1 CAPITOLO 1 : INTRODUZIONE

1. **Cos'è un protocollo ?**
   Possiamo paragonare il funzionamento delle reti informatiche alle interazioni della vita quotidiana, utilizzando il concetto di "protocollo delle buone maniere". Immaginiamo di voler chiedere l'ora a un nostro collega. Prima di tutto, lo salutiamo dicendo "Ciao", che può essere visto come una richiesta di connessione. Il collega può scegliere se rispondere o meno. Se risponde con un altro "Ciao", significa che la connessione è stata instaurata con successo. A questo punto, possiamo procedere con la nostra richiesta: "Che ore sono?". Se il collega decide di rispondere, ci fornirà l'informazione richiesta, completando così lo scambio di dati. Allo stesso modo, nelle reti informatiche, i dispositivi devono rispettare protocolli di comunicazione ben definiti per garantire uno scambio di dati corretto ed efficiente.Per definizione, un protocollo è un insieme di regole e convenzioni che stabiliscono come due entità comunicano tra loro. Queste regole definiscono il formato e l'ordine dei messaggi scambiati, oltre alle azioni da intraprendere durante la trasmissione e la ricezione. Ad esempio, nel protocollo TCP, un dispositivo deve inviare un messaggio di SYN per avviare una connessione, l'altro risponde con SYN-ACK, e infine il primo conferma con un ACK, instaurando così una comunicazione stabile.

2. **Commutazione a pacchetto vs Commutazione a Circuito**
   La commutazione a pacchetto è una tecnica di comunicazione in cui i messaggi vengono suddivisi in pacchetti più piccoli e trasmessi attraverso una rete. Ogni pacchetto viaggia indipendente e può seguire percorsi diversi per raggiungere la destinazione. Quando un'applicazione distribuita (come un'email,un file MP3 o un'immagine JPEG) deve inviare un messaggio, questo viene suddiviso in pacchetti più piccoli se è troppo grande. Ciascun pacchetto viene trasmesso attraverso collegamenti e commutatori a una velocità pari alla capacità del collegamento stesso. I pacchetti vengono memorizzati temporaneamente nei buffer dei commutatori. Questa tecnica è definita "store and forward" (memorizza e inoltra) e viene utilizzata dalla maggior parte dei commutatori. Ciò significa che il commutatore deve ricevere l'intero pacchetto prima di poterlo trasmettere sul collegamento in uscita. I bit vengono memorizzati in un buffer e solo dopo aver ricevuto tutti i bit, il commutatore può inoltrare il pacchetto. Questo processo genera un ritardo, noto come ritardo di memorizzazione e inoltro. I vantaggi della commutazione di pacchetto sono la sua semplicità,efficienza e costo ridotto. Tuttavia, ci sono anche degli svantaggi, come la perdita di pacchetti. Ogni commutatore connette più collegamenti e per ciascuno di questi collegamenti, il commutatore contiene un buffer di output. Se il buffer è pieno, i pacchetti vengono scartati,causando una perdita di pacchetti

   Esistono diversi tipi di ritardo nella commutazione di pacchetto:

- **Ritardo di elaborazione** : il tempo necessario per esaminare l'intestazione del pacchetto.
- **Ritardo di accodamento**: il tempo che un pacchetto passa in coda prima di essere trasmesso, che dipende dal numero di pacchetti precedentemente arrivati.
- **Ritardo di trasmissione**: il tempo necessario per trasmettere tutti i bit di un pacchetto.
- **Ritardo di propagazione**: il tempo richiesto per far viaggiare i bit attraverso il mezzo fisico, che dipende dalla distanza tra i router e dal tipo di mezzo utilizzato.

Se la rete è congestionata, si può verificare un buffer overflow, ovvero il riempimento della coda del buffer, che porta alla perdita di pacchetti. Per gestire questo problema, i protocolli di trasporto come TCP implementano meccanismi di ritrasmissione per garantire l'affidabilità della comunicazione. Tuttavia, protocolli come UDP non prevedono ritrasmissione, rendendoli più adatti a servizi in tempo reale (es. streaming video o chiamate VoIP).

