Università di Roma Tor Vergata
Corso di Laurea triennale in Informatica

Sistemi operativi e reti

A.A. 2016-17

Pietro Frasca

Parte II: Reti di calcolatori

Lezione 13 (37)

Venerdì 21-04-2017

Trasferimento affidabile dei dati

- Il protocollo IP dello strato di rete è inaffidabile: non garantisce né la consegna dei **datagram** e nemmeno l'integrità dei dati in essi contenuti. I datagram possono arrivare fuori ordine, e i bit nei datagram possono subire alterazioni passando da 0 a 1 e viceversa. Poiché i segmenti dello strato di trasporto costituiscono il **campo dati** dei datagram IP, anch'essi possono essere soggetti a questi problemi.
- Il TCP realizza un **servizio di trasferimento affidabile dei dati** utilizzando il servizio inaffidabile fornito da IP.
- Il servizio di trasferimento affidabile dati di TCP assicura che il flusso di dati inviato da un mittente è esattamente lo stesso che giunge al destinatario.

- Vediamo come il TCP realizza un trasferimento affidabile di dati, descrivendo un caso semplificato in cui un mittente TCP ritrasmette segmenti solo allo scadere del timeout. Faremo in seguito una descrizione più completa che usa riscontri duplicati oltre ai timeout per recuperare i segmenti persi.
- Il seguente pseudo codice è relativo a una descrizione molto semplificata di un **mittente TCP**, non considerando la frammentazione del messaggio, il controllo del flusso e della congestione.

```
numSequenza_min = numSequenza_iniziale
numSequenza = numSequenza_iniziale
while (true) {
  switch(evento)
    case evento1: /* dati ricevuti dallo strato applicativo (messaggio) */
      <crea il segmento TCP con il numero di sequenza numSequenza>
      if (<timer non è avviato>)
            <avvia il timer>
      <passa il segmento a IP>
      numSequenza = numSequenza + lunghezza(dati)
      break;
    case evento2: /* timer timeout (il timeout è dato dal valore di
     intervalloTimeout) */
      <ritrasmette il  segmento non ancora riscontrato
        con numero di sequenza sequenza_min (il più piccolo numero di
     sequenza)>
      <avvia il timer>
      break;
    case evento3: /* ACK ricevuto, con valore del campo numero di riscontro
     = numRiscontro */
      if (numRiscontro > numSequenza_min) {
        numSequenza_min = numRiscontro
        if ( <ci sono segmenti non ancora riscontrati>)
          <avvia timer>
      }
}
```

- Vediamo che ci sono tre principali eventi legati alla trasmissione e ritrasmissione dei dati nel mittente TCP:
  1. **ricezione di dati dall'applicazione;**
  2. **timeout del timer;**
  3. **ricezione di un ACK.**

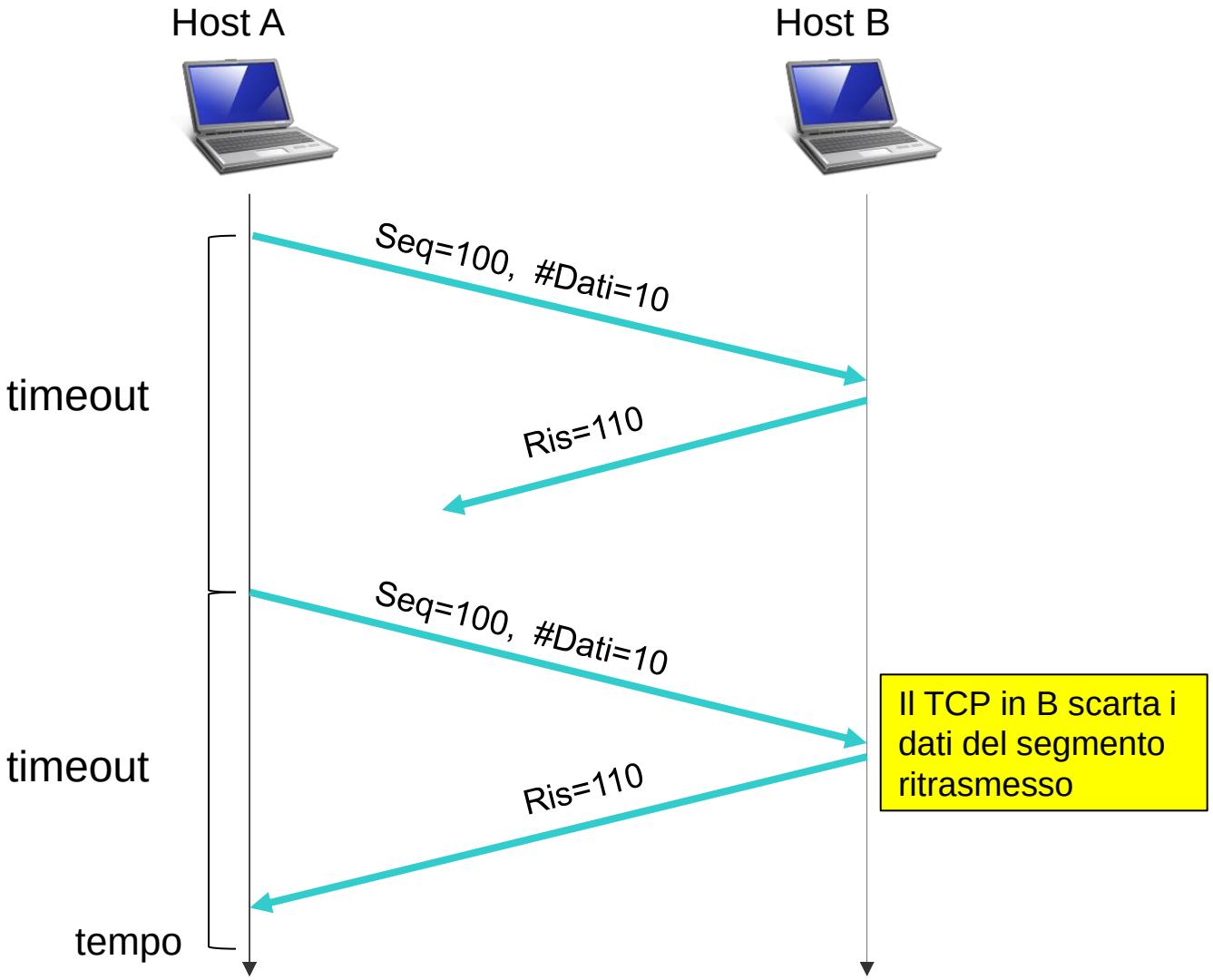
1. Al verificarsi del primo degli eventi principali, il TCP riceve i dati dall'applicazione, li incapsula in un segmento, e passa il segmento a IP. Ogni segmento ha un numero di sequenza. Se il timer non è già stato avviato per qualche altro segmento, il TCP avvia il timer nel momento in cui il segmento viene passato a IP. Il valore di scadenza per questo timer è dato da **intervalloTimeout**, che è calcolato da **RTTstimato** e **DevRTT** come descritto precedentemente.

