Documentazione progetto

Queue Simulator

Progetto C3

Componenti:

Daniele Gotti - 1079011

Amin Borqal - 1073928

Laurea Magistrale in Ingegneria Informatica

Corso di Reti di Telecomunicazioni

Prof. Fabio Martignon

Anno Accademico 2024/2025

Indice

Obiettivo

Progetto C3:

- Si consideri un collegamento tra due nodi modellato mediante sistema a coda con un servente e coda infinita. Gli arrivi siano di Poisson con tasso λ, e i servizi abbiano durata v. c. esponenziale negativa con valor medio $1/\mu$.
- Durante la trasmissione si verifica un errore con probabilità p. Il ricevitore invia i riscontri su un canale a parte in istanti di Poisson con tasso δ. Il trasmettitore invia un pacchetto ed attende il riscontro prima di procedere oltre (Stop-and-Wait).
- Il software riceve in ingresso tutti i parametri e calcola il tempo medio di attraversamento.

Stop-and-Wait

Il protocollo Stop-and-wait è uno dei più semplici meccanismi di controllo del flusso e affidabilità nelle comunicazioni punto-punto. È particolarmente adatto per collegamenti half-duplex, in cui il canale di trasmissione può essere utilizzato in una sola direzione per volta, mentre risulta inefficiente in ambienti full-duplex ad alta latenza. Il suo funzionamento si basa sull'invio di un singolo pacchetto alla volta: il mittente attende un riscontro (ACK) prima di trasmettere il successivo. In caso di errore, rilevato tramite un meccanismo di timeout o tramite ricezione di un NACK, il pacchetto viene ritrasmesso.

Per evitare ambiguità in caso di duplicazioni (ad esempio quando un ACK va perso e il pacchetto viene ritrasmesso inutilmente), è necessaria una numerazione delle trame: un semplice bit (Sequence Number, SN = 0 o 1) è sufficiente per distinguere tra due trame consecutive. Analogamente, l'ACK può contenere un bit di richiesta (Request Number, RN), per segnalare quale pacchetto è atteso.

Nel nostro progetto, il collegamento è modellato come un sistema a coda con un solo servente e coda infinita, con arrivi di pacchetti secondo un processo di Poisson con tasso λ e tempi di servizio esponenziali con media $1/\mu$. La trasmissione può fallire con probabilità p, e in tal caso il pacchetto viene ritrasmesso finché non viene ricevuto correttamente. Gli ACK vengono generati su un canale separato secondo un processo di Poisson con tasso δ, a cui abbiamo deciso di aggiungere anche un tempo di elaborazione medio esponenziale pari a $1/\mu_{ack}$, per modellare più realisticamente eventuali ritardi di processing da parte del ricevitore.

In questa configurazione, il tempo medio di attraversamento di un pacchetto (cioè il tempo che intercorre tra l'inizio della sua trasmissione e il ricevimento dell'ACK corretto) può essere calcolato considerando sia le ritrasmissioni dovute agli errori, sia il tempo di attesa dei riscontri. Se t_T è il tempo necessario a inviare un pacchetto e ricevere il relativo ACK senza errori, e la probabilità di errore sulla trama è p, allora il tempo medio di attraversamento t_v è dato dalla somma delle attese geometriche sulle ritrasmissioni:

$\mathbf{t_v = t_T / (1- p)}$ dove $\mathbf{t_T = t_{tx} + t_{ack} + t_{proct}}$ e i termini rappresentano rispettivamente:

- $t_{tx} = 1/\mu$: tempo medio di trasmissione del pacchetto;
- $t_{ack} = 1/\delta + 1/\mu_{ack}$: tempo medio di attesa e processing per il riscontro;
- t_{proct}: tempo medio di elaborazione locale (eventualmente trascurabile o incluso nei precedenti).

