REALIZZAZIONE DI UN MODULO IN OMNeT PER LA SIMULAZIONE DI UNA INTERFACCIA SPI COMUNICANTE CON IL LIVELLO FISICO IEEE 802.15.4 E RELATIVA VALUTAZIONE DELLE PRESTAZIONI

-

Corso di Reti per l'automazione industriale
A.A. 2015/16

Gruppo 1 (realizzazione del modulo):

Niccolò Fabrizio Pagano – O55000281
Enrico Verzì – O55000271

Gruppo 2 (valutazione delle prestazioni):

Laura Bartolone - O55000296
Miriam Benbachir - O55000272

Indice generale

Introduzione

La presente relazione contiene la descrizione dei passi che hanno condotto alla realizzazione di un modulo che simula un'interfaccia di comunicazione SPI, che si trova al di sopra e che comunica con il livello fisico dello standard IEEE 802.15.4, facendo uso del simulatore ad eventi discreti OMNeT++.
Per la simulazione del livello fisico è stato utilizzato il modulo Ieee802154phy, presente nella libreria INETMANET.
Per la generazione del traffico, è stata prevista la progettazione ed implementazione di uno strato che si collochi al di sopra del livello dell'interfaccia SPI e che faccia uso delle primitive offerte da quest'ultimo. Queste sono riportate nella tabella sottostante:

Primitive dal livello superiore verso il livello fisico.		
Nome Primitiva	**Tempo (s)**	**Descrizione**
setTxMode(int channel, cPacket *frame)	T_{tx} = bitLength/1e6 + T_{txm}	Questa primitiva trasmette la frame al buffer di trasmissione e setta il canale su cui deve impostarsi il transceiver per trasmetterla (T_{txm} è un valore parametrizzabile con valore di default pari a 350 us, ossia 0,00035 s).
startTx	T_{stx} = default(50us)	Questa primitiva avvia la trasmissione della frame che si trova sul buffer di trasmissione.
setRxMode(int channel)	T_{srx} = default(50us)	Questa primitiva configura il transceiver in ricezione su un determinato canale.
getReceivedData	T_{rx} = bitLength/1e6 + T_{rxm}	Questa primitiva richiede al livello fisico di trasmettere al livello superiore la frame ricevuta. La risposta è la frame nel buffer di ricezione che deve essere trasmessa ai livelli superiori dopo un tempo T_{rx} dalla ricezione di questa primitiva. Oppure deve essere trasmessa una primitiva apposita che indica che non ci sono frame ricevute.
Primitive dal livello fisico verso i livelli superiori		
TxDone	0	Inviata quando la trasmissione di una frame è conclusa.
RxReady	0	Inviata quando è stata ricevuta una frame valida.

Tabella 1: primitive offerte dalla SPI verso livello applicazione e fisico

Per verificare il corretto funzionamento del modulo verranno eseguite opportune simulazioni che mostreranno che i tempi delle specifiche vengono rispettati, raccogliendo statistiche su:

- Durata del processamento (simulato) delle primitive setTxMode, startTx, setRxMode, getReceivedData.
- Raccolta statistiche sul throughput, ossia num. totale di bit ricevuti da un nodo (sink) diviso il tempo totale della simulazione.

Interfaccia SPI

In questo paragrafo verranno descritti i passi che hanno condotto alla realizzazione del Simple Module rappresentante l'interfaccia SPI. Tale progettazione ha tenuto conto della necessaria interoperabilità tra interfaccia SPI e livello fisico IEEE 802.15.4.

Primitive offerte dall'interfaccia SPI

Data la tabella 1 sono state definite le seguenti primitive che regolano lo scambio di frame dati tra livello applicativo e livello fisico:

1. `setTxModePrimitive`: questa primitiva consente al livello applicativo di impostare il canale del livello fisico ed inserire la frame nel buffer di trasmissione dell'interfaccia SPI. Tale frame verrà successivamente inviata dietro ricezione della primitiva `startTx`;
2. `confirmSetTxModePrimitive`: questa primitiva consente al livello applicativo di sapere che la richiesta di `setTxMode` verso i livelli sottostanti è stata completata. Tale necessità è dovuta al fatto che la `setTxMode` richiede un tempo per la commutazione di modalità da ricezione a trasmissione del livello fisico;
3. `startTxPrimitive`: questa primitiva consente al livello applicativo di avviare la trasmissione della frame che si trova nel buffer di ricezione dell'interfaccia SPI verso il livello fisico;

4. `TxDonePrimitive`: questa primitiva viene inviata al livello applicativo quando la trasmissione della frame è stata completata con successo da parte del livello fisico;
5. `setRxModePrimitive`: questa primitiva consente al livello applicativo di impostare il canale del livello fisico e la modalità ricezione;
6. `confirmSetRxModePrimitive`: questa primitiva consente al livello applicativo di sapere che la richiesta di `setRxMode` verso i livelli sottostanti è stata completata. Tale necessità è dovuta al fatto che la `setRxMode` richiede un tempo per la commutazione di modalità da trasmissione a ricezione del livello fisico;
7. `getReceivedDataPrimitive`: questa primitiva permette al livello applicativo di richiedere il trasferimento della frame ricevuta dal livello fisico e attualmente disponibile nel buffer di ricezione dell'interfaccia SPI;
8. `RxNoFramePrimitive`: si tratta di una primitiva di risposta alla primitiva `getReceivedData` e viene inviata quando il buffer di ricezione dell'interfaccia SPI è vuoto, per notificare al livello applicativo che non vi sono frame disponibili;
9. `RxReadyPrimitive`: si tratta di una primitiva che notifica al livello applicativo che una frame è disponibile nel buffer di ricezione dell'interfaccia SPI. Alla ricezione di questa primitiva, il livello applicativo potrà effettuare una richiesta di `getReceivedData` per ottenere la suddetta frame.

Progettazione della FSM dell'interfaccia SPI

Di seguito viene riportato il diagramma della macchina a stati dell'interfaccia SPI. Sugli archi sono indicati gli eventi che generano il cambio di stato e la relativa azione che viene intrapresa dall'interfaccia.

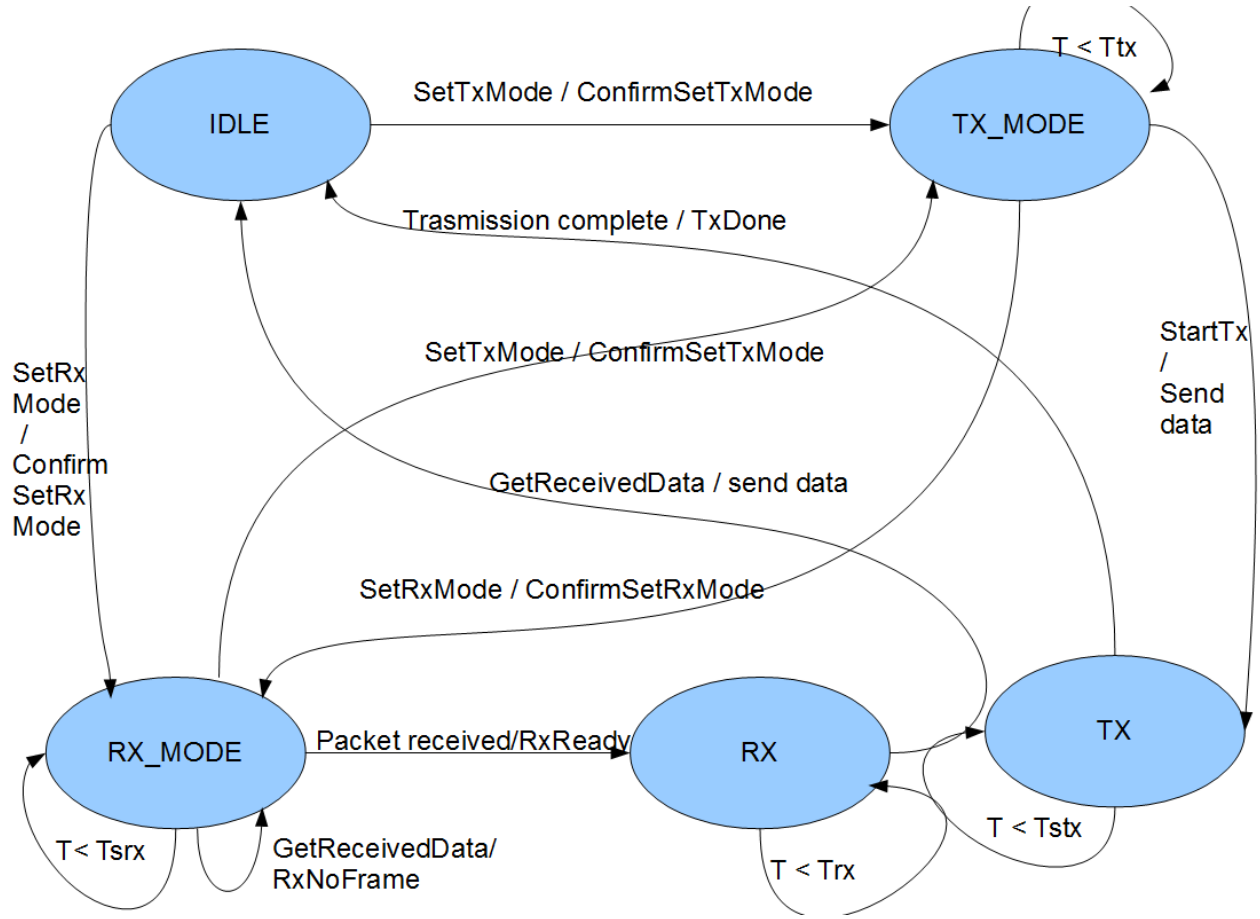


Illustrazione 1: diagramma della FSM della interfaccia SPI

Lo stato iniziale in cui si trova l'interfaccia è lo stato IDLE. In tale stato gli unici due eventi che possono verificarsi sono la ricezione di una setTxMode o la ricezione di una setRxMode.

