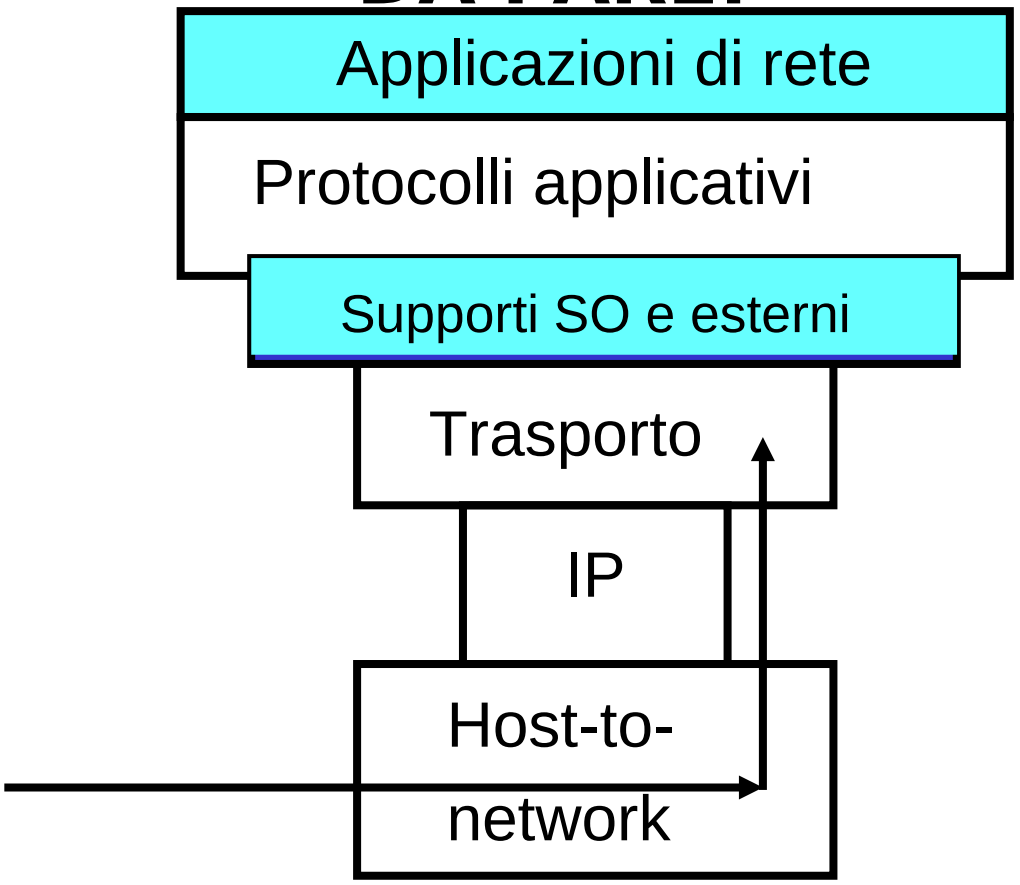
PARTE 5a

LIVELLO TRASPORTO

Dove ci troviamo?

- **Abbiamo introdotto i termini ed i concetti fondamentali del corso**
- **Abbiamo trattato il livello h2n**
- **Abbiamo trattato il livello IP**

DA FARE:



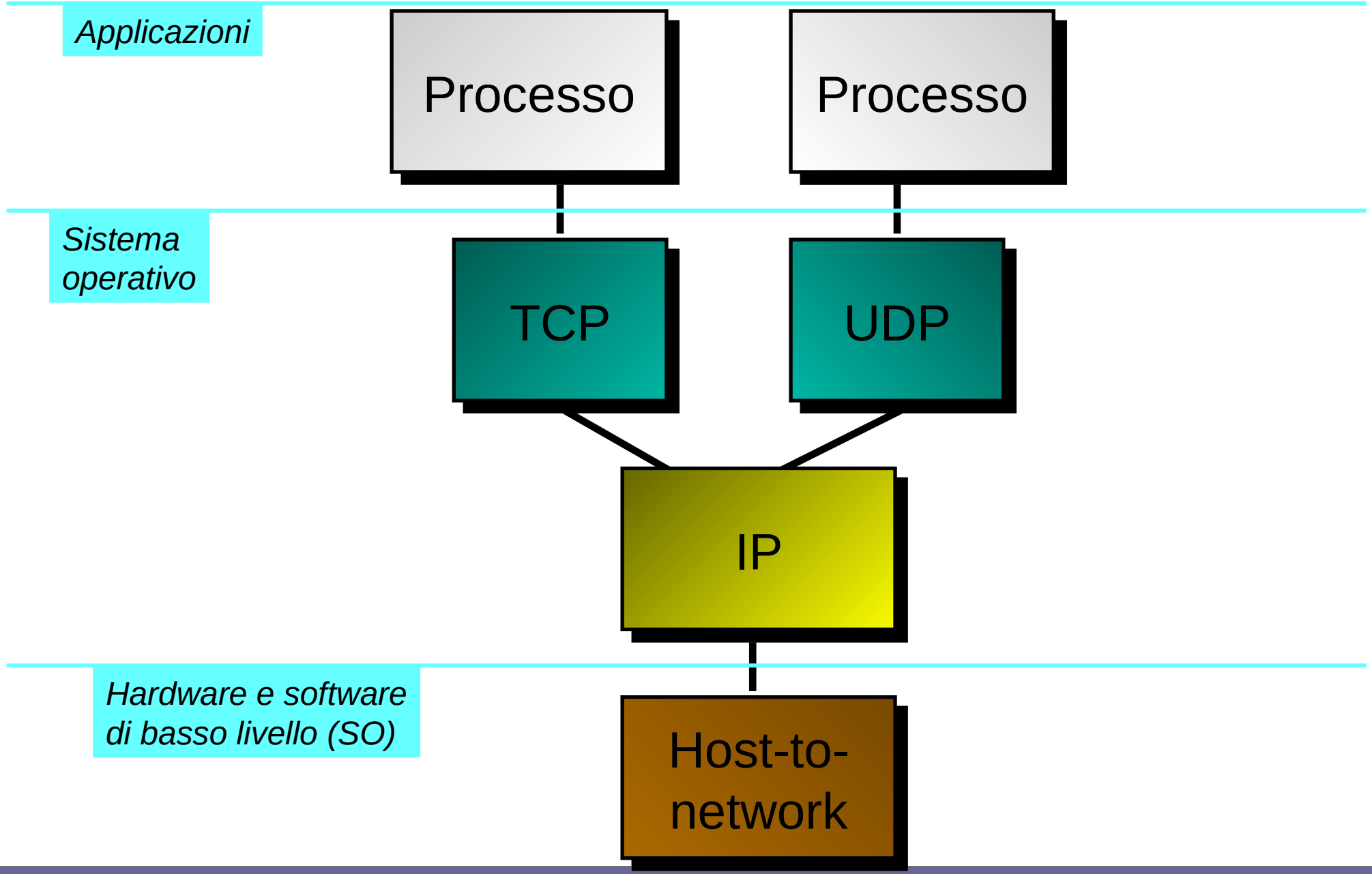
Livello 4 (transport)

- **Il livello trasporto estende il servizio di consegna con impegno proprio del protocollo IP tra due host terminali ad un servizio di consegna a due processi applicativi in esecuzione sugli host**
- **Se IP è il protocollo di rete, TCP sarà il protocollo di trasporto? → NO!**
- **TCP è solo un componente del livello di trasporto della suite TCP/IP**
- **L'altro componente è costituita dal protocollo UDP (User Datagram Protocol)**

Livello 4 (transport)

- **Il livello transport estende il servizio di consegna con impegno proprio del protocollo IP tra due host terminali ad un servizio di consegna a due processi applicativi in esecuzione sugli host**
- **Servizi aggiuntivi rispetto a IP**
  - multiplazione e demultiplazione messaggi tra processi
  - rilevamento dell'errore (mediante checksum)
- **Esempi di protocolli transport**
  - UDP (User Datagram Protocol)
  - TCP (Transmission Control Protocol): offre servizi aggiuntivi rispetto a UDP

Livelli dello stack TCP/IP: chi gestisce?