2. Il secondo evento principale è l'evento **timeout**. Il TCP risponde all'evento timeout ritrasmettendo il segmento che ha causato il timeout stesso. Il TCP quindi riavvia il timer.
3. Il terzo evento è l'arrivo di un segmento di riscontro (ACK) dal ricevente. Al verificarsi di questo evento, il TCP confronta il valore **numRiscontro** contenuto nel campo numero di riscontro con la sua variabile **numSequenza_min**. La variabile di stato **numSequenza_min** è il **numero di sequenza del più vecchio byte non riscontrato**. Come indicato in precedenza il TCP usa riscontri cumulativi, così che **numRiscontro** riscontra la ricezione di tutti i byte prima del byte numero **numRiscontro**. Se **numRiscontro > numSequenza_min**, allora ACK sta riscontrando uno o più segmenti non riscontrati prima. Quindi il TCP mittente aggiorna la sua variabile **numSequenza_min**. Inoltre riavvia il timer se ci sono segmenti attualmente non ancora riscontrati.

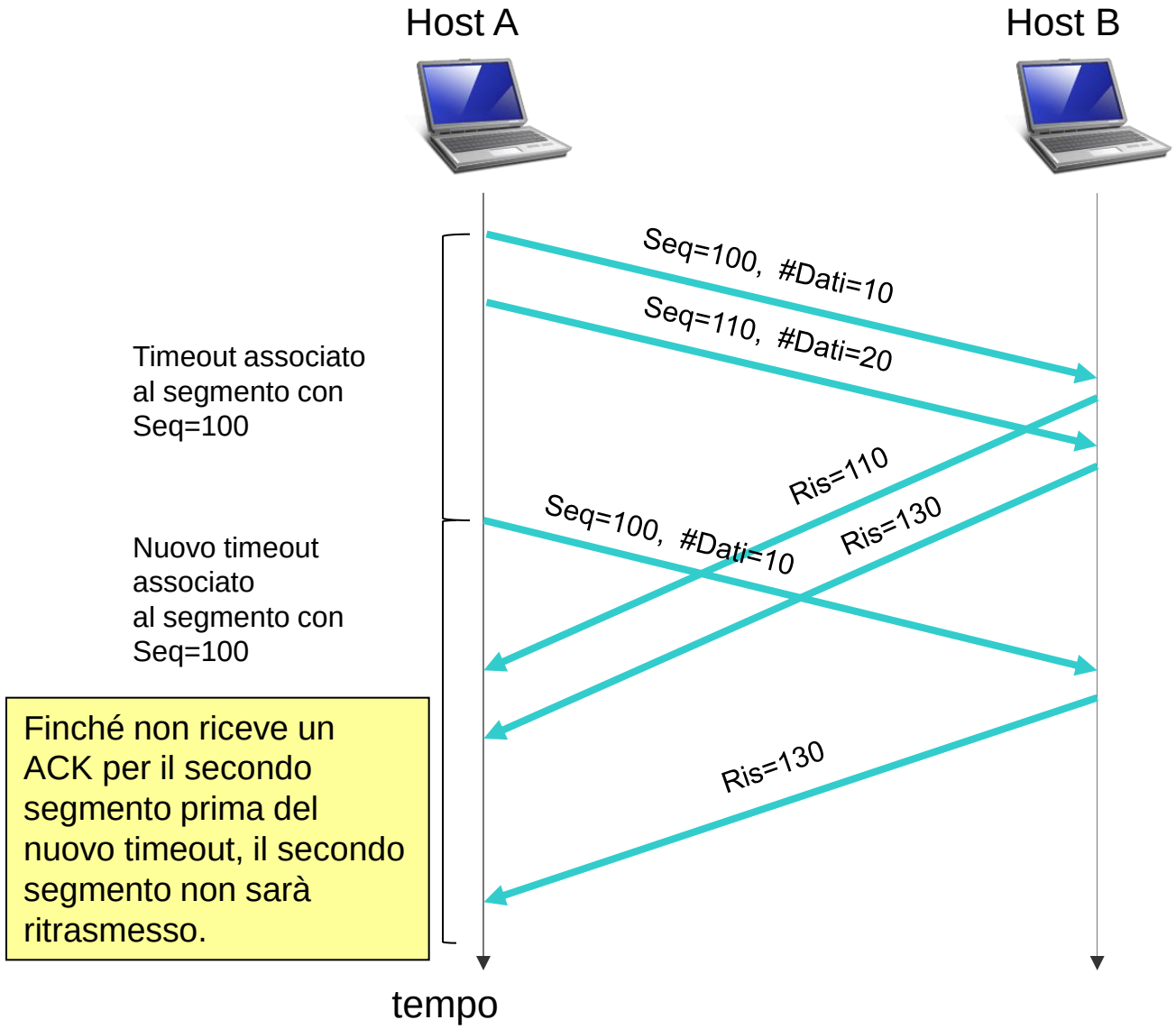
Alcuni tipici scenari

- Per avere un idea più chiara sul funzionamento del TCP esaminiamo ora alcuni tipici scenari.
- Il primo scenario è illustrato nella figura seguente in cui l'host **A** invia un segmento all'host **B**.
- Supponiamo che questo segmento abbia numero di sequenza 100 e che contenga 10 byte di dati.
- Dopo l'invio di questo segmento, l'host **A** aspetta da **B** un segmento con il numero di riscontro uguale a 110. Nonostante il segmento di **A** sia stato ricevuto da **B**, il riscontro da **B** ad **A** si è perso. In questo caso il timer scade, e l'host **A** ritrasmette lo stesso segmento. Ovviamente, quando l'host **B** riceve la ritrasmissione, rileverà dal numero di sequenza che il segmento è già stato ricevuto. Allora, il TCP nell'host **B** scarterà i dati contenuti nel segmento ritrasmesso.

Scenario 1: ritrasmissione a causa di un riscontro perso.

