Contents

1 Modello A

Sono stati usati tre template per rappresentare un sender, un link wd un receiver

```
sender = Sender();
receiver = Receiver();
link = Link();
system sender, receiver, link;
```

1.1 Sender

Lo stato iniziale dell'automata è *wait* che simula l'attesa dell'arrivo di un pacchetto che viene processato nell'intervallo [1;2] del clock *sc*. Lo stato successivo è $send_{pkg}$ che invia il pacchetto sul canale di sincronizzazione urgente *ML*. Il sender passa allo stato $wait_{ack}$ che simula l'attesa di un pacchetto sul canale di sincronizzazione urgente *LM* e resetta il clock *sc*. Viene simulata la verifica dell'ack in un intervallo di tempo [2;4] e successivamente l'automa torna allo stato iniziale resettando il clock *sc*.

1.2 Receiver

Il receiver è simmetrico al sender. Dallo stato iniziale di *wait* viene simulato l'arrivo di un pacchetto sul canale di sincronizzazione urgente *LR* che permette al receiver di resettare il clock *rc* e avanzare nello stato $recv_{pkg}$, simulando la preparazione del pacchetto in un intervallo di tempo [2;4]. La transizione dallo stato $done_{pkg}$ a $send_{ack}$ simula la generazione di un pacchetto di ack in un intervallo di tempo [2;4] ed tornare allo stato iniziale dopo aver inviato il pacchetto nel canale di sincronizzazione urgente *RL*.

1.3 Link

Il link è modellato come un canale perfetto senza perdite. Lo stato interno *idle* simula l'attesa di un pacchetto da parte del receiver o del sender. Lo stato $received_{pkg}$ simula l'arrivo di un pacchetto da parte del sender sul canale *ML* che viene processato e spedito al receiver sul canale *LR*. Simmetricamente le stesse operazioni di attesa, processing e invio avvengono con i pacchetti di ack di cui si simula l'arrivo nello stato $received_{ack}$ dal canale di trasmissione *RL* e l'invio verso il sender dallo stato $resend_{ack}$ attraverso

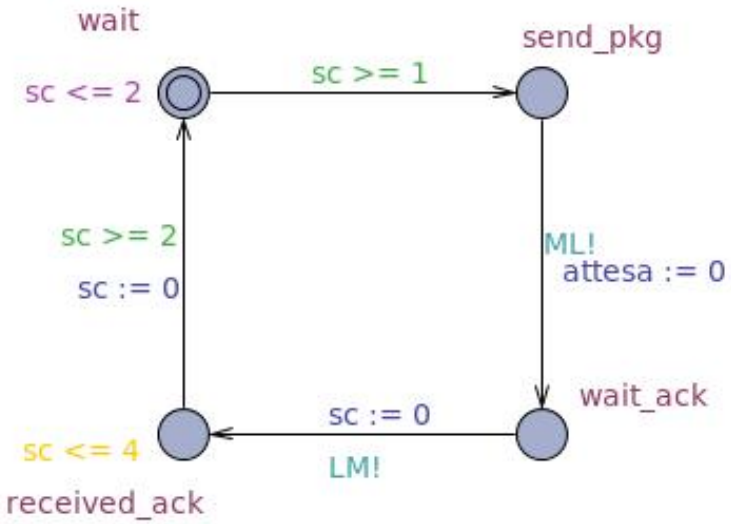


Figure 1: Sender protocollo A

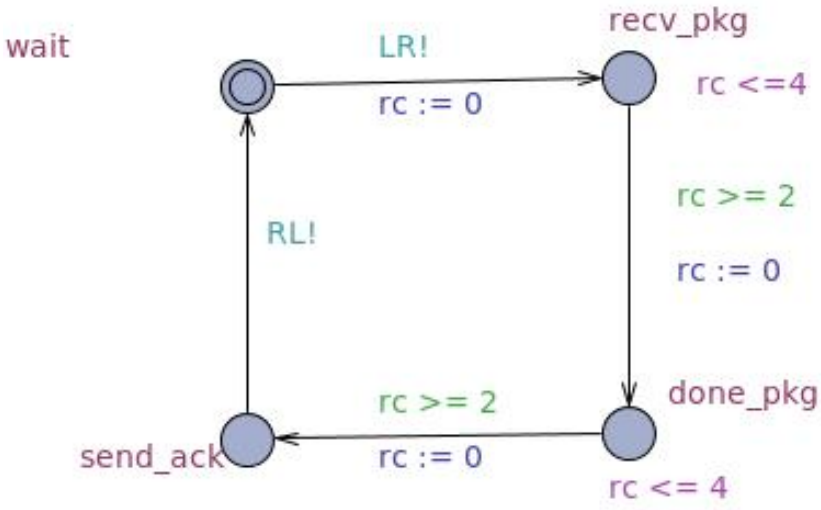


Figure 2: Receiver protocollo A

il canale di trasmissione *LM*. Per il processing dei pacchetti l'intervallo di tempo è sempre [2;4].