Servizi del livello di trasporto

Servizi comuni a UDP e TCP:

Estensione del servizio di consegna del protocollo IP tra due nodi terminali ad un servizio di consegna a due processi applicativi in esecuzione sui nodi terminali

- **multiplazione e demultiplazione**
- **rilevamento dell'errore (non correzione!)**

Servizi del livello di trasporto

Servizi aggiuntivi di TCP:

- **Trasferimento affidabile dei dati**
  - → controllo di flusso, numeri di sequenza, acknowledgement e timer
- **Controllo di congestione**
  - → regola il tasso di invio dei segmenti da parte del mittente

Modulo 1: Multiplazione e demultiplazione

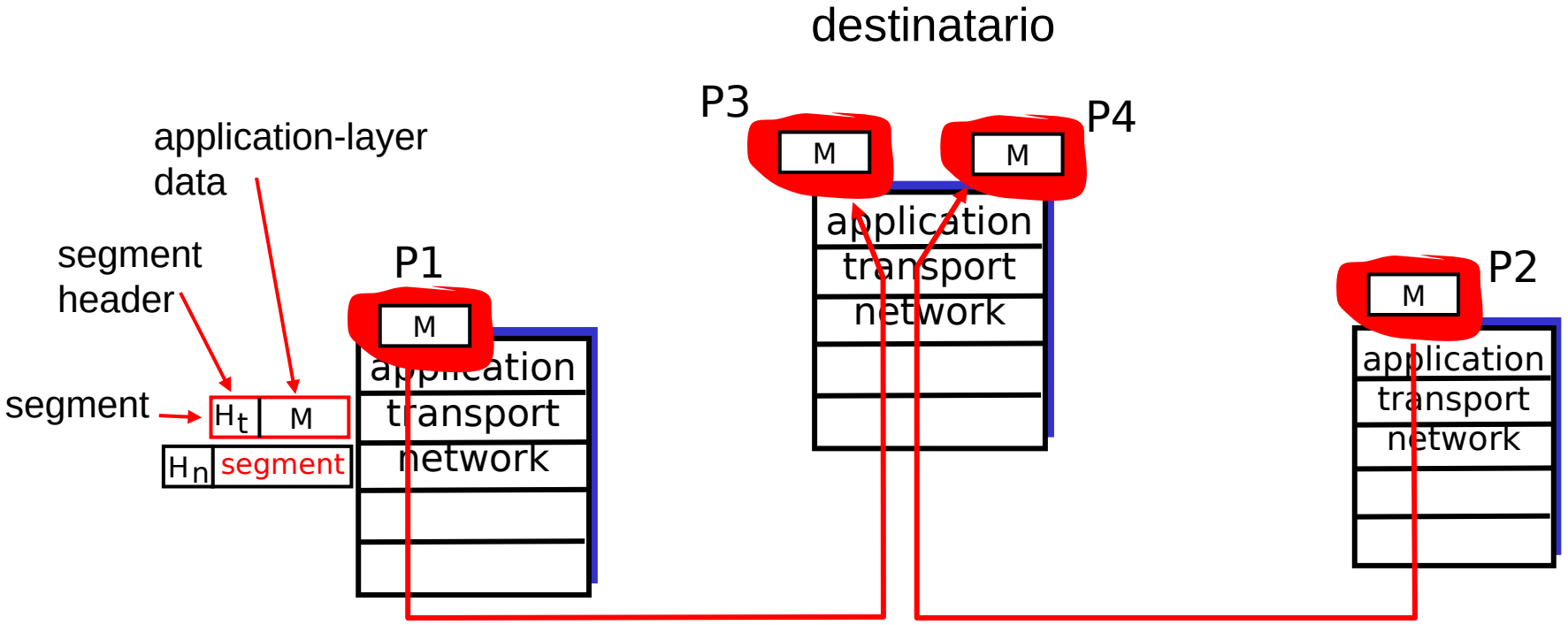
1. Multiplazione e demultiplazione

- **Il protocollo IP non consegna i dati tra processi applicativi in esecuzione sui nodi terminali (un indirizzo IP per identificare ogni interfaccia di un nodo terminale)**
  - → compito del protocollo di trasporto
- **Ogni segmento dello strato di trasporto possiede un campo contenente l'informazione usata per determinare a quale processo deve essere consegnato il segmento**
  - → demultiplazione
- **La demultiplazione avviene dal lato del nodo destinatario**

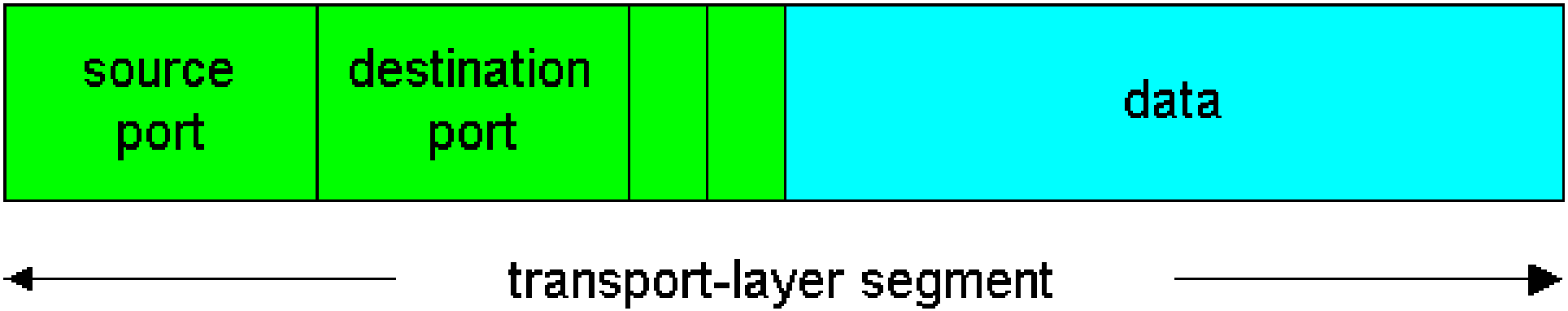
1. Multiplazione e demultiplazione

- **Creazione dei segmenti provenienti dai messaggi di diversi processi applicativi**
  - → multiplazione
- **La multiplazione avviene dal lato del nodo mittente**

Esempio


destinatario
P3
P4
P1
P2
application-layer
data
segment
header
segment
M
application
transport
network
H_t
H_n
segment

Multiplazione e demultiplazione



UDP e TCP attuano la multiplazione/demultiplazione includendo due campi speciali nell'header del segmento:

- **il numero di porta del mittente**
- **il numero di porta del destinatario**

Permettono di identificare in modo univoco i due processi applicativi, residenti su due nodi terminali e comunicanti tra loro

Multiplazione e demultiplazione

- **Numero di porta: numero di 16 bit compreso tra 0 e 65535**
- **Numeri di porta noti (well-known ports, assigned numbers in [RFC 1700]): tra 0 e 1023 riservati per protocolli applicativi noti (ad es., HTTP e FTP)**
  - HTTP: numero di porta 80
  - Telnet: numero di porta 23
  - SMTP: numero di porta 25
  - DNS: numero di porta 53

Multiplazione e demultiplazione

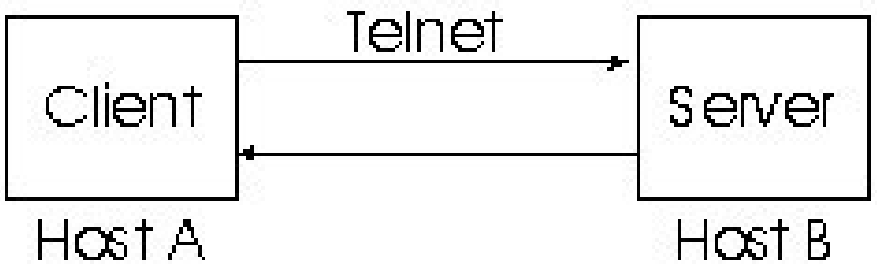
- **File /etc/services su sistemi Unix per conoscere i numeri di porta principali**
- **Quando si realizza un nuovo servizio di rete è necessario assegnargli un nuovo numero di porta**
- **Occorre sia il numero di porta del mittente sia quello del destinatario per distinguere processi dello stesso tipo ed in esecuzione negli stessi istanti**

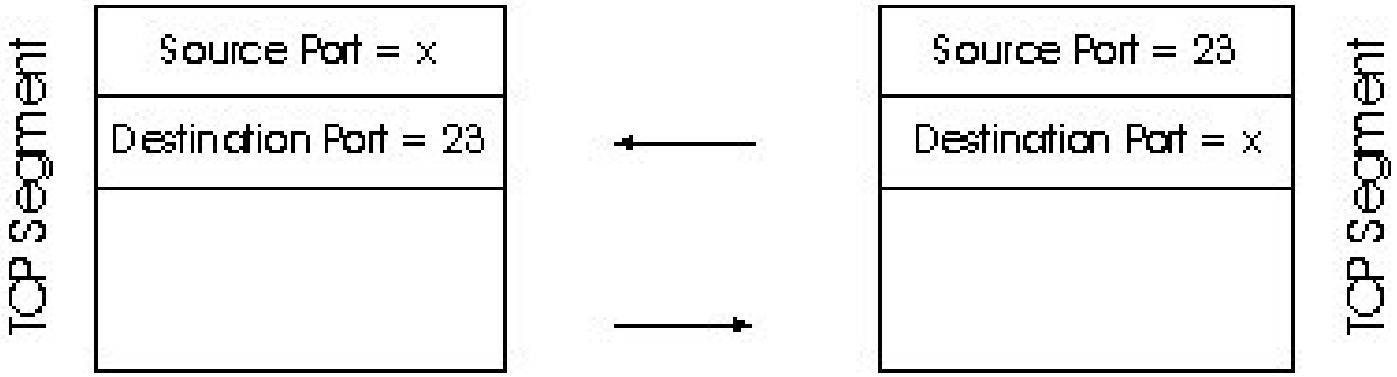
Categorie di numeri di porta

- **0-1023 → Well Known Ports**
  - NON DEVONO essere usate senza una precedente autorizzazione da IANA [RFC4340]. Nella maggior parte dei sistemi, possono essere usate solo da processi con privilegi di root o simili
- **1024-49151 → Registered Ports**
  - NON DEVONO essere usate senza una precedente autorizzazione da IANA [RFC4340]. Nella maggior parte dei sistemi, possono essere usate da qualsiasi processo
- **49152-65535 → Dynamic or Private Ports**

Esempio: telnet

Uso dei numeri di porta in un'applicazione client/server (es. Telnet, con numero di porta 23):





Assegnazione dei numeri di porta

Modello client/server

- **Numero di porta del destinatario nel segmento inviato dal client al server corrisponde al numero di porta del servizio richiesto (ad es., 80 per HTTP)**

- **Numero di porta del mittente nel segmento inviato dal client al server corrisponde ad un numero di porta scelto tra quelli non in uso sul client**

Assegnazione dei numeri di porta

Modello client/server

- **Numero di porta del mittente nel segmento inviato dal server al client corrisponde al numero di porta del servizio richiesto (ad es., 80 per HTTP)**

- **Numero di porta del destinatario nel segmento inviato dal server al client corrisponde al numero di porta indicato dal client nel messaggio precedentemente inviato**