Compilazione e avvio

1. Aprire il terminale nella cartella `code`

cd …/code

2. Eseguire prima Make clean per accertarsi che eventuali file precedenti siano rimossi

make clean

3. Compilare il progetto con il comando seguente

make

4. Avviare il simulatore

./queue

5. Selezionare il simulatore desiderato

All'avvio, il programma mostrerà il menù seguente dove il primo simulatore ("Mode 1") è quello già presente nell'originale, mentre il secondo ("Mode 2") è stato implementato in questo progetto.

```
**********************************************************

                G/G/1 QUEUE SIMULATION PROGRAM

**********************************************************

                Select simulation mode:
                1 - Simple queue simulator
                2 - Stop-and-wait simulator
                Mode [1] >
```

Struttura del codice

Per introdurre la modalità Stop-and-Wait nel simulatore esistente, è stato necessario intervenire esclusivamente sul file contenente la funzione `main`. In particolare, è stato aggiunto un semplice meccanismo di selezione iniziale, che consente all'utente di scegliere tra due modalità operative: "1 - Simple queue simulator", per eseguire il simulatore originale senza alcuna modifica, e "2 - Stop-and-Wait simulator", per attivare la nuova modalità di simulazione. Nel primo caso, il comportamento del simulatore rimane invariato rispetto alla versione di partenza. Nel secondo caso, invece, il simulatore esegue la logica propria dello Stop-and-Wait, facendo uso delle nuove funzionalità implementate nei file `stopandwait.h` e `stopandwait.cpp`.

Il file `stopandwait.h` definisce la classe `StopAndWaitSimulator`, derivata dalla classe base `simulator`. La classe gestisce l'intero ciclo simulativo e include parametri fondamentali come il carico di traffico ρ (Erlang), la durata si servizio media 1/mu (s), la probabilità di errore p, il tasso di arrivo dei riscontri δ (1/s) e il tempo medio di servizio mu_{ack} (s). Oltre alla gestione dei parametri temporali e della durata della simulazione, la classe raccoglie statistiche su ritardi, numero di pacchetti trasmessi e ritrasmissioni. Sono inoltre presenti metodi ausiliari per l'inizializzazione, l'esecuzione e la raccolta dei risultati della simulazione.

Il file `stopandwait.cpp` implementa la logica definita nell'intestazione `stopandwait.h`. La classe `StopAndWaitSimulator` eredita dalla classe `simulator` e utilizza la libreria `easyio` per l'input interattivo dei parametri e la stampa dei risultati. Le funzioni principali sono:

- **`input()`**: legge i parametri del modello descritti sopra, oltre ai parametri della simulazione come durata transiente (s), durata run (s) e numero di run. Calcola da questi input i valori `lambda` e `mu`.
- **`run()`**: esegue la simulazione vera e propria. Per ogni run:
  - Genera il tempo inter-arrivo dei pacchetti secondo una distribuzione esponenziale negativa (modello Poisson).
  - Per ogni pacchetto, simula eventuali ritrasmissioni dovute ad errori. Ogni tentativo ha tempo di servizio esponenziale e probabilità di errore `p`.
  - Una volta ricevuto correttamente, aggiunge il tempo di attesa per l'ACK e il suo tempo di servizio, entrambi generati esponenzialmente.
  - Calcola e accumula il ritardo totale per ciascun pacchetto trasmesso con successo e aggiorna i contatori statistici.
- **`update_stats()`**: aggiorna l'oggetto statistico `Sstat` con il ritardo medio per run.
- **`print_trace()`** e **`results()`**: stampano rispettivamente i risultati parziali e finali. Oltre al ritardo medio simulato, `results()` calcola anche il valore teorico del ritardo medio secondo la formula:

  $$t_v = t_T / (1- p) \text{ dove } t_T = 1/\mu + 1/ \delta + 1/\mu_{ack}$$

  Dove t_T è il tempo massimo per ogni tentativo e t_v è il tempo medio per una trasmissione corretta, tenendo conto delle ritrasmissioni.