Nel caso di una setTxMode, l'interfaccia commuta stato verso lo stato TX_MODE. Al fine di simulare il tempo necessario al cambio di modalità, viene avviato un timer (Ttx), allo scadere del quale l'interfaccia informa lo strato superiore con una ConfirmSetTxMode dell'avvenuto cambio di modalità.

A questo punto l'interfaccia può ricevere una primitiva di tipo startTx, che la porta verso lo stato TX di trasmissione. Viene quindi avviato un timer (Tstx) che simula il tempo di trasmissione della frame da inviare lungo il canale, l'interfaccia SPI si riporta nello stato di IDLE e comunica ai livelli soprastanti che la trasmissione è stata completata con una TxDone.

Sia nello stato di TX_MODE che nello stato di IDLE possono giungere all'interfaccia primitive di tipo setRxMode. L'interfaccia a questo punto avvia il timer che simula il tempo di commutazione di stato (Tsrx) e al suo scadere notifica ai livelli soprastanti che l'operazione è stata completata con una ConfirmSetRxMode.

A questo punto, se il livello superiore desidera ricevere eventuali frame presenti nel buffer di ricezione della SPI, fa uso della primitiva getReceivedData. La relativa risposta dipenderà dallo stato in cui si troverà la SPI al momento della ricezione della primitiva. In particolare:

- se lo stato è RX_MODE, viene restituita ai livelli superiori una primitiva RxNoFrame;
- se lo stato è RX (stato di ricezione) significa che il buffer di ricezione contiene una frame valida e pertanto essa viene inoltrata ai livelli superiori. Si è infatti previsto che l'interfaccia SPI si sposti dallo stato RX_MODE allo stato RX non appena riceva una frame valida e notifichi la disponibilità di servire una getReceivedData con la primitiva RxReady. La ricezione impiega in ogni caso un tempo Trx dipendente dalla lunghezza della frame. Al termine di questo tempo la SPI si sposta autonomamente nello stato di IDLE.

Implementazione della FSM dell'interfaccia SPI

I frammenti di pseudo-codice di sotto riportati costituiscono una descrizione alternativa del comportamento dell'interfaccia SPI, precedentemente definito:

```
case IDLE:

       if (setTxModePrimitive) {//caso ricezione primitiva setTxMode

            state = TX_MODE; //prossimo stato = TX_MODE

            //decapsulamento della dataframe
            //dalla primitiva setTxMode

            //bufferizzazione della dataframe

            //avvio del timer che simula il cambio di stato


        }

        if (setRxModePrimitive) {//caso ricezione primitiva setRxMode

             //prossimo stato = RX_MODE
             state = RX_MODE;

             //avvio del timer che simula il
             //cambio di stato

         }
```

Codice 1: pseudocodice relativo al case IDLE dell'interfaccia SPI

```
case TX_MODE:

     if (timeout TX_MODE timer) { //completato passaggio a
                                  //modalita' trasmissione

          //notifica al livello app
          //dell'avvenuto cambio di stato

          //impostazione del canale in trasmissione
          //e invio al livello fisico della  dataframe
          setTxMode(channel, dataframe);

          confirmSetTxMode();

          return;
     }

     if (startTxPrimitive) {//richiesta di startTx

          state = TX;  //prossimo stato = TX

          //avvio del timer che
          //simula la trasmissione

          return;
     }


     if (setRxModePrimitive) {//richiesta di setRxMode

         //prossimo stato = RX_MODE

         state = RX_MODE;

          //avvio del timer che simula
          //il cambio di stato

         return;

     }
```

Codice 2: pseudocodice relativo al case TX_MODE dell'interfaccia SPI

```
case RX_MODE:

     if (timeout RX MODE timer) {//gestione del timer avvenuto
                                 // passaggio allo stato RX_MODE

              //impostazione del canale in ricezione
               setRxMode(channel);
              //notifica verso il livello app dell'avvenuto cambio di stato
              confirmSetRxMode();
              return;
      }

      if (dataframe coming from PHY layer) {
              //gestione dell'arrivo di una dataframe
              //dal livello fisico quando l'interfaccia
              //ha già completato il passaggio allo stato RX MODE

              //bufferizzazione della dataframe

              RxReady();
              //notifica al livello applicazione
              //della disponibilita' di una dataframe

              state = RX; //prossimo stato = RX
              return;
       }

       if (setTxModePrimitive) { //richiesta di setTxMode

                state = TX_MODE; //prossimo stato = TX_MODE

                //decapsulamento della dataframe
                //all'interno della primitiva ricevuta

                //impostazione del canale in trasmissione
                //ed invio al livello fisico della dataframe
                setTxMode(channel, dataframe);

                //avvio del timer che simula la trasmissione
                return;
        }

        if (getReceivedDataPrimitive) {//ricezione di una getReceivedData

                //avvio del timer che simula i tempi
                //di trasmissione della primitiva RxNoFrame
                return;
        }

        if (timeout RxNoFrame timer) { //gestione del timeout del timer
                                       //relativo alla RxNoFrame
                RxNoFrame();
                //notifica al livello applicativo che non vi sono dataframe
                //disponibili nel buffer di ricezione
                return;
        }
```

Codice 3: pseudocodice relativo al case RX_MODE dell'interfaccia SPI

```
case RX:

     if (getReceivedDataPrimitive) { //ricezione di una getReceivedData

          //avvio del timer che simula i
          //tempi di trasmissione della frame
          //verso il livello applicazione

          return;
      }

      if (timeout getReceivedData timer) {//gestione del timeout del timer
                                          //relativo alla getReceivedData

          getReceivedData();
          //invio della dataframe presente
          //nel buffer di ricezione verso
          //il livello applicazione

          state = IDLE; //prossimo stato = IDLE

          return;

       }


case TX:

     if (timeout startTx timer) { //gestione del timeout del timer che
                                  //simula l'avvio della trasmissione

          startTx();
          //avvio della trasmissione della
          //frame presente nel buffer di
          //trasmissione verso il livello fisico

          return;

     }
```

Codice 4: pseudocodice relativo ai case RX e TX dell'interfaccia SPI

Per ulteriori dettagli implementativi, fare riferimento ai file SPIInterface.h e SPIInterface.cc, presenti nel package src del progetto.

Livello applicativo

Sulla base dei servizi offerti dall'interfaccia SPI, è stato progettato ed implementato un livello applicazione che ben si adatti ai livelli sottostanti.

Dal momento che la rete si compone di un unico trasmettitore ed un unico ricevitore, il comportamento di tale livello dipende dalla natura del nodo.

In particolare, nel caso del nodo trasmettitore, il livello si comporta come un generatore di traffico che dipende dai messaggi provenienti dall'interfaccia SPI. Dal momento che l'obiettivo delle simulazioni, che verranno effettuate, è quello di valutare le prestazioni massime della rete in presenza dell'interfaccia SPI, il trasmettitore genera traffico alla massima velocità possibile che la SPI sottostante riesce a sostenere, ovvero genera traffico ad ogni ricezione di una TxDone. La relativa macchina ad eventi è descritta dai seguenti diagrammi di sequenza, in cui viene evidenziata lo scambio di messaggi tra i tre livelli (applicazione, SPI e fisico) del nodo trasmettitore:

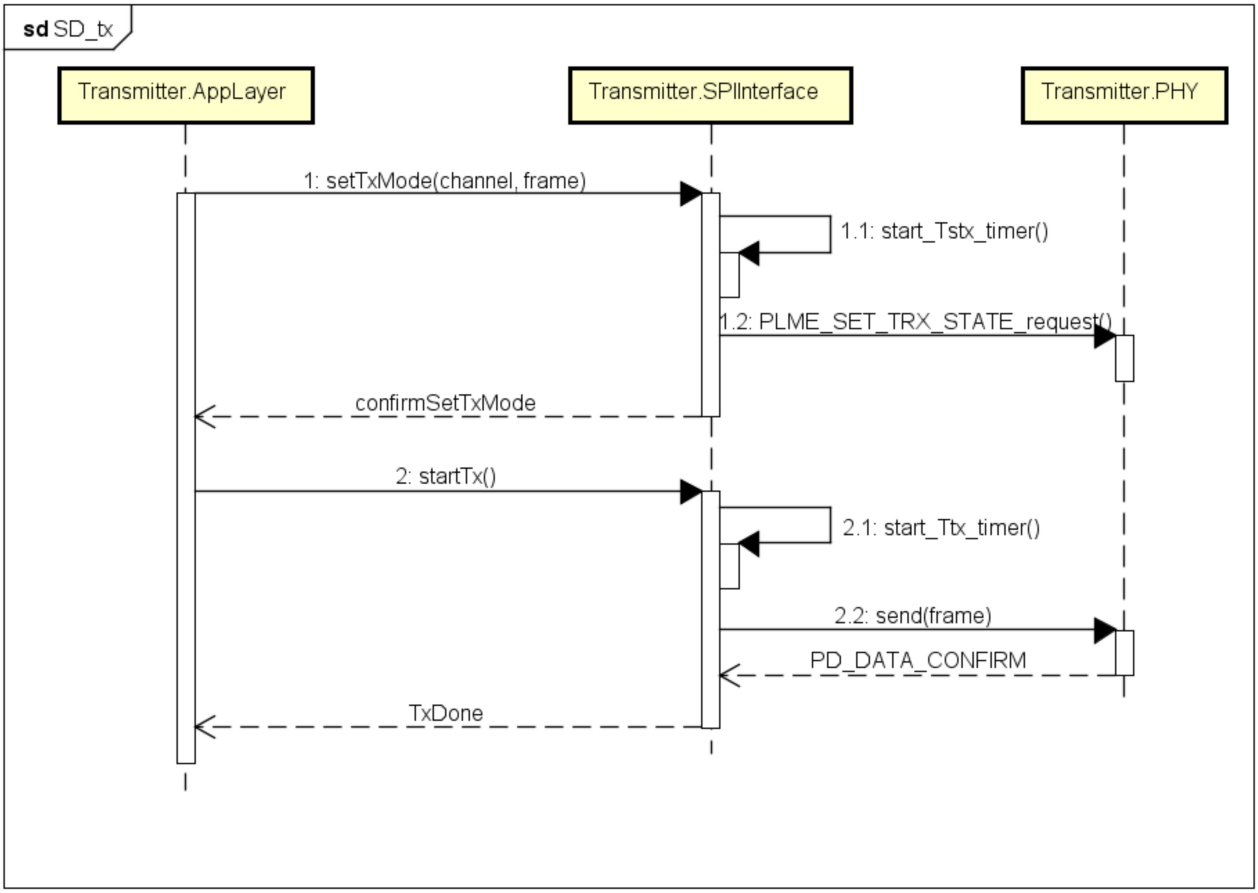


Illustrazione 2: diagramma di sequenza del livello app del nodo trasmettitore

Il livello applicativo del nodo trasmettitore dapprima invia una setTxMode all'interfaccia SPI. Questa, secondo la propria FSM sopra illustrata, imposta il canale del livello fisico e risponde al livello applicativo con una confirmSetTxMode. Questa ricezione, da parte del livello applicativo, costituisce un nuovo evento, la cui conseguenza è l'invio della startTx. Completata la trasmissione, il livello applicativo riceverà una TxDone, a cui reagirà con una nuova setTxMode.

Nel caso del nodo ricevitore (il sink), invece, sono previste due modalità di funzionamento:

- **modalità interrupt**: il livello applicazione non chiama nessuna primitiva getReceivedData finchè non riceve la notifica di disponibilità di dataframe nel buffer di ricezione della SPI (tramite una primitiva RxReady);
- **modalità periodica**: il livello applicazione con un periodo pari a metà del tempo di arrivo delle dataframe alla SPI chiama la primitiva getReceivedData. In tal modo, su N richieste, la metà saranno soddisfatte mentre le rimanenti scateneranno la ricezione, a livello applicativo, di primitive RxNoFrame.

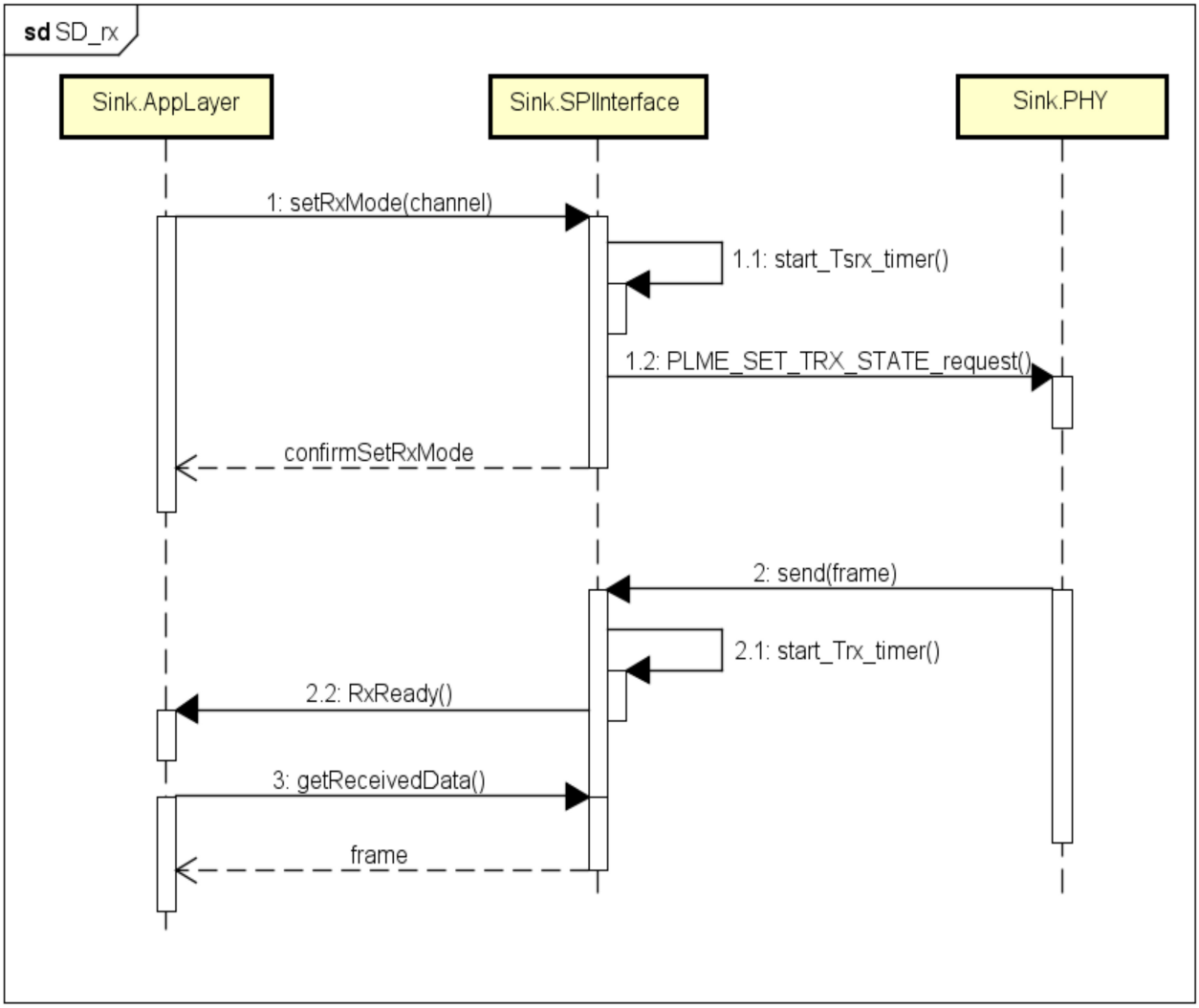


Illustrazione 3: diagramma di sequenza del livello app del nodo sink (mod. interrupt)

Il livello applicativo del nodo ricevitore dapprima invia alla SPI una setRxMode. Completato il cambio di canale, al livello applicativo viene inviata una confirmSetRxMode. A questo punto, poichè la modalità interrupt prevede che il livello applicazione richieda una dataframe solo dietro ricezione di una primitiva RxReady proveniente dalla SPI, il livello applicazione si blocca in attesa della ricezione di questa primitiva. La ricezione della RxReady scatenerà la richiesta della frame disponibile tramite una getReceivedData.