La commutazione a circuito è una tecnica di comunicazione in cui viene creato un canale di comunicazione dedicato tra due dispositivi (ad esempio due telefoni o due computer) prima che inizi lo scambio dei dati . Questo canale rimane attivo e riservato per tutta la durata della comunicazione,indipendentemente dal fatto che ci siano dati da trasmettere o meno. Per capire meglio, possiamo usare un'analogia con due ristoranti:

Il primo ristorante richiede una prenotazione. Prima di andare,dobbiamo chiamare e prenotare un tavolo.Una volta arrivati,possiamo sedersi immediatamente perché il tavolo è riservato solo per noi. Questo è simile alla commutazione a circuito,dove il percorso di comunicazione è riservato fin dall'inizio. Il secondo ristorante non accetta prenotazioni. Quando arriviamo, potremmo dover aspettare che si liberi un tavolo. Questo è simile alla commutazione a pacchetto,dove le risorse vengono condivise tra più utenti.

Un esempio pratico di commutazione a circuito sono le reti telefoniche tradizionali. Quando effettuiamo una chiamata, viene stabilito un circuito dedicato tra i due telefoni, che rimane attivo per tutta la conversazione, anche nei momenti di silenzio. Sebbene la commutazione a circuito garantisca una connessione stabile e affidabile, è meno efficiente rispetto alla commutazione a pacchetto, poiché riserva risorse anche quando non vengono utilizzate. Per questo motivo, le moderne reti di comunicazione (inclusa la telefonia VoIP) hanno abbandonato la commutazione a circuito a favore della commutazione a pacchetto.

3. **Non tutti gli indirizzi IP possono essere assegnati agli host: ve ne sono alcuni, definiti "Indirizzi speciali" che hanno un particolare utilizzo. Tra questi,vi sono: l'indirizzo composto da tutti**

"1",l'indirizzo composto da tutti "0",l'indirizzo in cui l'host ID è composto da tutti "1",l'indirizzo in cui l'host ID è composto da tutti "0".Per ciascuno dei 4 indirizzi si spieghi il loro utilizzo

(a) **Indirizzo composto da tutti "0"** (`0.0.0.0`):
Rappresenta un indirizzo **non specificato**. Viene utilizzato da un host che non conosce ancora il proprio indirizzo IP, ad esempio durante l'avvio o la richiesta DHCP. È anche usato nei router per indicare la *default route* (`0.0.0.0/0`).
Esempio: un host che invia una richiesta DHCP usa `0.0.0.0` come indirizzo sorgente.

(b) **Indirizzo composto da tutti "1"** (`255.255.255.255`):
È l'indirizzo di **broadcast limitato**, usato per inviare pacchetti a tutti gli host presenti nella stessa rete locale (LAN), senza attraversare router.
Esempio: un host può inviare un pacchetto a `255.255.255.255` per raggiungere ogni dispositivo sulla rete locale.

(c) **Indirizzo con host ID composto da tutti "1"**:
Rappresenta il **broadcast di rete** per una rete specifica, ottenuto impostando a 1 tutti i bit della parte host. Identifica tutti gli host della rete.
Esempio: nella rete `192.168.1.0/24`, l'indirizzo `192.168.1.255` è il broadcast della rete.

(d) **Indirizzo con host ID composto da tutti "0"**:
Identifica l'**indirizzo di rete** (network address). Serve per rappresentare la rete in sé e non un singolo host.
Esempio: `192.168.1.0/24` è l'indirizzo della rete in cui si trovano gli host da `192.168.1.1` a `192.168.1.254`.

Nota: Questi indirizzi non possono essere assegnati agli host perché hanno significati riservati all'interno del protocollo IP. Il loro utilizzo scorretto potrebbe compromettere il corretto funzionamento della rete.

4. **Definizione di indirizzo privato e indirizzo pubblico, spiegando come vengono utilizzate.**

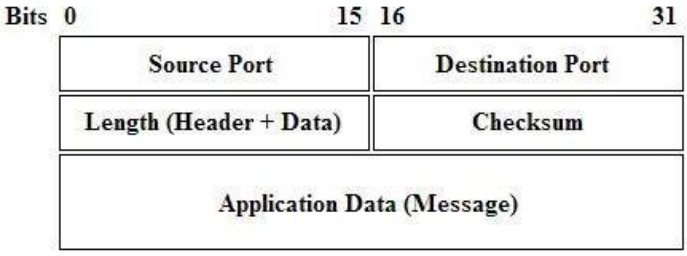
   Un **indirizzo pubblico** è un indirizzo IP unico a livello globale, assegnato da autorità come l'IANA, e viene utilizzato per dispositivi che devono essere raggiungibili su Internet. Esempi includono server web e email.