Inoltre, rispetto al simulatore originale, questa versione introduce, negli output finali, ulteriori informazioni utili per l'analisi delle prestazioni del sistema. In particolare, oltre al ritardo medio simulato, vengono calcolati:

- Il tempo teorico medio di attraversamento di un pacchetto (s), ottenuto considerando sia i tempi di trasmissione che quelli di ricezione dell'ACK, tenendo conto della probabilità di errore.
- La differenza tra il ritardo medio simulato e quello teorico (s), utile per valutare la coerenza della simulazione con il modello analitico di riferimento.
- Il numero totale di pacchetti ritrasmessi, che fornisce una misura diretta dell'impatto degli errori sul protocollo di trasmissione.

Per ulteriori dettagli sul funzionamento del codice, si rimanda alla consultazione dei file sorgente `stopandwait.h` e `stopandwait.cpp`, spiegati attraverso commenti.

Risultati

I risultati della simulazione sono stati analizzati attraverso quattro grafici, realizzati con codice python a partire dai dati raccolti dalle simulazioni, ognuno dei quali mette in evidenza l'effetto di un parametro diverso sul ritardo medio in coda e sui relativi intervalli di confidenza.

Grafico 1 - Sistema a coda semplice

L'asse delle ascisse rappresenta il carico di traffico ρ, mentre l'asse delle ordinate mostra il ritardo medio in coda. Si osserva che all'aumentare di ρ cresce in modo significativo sia il ritardo medio che gli intervalli di confidenza. Questo comportamento è coerente con la teoria delle code, poiché un aumento del carico comporta un maggior affollamento e quindi un aumento del tempo di attesa.

Ritardo medio = y
ρ = x
$1/\mu$ = 0.1 s
p = /
δ = /
$1/\mu_{ack}$ = /
ρ_{ack} = /
Simulation transient len = 1000 s
Simulation RUN len = 1000 s
Simulation number of RUNs = 10

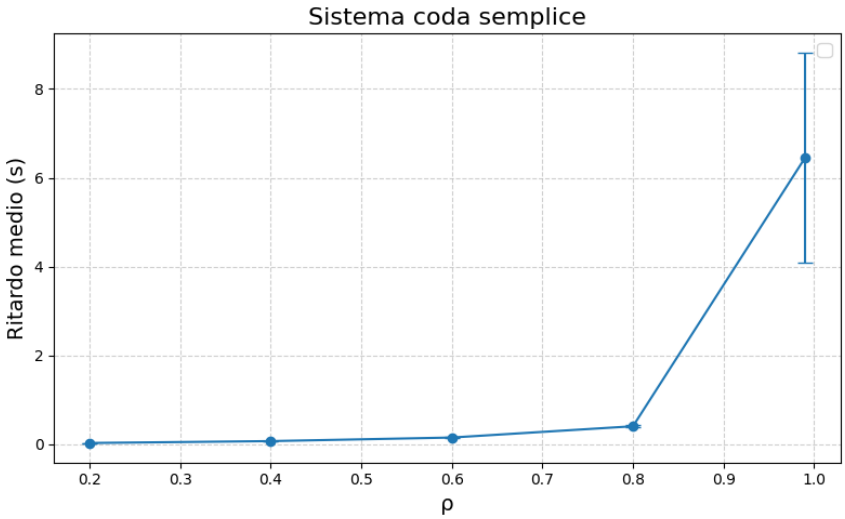


Grafico 2 - Sistema Stop-and-Wait, variando ρ

Anche in questo caso l'asse x indica il carico di traffico ρ, mentre sull'asse y è riportato il ritardo medio in coda. Tuttavia, mantenendo costanti i parametri relativi agli ACK e alla probabilità di errore, si nota che il ritardo medio e i suoi intervalli di confidenza rimangono sostanzialmente costanti al variare di ρ. Ciò accade perché la velocità di uscita del sistema è limitata dalla velocità di arrivo degli ACK, che è indipendente da ρ, causando un collo di bottiglia nel sistema.