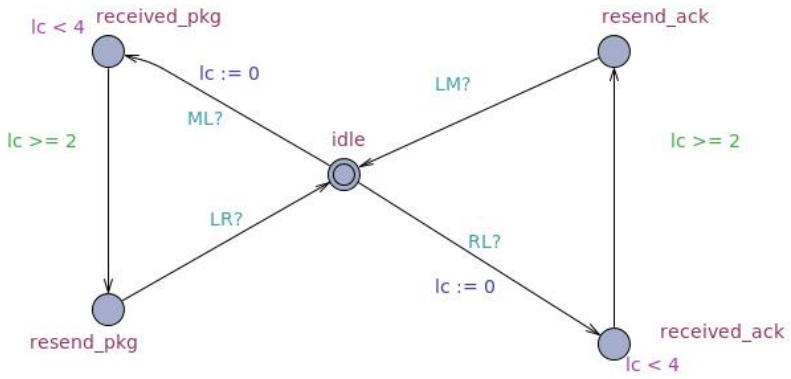


Figure 3: Link protocollo A

1.4 Proprietà

Sono state verificate tre proprietà TCTL sul modello:

- Assenza di deadlock

  $$\text{A[] (not deadlock)}$$

  che viene rispettata e indica che la traccia del sistema è infinita

- Il sender riceverà sempre un ack

  $$\text{AF sender.received}_{\text{ack}}$$
  $$\text{A<> sender.received}_{\text{ack}}$$

  la proprietà risulta vera in quanto il link non ha perdite.

- Il receiver riceverà sempre un pacchetto

  $$\text{AF receiver.recv}_{\text{pkg}}$$
  $$\text{A<> receiver.recv}_{\text{pkg}}$$

  è una proprietà rispettata in quando non ci sono constraint sul tempo di attesa del pacchetto.

1.5 Tempo attesa

Il tempo di attesa viene calcolato attraverso il simulatore di Uppaal. È stato inserito un clock che viene resettato ogni qualvolta viene inviato un pacchetto e una volta processato l'ack. L'intervallo di tempo che intercorre dall'invio del messaggio alla recezione del suo ack è [11, 22).

2 Modello B

È stato riutilizzato il sistema A e modificato il link per simulare la possibilità di perdita o corruzione di un pacchetto.

2.1 Link

Come si denota dall'immagine il link è simile a quello del modello A eccezion fatta per la possibilità di procedere non deterministicamente dagli spazi $received_{pkg}$ e $received_{ack}$ verso *loss* Mostriamo un trace in cui il link

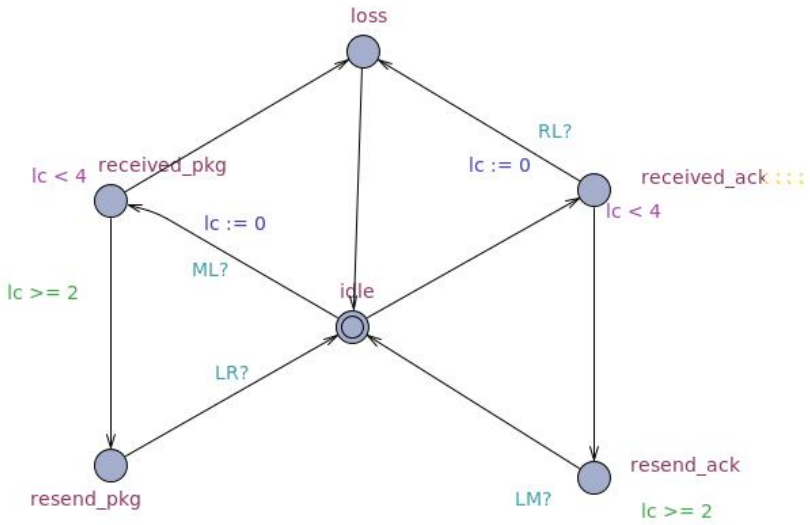


Figure 4: Link protocollo B

perde un pacchetto di ack inviato dal sender e il sistema va in deadlock.

Dalla figura possiamo notare la transizione che simula la perdita, ovvero l'arco $received_{ack} \rightarrow loss$.