   Un **indirizzo privato** è un indirizzo riservato per uso nelle reti locali (LAN), non accessibile direttamente su Internet. Viene utilizzato all'interno di reti aziendali o domestiche e richiede tecniche come NAT per accedere a Internet. Gli indirizzi privati appartengono a specifici intervalli, come 10.0.0.0 - 10.255.255.255 (IPv4).

2 CAPITOLO 3 : LIVELLO DI TRASPORTO

1. **Spiega il concetto di numero di porta noti**
   I numeri di porta noti sono un intervallo di porte compreso tra 0 e 1023, assegnato a servizi e protocolli di rete ben definiti. Questi numeri, gestiti dalla IANA (Internet Assigned Numbers Authority), permettono di identificare in modo univoco le applicazioni di rete più comuni, garantendo una comunicazione efficiente tra client e server. La loro introduzione è stata necessaria per standardizzare il modo in cui i dispositivi si connettono ai vari servizi, evitando ambiguità e semplificando le connessioni. Senza un sistema di porte predefinite, un client non saprebbe su quale porta contattare un determinato servizio, rendendo la comunicazione più complessa e meno affidabile. Ad esempio, un browser che vuole accedere a un sito web tramite HTTP deve connettersi alla porta 80, mentre per HTTPS utilizza la porta 443. Questo meccanismo garantisce che ogni servizio sia raggiungibile in modo prevedibile, evitando conflitti tra applicazioni e migliorando l'interoperabilità tra dispositivi e sistemi operativi diversi. Tra i numeri di porta noti più utilizzati troviamo la porta 21 per FTP, utilizzata per il trasferimento di file, la porta 53 per DNS, responsabile della risoluzione dei nomi di dominio, e la porta 25 per SMTP, usata per l'invio delle email. Queste porte sono documentate nell'RFC 1700 e nei suoi aggiornamenti, e rappresentano uno standard fondamentale per il corretto funzionamento delle reti informatiche.

2. **L'header del protocollo UDP contiene solo 4 campi: Source Port, Destination Port, Length e Checksurn. Si spieghi brevemente a cosa servono tali campi**
   La struttura del Protocollo UDP è la seguente : La struttura del prtocollo



   UDP è composta da 4 campi fondamentali:

   - **Source Port / Destination Port** : identificano rispettivamente il processo mittente e il processo destinatario dell'host. Questi campi permettono all'host di consegnare correttamente i dati all'applicazione appropriata.
   - **Length** : indica la lunghezza totale del segmento UDP (header + dati), espressa in byte. È necessaria perché la dimensione del campo

dati può variare da un segmento all'altro.

- **Checksum** : utilizzato per il rilevamento degli errori. Il mittente calcola il complemento a uno della somma di tutte le parole a 16 bit presenti nel segmento (inclusi header e dati). In alcuni casi, si include anche il riporto finale. Al ricevimento, il destinatario effettua lo stesso calcolo: se il risultato è una sequenza di soli 1 (cioè 1111111111111111), il segmento è considerato integro; in caso contrario, si presume che ci siano stati errori durante la trasmissione.

Dati (non parte dell'header, ma del segmento UDP): contiene il messaggio dell'applicazione, che può essere una richiesta o una risposta, oppure contenuti multimediali come audio o video.

3. **Si descriva la modalità di instaurazione di una connessione TCP, specificando i messaggi scambiati e i campi più significativi dell'header utilizzati durante tale fase.** Supponiamo che un processo client, in esecuzione su un host, desideri iniziare una connessione con un altro processo su un host remoto. Il processo applicativo client informa il protocollo TCP del lato client che desidera stabilire una connessione verso il processo server. A questo punto, il TCP sul lato client avvia la procedura di instaurazione della connessione TCP con il TCP sul lato server, utilizzando il meccanismo del three-way handshake (stretta di mano a tre vie).