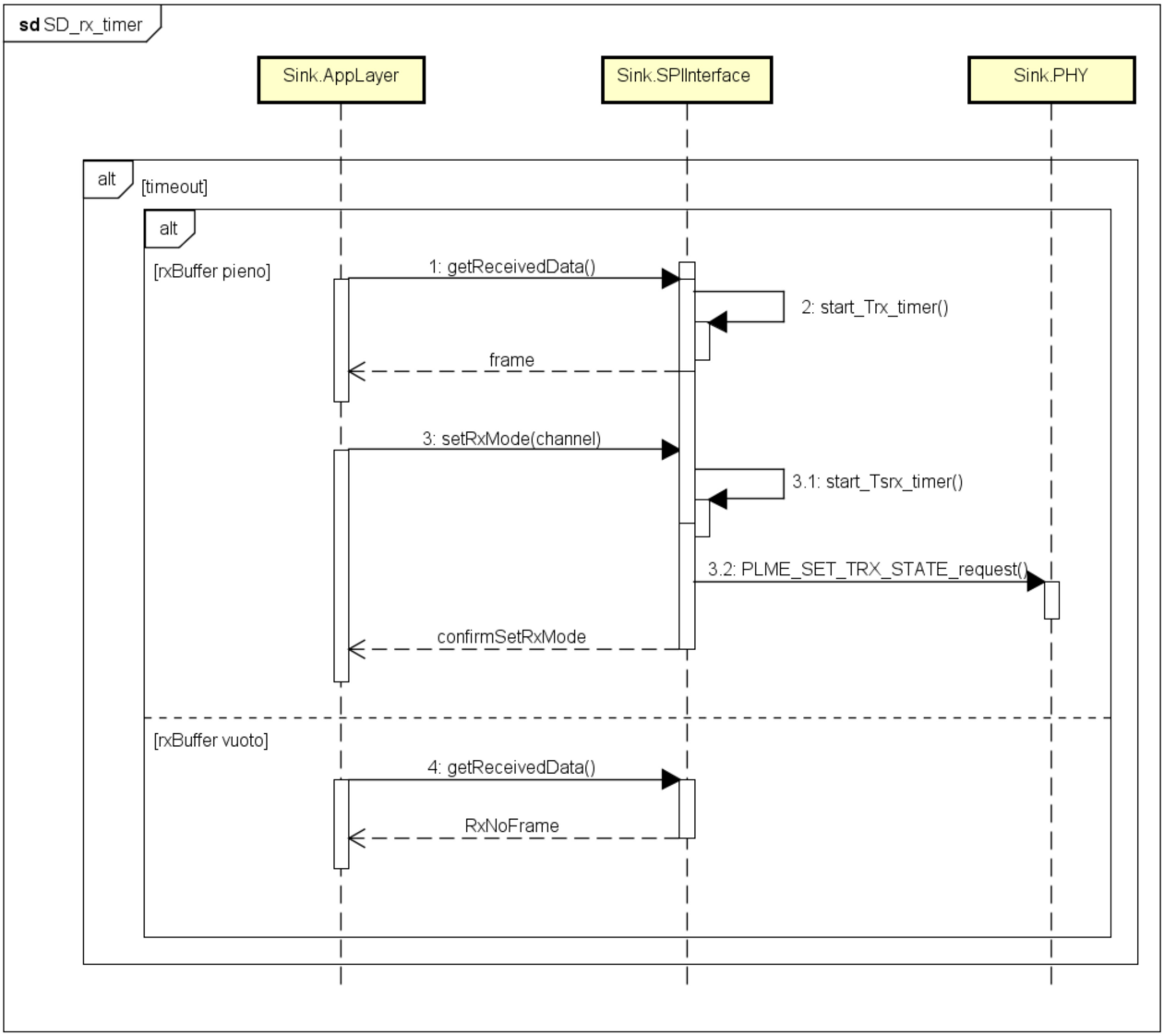


Illustrazione 4: diagramma di sequenza del livello app del nodo sink (mod. periodica)

Quando la modalità è periodica, il comportamento del livello applicativo dipende dall'eventuale presenza di frame dati nel buffer di ricezione della SPI al momento del timeout.

In particolare, se il buffer di ricezione è vuoto, la SPI risponderà alla getReceivedData con una primitiva RxNoFrame. Se il buffer di ricezione è pieno, invece, la SPI risponderà alla getReceivedData con una frame dati e, a ricezione completata, il livello applicazione setterà nuovamente lo stato del livello fisico alla modalità ricezione.

Implementazione del livello applicazione

I frammenti di pseudo-codice di sotto riportati costituiscono una descrizione alternativa del comportamento del livello applicazione, precedentemente discusso:

```
if (!sink) {//codice del trasmettiore (possibili stati SPI: IDLE, TX_MODE, TX)

   if (confirmSetTxModePrimitive) {//gestione della ricezione della
                                   //primitiva di conferma di avvenuto
                                   //passaggio di stato a TX_MODE

           //generazione della primitiva startTx
           //invio della primitiva

           return;
     }

    if (TxDonePrimitive) {//gestione della ricezione della
                          //primitiva di notifica di avvenuta trasmissione

            //generazione della primitiva setTxMode
            //generazione di una nuova dataframe


            //invio della primitiva

            return;
      }

}
```

Codice 5: pseudocodice relativo al livello app del trasmettitore

```
if (sink) {//codice del ricevitore  (modalità periodica)

   if (timeout getReceivedDataTimer) {

                //riavvio del prossimo timer

                //generazione della primitiva getReceivedData
                //invio della primitiva

               return;
   }


    if (dataframe) {

                //generazione della primitiva setRxMode
                //invio della primitiva

                return;
    }


}
```

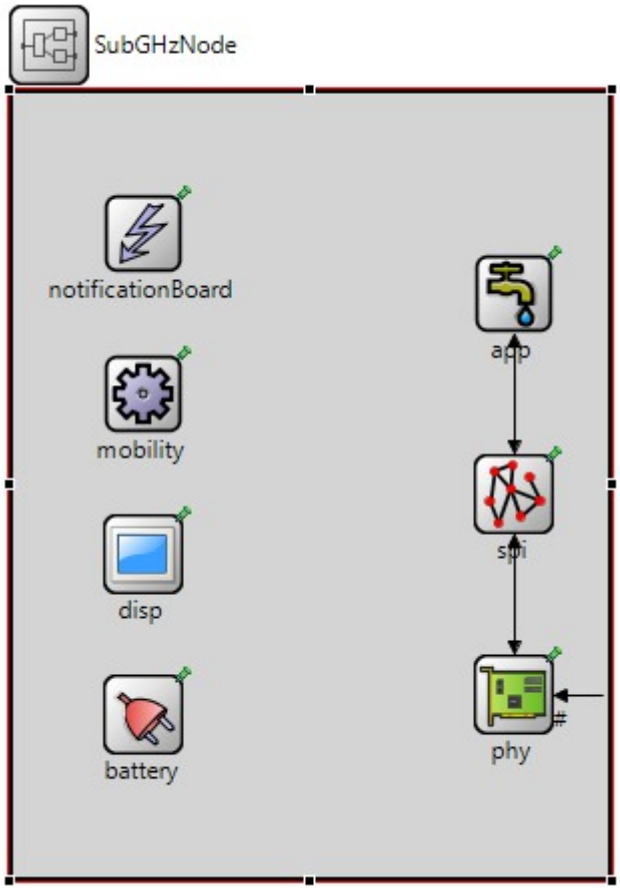
Codice 6: pseudocodice relativo al livello app del ricevitore (modalità periodica)

```
if (sink) {//codice del ricevitore  (modalità interrupt)

   if (RxReadyPrimitive ) {//gestione della ricezione di una
                           //primitiva di notifica di
                           //disponibilita' di una dataframe
                           //nell'interffacia SPI

          //generazione della primitiva getReceivedData
          //invio della primitiva

          return;
    }

    if (dataframe ) {//gestione della ricezione
                     //di una nuova dataframe

           //generazione della primitiva setRxMode
           //invio della primitiva



           return;
    }

}
```

Codice 7: pseudocodice relativo al livello app del ricevitore (modalità interrupt)

Definizione della rete

Una volta implementati lo strato applicazione e lo strato dell'interfaccia SPI si è proceduto all'aggregazione di tali strati con il livello fisico Ieee802154phy. In particolare si è definito un nodo SubGHzNode, rappresentante il generico nodo della rete. La struttura di questo Compound Module è illustrata nelle immagini sottostanti:



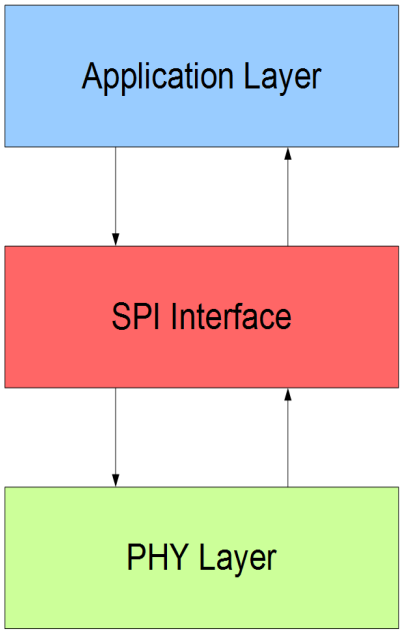


Illustrazione 5: schema di composizione del compound module SubGHzNode

Ogni livello comunica con il livello sottostante tramite un gate di input ed un gate di output.

Per quanto riguarda la definizione della rete da osservare si sono considerati solo due nodi, un trasmettitore ed un ricevitore (sink). Ciò è dovuto al fatto che, non essendo presente un livello MAC, non sarebbe possibile gestire le collisioni quando il numero di nodi trasmettitori è due o superiore. La rete è stata definita

prendendo spunto dalla rete StarNet di INETMANET, dando la possibilità di configurarla opportunamente per le esigenze di simulazione. Nel codice sottostante, contenuto nel file omnetpp.ini della directory simulation, sono riportati i parametri che è possibile variare per le diverse simulazioni:

```
**.host[*].app.GetReceivedTimePeriod = 0.0018961667816s
**.host[*].app.gestioneConInterrupt = false
**.host[*].app.payloadSize = 640 #variare da 80 a 640 con passo di 80 bit
```

Codice 8: parametri variabili principali presenti all'interno del file di configurazione omnetpp.ini

Il parametro payloadSize rappresenta la dimensione delle frame dati generati in bit. Quando il campo gestioneConInterrupt è settato a true, il ricevitore commuta nella modalità interrupt, precedentemente esposta. In tal caso il campo GetReceivedTimePeriod viene ignorato.