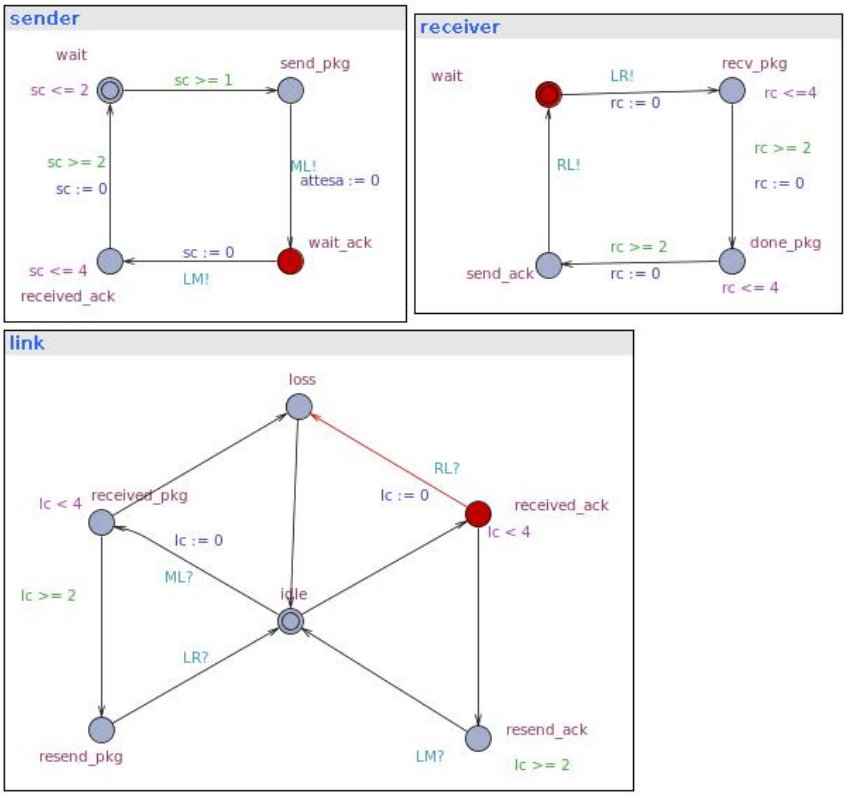
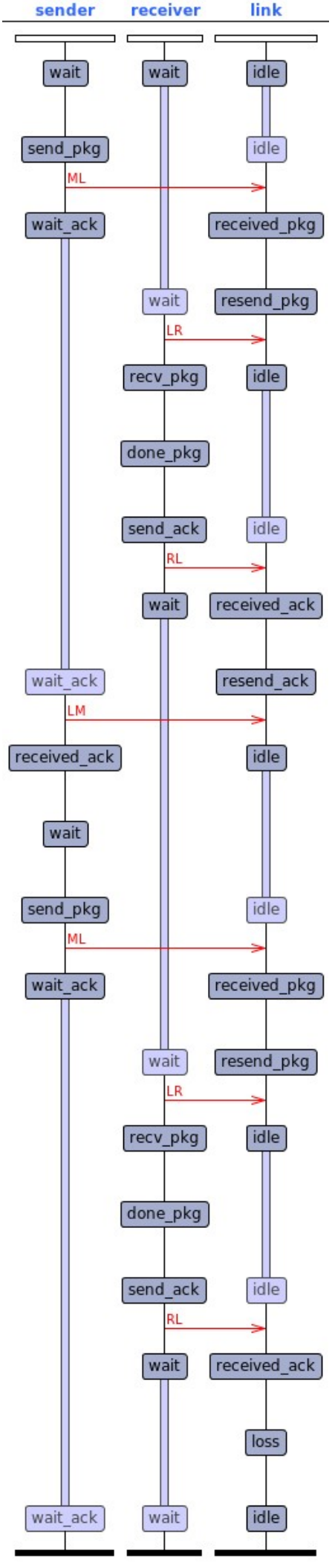


Figure 5: Trace con deadlock


sender
receiver
link
wait
wait
idle
send_pkg
idle
ML
wait_ack
received_pkg
wait
resend_pkg
LR
recv_pkg
idle
done_pkg
send_ack
idle
RL
wait
received_ack
wait_ack
resend_ack
LM
received_ack
idle
wait
send_pkg
idle
ML
wait_ack
received_pkg
wait
resend_pkg
LR
recv_pkg
idle
done_pkg
send_ack
idle
RL
wait
received_ack
loss
wait_ack
wait
idle

2.2 Proprietà

Sono state verificate quattro proprietà TCTL sul modello:

- Assenza di deadlock

  $$\text{A[] (not deadlock)}$$

  che non è rispettata

- Il sender compierà sempre del progresso

  $$\text{sender.waitACK} \dashrightarrow \text{(sender.receivedACK)}$$

  la proprietà non risulta vera in quanto il link ha perdite

- Il receiver riceverà sempre un pacchetto

  $$\text{AF receiver.recv}_{\text{pkg}}$$
  $$\text{A<> receiver.recv}_{\text{pkg}}$$

  che come per la proprietà precedente non può risultare vera

- Il receiver può ricevere un pacchetto

  $$\text{EF receiver.recv}_{\text{pkg}}$$
  $$\text{E<> receiver.recv}_{\text{pkg}}$$

  la proprietà risulta vera perché il link in alcuni path riesce ad inviare il pacchetto (non avvengono perdite sul canale)

2.3 Tempo attesa

Non si devono fare modifiche al modello rispetto al precedente per stabilire il tempo che intercorre dall'invio di un messaggio all'arrivo dell'ack. Bisogna però notare che in caso di deadlock l'ack non arriva mai al mittente e quindi effettivamente possiamo calcolare solo il tempo di ricezione minimo, ovvero 11 cicli del clock.

3 Modello C

È stato riutilizzato il sistema B e modificati sender e receiver per simulare la possibilità di recovery in seguito alla perdita o corruzione di uno o più pacchetti. Vengono anche mantenute due flag binarie globali, *ack* e *frame* per tenere traccia dei pacchetti inviati e degli ack ricevuti.

3.1 Receiver

Di seguito viene mostrato il receiver del modello C. Rispetto al Modello B il receiver utilizza una flag binaria locale r_{frame} per tenere traccia degli pacchetti già ricevuti e processati.

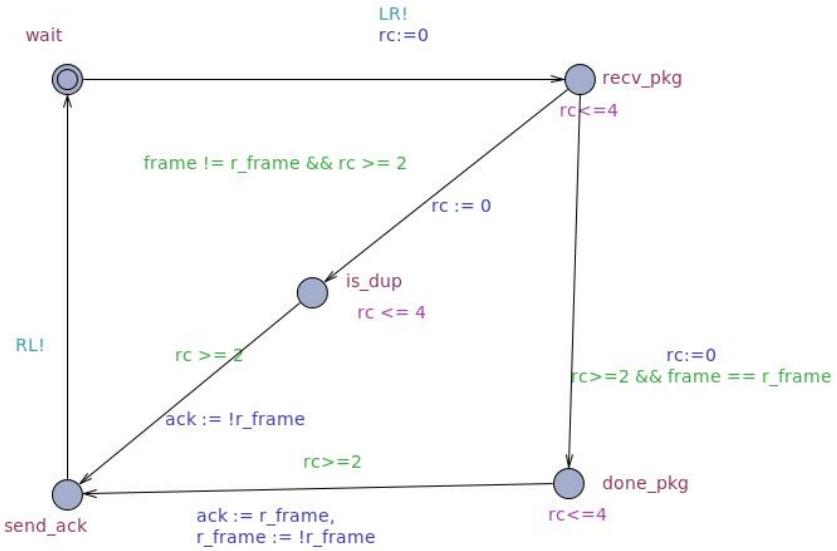


Figure 6: Receiver protocollo C

3.2 Sender

Il sender è stato modellato raddopiando il numero di posti e archi, escludendo per lo stato iniziale, in modo da poter processare nella parte destra i frame di tipo 0 e nella parte sinistra i frame di tipo 1. Inoltre sono stati aggiunti due timer che, nel momento in cui l'automa è fermo su $wait_{ack}$ allo scadere del timeout, permettono di spostarsi nello spazio *lost* e riprovare l'invio del pacchetto. Si nota subito che i due timer sono indipendenti l'uno dall'altro e vi è la possibilità di modellare i sender senza duplicare il numero di spazi.
La variabile che regola il timeout è un numero intero uguale o maggiore del più grande fra i seguenti valore:

- il massimo tempo necessario all'invio di un pacchetto che il sender non ha processato