   (a) Il TCP lato client invia un segmento speciale al TCP lato server. Questo segmento non contiene dati a livello applicativo, ma ha il bit SYN impostato a 1 per indicare l'intenzione di stabilire una connessione. Inoltre, il client sceglie casualmente un numero di sequenza iniziale, chiamato `client_ISN`, e lo inserisce nel campo `Sequence Number` del segmento SYN. Questo segmento viene poi incapsulato in un datagramma IP e inviato al server.

   (b) Quando il datagramma IP arriva a destinazione (ammesso che arrivi), il server estrae il segmento dal datagramma, alloca i buffer e le variabili TCP necessarie alla connessione e invia un segmento di conferma di connessione al TCP client. Anche questo segmento non contiene dati a livello applicativo, ma ha il bit `SYN` impostato a 1. Il campo `ACK` assume il valore `client_ISN + 1`, per confermare la ricezione del segmento SYN del client. Inoltre, il server sceglie un proprio numero di sequenza iniziale, detto `server_ISN`, e lo inserisce nel campo `Sequence Number`. Questo segmento di conferma è detto **segmento SYNACK**.

   (c) Alla ricezione del segmento `SYNACK`, anche il client alloca i buffer e le variabili TCP necessarie alla connessione. L'host client risponde al segmento di conferma ricevuto dal server inviando un terzo segmento. In questo segmento, il campo `ACK` viene impostato a `server_ISN +`

`1`, per confermare la ricezione del segmento `SYNACK`. Il bit `SYN` è impostato a 0, poiché la richiesta di connessione è già stata effettuata, e il segmento può eventualmente contenere dati applicativi da inviare subito al server.

Una volta completati questi tre passi, la connessione TCP è stabilita. Gli host client e server possono ora scambiarsi segmenti contenenti dati a livello applicativo. In ciascuno dei segmenti futuri, il bit `SYN` sarà impostato a 0, poiché la fase di instaurazione della connessione è terminata.
I campi piu significativi dell'header sono:

- **Sequence Number (Numero di sequenza):** indica il numero di sequenza del primo byte di dati nel segmento. Nel primo segmento SYN inviato dal client, questo campo contiene un numero casuale iniziale detto `client_ISN` (Initial Sequence Number).
- **Acknowledgment Number (Numero di riscontro):** quando il flag ACK è attivo, questo campo contiene il numero di sequenza che il mittente si aspetta di ricevere come prossimo byte. Nel segmento SYN-ACK inviato dal server, ad esempio, esso contiene `client_ISN + 1` per confermare la ricezione del SYN del client.
- **Control Bits (flag):** tra questi, i più importanti nella fase di handshake sono:
  - `SYN`: viene impostato a 1 per indicare una richiesta di connessione (segmenti 1 e 2).
  - `ACK`: viene impostato a 1 nei segmenti 2 e 3 per confermare la ricezione dei segmenti precedenti.
- **Window Size (Finestra):** indica quanti byte il destinatario è disposto a ricevere oltre al byte confermato. Anche se non è specifico dell'handshake, è importante perché stabilisce la dimensione della finestra iniziale.
- **Checksum:** usato per rilevare errori nel segmento TCP. Calcolato su header, dati e un pseudo-header IP. Garantisce l'integrità dei dati scambiati fin dalla fase iniziale.

4. **Si descriva la fase di chiusura della connessione nel TCP, indicando i messaggi scambiati e principali campi dell'header utilizzati durante tale fase.**
   Quando una connessione TCP deve essere chiusa, il processo può avvenire in due modi principali: la **chiusura unilaterale** e la **chiusura simultanea**. Entrambi i casi prevedono lo scambio di messaggi tra il client e il server per garantire che tutte le risorse vengano liberate correttamente e che non ci siano dati persi.

   Nel caso della **chiusura unilaterale**, il client è il primo a decidere di chiudere la connessione. In questa fase, il client invia un segmento TCP

con il flag FIN impostato su 1, il che significa che non ha più dati da inviare. Una volta che il server riceve questo segmento, risponde con un segmento che ha il flag ACK attivato, per confermare la ricezione del FIN. A questo punto, la connessione dal client al server è stata chiusa, ma il server può ancora inviare dati. Il client entra nello stato `FIN_WAIT_1`, in attesa della conferma dal server.