Valutazione delle prestazioni

Con il termine Valutazione delle prestazioni si intende un insieme di attività rivolte alla determinazione delle caratteristiche di un sistema sulla base del suo comportamento. Attraverso la valutazione di prestazioni di un sistema è possibile sapere “cosa” esso può offrire sotto determinate condizioni operative, in modo da delimitare l’area applicativa dove può essere utilizzato con successo.

Le prestazioni della rete sono state valutate in due differenti casi di sviluppo:

1. Il caso con timer in cui il ricevitore richiede periodicamente i dati all'interfaccia SPI tramite la getReceivedData. In questo caso sono state effettuate diverse simulazioni con bitlenght variabile nell'intervallo tra 10 e 80 Bytes. Il timer è stato impostato a metà dei tempi di arrivo delle frame a livello SPI.

2. Il caso di gestione degli Interrupt, in cui il nodo ricevitore (sink) richiede i dati alla SPI solo previa ricezione di un Interrupt (RxReady) cioè solo quando vi sono dati disponibili. In questo secondo caso il caso di RxNoFrame non si verifica mai e ciò dimostra che l'Interrupt funziona perfettamente.

E' possibile scegliere quale simulazione avviare (caso 1 o caso 2) settando il flag gestioneConInterrupt nel file .ini.

Segnali utilizzati per la valutazione delle prestazioni

Al fine di confrontare i tempi delle primitive simulati con i tempi da noi impostati e calcolati sono stati emessi i segnali relativi ad ogni primitiva. Ogni segnale indica il ritardo dall'omonima primitiva.

1. **setRxMode_time**

```
simsignal_t sig = registerSignal("setRxMode_time");
setRxModePrimitive *prim =
dynamic_cast<setRxModePrimitive *>(primBuffer->decapsulate());
emit(sig, simTime().dbl() - prim->getGenTime());
```

Codice 9: segnale emesso dalla SPI quando scade il timer relativo al passaggio in RxMode.

2. **startTx_time**

```
simsignal_t sig = registerSignal("startTx_time");
startTxPrimitive *prim =
dynamic_cast<startTxPrimitive *>(primBuffer->decapsulate());
emit(sig, simTime().dbl() - prim->getGenTime());
```

Codice 10: segnale emesso dalla SPI quando scade il timer Tstx che simula l'avvio di una trasmissione.

3. **getReceivedData_time**

```
simsignal_t sig=registerSignal("getReceivedData_time");
getReceivedDataPrimitive *prim =
dynamic_cast<getReceivedDataPrimitive *>(primBuffer->decapsulate());
emit(sig, simTime().dbl() - prim->getGenTime());
```

Codice 11: segnale emesso dalla SPI quando scade il timer relativo alla getReceivedData.

4. setTxMode_time

```
simsignal_t sig = registerSignal("setTxMode_time");
setTxModePrimitive *prim =
dynamic_cast<setTxModePrimitive *>(primBuffer->decapsulate());
emit(sig, simTime().dbl() - prim->getGenTime());
```

Codice 12: segnale emesso dalla SPI quando scade il timer relativo al passaggio in TxMode.

5. DataFrameArrivalPeriod

Il traffico della rete non è periodico ma è dettato dagli eventi che le primitive gestiscono. Ogni volta che il trasmettitore genera un pacchetto si salva il tempo di generazione dello stesso. Per calcolare i tempi di arrivo delle frame a livello SPI si è utilizzato il segnale `DataframeArrivalPeriod`, che viene generato dall'interfaccia SPI ogni qual volta arriva una frame dati all'interfaccia:

```
simsignal_t sig2 = registerSignal("DataframeArrivalPeriod");
emit(sig2, (double) (simTime().dbl() - df->getGenTime()));
```

Codice 13: segnale per l'emissione della DataframeArrivalPeriod

Questo DataFrameArrivalPeriod è stato calcolato settando (nel file .ini) il flag gestioneConInterrupt uguale a true e quindi misurando i tempi di arrivo delle frame nel caso di gestione con interrupt. Il valor medio di questo segnale, ottenuto per uno specifico bitLenght (parametrizzabile), è stato poi dimezzato ed utilizzato per impostare il timer del caso 1. Così facendo il timer del livello applicativo richiede i dati all'interfaccia SPI con una frequenza doppia rispetto a quella di generazione delle frame. Ciò comporta che riceverà una frame dati con una probabilità del 50% e nell'altro caso riceverà la primitiva RxNoFrame.

6. Workload e Throughput

Al fine di valutare le prestazioni sono stati emessi dei segnali ausiliari, non direttamente riguardanti i tempi di ritardo delle primitive o il throughput. In particolare per ogni TxDone ricevuta è stato emesso il segnale `Workload` dal livello applicativo:

```
simsignal_t sig = registerSignal("Workload");
emit(sig, (double) (payloadSize) /
((double)(simTime().dbl() - dataframeGenTime)));
```

Codice 14: segnale per l'emissione del segnale workload

Il workload rappresenta il totale di bit trasmessi diviso il tempo impiegato per trasmetterli. Il livello applicativo del nodo trasmettitore (host [1]) spedisce la frame ogni qual volta riceve la primitiva TxDone. Questo segnale è stato calcolato come la dimensione del payload trasmesso (payloadSize) fratto i tempi di trasmissione del payload stesso. A dimostrazione del corretto funzionamento della rete si è osservato che **il Workload, fissata la bitLenght, resta costante nel tempo ed è sempre maggiore del throughput.**

Per misurare il Throughput è stato utilizzato il segnale `Throughput`, emesso dal livello applicativo del nodo ricevitore:

```
totbit = totbit + df->getBitLength();
simsignal_t sig = registerSignal("Throughput");
double thr = (double) totbit / simTime().dbl();
emit(sig, thr);
```

Codice 15: codice per l'emissione del segnale Throughput

Questo segnale è calcolato dal nodo ricevitore che somma tutti i bit delle varie frame ricevute nella variabile totbit e poi lo divide per il tempo attuale.

7. **RxNoFrame**

Il segnale `RxNoFrame_time` è invece emesso dalla SPI quando scade il timer relativo dell'RxNoFrame:

```
simsignal_t sig = registerSignal("RxNoFrame_time");
getReceivedDataPrimitive *prim =
dynamic_cast<getReceivedDataPrimitive*>(
                    primBuffer->decapsulate());
emit(sig, simTime().dbl() - prim->getGenTime());
```

Codice 16: codice per l'emissione del segnale RxNoFrame

Simulazione

Tutte le simulazioni saranno svolte con Workload massimo, pari al massimo traffico generato che l'interfaccia SPI riesce a supportare.

In tutte le simulazioni i nodi sono a 100 m di distanza.

SIMULAZIONI CON INTERRUPT

In questo caso il nodo ricevitore (sink) richiede i dati alla SPI solo previa ricezione di un Interrupt (RxReady) cioè solo quando vi sono dati disponibili. Dall'analisi dei dati sotto riportati è emerso che non è stata ricevuta dal livello applicativo neppure una RxNoFrame e ciò dimostra che l'Interrupt funziona perfettamente. In tabella si riporta solo il caso intermedio a 40 Bytes, i dati relativi alle altre simulazioni del caso Interrupt sono in allegato.

SIMULAZIONE

Caso con Interrupt (senza timer)

Simulazione con **bitLenght=40 Bytes**

durata simulazione 22 s

Nome	Num. campioni	Valor Medio (o Valore singolo nel caso di uno Scalare)	Deviazione Standard	Varianza
SetRxMode :vett	10037	4.9999999E-5	2.22922408236576 4E-11	4.969440009399482E -22
SetTxMode :vett	10037	6.699999999999999E-4	0	0
StartTx :vett	10037	4.9999999000000996E-5	2.22998116307441 88E-11	4.972815987666737E -22
Workload :vett	10036	145985.40152645478	0	0
DataFrame ArrivalPeriod :vett	10036	0.002192333563000000 2	9.89155693484115 E-10	9.784289859520407E -19
GetReceivedData :vett	10036	6.7E-4	0	0
Throughput :scalare		145980.95330953		
RxNoFrame	0			

In tutte queste simulazioni il throughput è sempre pochissimo minore del Workload e non si riceve neppure una RxNoFrame in quanto siamo nel caso Interrupt.

Analizziamo adesso nel caso intermedio (**BitLenght=40Bytes**) la durata del processamento (simulato) delle primitive setTxMode, StartTx, setRxMode e getReceivedData.
SetTxMode_time, StartTx_time, setRxMode_time e getReceivedData_time : indicano il tempo di processamento delle omonime primitive.

SetTxMode_Time

SetTxMode_time: **valor medio** 6.699999999999999E-4 s ricavato in simulazione.