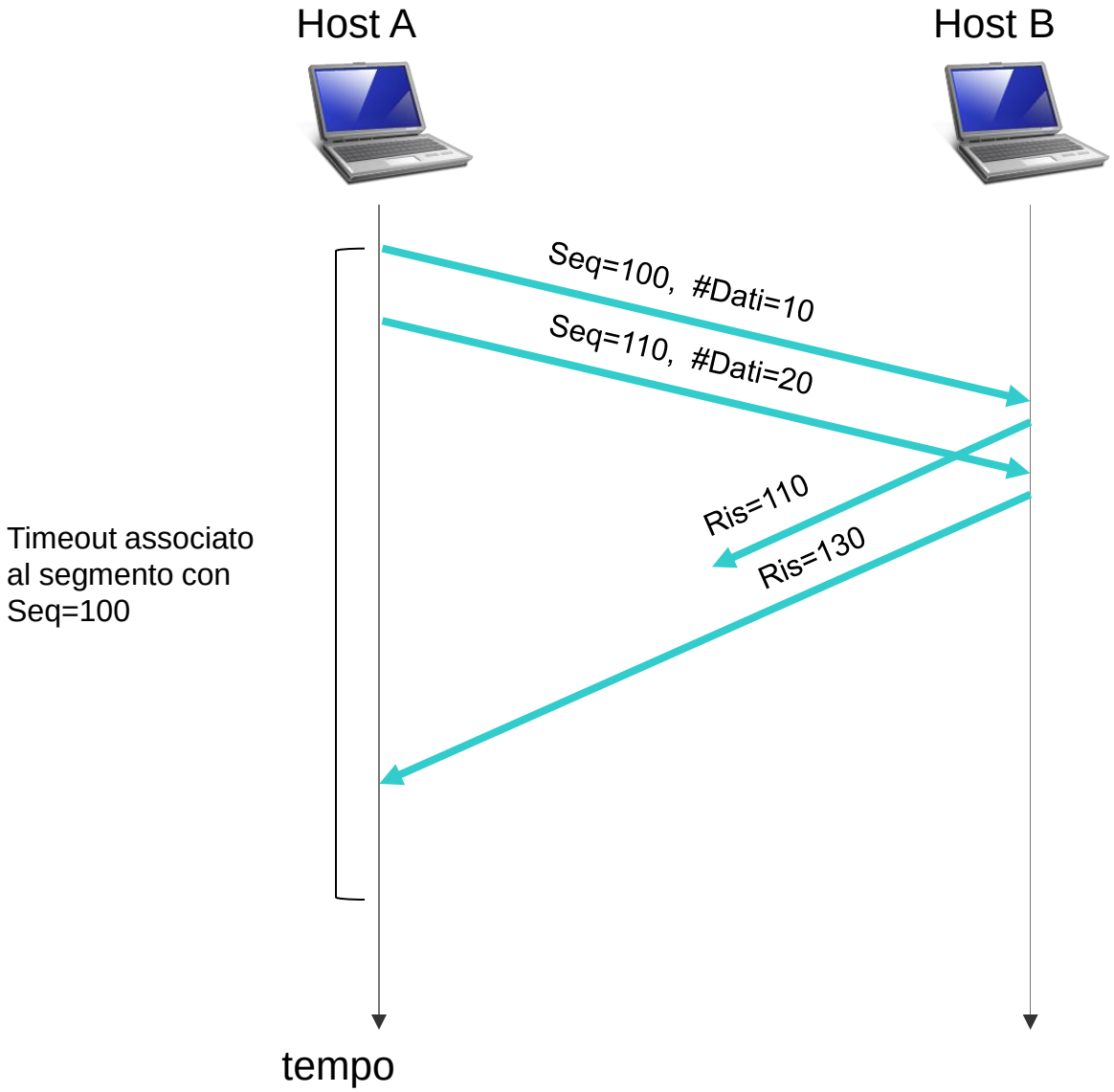


Scenario 2: segmento non ritrasmesso.



- Il primo segmento ha numero di sequenza 100 e 10 byte di dati. Il secondo segmento ha numero di sequenza 110 e 20 byte di dati. Supponiamo che entrambi i segmenti arrivino a B e che B invii due riscontri separati per ciascuno di questi segmenti. Il primo di questi riscontri ha numero di riscontro 110; il secondo ha numero di riscontro 130. Supponiamo ora che nessuno dei riscontri arrivi all'host A prima del timeout del primo segmento. Quando il timer scade, l'host A rispedisce il primo segmento con numero di sequenza 100 e riavvia il timer. **Finché non riceve un ACK per il secondo segmento prima del nuovo timeout, il secondo segmento non sarà ritrasmesso.**

Scenario 3: riscontro cumulativo che evita la ritrasmissione del primo segmento.



- Il riscontro per il primo segmento si perde nella rete, ma appena prima del timeout di questo segmento, l'host A riceve un riscontro con numero di riscontro 130. L'host A perciò sa che l'host B ha ricevuto tutti i dati fino al byte 129 e non rispedisce nessuno dei due segmenti.

Raddoppio dell'intervallo di timeout

- Quando si verifica un evento di timeout, il TCP ritrasmette il segmento non ancora riscontrato che ha il più piccolo numero di sequenza, come descritto prima.
- Ma a ogni ritrasmissione il TCP raddoppia il valore del successivo timeout rispetto al valore precedente, invece di determinarlo con la relazione:

  intervalloTimeout = RTTstimato + 4·DevRTT

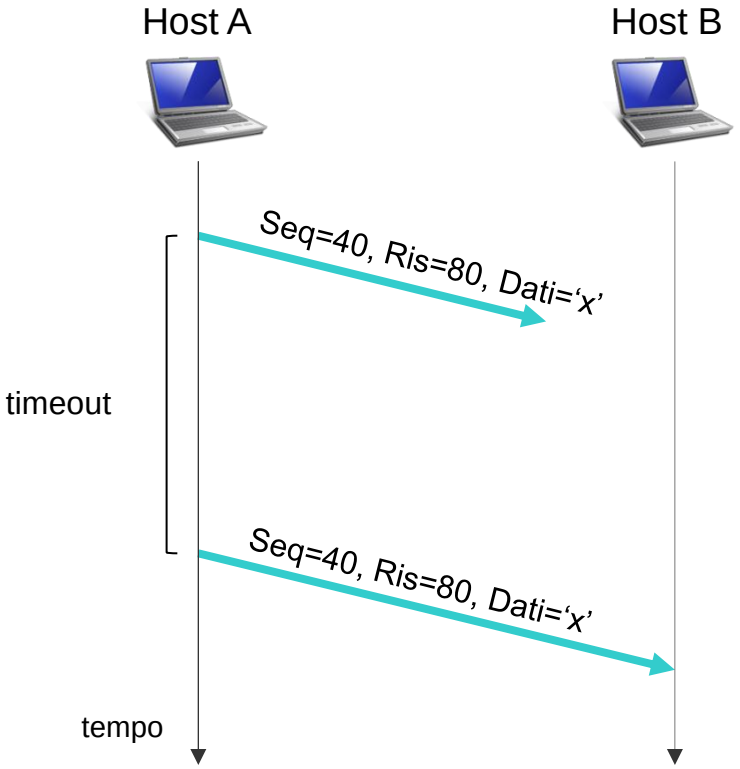
  Ad esempio, supponiamo che all'evento di timeout il valore di **intervalloTimeout** relativo al segmento non riscontrato, con numero di sequenza più piccolo, sia **0.25 s**. Il TCP ritrasmetterà questo segmento e assegnerà il nuovo valore di timeout a **0.5 s**. Se si verifica un nuovo timeout, dopo 0.5 secondi, il TCP ritrasmette di nuovo il segmento, assegnando ora a intervalloTimeout il valore di **1 s**.