Il server, intanto, passa nello stato `CLOSE_WAIT`, indicando che ha ricevuto il FIN dal client e che sta per chiudere la connessione dal suo lato. Quando il server non ha più dati da inviare, invia a sua volta un segmento con il flag FIN impostato su 1. Il client riceve il FIN del server e invia un ACK per confermare la chiusura della connessione. Dopo aver ricevuto l'ACK, il client entra nello stato `TIME_WAIT`, in attesa di un periodo di tempo per assicurarsi che il messaggio di ACK sia stato correttamente ricevuto dal server.

D'altra parte, nel caso della **chiusura simultanea**, sia il client che il server decidono contemporaneamente di chiudere la connessione. Il client invia un segmento con il flag FIN attivato, lo stesso comportamento della chiusura unilaterale, ma stavolta il server risponde con un segmento che ha sia il flag ACK che il flag FIN impostati. Questo significa che anche il server ha finito di trasmettere i suoi dati e sta chiudendo la connessione. Il client, dopo aver ricevuto questo segmento con il FIN, invia un ACK di conferma.

Dopo che il client invia l'ACK, la connessione è considerata chiusa su entrambi i lati, e il client entra nello stato `TIME_WAIT` per garantire che eventuali ritrasmissioni non causino conflitti. Il server, dal suo lato, entra nello stato `CLOSED` e la connessione è definitivamente chiusa.

Gli stati TCP durante il processo di chiusura sono cruciali per assicurare che ogni lato della connessione gestisca correttamente i messaggi e non ci siano conflitti. Durante la chiusura unilaterale, il client e il server attraversano diversi stati come `FIN_WAIT_1`, `CLOSE_WAIT`, `FIN_WAIT_2`, `LAST_ACK`, `TIME_WAIT` e infine `CLOSED`. Questi stati permettono di tracciare il progresso della chiusura e di garantire che la connessione venga chiusa in modo affidabile e ordinato.

5. **Si consideri una conessione TCP già instaurata e lo scambio di segmenti dovuti ad una HTTP GET. Si indichi il valore assunto dai campi dell'header TCP "Sequence number" (SN) e "Acknowledge Number" (AckN) in entrambe le direzioni,spiegandone le ragioni dei valori indicati**
   Dopo che una connessione TCP è stata instaurata mediante il *three-way handshake*, i due host (client e server) iniziano lo scambio di dati. Analizziamo cosa succede ai campi **Sequence Number (SN)** e **Acknowledgment Number (AckN)** in seguito a una richiesta `HTTP GET`. Viene inviato un segmento HTTP GET dal client. Il client ha scelto un numero

di sequenza iniziale, ad esempio:

$$\text{ISN}_{\text{client}} = 1001$$

Il server, nel handshake, ha inviato il proprio ISN, ad esempio:

$$\text{ISN}_{\text{server}} = 3001$$

Quando il client invia il segmento con la richiesta HTTP GET, che supponiamo essere lunga 200 byte, esso imposta:

$$\text{SN} = 1001, \quad \text{AckN} = 3002$$

(cioè ISN del server + 1).

Il server riceve i 200 byte della richiesta e invia la risposta HTTP. Supponiamo che la risposta sia lunga 1000 byte. Allora il server imposta:

$$\text{SN} = 3002, \quad \text{AckN} = 1201$$

(perché il client ha inviato 200 byte, quindi 1001 + 200 = 1201). Dopo aver ricevuto correttamente i dati, il client invia un segmento ACK senza dati. Quindi il numero di sequenza rimane invariato:

$$\text{SN} = 1201, \quad \text{AckN} = 4002$$

(dato che il server ha inviato 1000 byte a partire da 3002).