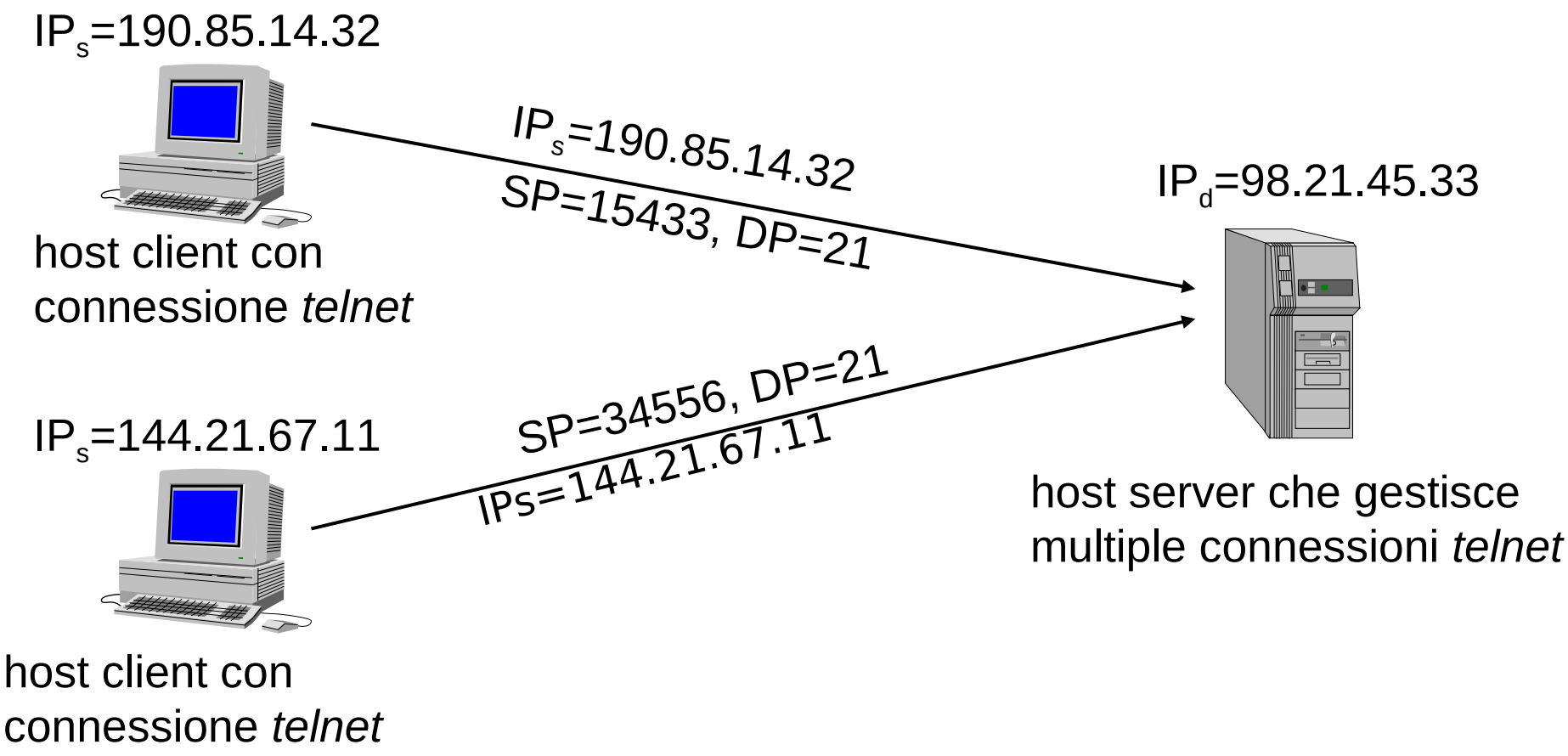
Identificatori

- **A livello network (IP)**
  [indirizzo IP1, indirizzo IP2]

- **A livello trasporto (UDP, TCP)**
  [(ind. IP1, porta1), (ind. IP2, porta2)]

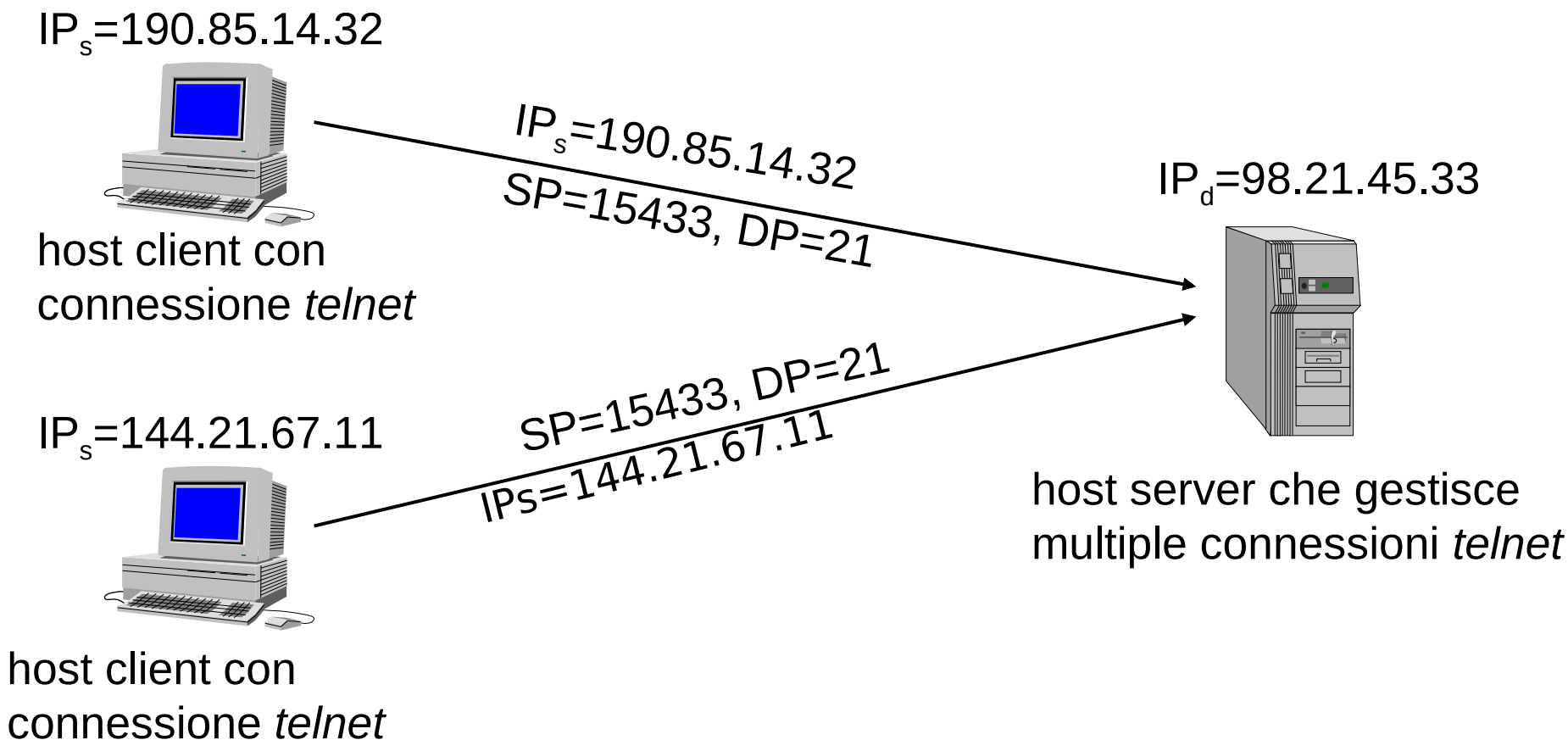
Indirizzo IP e numero di porta

Due processi client, residenti su host diversi per comunicare con lo stesso servizio applicativo sono sempre distinti in base al loro indirizzo IP (sorgente) e possono essere distinti in base al numero di porta sorgente (SP):



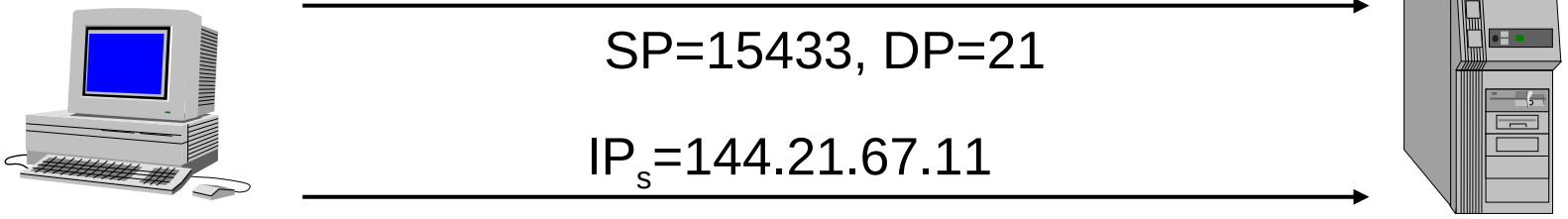
Indirizzo IP e numero di porta

Due processi client, residenti su host diversi e che, per eventualità, usano lo stesso numero di porta sorgente (SP) per comunicare con lo stesso servizio applicativo, sono distinti in base al loro indirizzo IP:



Indirizzo IP e numero di porta

Due processi client, residenti sullo stesso host per comunicare con lo stesso servizio applicativo, non essendo distinti in base al loro indirizzo IP, useranno diversi numeri di porta sorgente (SP) grazie al sistema operativo:



host client con due connessioni *telnet*

host server che gestisce multiple connessioni *telnet*

Modulo 2:
Protocollo UDP

Caratteristiche protocollo UDP (cosa ha)