Ritardo medio = y
ρ = x
$1/\mu$ = 0.1 s
p = 0.01
δ = 1 s^{-1}
$1/\mu_{ack}$ = 0.1 s
ρ_{ack} = 0.1 Erlang
Simulation transient len = 1000 s
Simulation RUN len = 1000 s
Simulation number of RUNs = 10

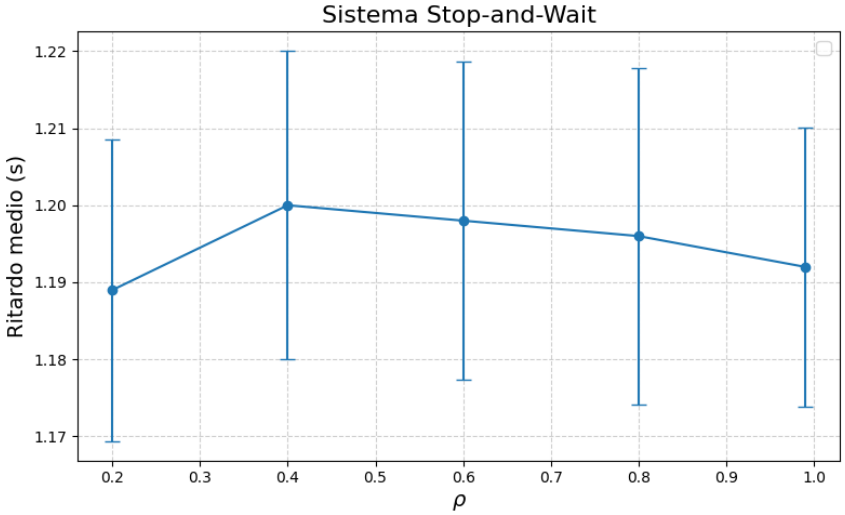


Grafico 3 - Sistema Stop-and-Wait, variando il carico degli ACK ρack

In questo grafico l'asse x rappresenta il carico di traffico relativo agli ACK, mentre sull'asse y è riportato il ritardo medio in coda. Tenendo costanti il carico dei pacchetti e la probabilità di errore, si osserva che all'aumentare di ρ_{ack} il ritardo medio e i relativi intervalli di confidenza diminuiscono. Questo perché un maggior numero di ACK disponibili riduce il tempo di attesa per la conferma di ricezione, velocizzando così la trasmissione complessiva.

Ritardo medio = y
ρ = 0.4 Erlang
$1/\mu = 0.4$ s
p = 0.01
δ = x
$1/\mu_{ack} = 1$ s
$\rho_{ack} = \delta/\mu_{ack} = \delta/1 = x$
Simulation transient len = 1000 s
Simulation RUN len = 1000 s
Simulation number of RUNs = 10