6. **In riferimento al livello di trasporto,si spieghi la differenza tra Smoothed Round Trip Time (SRTT) e Rount Trip Time (RTT), e come ciascuno di questi viene calcolato.Infine,si spieghi cos'è il Retransmission Timeout (RTO) e come viene calcolato**
   Nel livello di trasporto, il protocollo TCP utilizza misurazioni temporali per gestire il controllo della ritrasmissione dei segmenti. Le grandezze fondamentali coinvolte in questo processo sono il Round Trip Time (RTT), lo Smoothed Round Trip Time (SRTT) e il Retransmission Timeout (RTO). Ognuna di queste ha uno scopo preciso ed è calcolata in modo specifico per ottimizzare le prestazioni e l'affidabilità della trasmissione dei dati. Il RTT, ovvero tempo di andata e ritorno, è il tempo che intercorre tra l'invio di un segmento TCP da parte del mittente e la ricezione del corrispondente acknowledgment (ACK) da parte del destinatario. Questo valore può variare notevolmente in base a fattori dinamici della rete come la congestione, la latenza variabile dei collegamenti e il routing.

   Ogni volta che un host riceve un ACK per un segmento che non è stato ritrasmesso, può calcolare un nuovo valore di RTT. Tuttavia, l'uso diretto di questi valori grezzi è rischioso a causa delle variazioni repentine tipiche delle reti reali.

Per gestire meglio queste fluttuazioni, TCP introduce una stima filtrata del RTT, chiamata SRTT, ovvero RTT levigato. Questa stima viene aggiornata a ogni nuovo campionamento di RTT, tenendo conto sia del valore attuale sia di quelli passati, tramite una media esponenziale ponderata.

In pratica, lo SRTT permette di ottenere una previsione stabile e affidabile del tempo che intercorre tra l'invio di un segmento e la ricezione del relativo ACK. A livello concettuale, questa operazione consiste nel "smussare" le variazioni troppo brusche del RTT, dando maggior peso ai valori recenti ma senza ignorare del tutto quelli passati. Ciò consente a TCP di reagire ai cambiamenti della rete in modo controllato, evitando oscillazioni improvvise nei meccanismi di ritrasmissione.

$$\text{SRTT} = (1 - \alpha) \cdot \text{SRTT}_{\text{precedente}} + \alpha \cdot \text{RTT}_{\text{campionato}}$$

Il valore di RTO, ovvero timeout di ritrasmissione, rappresenta il tempo massimo che il mittente attende un acknowledgment prima di decidere che il segmento è stato perso e deve quindi essere ritrasmesso. Un RTO troppo breve porterebbe a ritrasmissioni inutili, aumentando il traffico e la congestione. Un RTO troppo lungo, invece, rallenterebbe inutilmente la comunicazione in caso di perdita effettiva di pacchetti.

Per questo motivo, il valore di RTO viene calcolato tenendo conto sia dello SRTT che della variazione del RTT, cioè quanto i valori di RTT oscillano nel tempo. Questa variazione, indicata nel libro come RTTVAR, serve ad aumentare o diminuire dinamicamente l'RTO in base alla stabilità della rete. Quando la rete è stabile, RTTVAR è piccolo e quindi l'RTO si avvicina allo SRTT; quando la rete è instabile, RTTVAR cresce e quindi anche l'RTO viene aumentato per prevenire ritrasmissioni premature.

$$\text{RTO} = \text{SRTT} + \max(G,\ 4 \cdot \text{RTTVAR})$$

$$\text{RTTVAR} = (1 - \beta) \cdot \text{RTTVAR}_{\text{precedente}} + \beta \cdot |\text{RTT}_{\text{campionato}} - \text{SRTT}|$$

3 Capitolo 4/5 : LIVELLO DI RETE

1. **L'header del protocollo IP contiene un campo chiamato "Time to live" (TTL) : si spieghi come viene utilizzato tale campo e il perchè è stato introdotto.**
   Il campo "Time to Live" (TTL) nell'header del protocollo IP è un campo fondamentale per la gestione del traffico di pacchetti nelle reti. Esso ha lo scopo principale di limitare la durata di vita di un pacchetto in rete, prevenendo i loop di routing e garantendo che i pacchetti non circolino indefinitamente tra i router, qualora ci siano errori nella configurazione del percorso.