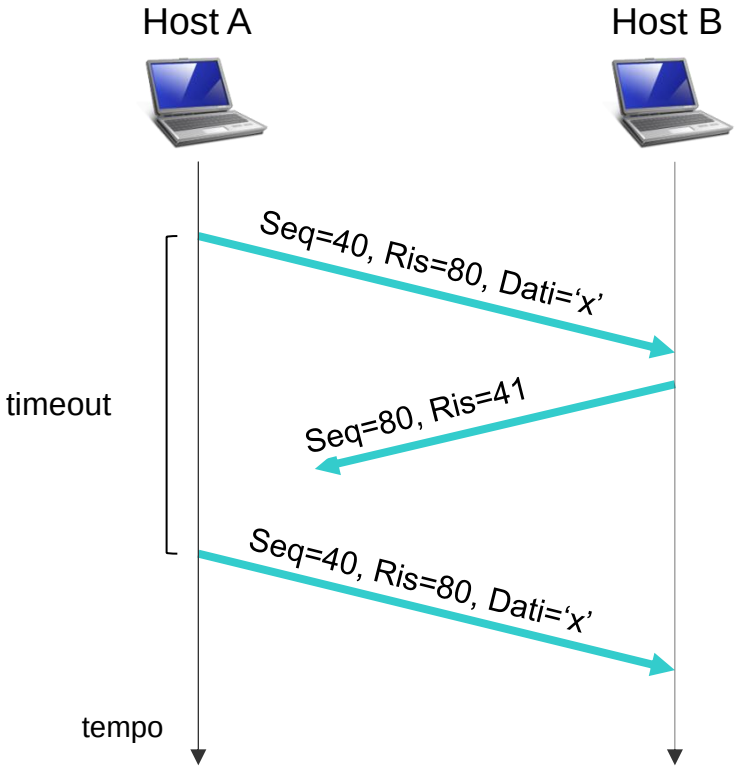
  In tal modo, **gli intervalli crescono esponenzialmente dopo ogni ritrasmissione**.

Tuttavia, se il timer viene riavviato per via dell'evento di ricezione di dati dall'applicazione o dall'evento di ricezione di un ACK, **intervalloTimeout** viene calcolato in base alla relazione descritta precedentemente.

Ritrasmissione rapida

- Un importante problema relativo alle ritrasmissioni riguarda la stima del periodo di timeout che potrebbe risultare troppo lungo.
- Quando si perde un segmento, si genera un lungo periodo di timeout che costringe il mittente a ritardare il rinvio del segmento perso, aumentando di conseguenza il ritardo di trasmissione.



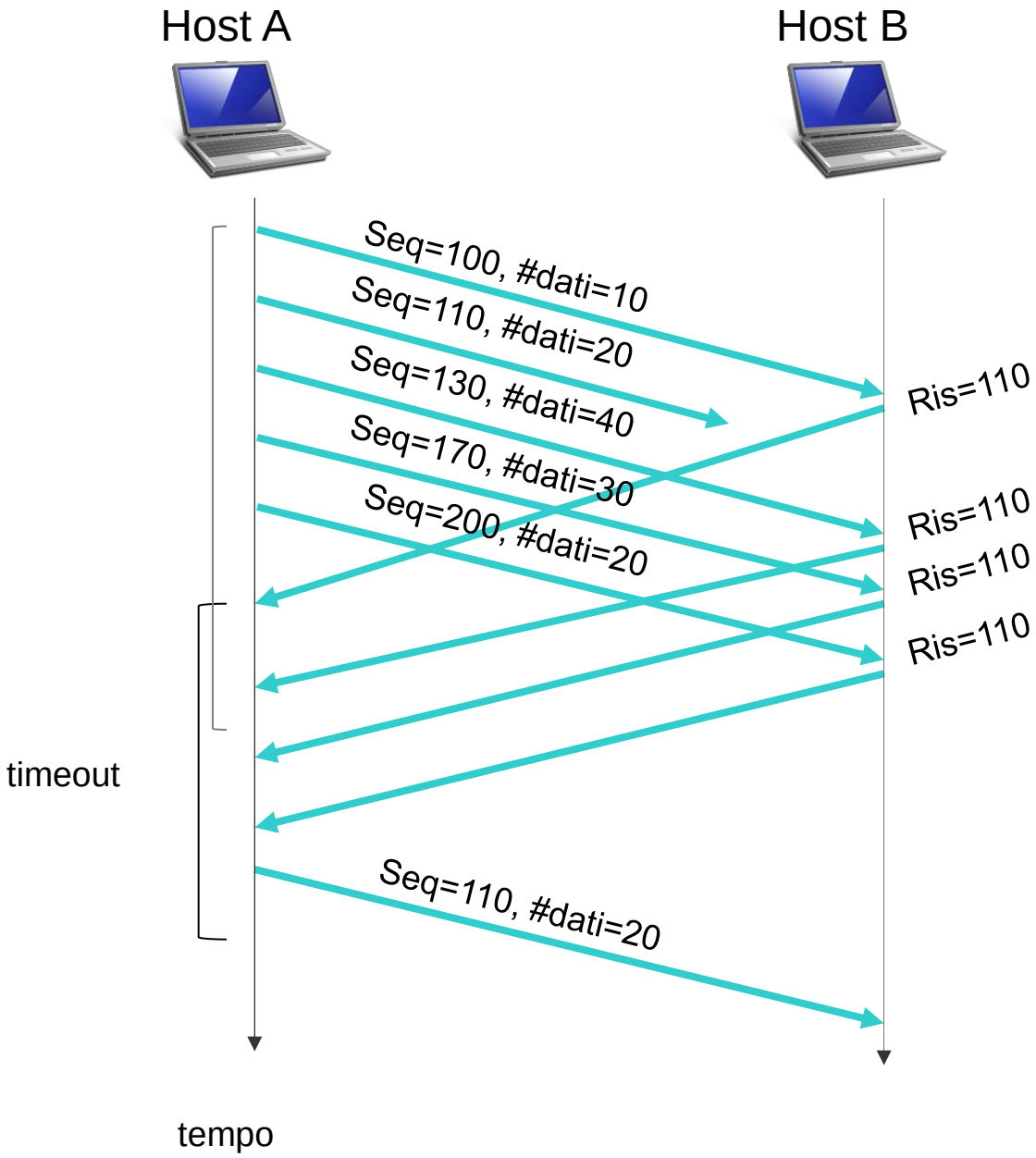


Spesso però, il mittente TCP può rilevare la perdita dei segmenti **prima che si verifichi l'evento di timeout** grazie agli **ACK duplicati** che riscontrano un segmento il cui ACK è già stato ricevuto dal mittente.

- Quando il destinatario TCP riceve un segmento con numero di sequenza superiore al numero di sequenza atteso, rileva che uno o più segmenti non sono arrivati. Il destinatario allora riscontra di nuovo il segmento che ha ricevuto in ordine corretto, duplicando così un ACK.
- Poiché il mittente spesso invia più segmenti, uno dopo l'altro, se un segmento si perde ci saranno probabilmente vari ACK duplicati.
- Se il mittente TCP riceve tre ACK duplicati per lo stesso segmento, considera questo evento come prova che il segmento, che ha inviato dopo il segmento riscontrato tre volte, è andato perso.

