dell'uscita assumendo la trasmissione del comando usando la priorità L, $\hat{\mathbf{y}}_{get}^L$, (nella stima si usa $\hat{\tau}_{CP}^H$);

3. calcolare la stima dell'uscita assumento la perdita del pacchetto, $\hat{\mathbf{y}}_{lost}$, (in questo caso si assume di applicare di nuovo il comando ricevuto per ultimo);

4. si pesano le stime con la probabilità di perdita derivata dalle statistiche sul packet loss rate.

$$\begin{aligned} \hat{y}^H(k) &= (1-\hat{p}^H_{CP}(k))\hat{y}^H_{get}(k) + \hat{p}^H_{CP}(k)\hat{y}_{lost}(k) \\ \hat{y}^L(k) &= (1-\hat{p}^L_{CP}k))\hat{y}^L_{get}(k) + \hat{p}^L_{CP}(k)\hat{y}_{lost}(k) \end{aligned}$$

5. si calcola la differenza tra le due stime precedenti

$$\hat{e}(k) = \hat{y}^L(k) - \hat{y}^H(k)$$

6. si utilizza una soglia per decidere la priorità di invio;

```
if |ê(k)| > E
      then π_k = H
      else π_k = L
```

dove E è un parametro deciso dal progettista e dipende dall'applicazione.

7 Esercizi

1. Lanciare una simulazione in cui sia il traffico NCS che la sorgente concorrente sono sempre in bassa priorità (etichetta 1). Occorre abilitare il task 3 ma non il task 7, e impostare nel modello Simulink *NCS_QoS* il blocco *priority_C2P* e *priority_P2C* a 1. Visualizzare l'andamento di packet loss rate e ritardo dei comandi e andamento del tracking error.

   - qual è il ritardo minimo dei comandi?
   - qual è il ritardo massimo dei comandi?
   - qual è la percentuale massima di comandi persi?
   - perché il ritardo dei comandi ad un certo punto inizia a crescere? per quanto tempo cresce?
   - usando lo script plr.sh in */tmp/scnsl/bin/* calcolare il packet loss rate complessivo del traffico concorrente;
   - qual è il bitrate del traffico generato da controllore a impianto considerando che il tempo di campionamento del sistema è $T_s = 0.02$ secondi e la dimensione dei pacchetti di comando è 1000 byte.
   - trovare una capacitá del bottleneck in modo tale che non ci siano perdite di pacchetti.
   - modificare il ritardo dei link e verificare come influenza il ritardo dei comandi.
   - modificare la dimensione delle code e verificare come influenza il ritardo dei comandi e la percentuale di perdita dei pacchetti.

2. Lanciare una simulazione alzando la priorità (etichetta 0) del traffico NCS mentre la sorgente concorrente rimane sempre in bassa priorità. Visualizzare l'andamento di packet loss rate e ritardo dei comandi e andamento del tracking error.

   - cosa succede circa l'andamento di ritardo, packet loss rate dei comandi e tracking error?
   - usando lo script plr.sh in */tmp/scnsl/bin/* calcolare il packet loss rate complessivo del traffico concorrente;
   - riportare su un foglio il valore complessivo del tracking error e confrontarlo con quello dell'esercizio 1.

3. Lanciare una simulazione alzando la priorità (etichetta 0) del traffico NCS e abilitando task 7 invece di task 3 (alta priorità). Visualizzare l'andamento di packet loss rate e ritardo dei comandi e andamento del tracking error.

   - cosa succede circa l'andamento di ritardo, packet loss rate dei comandi e tracking error?
   - usando lo script plr.sh in */tmp/scnsl/bin/* calcolare il packet loss rate complessivo del traffico concorrente;
   - riportare su un foglio il valore complessivo del tracking error.
   - i valori di queste statistiche sono simili all'esperimento 1 o 2?

4. Lanciare una simulazione usando il marker intelligente (*NCS_QoS_marker.mdl*) con sorgente concorrente in bassa priorità; Visualizzare l'andamento di packet loss rate e ritardo dei comandi e andamento del tracking error.

   - cosa succede circa l'andamento di ritardo, packet loss rate dei comandi e tracking error?
   - usando lo script plr.sh in */tmp/scnsl/bin/* calcolare il packet loss rate complessivo del traffico concorrente;
   - riportare su un foglio il valore complessivo del tracking error; come si posiziona rispetto agli esperimenti 1 e 2?

5. Lanciare una simulazione usando il marker intelligente (*NCS_QoS_marker.mdl*) con sorgente concorrente sia in bassa priorità e in alta priorità; Modificare il file *DefineList.hh* in modo che il traffico concorrente generi in totale 1.350.000 bit/sec. Visualizzare l'andamento di packet loss rate e ritardo dei comandi e andamento del tracking error.

   - cosa succede circa l'andamento di ritardo, packet loss rate dei comandi e tracking error?
   - usando lo script plr.sh in */tmp/scnsl/bin/* calcolare il packet loss rate complessivo del traffico concorrente;
   - riportare su un foglio il valore complessivo del tracking error.