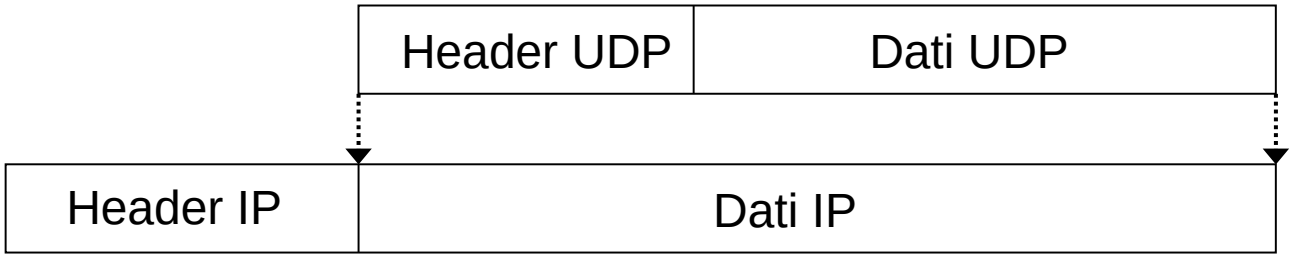
- **User Data Protocol (UDP), definito in**
- **[RFC 768], è un protocollo di trasporto leggero, ovvero dotato delle funzionalità minime del trasporto:**
- **Servizio di multiplazione/demultiplazione**
  - UDP aggiunge al messaggio proveniente dal livello applicativo il numero di porta del mittente e del destinatario
- **Controllo di errore**
  - UDP include nell'header un campo di checksum

Caratteristiche protocollo UDP (cosa non ha)

- **Servizio di consegna non garantito, ma solo di tipo best effort**
  - i segmenti UDP possono essere persi, duplicati, consegnati senza ordine
- **Servizio senza connessione (connectionless)**
  - non vi è handshaking tra mittente e destinatario del segmento UDP
  - ogni segmento UDP è trattato in modo indipendente dagli altri

Formato segmento UDP

Segmento UDP (o *user datagram*) incapsulato in un datagramma IP



Segmento UDP

32 bit

numero porta mittente	numero porta destinatario
lunghezza	checksum
dati dell'applicazione (messaggio)	

Campi del segmento UDP

- **numero di porta del mittente (16 bit)**
- **numero di porta del destinatario (16 bit)**
- **lunghezza (16 bit): dimensione in byte del segmento**
  - lunghezza = header + dati
  - header: dimensione pari a 8 byte

Campi del segmento UDP

- **checksum**
  - Non tutti i link forniscano rilevazione errori a livello H2N
  - Operazioni a livello IP non coperte da livello H2N
  - Controllo errori end-to-end
  - Checksum a livello IP limitato all'header del datagram IP
  - Non c'è recupero dell'errore
- **dati: contiene il messaggio fornito dal livello applicativo**

Checksum UDP

Scopo: individuare “errori” (es., bit modificati) nel segmento trasmesso

Mittente

- **Tratta i contenuti del segmento come sequenza di interi a 16 bit**
- **Checksum=somma dei contenuti dei segmenti con complemento a 1**
- **Il mittente invia il valore del checksum nel campo checksum del segmento UDP**

Destinatario

- **Calcola il checksum del segmento ricevuto**
- **Controlla se il valore del checksum calcolato è uguale al valore del campo checksum:**
  - **NO** → errore
  - **SI** → non si individua errore. Ci può essere lo stesso un errore?

Checksum UDP

Calcolato usando un maggior numero di informazioni di quelle presenti nell'header UDP → definizione di uno *pseudo-header* UDP

- *zero padding*: dimensione dello pseudo-header (multiplo di 32 bit)
- *protocollo*: campo protocollo del datagram IP
- pseudo-header anteposto al segmento UDP
- checksum calcolato su pseudo-header e intero segmento UDP
- lo pseudo-header *non* è trasmesso dal mittente

Pseudo header UDP e UDP6

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source IPv4 Address																															
4	32	Destination IPv4 Address																															
8	64	Zeroes								Protocol								UDP Length															
12	96	Source Port																Destination Port															
16	128	Length																Checksum															

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source IPv6 Address																															
4	32																																
8	64																																
12	96																																
16	128	Destination IPv6 Address																															
20	160																																
24	192																																
28	224																																
32	256	UDP Length																															
36	288	Zeroes																								Next Header							
40	320	Source Port																Destination Port															
44	352	Length																Checksum															

Checksum UDP

Calcolato usando il complemento ad 1 della somma di tutti i campi dello pseudo-header e del segmento UDP

Esempio

3 parole da 16 bit l'una

0110011001100110
0101010101010101
0000111100001111

Somma delle 3 parole

0110011001100110
0101010101010101
0000111100001111
1100101011001010

- complemento ad 1di 1100101011001010 → 0011010100110101
- campo checksum nel segmento UDP trasmesso = 0011010100110101
- il destinatario calcola il suo checksum su pseudo-header e segmento UDP ricevuto (senza calcolare il complemento a 1)
- checksum dest. + checksum UDP = 1111111111111111 → no errore
- checksum dest. + checksum UDP ≠ 1111111111111111 → errore

Calcolo del checksum UDP

- **Conoscenza dell'indirizzo IP del mittente e del destinatario**
- **il processo mittente a livello UDP non può acquisire l'indirizzo IP del destinatario dall'applicazione di livello superiore**
- **il processo mittente a livello UDP chiede al livello IP di costruire lo pseudo-header, calcolare il checksum UDP ed eliminare lo pseudo-header**

→ Violazione del principio di indipendenza funzionale per protocolli appartenenti a livelli diversi