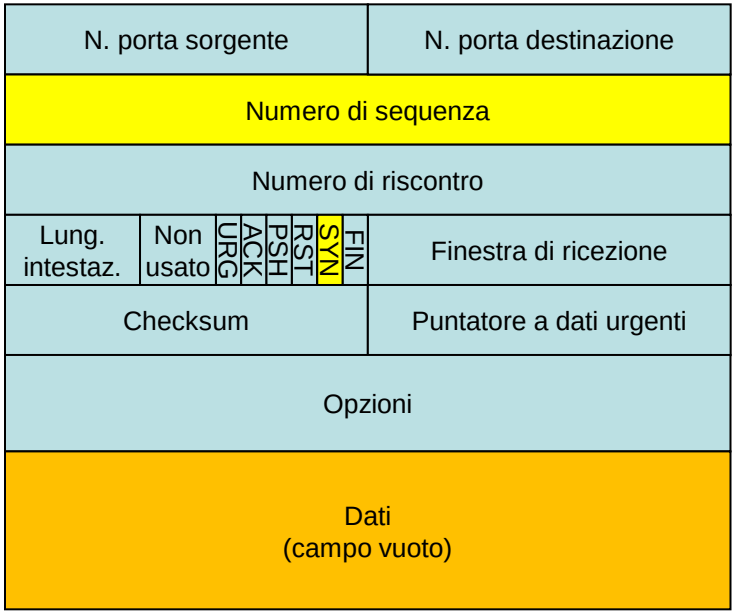
- Nel caso in cui siano stati ricevuti tre ACK duplicati, il mittente TCP effettua una **ritrasmissione rapida**, rinviando il segmento perso prima che scada il timer (timeout).
- Le procedure precedentemente descritte relative ai timer TCP sono complesse e hanno subito molti cambiamenti e sono il risultato di più venti anni di esperienza.

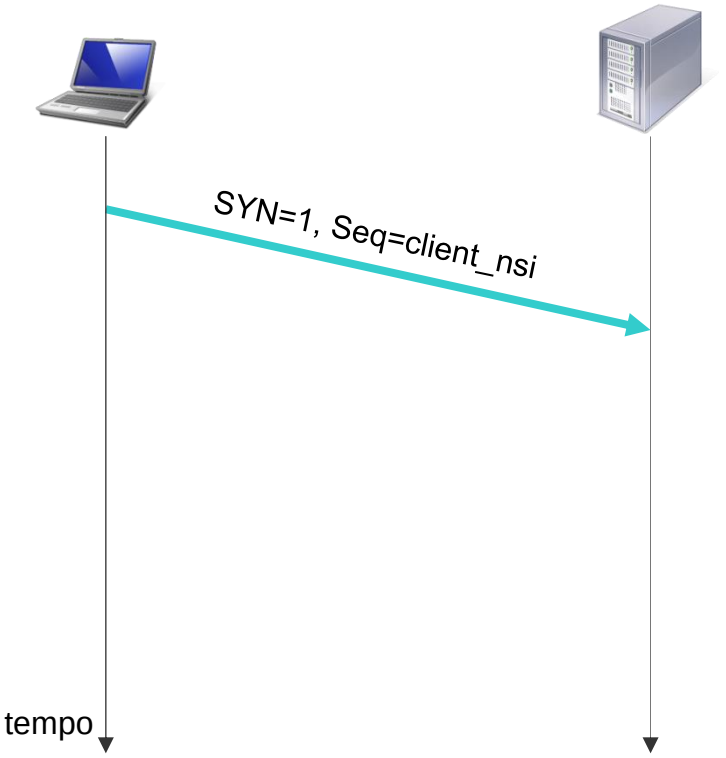
```
evento3: /* ACK ricevuto, con valore del campo numero di riscontro = numRiscontro */
if (numRiscontro > numSequenza_min) {
   numSequenza_min = numRiscontro
   if ( <ci sono segmenti non ancora riscontrati>)
      <avvia timer>
   }
else { /* un ACK duplicato per un segmento già riscontrato */
   <incrementa il numero di ACK duplicati ricevuti per numRiscontro>
   if ( <numero di ACK duplicati per numRiscontro == 3>) {
      /* ritramissione rapida */
      <rinvia il segmento con numero di sequenza numRiscontro>
    }
 }
```


Host A
Host B
Seq=100, #dati=10
Seq=110, #dati=20
Seq=130, #dati=40
Seq=170, #dati=30
Seq=200, #dati=20
Ris=110
Ris=110
Ris=110
Ris=110
timeout
Seq=110, #dati=20
tempo

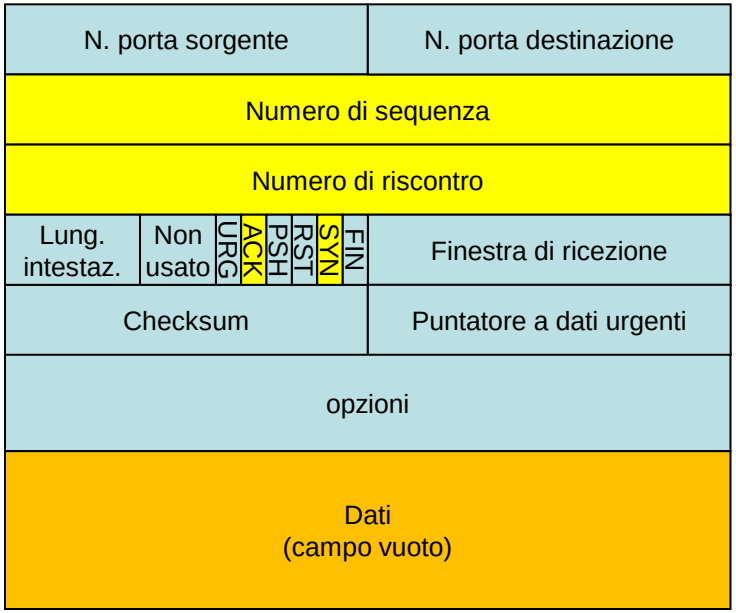
Instaurazione della connessione TCP

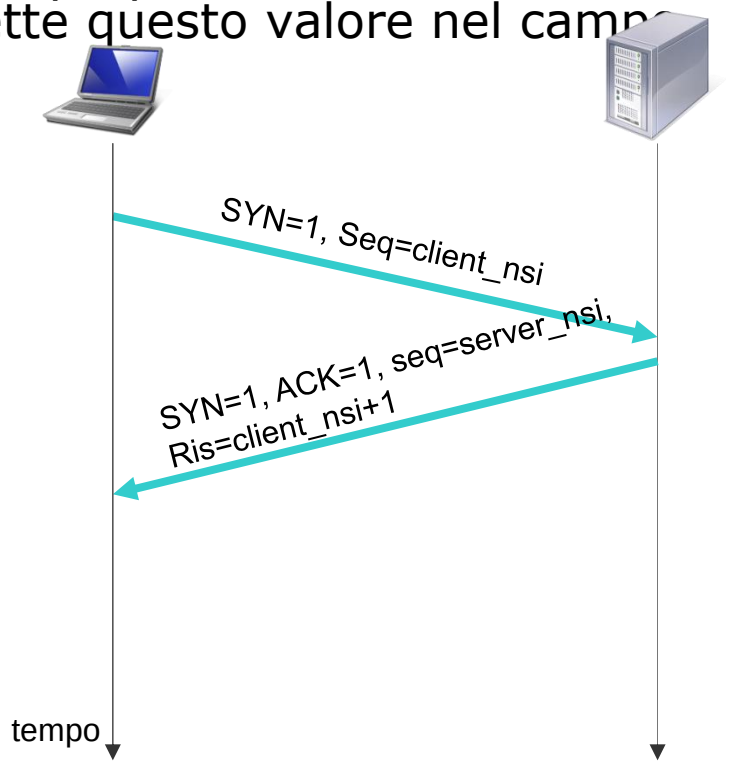
- Il TCP lato client stabilisce una connessione con il TCP lato server nel seguente modo:
- **Passo 1**. Il client invia uno speciale segmento al server, detto **segmento SYN,** che è caratterizzato dall'avere il flag **SYN = 1** e non contiene dati dello strato di applicazione. Inoltre, il client genera casualmente un numero di **sequenza iniziale (client_nsi**) e lo inserisce nel campo **numero di sequenza**.