   Il funzionamento del TTL è semplice: quando un pacchetto IP viene inviato da un host verso un altro, il suo campo TTL viene inizialmente impostato a un valore numerico, solitamente compreso tra 64 e 255, a seconda della configurazione del sistema di origine. Ogni volta che il pacchetto attraversa un router, il valore TTL viene decrementato di uno. Se, durante il suo viaggio, il valore di TTL arriva a zero, il pacchetto viene scartato dal router e viene inviato un messaggio ICMP Time Exceeded al mittente del pacchetto, informandolo che il pacchetto non è riuscito a raggiungere la destinazione prima che il TTL scadesse. Il TTL è stato introdotto per evitare il problema dei loop di routing. Senza un campo TTL, pacchetti mal indirizzati potrebbero continuare a circolare in una rete, bloccando risorse e causando congestionamenti, se ad esempio i router sono configurati in modo errato o se c'è un errore nelle tabelle di routing. Il TTL assicura che un pacchetto non possa mai rimanere bloccato indefinitamente in un ciclo di routing. In pratica, ogni passaggio del pacchetto tra i router riduce il TTL, facendo sì che, prima o poi, il pacchetto venga scartato, evitando così il sovraccarico della rete.

2. **Si spieghi brevemente la funzionalità di frammentazione dei pacchetti IP, mostrando un caso in cui essa è necessaria, gli apparati coinvolti sia nella frammentazione che nel riassemblaggio e in principali campi dell'header coinvolti.**
   La **frammentazione dei pacchetti IP** è una funzionalità che permette di dividere un pacchetto IP in più frammenti più piccoli per facilitarne il trasporto attraverso una rete che supporta una **Dimensione Massima del Pacchetto (MTU)** inferiore a quella del pacchetto originale. Questo processo è necessario quando il pacchetto da trasmettere è troppo grande per essere inviato in un singolo blocco attraverso un determinato segmento della rete, in quanto la MTU di alcune tecnologie di rete (come Ethernet) potrebbe essere inferiore alla dimensione del pacchetto originale. La frammentazione è necessaria quando un pacchetto supera la MTU della rete attraverso cui deve passare. Ad esempio, se un pacchetto IP ha una dimensione di 2000 byte e la MTU di una rete è di 1500 byte, il pacchetto deve essere frammentato in modo che i singoli frammenti non superino la dimensione di 1500 byte. La **frammentazione** e il **riassemblaggio**

dei pacchetti IP coinvolgono principalmente i **router** e il **destinatario finale**:

- **Router**: Se un pacchetto è troppo grande per una rete specifica, il router che sta instradando il pacchetto esegue la frammentazione. Ogni frammento contiene un'intestazione IP separata e viene inviato come un pacchetto separato.
- **Host di destinazione**: Al ricevimento dei frammenti, l'host destinatario si occupa del **riassemblaggio** dei frammenti in un unico pacchetto completo. Questo processo avviene solo quando tutti i frammenti sono stati ricevuti.

Alcuni campi dell'header IP sono cruciali per la frammentazione e il riassemblaggio:

- **Identificazione (Identification)**: Un campo di 16 bit che viene impostato uguale per tutti i frammenti di un pacchetto. Serve a identificare i frammenti appartenenti allo stesso pacchetto originale.
- **Flag (Flags)**: Un campo di 3 bit in cui i primi due bit sono usati per il controllo della frammentazione:
  - Il bit **DF (Don't Fragment)** indica che il pacchetto non deve essere frammentato. Se questo bit è impostato a 1, e la dimensione del pacchetto supera la MTU, il pacchetto viene scartato.
  - Il bit **MF (More Fragments)** è impostato a 1 in tutti i frammenti tranne l'ultimo, per indicare che ci sono altri frammenti successivi.
- **Offset di frammento (Fragment Offset)**: Un campo di 13 bit che specifica la posizione di un frammento all'interno del pacchetto originale. Ogni frammento (eccetto il primo) ha un valore di offset che indica la sua posizione relativa rispetto agli altri frammenti.
- **Lunghezza totale (Total Length)**: Indica la lunghezza totale del pacchetto (header più dati) e viene aggiornata per ogni frammento.