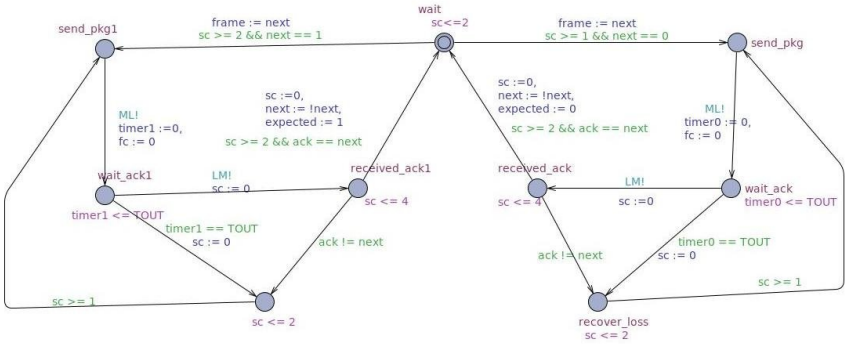


Figure 7: Sender protocollo C (due timer)

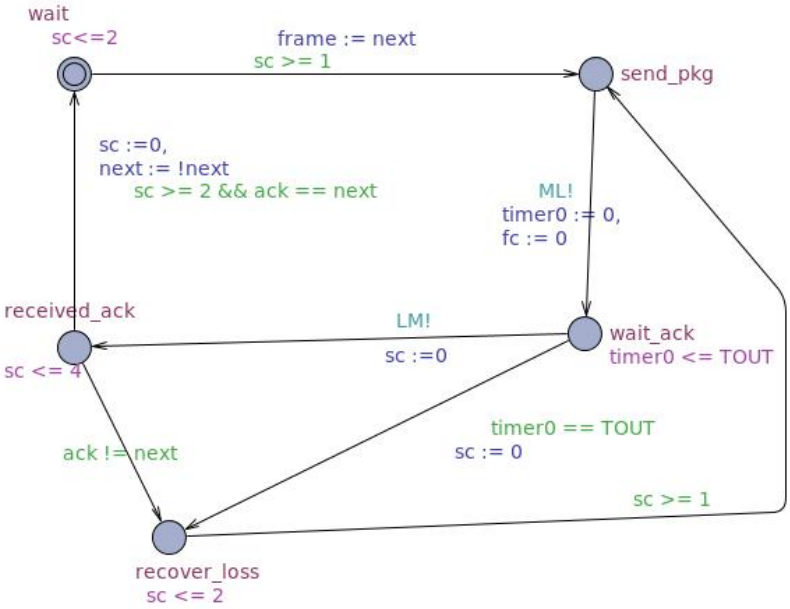


Figure 8: Sender protocollo C (un timer)

- il massimo tempo necessario all'invio di un pacchetto che il sender ha già processato

Se non si rispetta questo constraint il sistema va in deadlock. Nel modello utilizzato lo spazio is_{dup} e $done_{pkg}$ del receiver hanno lo stesso invariante e il timeout $TOUT$ ha valore minimo 16.

3.3 Proprietà

Sono state verificate cinque proprietà TCTL sul modello:

- In caso di perdita il sender o il receiver tentano nuovamente di inviare il pacchetto

  $$\text{link.loss} \dashrightarrow (\text{sender.send}_{\text{pkg}} \quad \text{sender.send}_{\text{pkg1}} \quad \text{receiver.send}_{\text{ack}})$$

  questa proprietà è rispttata.

- Assenza di deadlock

  $$\text{A[] (not deadlock)}$$

  questa proprietà è rispettata.

- Nonostante una perdita nel link, il sender o il receiver riceveranno il pacchetto

  $$\text{link.loss} \dashrightarrow (\text{sender.send}_{\text{pkg}} \quad \text{receiver.send}_{\text{ack}})$$

  la proprietà è rispettata

- Il receiver riceverà sempre un pacchetto

  $$\text{sender.wait}_{\text{ack}} \quad \text{sender.wait}_{\text{ack1}} \dashrightarrow (\text{sender.received}_{\text{ack}} \quad \text{sender.received}_{\text{ack1}})$$

  questa proprietà non risulta vera dato che ci possono essere delle perdite nel canale

- C'è almeno un caso in cui l'ack ricevuto è quello atteso

  $$\text{E<> (ack == expected)}$$

  questa proprietà risulta vera e conferma la correttezza del meccanismo di recovery loss.

3.4 Tempo attesa

In questo modello se si considera l'intervallo di tempo che intercorre dal primo tentativo di invio alla ricezione dell'ack, non è possibile calcolare il valore massimo dell'intervallo. Questo perché non è possibile prevedere quanti tentativi saranno necessari per un corretto invio. Se invece si decide di resettare il clock ad ogni tentativo di invio, ovvero di calcolare l'intervallo di tempo che intercorre fra un pacchetto inviato in una situazione in cui il link non avrà perdite fino all'invio del relativo ack, allora si può considerare il modello equivalente al modello A, e quindi l'intervallo è il medesimo.