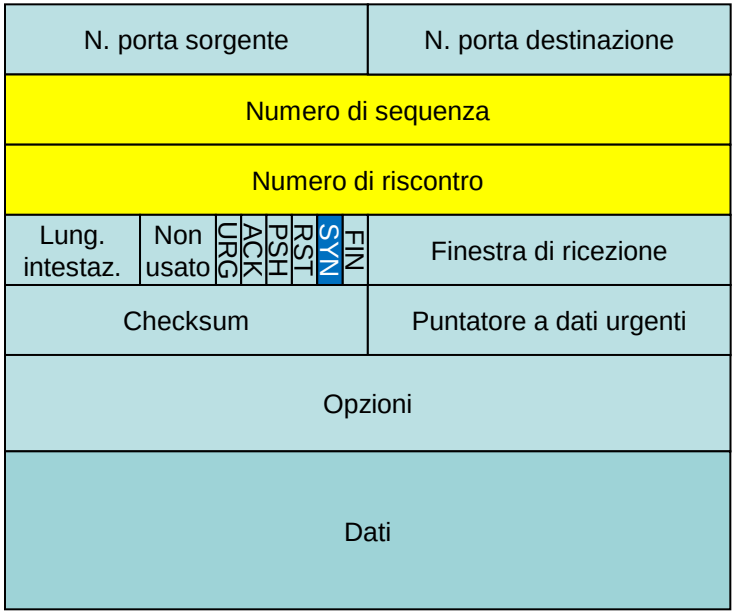


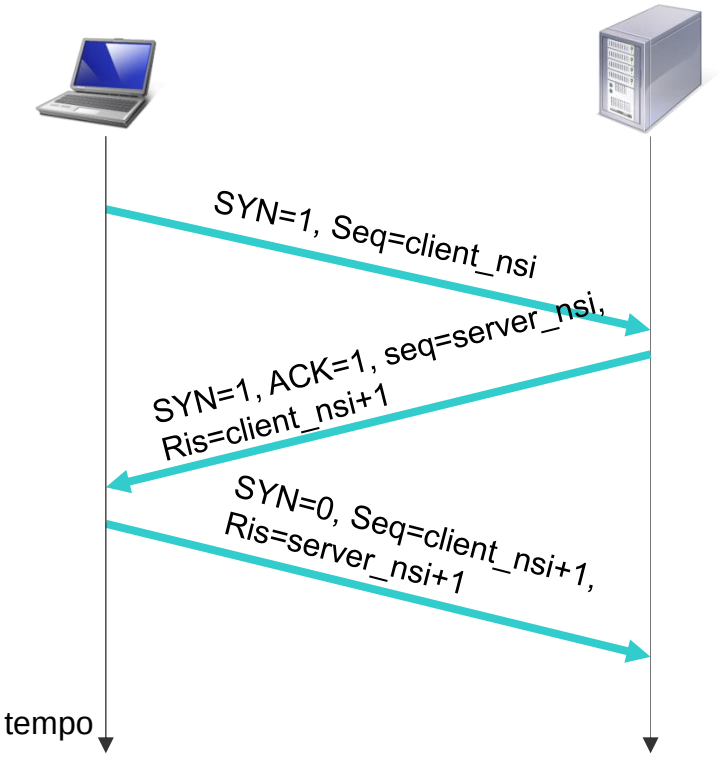
- **Passo 2**. Il server TCP, quando riceve il **segmento SYN** dal client, crea le variabili e i buffer per la connessione, e invia al client TCP un segmento, detto **segmento SYNACK**, che autorizza la connessione. Anche questo **segmento** non contiene dati dello strato di applicazione. La sua intestazione è caratterizzata dall'avere:
  - i flag **SYN e ACK settati a 1**
  - il campo **numero di riscontro** è posto a **client_nsi + 1**.
  - il server genera casualmente il proprio numero iniziale di sequenza **(server_nsi**) e mette questo valore nel campo **numero di sequenza**.



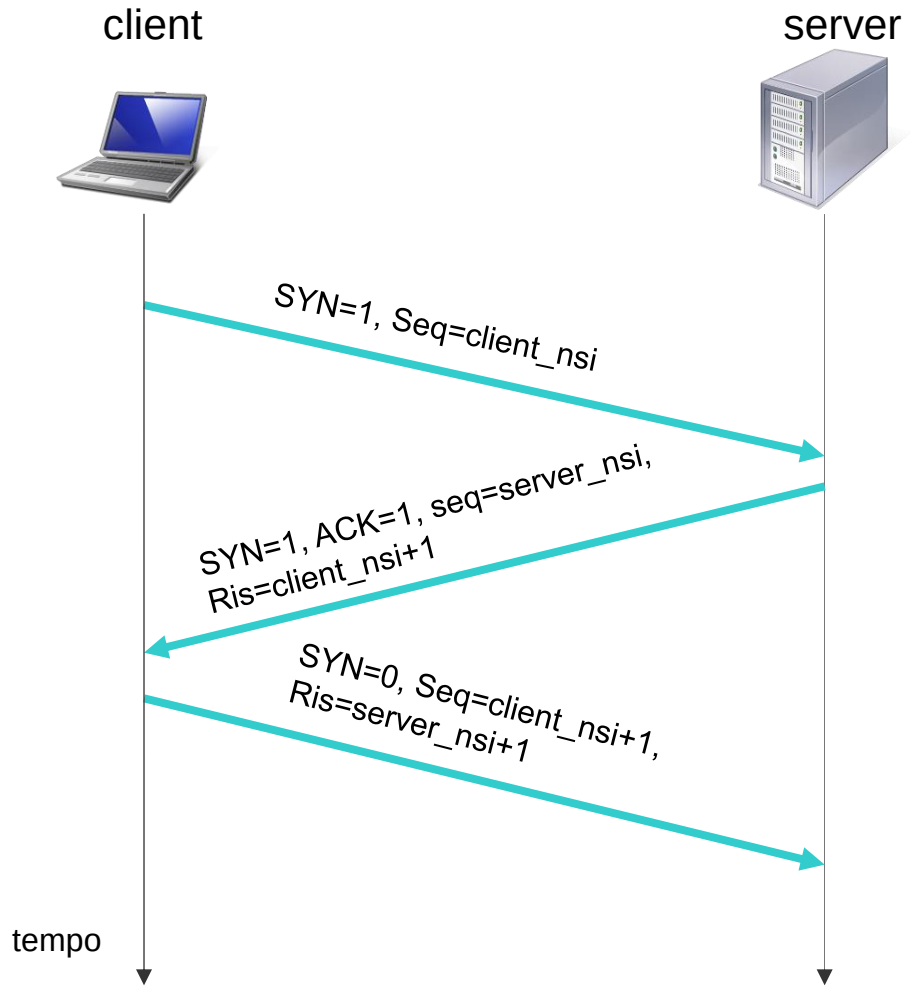


- **Passo 3**. Dopo la ricezione del segmento **SYNACK**, anche il **client** crea variabili e buffer per la connessione. Il client invia allora al server un ulteriore segmento che **riscontra il segmento SYNACK,** inserendo il valore **server_nsi + 1** nel campo numero di riscontro dell'intestazione del segmento TCP. Il bit **SYN è posto a 0,** poiché la connessione è stabilita. Questo segmento può contenere dati dell'applicazione.