Modulo 3:
Protocolli su canale affidabile

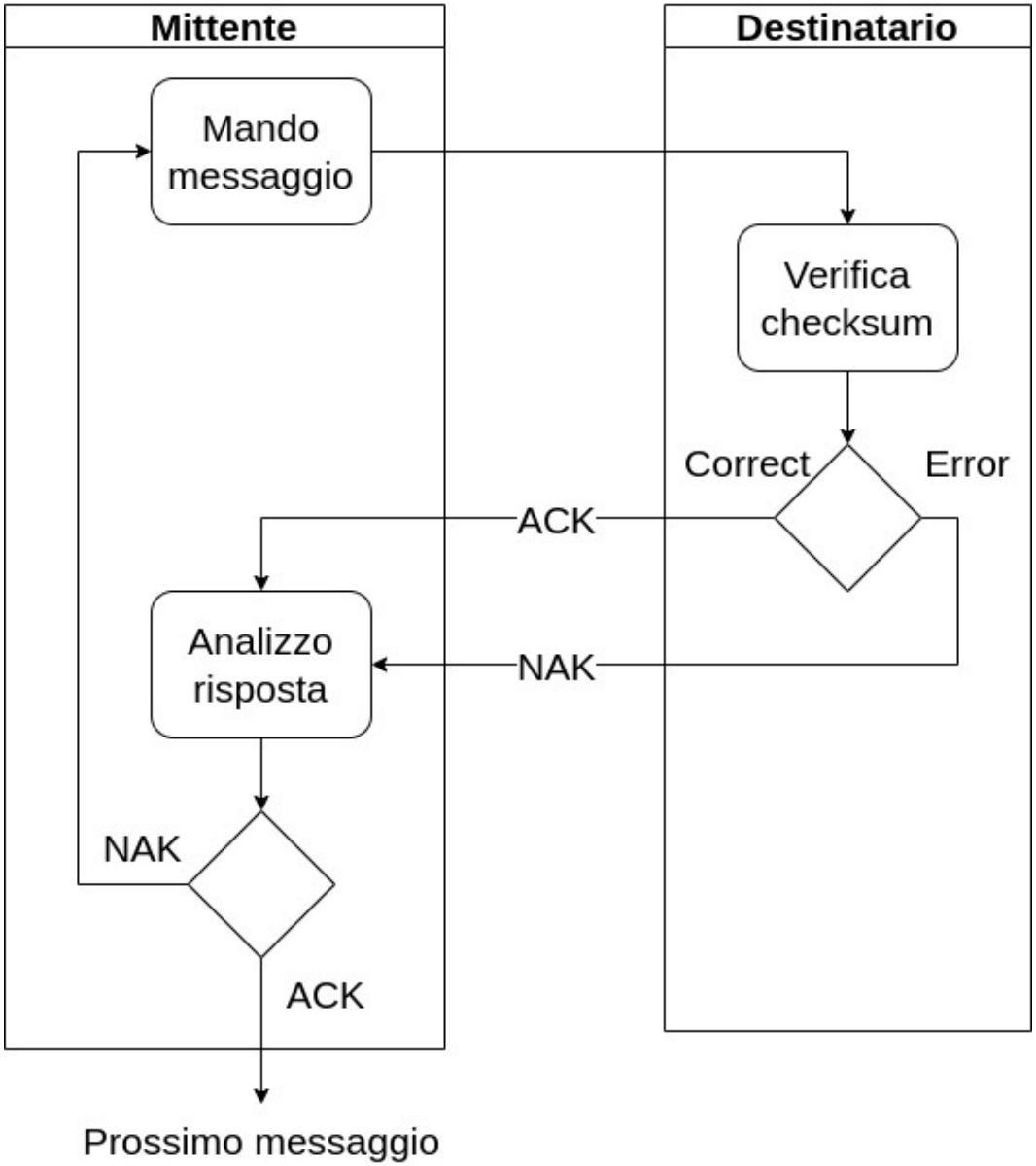
Concetto di affidabilità

- **Affidabilità → tolleranza a errori**
- **Serve modello dei possibili errori per implementare le contromisure**
- **Esempio di modelli di errori diversi**
- **Scenario**
  - Trasmissione Stop-and-Wait
  - Mando un pacchetto per volta

Errore 1: Corruzione del messaggio

- **Caso molto semplice**
- **Devo identificare l'errore**
  - Introduce checksum (già visto per UDP)
- **Voglio affidabilità**
  - Se ho errore → ritrasmetto
- **Devo notificare al mittente la presenza di errori**
  - Messaggio di acknowledgement
  - ACK o NAK

Errore 1: Corruzione del messaggio



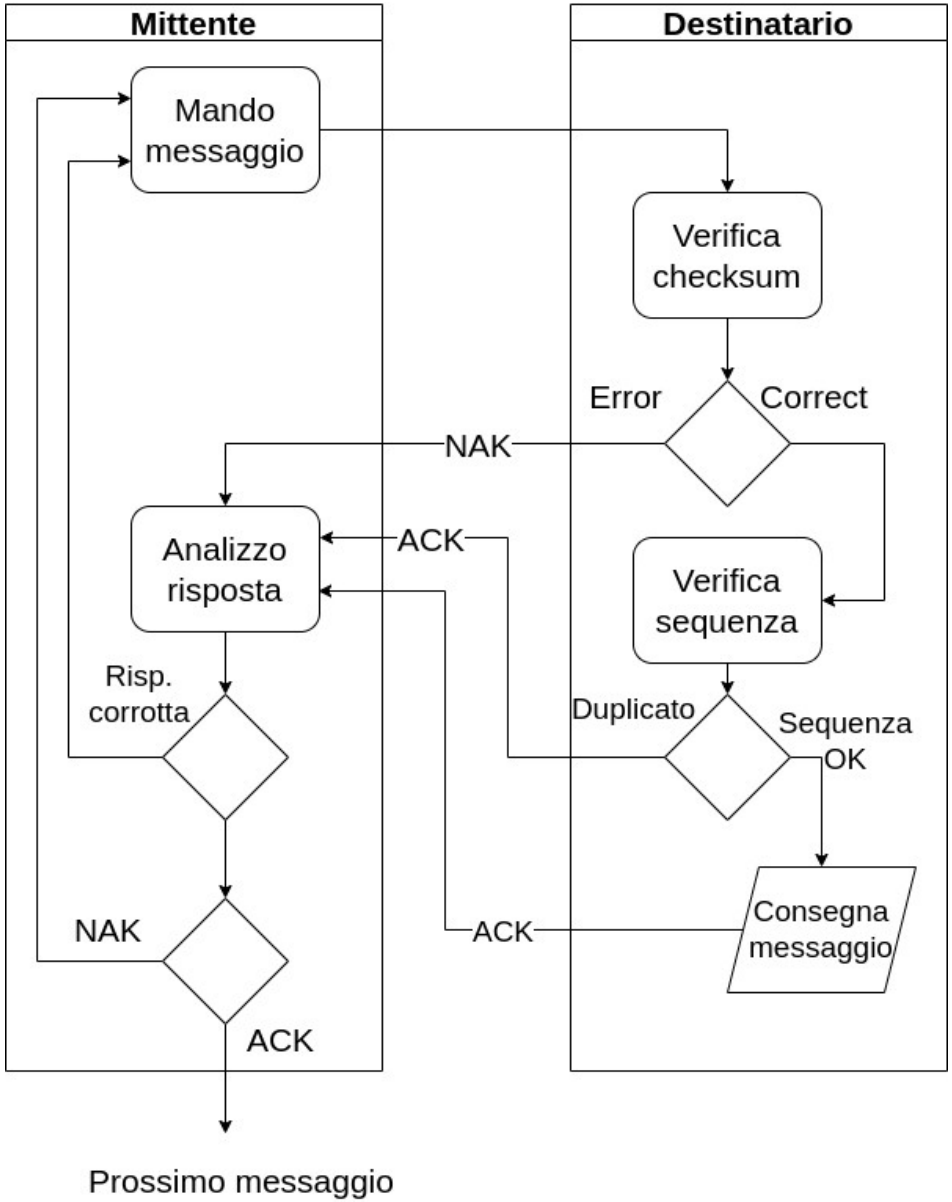
Errore 2: Corruzione delle risposte

- **Complica la situazione precedente**
- **Anche il messaggio ACK/NAK può essere corrotto**
  - Devo aggiungere checksum anche a questi messaggi
  - Devo verificare se la risposta è corrotta
- **Cosa succede se la risposta non è integra?**
  - Se era NAK e non ritrasmetto → perdo dati
  - Se era ACK e ritrasmetto → duplicazione

Errore 2: Corruzione delle risposte

- **Evitare duplicati**
- **Devo aggiungere numero di sequenza**
  - Per ora solo ai messaggi
- **Se messaggio di risposta è corrotto**
  - Ritrasmetto sempre
- **So che messaggio stavo aspettando**
  - Se arriva numero di sequenza che ho già ricevuto
    → scarto il duplicato
    → mando ACK