Calcolato manualmente con la formula:

SetTxMode_time = BitLenght / 10^6 + Ttxm =

40*8/10^6+0.00035=0.00067

In cui Ttxm è fissato a 0.00035 s.

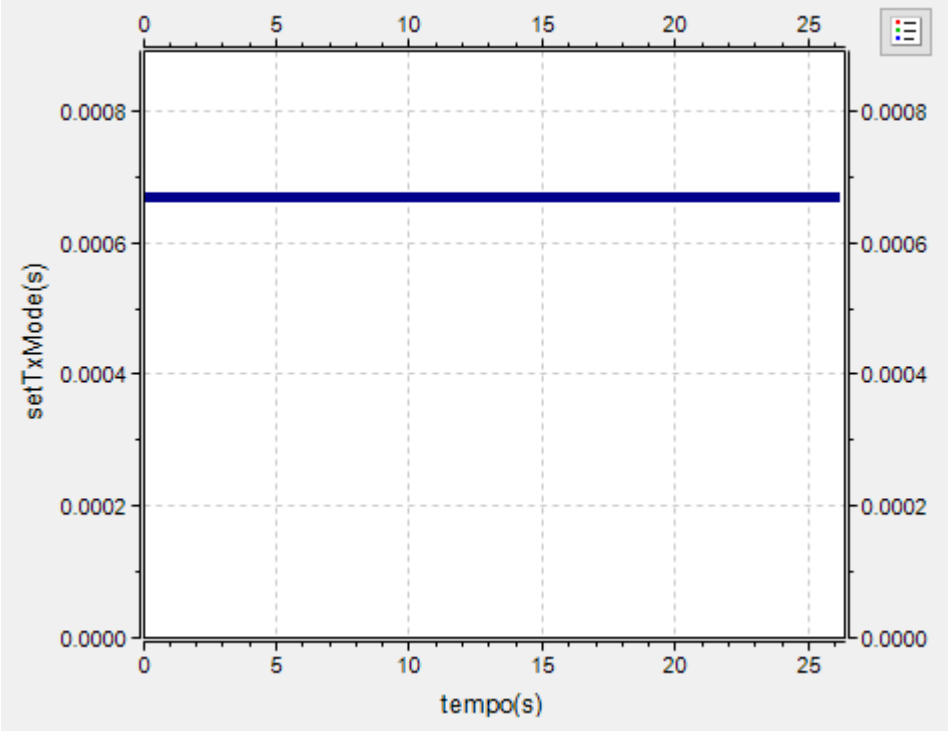


Figura 1: andamento del setTxMode con ByteLenght=40 Bytes

I due risultati calcolato e simulato coincidono.

StartTx_Time

StartTx_time: **valor medio** 4.9999999000000996E-5 s ricavato in simulazione. Doveva risultare essere uguale a 50 micro secondi proprio come è emerso in simulazione.

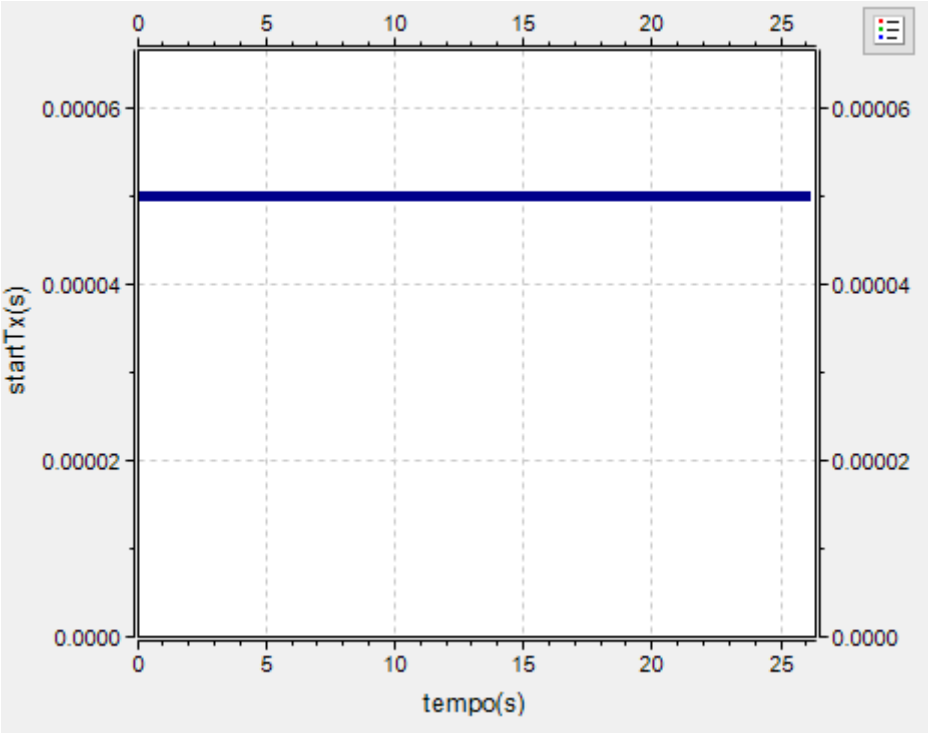


Figura 2: andamento dello startTx_time con Bytelenght=40Bytes

SetRxMode_Time

SetRxMode_time: **valor medio** 4.9999999E-5 ricavato in simulazione. Doveva risultare essere uguale a 50 µs proprio come è emerso in simulazione.

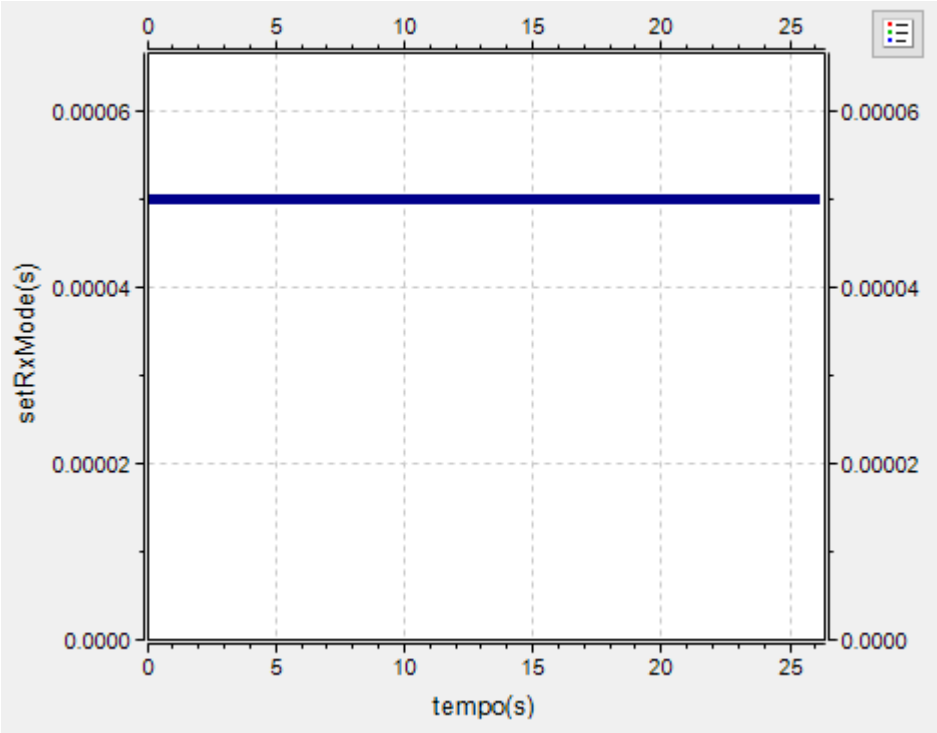


Figura 3: setRxMode

GetReceivedData_Time

GetReceivedData_time: **valor medio** 6.7E-4 ricavato in simulazione.

Calcolato manualmente con la formula:

GetReceivedData_time = BitLenght / 10^6 + Trxm = 40*8/10^6+0.00035=0.00067

In cui Trxm è fissato nel file .ini a 350 μs

I due risultati calcolato e simulato coincidono anche in questo caso.