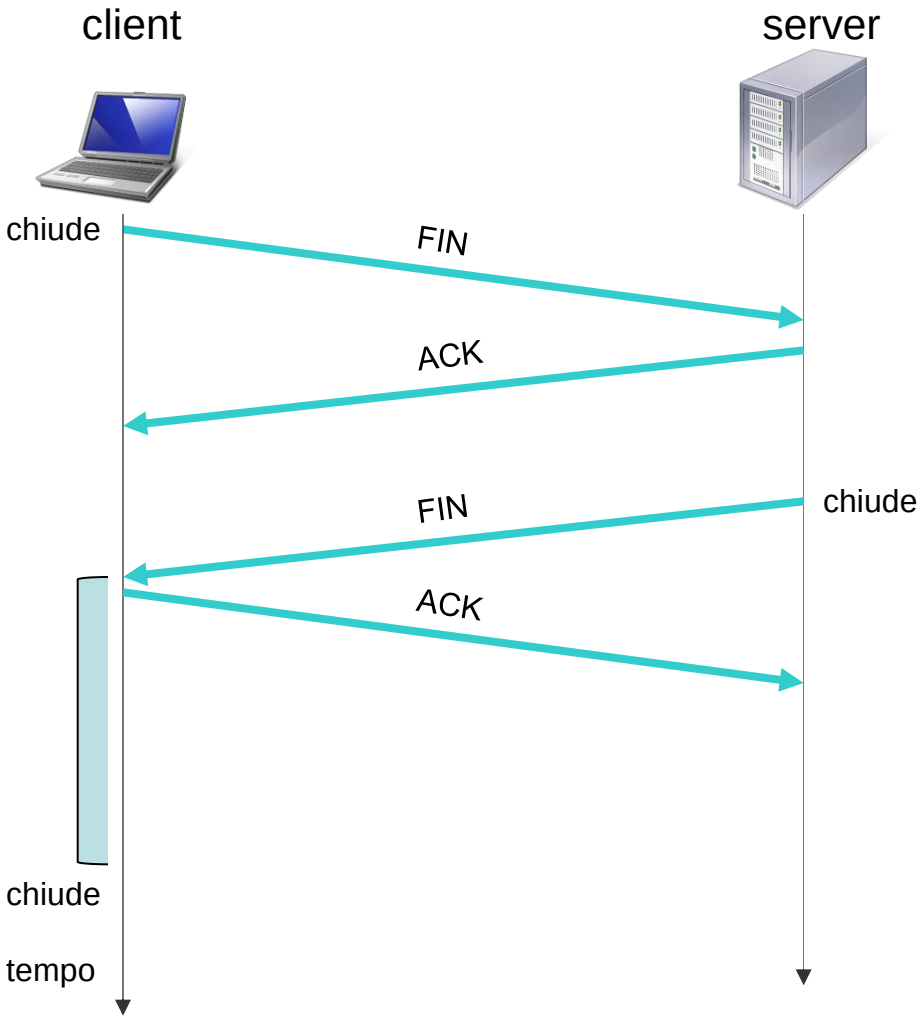




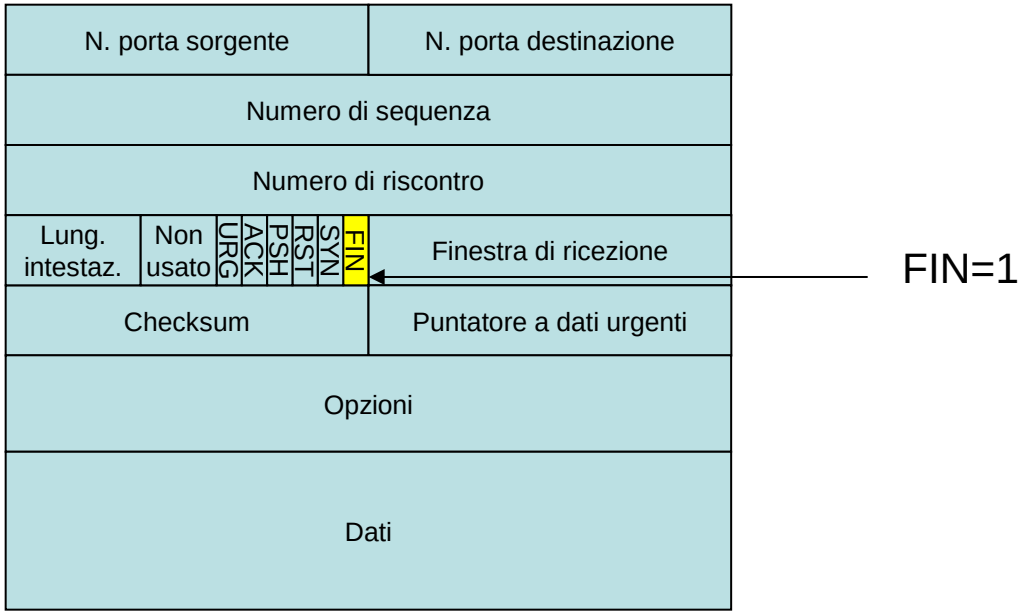
- Quindi, una volta completati i primi due passi, client e server possono scambiarsi segmenti contenenti dati.
- Tutti i segmenti successivi avranno il bit SYN a zero.