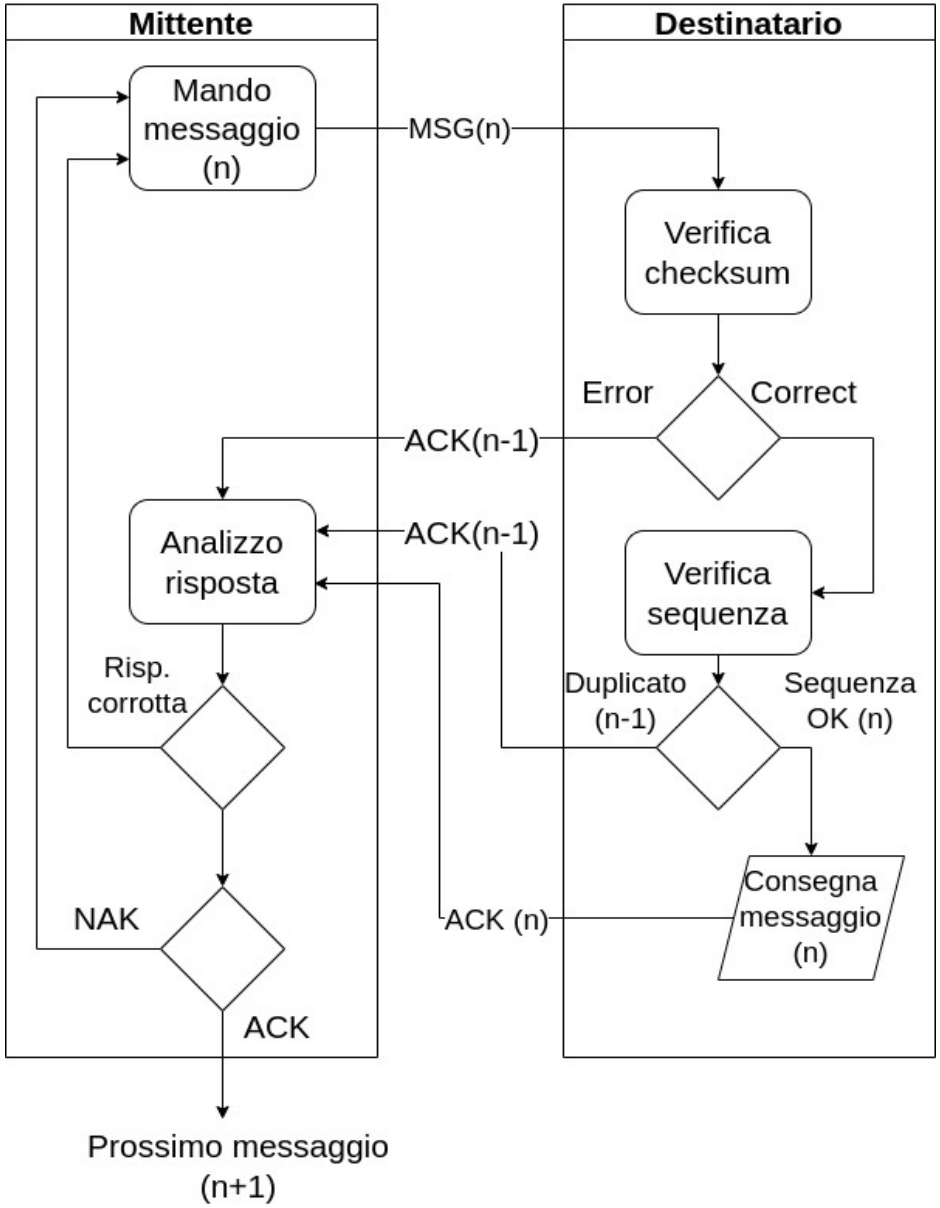
Errore 2: Corruzione delle risposte



Errore 2: Corruzione delle risposte

- **Evoluzione del modello**
- **Evitare due tipi di messaggio**
  - ACK
  - NAK
- **Aggiungere il numero di sequenza a ACK**
  - ACK(n-1)=NAK(n)

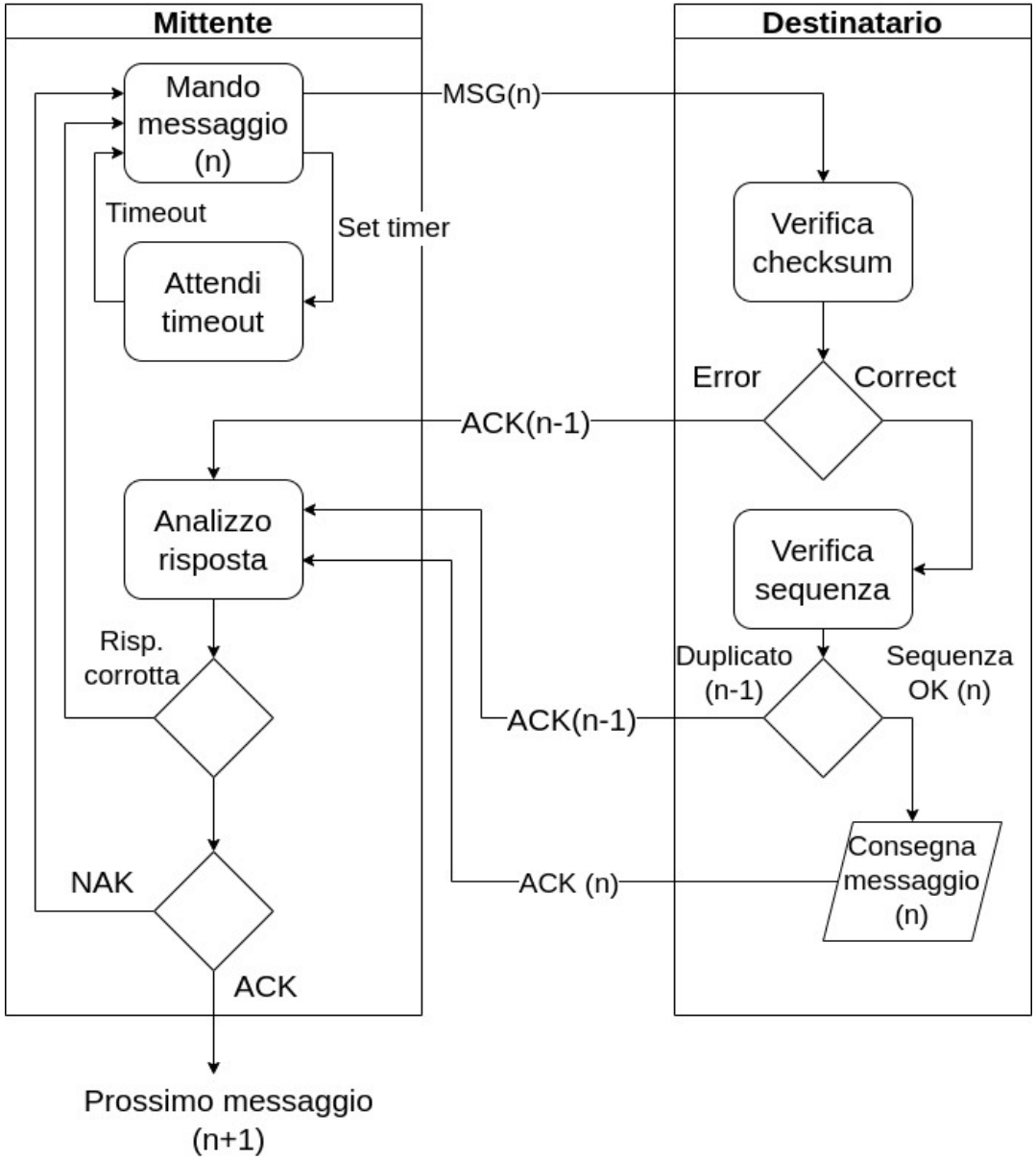
Errore 2: Corruzione delle risposte



Errore 3: perdita di messaggi

- **Un messaggio si può perdere**
  - Messaggio vero e proprio
  - Risposta (ACK)
- **Rischio di attesa infinita**
  - Non ho mai una risposta
  - Posso agire solo quando ricevo una risposta dal destinatario
- **Come operare?**
  - Introduco timeout