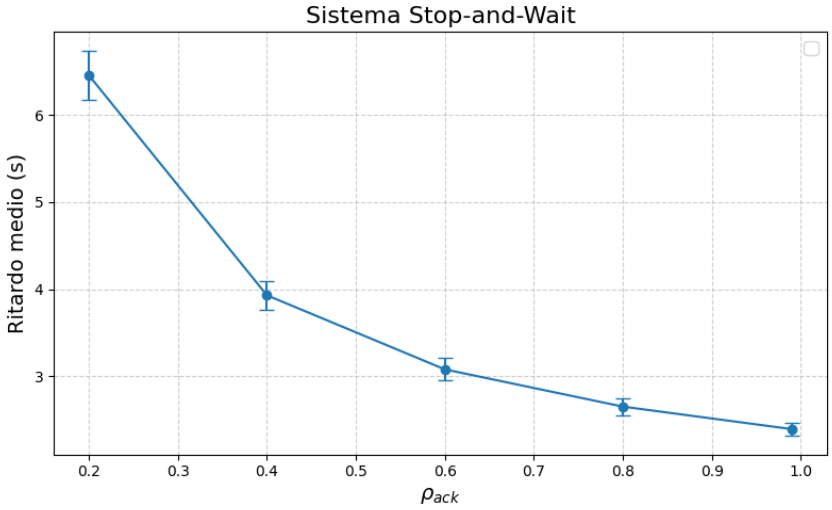
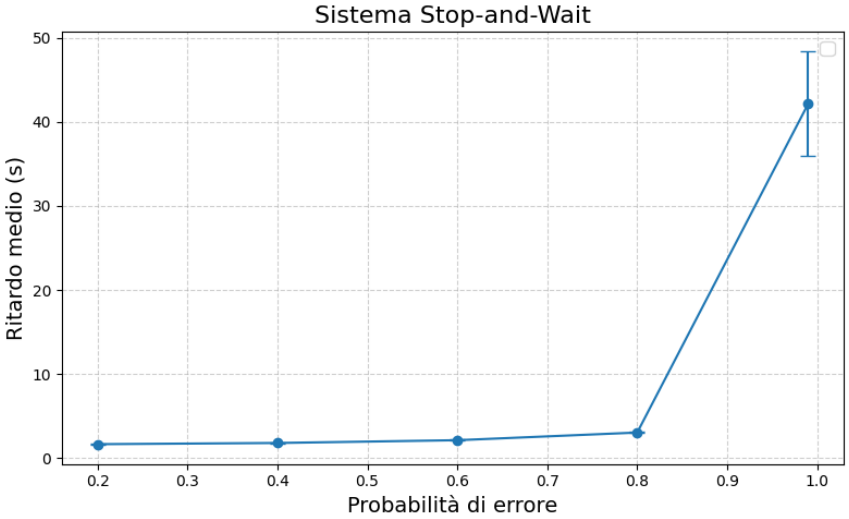


Grafico 4 - Sistema Stop-and-Wait, variando la probabilità di errore p

L'asse x indica la probabilità di errore, mentre sull'asse y si riporta il ritardo medio in coda. Mantenendo costanti il carico di pacchetti e il carico degli ACK, si osserva che all'aumentare della probabilità di errore il ritardo medio e i suoi intervalli di confidenza aumentano. Questo incremento è dovuto al maggior numero di ritrasmissioni necessarie per garantire la corretta ricezione dei dati, che comportano inevitabilmente un aumento del tempo totale di attraversamento.

Ritardo medio = y
ρ = 0.4 Erlang
$1/\mu = 0.4$ s
p = x
$\delta = 1\ s^{-1}$
$1/\mu_{ack} = 0.1$ s
$\rho_{ack} = 0.1$ Erlang
Simulation transient len = 1000 s
Simulation RUN len = 1000 s
Simulation number of RUNs = 10



Conclusioni

Il progetto ha permesso di analizzare e confrontare il comportamento di un sistema a coda semplice con quello di un protocollo Stop-and-Wait, evidenziando i vantaggi e i limiti di entrambi. In particolare, è emerso che la gestione degli ACK e la probabilità di errore influenzano in modo significativo il ritardo medio in coda: una frequenza elevata di ACK riduce i tempi di attesa, mentre un'elevata probabilità di errore comporta ritrasmissioni frequenti e quindi maggiori ritardi. Sebbene il modello a coda semplice risulti più performante in condizioni ideali, il protocollo Stop-and-Wait offre maggiore robustezza e realismo in scenari di rete con perdita di pacchetti.

Il simulatore rappresenta una base flessibile per future estensioni e approfondimenti. A partire da questa struttura, sarà possibile aggiungere altri protocolli di comunicazione come Go-Back-N e Selective Repeat, ampliando così lo spettro delle tecnologie analizzabili. Inoltre, potranno essere integrati strumenti per facilitare il confronto diretto tra le diverse tecniche.