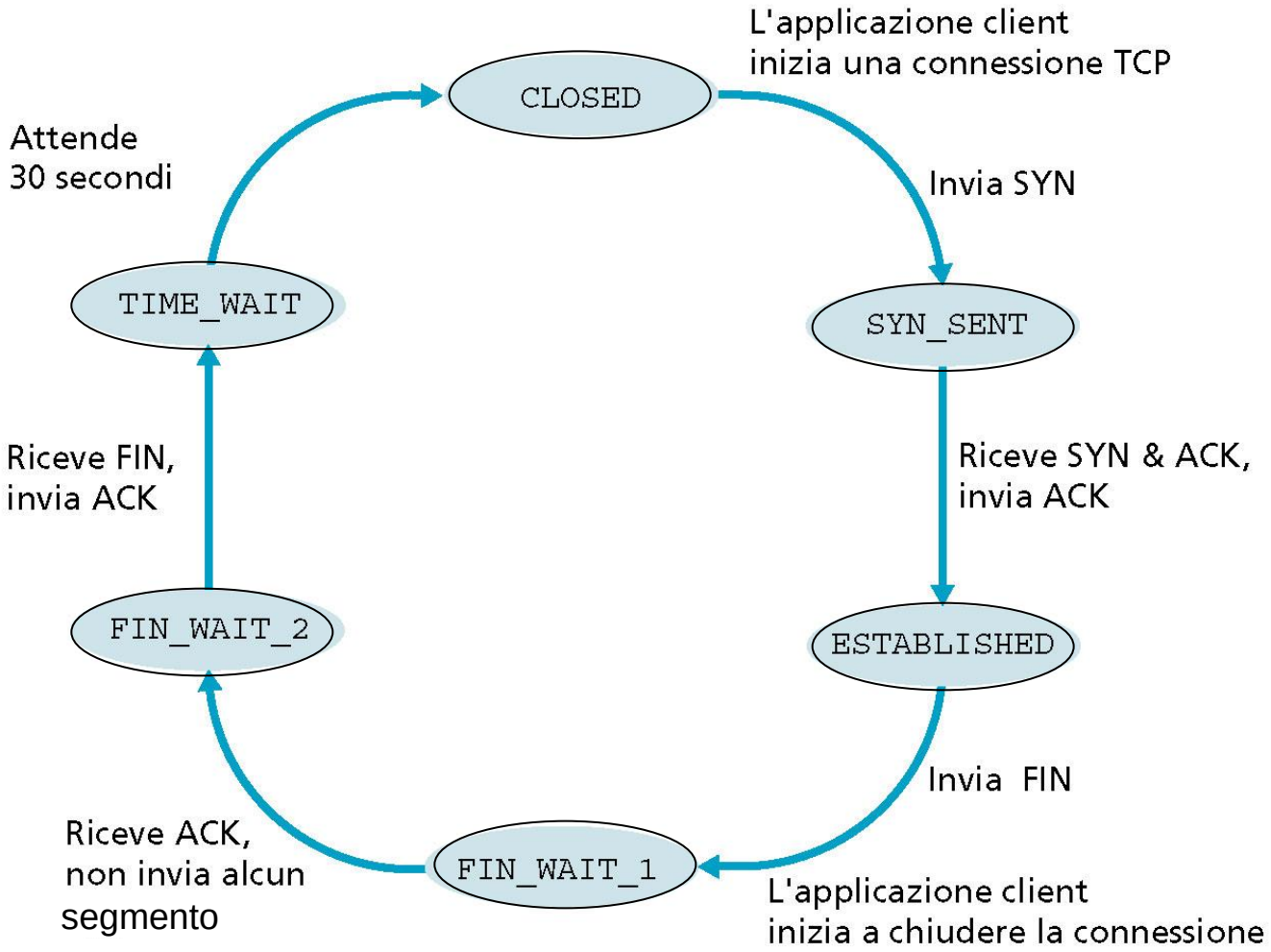
- Uno dei due processi può chiudere la connessione. Al termine della connessione, le "risorse" allocate dai processi, cioè variabili e buffer vengono rilasciate.

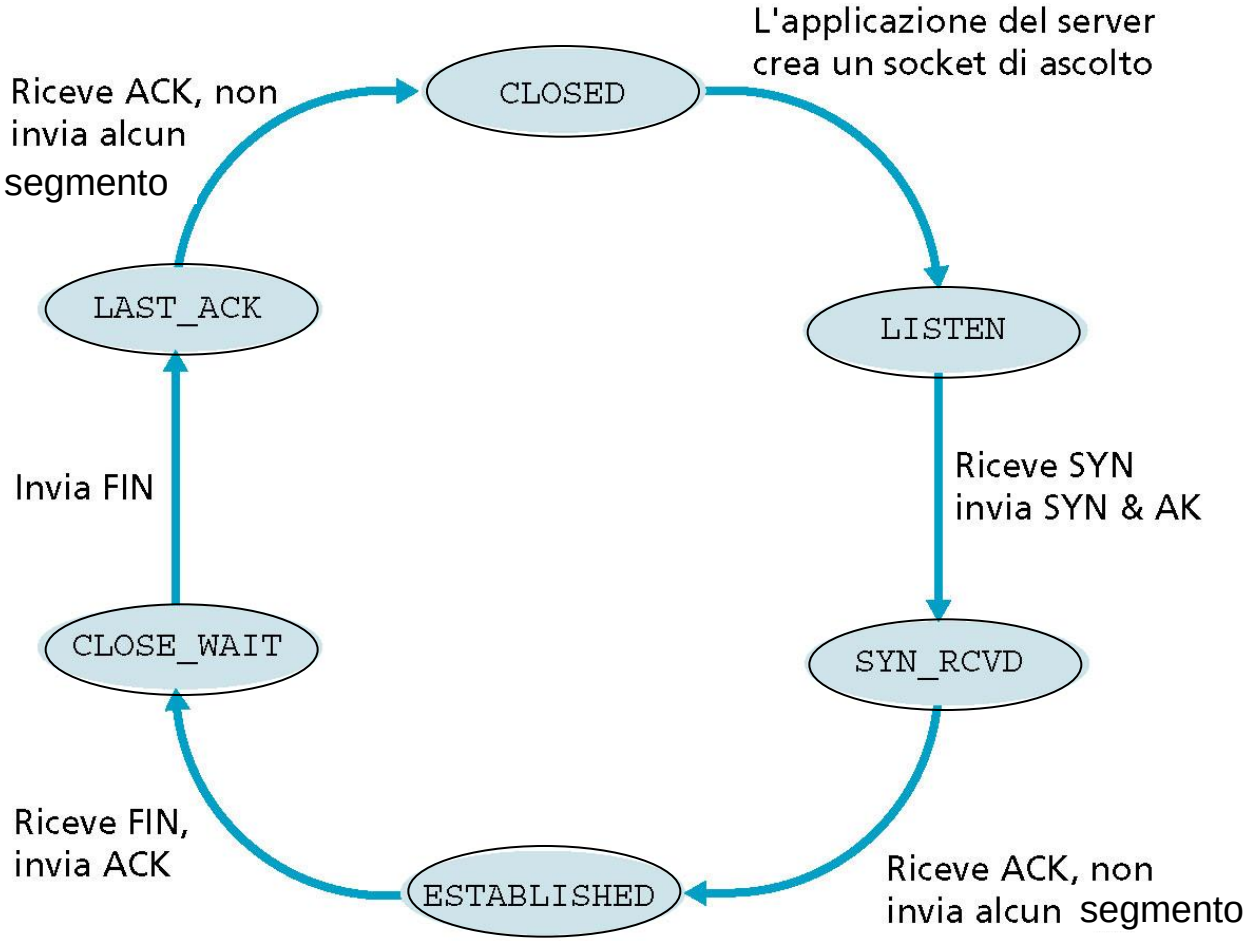


- La figura precedente, mostra il caso in cui è il client a chiudere la connessione. Dapprima il client invia uno speciale segmento, caratterizzato dall'avere il bit **FIN** del campo flag settato a 1. Quando il server riceve questo segmento, risponde al client con un segmento di riscontro (**ACK=1**). Il server invia poi il suo segmento di chiusura, che ha il bit FIN posto a 1. Infine, il client riscontra il segmento di chiusura del server. A questo punto, tutte le risorse allocate nei due host sono state rilasciate.



- Durante il periodo di esistenza della connessione, il TCP esegue transizioni di stato. Le figure seguenti mostrano una tipica sequenza degli stati eseguita dal TCP client e server.





Una tipica sequenza degli stati per i quali passa un TCP del server