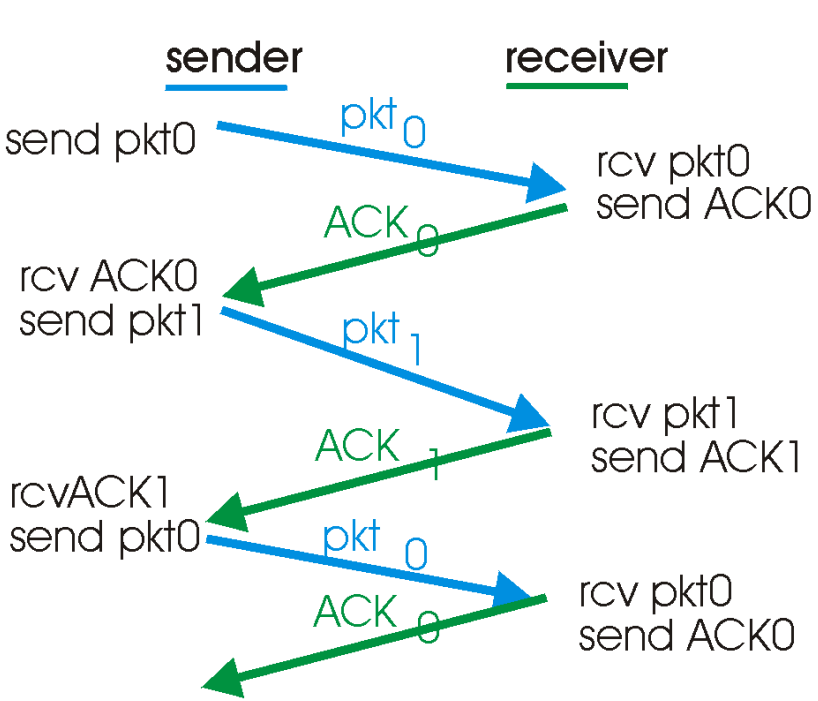
Errore 3: perdita di messaggi



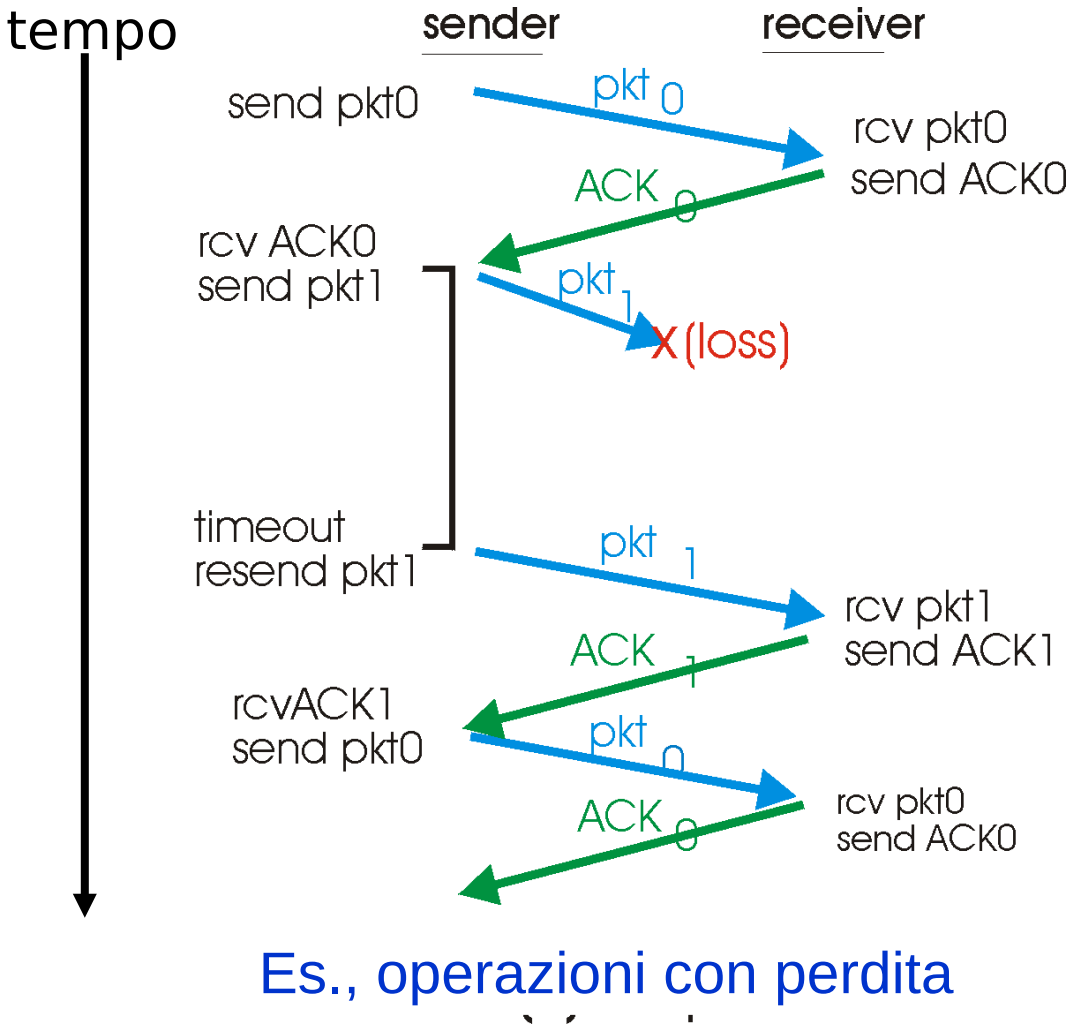
Errore 3: perdita di messaggi

- **Nota sul funzionamento**
- **Perdita di messaggio**
  - Timeout fa rispedire il messaggio
  - Tutto funziona correttamente
- **Perdita di ACK**
  - Causa ritrasmissione
  - Crea duplicato
  - Gestito come corruzione ACK
  - Tutto funziona correttamente

Sequenza operazioni in rdt3.0



Es. operazioni senza perdite



Es., operazioni con perdita

Il meccanismo send-ack per singolo pacchetto è molto semplice ed affidabile, ma INEFFICIENTE

Prossimo obiettivo

- **Utilizzare questi risultati teorici per analizzare quali meccanismi utilizza il protocollo TCP per realizzare una “comunicazione affidabile” e “orientata alla connessione” su di un “canale inaffidabile”**