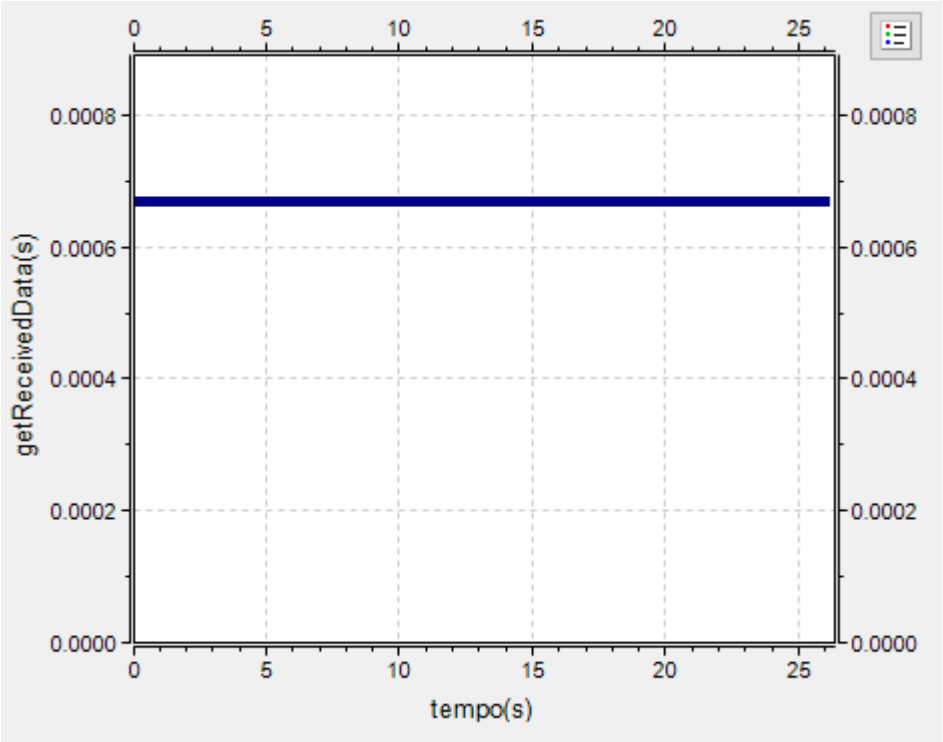


Figura 4: getReceivedData(s)

I calcoli sono stati effettuati pure nei casi con bitLenght differenti e coincidono con quelli ottenuti in simulazione.

SIMULAZIONI CON TIMER

SIMULAZIONE 1

BitLenght=10 Bytes

Il timer è stato impostato a metà del DataframeArrivalPeiod nel caso 10 Bytes cioè pari a :

`0.00004961667815 s`

Nome	Num. campioni	Valor Medio (o Valore singolo nel caso di uno Scalare)	Deviazione Standard	Varianza
SetRxMode :vett	10080	4.999999899999999E-5	0	0
SetTxMode :vett	10081	4.3000000000000004E-4	0	0
StartTx :vett	10081	4.9999999E-5	0	0
Workload :vett	10080	80645.16137161608	0.013028917976610117	1.6975270364123424E-4
DataFrame ArrivalPeriod :vett	10080	9.92333563E-4	3.790963849711412 6E-10	1.4371406909818772E-19
GetReceivedData :vett	10079	4.3E-4	0	0
Throughput :scalare		80638.588408847		
RxNoFrame	9175	4.9999999E-5	0	0

SIMULAZIONE 2

BitLenght=20 Bytes

Il timer è stato impostato a metà del DataframeArrivalPeiod nel caso 20 Bytes cioè pari a :

`0.00069616678149999995 s.`

Nome	Num. campioni	Valor Medio (o Valore singolo nel caso di uno Scalare)	Deviazione Standard	Varianza
SetRxMode :vett	10057	4.9999999E-5	0	0
SetTxMode :vett	10058	5.099999999999999E-4	0	0
StartTx :vett	10058	4.9999999E-5	0	0
Workload :vett	10057	114942.52881820624	0.020482325461836173	4.195256563245823E-4
DataFrame ArrivalPeriod :vett	10057	0.0013923335629999999	2.8954804049251606E-10	8.383806775305571E-20
GetReceivedData :vett	10056	5.1E-4	0	0
Throughput :scalare		114933.66414589		
RxNoFrame	9452	4.9999999E-5	0	0

SIMULAZIONE 3

BitLenght=30 Bytes

Il timer è stato impostato a metà del DataframeArrivalPeiod nel caso 30 Bytes cioè pari a : `0.0008961667815 s`

Nome	Num. campioni	Valor Medio (o Valore singolo nel caso di uno Scalare)	Deviazione Standard	Varianza
SetRxMode :vett	10044	4.9999999E-5	0	0
SetTxMode :vett	10045	5.9E-4	0	0
StartTx :vett	10045	4.9999999000000996E-5	0	0
Workload :vett	10044	133928.57150330546	0.019161710883812598	3.6717116399482224E-4
DataFrame ArrivalPeriod :vett	10044	0.0017923335630000003	0	0

GetReceivedData :vett	10043	5.9E-4	0	0
Throughput :scalare		133919.25907359		
RxNoFrame	9591	4.9999999E-5	0	0

SIMULAZIONE 4

BitLenght=40 Bytes

Il timer è stato impostato a metà del DataframeArrivalPeiod nel caso 40 Bytes cioè pari a :

`0.0010961667815000001 s`

Nome	Num. campioni	Valor Medio (o Valore singolo nel caso di uno Scalare)	Deviazione Standard	Varianza
SetRxMode :vett	10037	4.9999999000000996E-5	0	0
SetTxMode :vett	10037	6.699999999999999E-4	0	0
StartTx :vett	10037	4.9999999000000996E-5	0	0
Workload :vett	10036	145985.40152645478	0	0
DataFrame ArrivalPeriod :vett	10036	0.002192333563000000 2	9.89155693484115 E-10	9.784289859520407E -19
GetReceivedData :vett	10036	6.7E-4	0	0
Throughput :scalare	10036	145980.56310695		
RxNoFrame	9582	4.9999999E-5	0	0

SIMULAZIONE 5

BitLenght=50 Bytes

Il timer è stato impostato a metà del DataframeArrivalPeiod nel caso 50 Bytes cioè pari a :

`0.0012961667815 s`

Nome	Num. campioni	Valor Medio (o Valore singolo nel caso di uno Scalare)	Deviazione Standard	Varianza
SetRxMode :vett	10031	4.999999899999801E-5	0	0
SetTxMode :vett	10031	7.5E-4	0	0
StartTx :vett	10031	4.999999900000199E-5	0	0
Workload :vett	10030	154320.98771385843	0.014556288399084561	2.1188553195732377E-4
DataFrame ArrivalPeriod :vett	10030	0.002592333563	2.381898407810678 7E-10	5.673440025131047E-20
GetReceivedData :vett	10030	7.5E-4	0	0
Throughput :scalare		154312.06472047		
RxNoFrame	9728	4.999999899999897E-5	0	0

SIMULAZIONE 6

BitLenght=60 Bytes

Il timer è stato impostato a metà del DataframeArrivalPeiod nel caso 60 Bytes cioè pari a :

`0.00149616678149999985 s`

Nome	Num. campioni	Valor Medio (o Valore singolo nel caso di uno Scalare)	Deviazione Standard	Varianza
SetRxMode :vett	10027	4.999999900000598E-5	0	0
SetTxMode :vett	10027	8.3E-4	0	0
StartTx :vett	10027	4.9999998999999E-5	0	0
Workload :vett	10026	160427.80754024535	0.03673895239989018	0.0013497506234413964

DataFrame ArrivalPeriod :vett	10026	0.002992333562999999 7	1.20677359976069 34E-9	1.4563025210793824 E-18
GetReceivedData :vett	10026	8.300000000000001E-4	0	0
Throughput :scalare		160421.48663382		
RxNoFrame	9724	4.999999900000163E-5	0	0

SIMULAZIONE 7

BitLenght=70 Bytes

durata simulazione =

Il timer è stato impostato a metà del DataframeArrivalPeiod nel caso 70 Bytes cioè pari a :

`0.0016961667815 s`

Nome	Num. campioni	Valor Medio (o Valore singolo nel caso di uno Scalare)	Deviazione Standard	Varianza
SetRxMode :vett	10023	4.999999899999801E-5	0	0
SetTxMode :vett	10024	9.100000000000001E-4	0	0
StartTx :vett	10024	4.999999900000199E-5	0	0
Workload :vett	10023	165094.33967131597	0	0
DataFrame ArrivalPeriod :vett	10023	0.003392333563000000 3	0	0
GetReceivedData :vett	10022	9.1E-4	0	0
Throughput :scalare		165081.92353966		
RxNoFrame	9870	4.9999999E-5	0	0

SIMULAZIONE 8

BitLenght=80 Bytes

durata simulazione = 38 s.

Il timer è stato impostato a metà del DataframeArrivalPeiod nel caso 80 Bytes cioè pari a :

`0.0018961667814999998 s`

Nome	Num. campioni	Valor Medio (o Valore singolo nel caso di uno Scalare)	Deviazione Standard	Varianza
SetRxMode :vett	10021	4.999999900000199E-5	0	0
SetTxMode :vett	10021	9.8999999999999E-4	0	0
StartTx :vett	10021	4.999999899999501E-5	0	0
Workload :vett	10021	168776.37135251972	0.04267736268127568	0.0018213572854291417
DataFrame ArrivalPeriod :vett	10021	0.003792333562999999 6	1.5181088834652846E-9	2.3046545820562132E-18
GetReceivedData :vett	10020	9.8999999999999E-4	0	0
Throughput :scalare		168765.55081066		
RxNoFrame	9869	4.9999999E-5	0	0

In tutte queste simulazioni il throughput è sempre pochissimo minore del Workload. Inoltre è possibile osservare che nel caso timer non sempre la SPI ha il buffer di ricezione pieno ciò determina la presenza delle RxNoFrame. Il numero di RxNoFrame è leggermente inferiore a quello delle getReceivedData. Ciò vuol dire che con un 50% di probabilità arriveranno i dati e nei rimanenti casi arriveranno RxNoFrame. Questo comportamento era prevedibile in quanto il timer è impostato a metà dei tempi dei arrivo delle frame.

Analizziamo adesso nel caso intermedio (**BitLenght=40Bytes**) la durata del processamento (simulato) delle primitive setTxMode, StartTx, setRxMode e getReceivedData.
SetTxMode_time, StartTx_time, setRxMode_time e getReceivedData_time : indicano il tempo di processamento delle omonime primitive.

SetTxMode_Time

SetTxMode_time: **valor medio** 6.699999999999999E-4 s ricavato in simulazione.

Calcolato manualmente con la formula:

SetTxMode_time = BitLenght / 10^6 + Ttxm = 40*8/10^6+0.00035=0,00067

In cui Ttxmè fissato a 0,00035 s

I due risultati calcolato e simulato coincidono.

StartTx_Time

StartTx_time: **valor medio** 4.9999999000000996E-5 s ricavato in simulazione.

Doveva risultare essere uguale a 50 micro secondi proprio come è emerso in simulazione.

SetRxMode_Time

SetRxMode_time: **valor medio** 4.9999999E-5 ricavato in simulazione.

Doveva risultare essere uguale a 50 μs proprio come è emerso in simulazione.

GetReceivedData_Time

GetReceivedData_time: **valor medio** 6.7E-4 ricavato in simulazione.

Calcolato manualmente con la formula:

GetReceivedData_time = BitLenght / 10^6 + Trxm = 40*8/10^6+0.00035=0,00067

In cui Trxm è fissato nel file .ini a 350 μs

I due risultati calcolato e simulato coincidono anche in questo caso.

I calcoli sono stati effettuati pure nei casi con bitLenght differenti e coincidono con quelli ottenuti in simulazione.

Grafici

Tutti i grafici sono stati realizzati con OpenOffice Calc. Tutte le prestazioni sono state valutate al limite della saturazione. Il grafico sottostante mostra come varia il Throughput della rete all'aumentare della lunghezza delle frame. In ascissa abbiamo la dimensione in Bytes delle frame e in ordinata il Throughput.

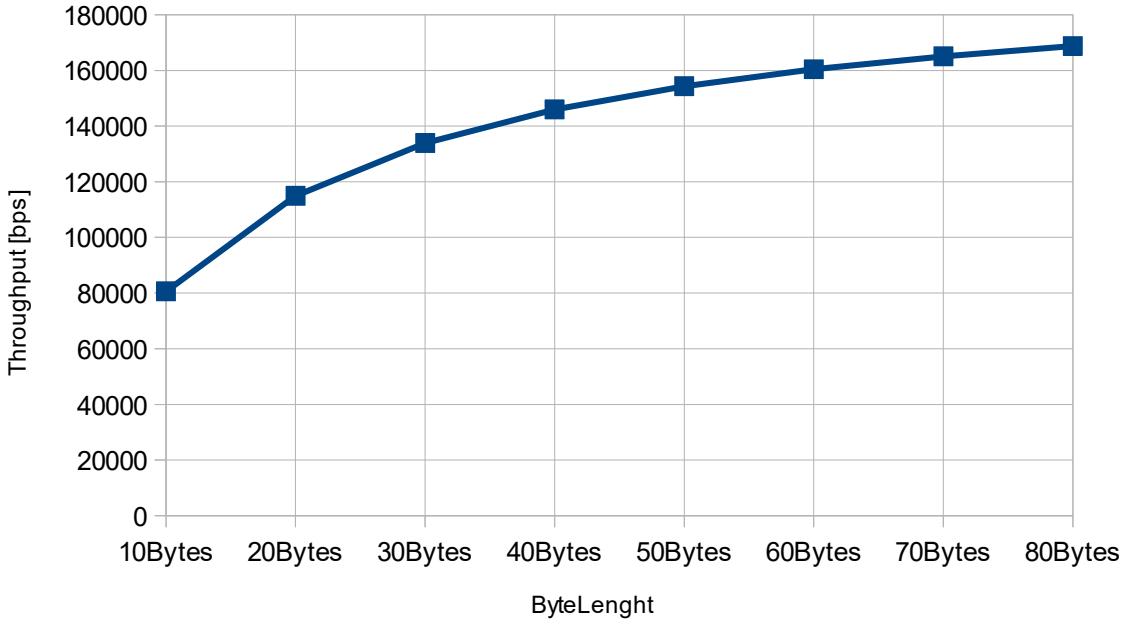


Figura 5:Throughput della rete nel caso Timer all'aumentare del ByteLenght.

Il grafico sottostante mostra l'aumentare del WorkLoad all'aumentare della dimensione delle frame.

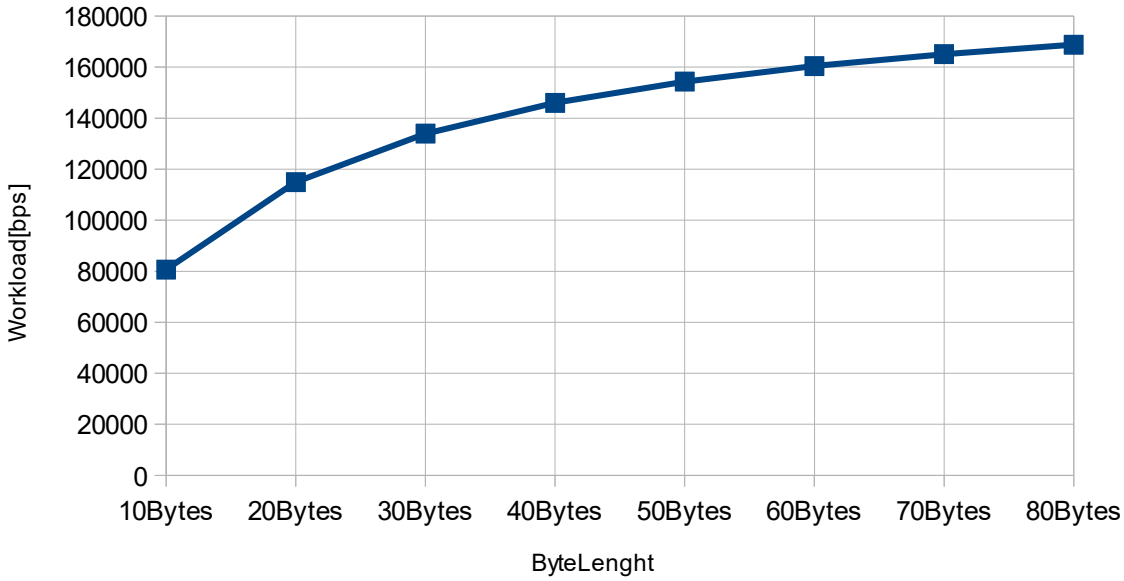


Figura 6: Workload al variare del ByteLenght nel caso timer

Il grafico sottostante mostra il variare del DataFrameArrival Period nel caso timer all'aumentare del ByteLenght.

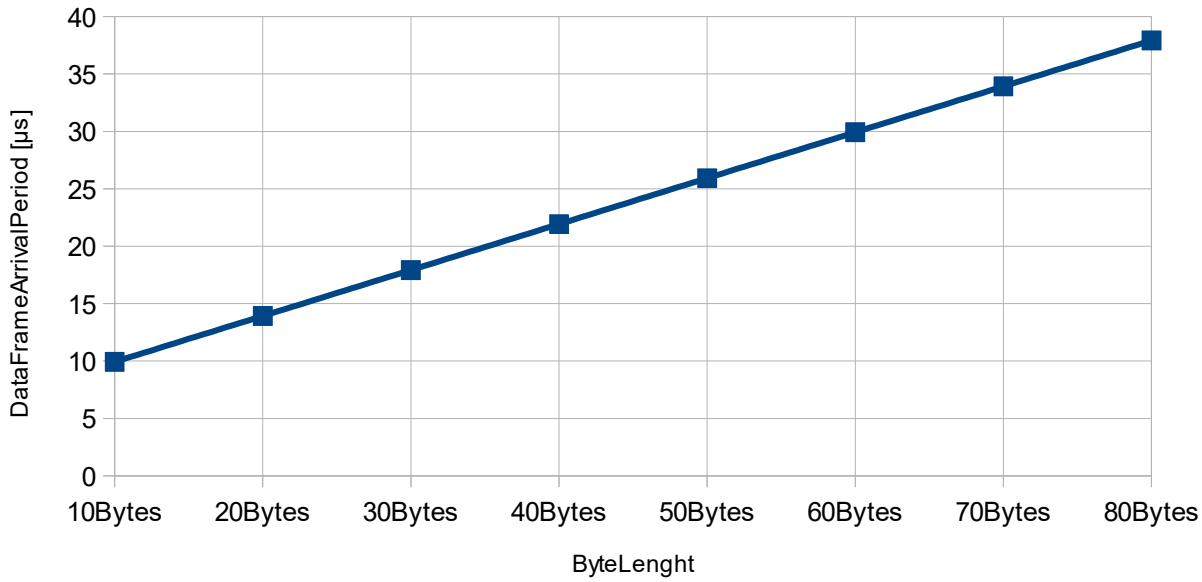


Figura 7:DataFrameArrival Period nel caso timer.