Supponiamo che un pacchetto originale abbia una dimensione di 4000 byte, mentre la MTU della rete è di 1500 byte. Il pacchetto verrà frammentato come segue:

- Il primo frammento avrà 1480 byte di dati e 20 byte di header, con l'offset di frammento impostato a 0. Il flag MF sarà impostato a 1.
- Il secondo frammento avrà 1480 byte di dati e 20 byte di header, con l'offset di frammento impostato a 1480 (poiché è il primo frammento di 1480 byte).
- Il terzo frammento avrà 1040 byte di dati e 20 byte di header, con l'offset di frammento impostato a 2960 (la posizione dove inizia questo frammento rispetto al pacchetto originale). Il flag MF sarà impostato a 0, indicando che è l'ultimo frammento.

3. **Nel protocollo IPV6 c'è un campo denominato "extension header", che a sua volta contiene altri sottocampi.Si spieghi quali sono tali sotto-campi, come vengono utilizzati e perchè il campo "extension header" è stato introdotto**
   Nel protocollo IPv6, gli Extension Header sono intestazioni opzionali introdotte per rendere la struttura del pacchetto più modulare, flessibile ed efficiente rispetto a IPv4. A differenza di IPv4, in cui le opzioni erano inglobate nell'intestazione principale e analizzate da tutti i nodi, in IPv6 solo i nodi interessati esaminano gli Extension Header, migliorando così le prestazioni e la scalabilità del protocollo.

   Ogni Extension Header è posizionato tra l'intestazione base di IPv6 e l'intestazione del protocollo di livello superiore (come TCP o UDP). Se presenti più Extension Header, essi vengono concatenati in una catena ordinata.

   Gli Extension Header contengono generalmente tre sottocampi fondamentali:

   (a) **Next Header:** Indica il tipo di intestazione successiva. Può puntare a un altro Extension Header oppure all'intestazione del protocollo di trasporto. Questo campo è essenziale per la costruzione della catena di intestazioni.

   (b) **Lenght:**Specifica la lunghezza dell'Extension Header, espressa in multipli di 8 byte (esclusi i primi 8). Serve a delimitare correttamente l'intestazione e a individuare quella successiva.

   (c) **Payload :**Variano a seconda del tipo di estensione. Possono includere informazioni per l'instradamento, opzioni per il destinatario, parametri di sicurezza, frammentazione o altro.

   L'introduzione degli Extension Header in IPv6 ha permesso di separare le funzionalità opzionali dal nucleo del protocollo, ottenendo così maggiore efficienza e la possibilità di estendere il protocollo in futuro senza modificarne l'intestazione base. Inoltre, poiché solo i nodi che necessitano di un certo Extension Header lo elaborano, si riduce il carico di elaborazione complessiva nella rete.

4. **Si spieghi il funzionamento del protocollo ARP (Adress Resolution Protocol),senza necessariamente entrare nei dettagli del protocollo stesso,specificando il motivo per cui è stato introdotto tale protocollo.**
   Il protocollo ARP (Address Resolution Protocol) è utilizzato per risolvere la relazione tra gli indirizzi IP e gli indirizzi MAC all'interno di una rete locale (LAN). La sua funzione principale è permettere ai dispositivi di scoprire l'indirizzo MAC associato a un determinato indirizzo IP. Questo processo è fondamentale perché mentre gli indirizzi IP vengono utilizzati per l'instradamento dei pacchetti a livello di rete, i pacchetti stessi devono

essere trasmessi tra i dispositivi utilizzando gli indirizzi MAC a livello di data link.

Quando un dispositivo di rete, come un computer, deve inviare dati a un altro dispositivo nella stessa rete locale, ma conosce solamente l'indirizzo IP del destinatario, si trova di fronte a un problema: ha l'indirizzo IP, ma non l'indirizzo MAC a cui inviare i dati. Per risolvere questo problema, il dispositivo invia una richiesta ARP (ARP Request) in modalità broadcast, cioè inviata a tutti i dispositivi nella rete. La richiesta contiene l'indirizzo IP del destinatario e chiede a chi ha quell'indirizzo IP di rispondere con il proprio indirizzo MAC.

Quando il dispositivo con l'indirizzo IP richiesto riceve la richiesta, risponde con un messaggio ARP Reply contenente il suo indirizzo MAC. Il dispositivo mittente, una volta ricevuta la risposta, può memorizzare l'associazione tra l'indirizzo IP e l'indirizzo MAC nella sua tabella ARP. In questo modo, non sarà necessario inviare nuovamente una richiesta ARP ogni volta che il dispositivo deve inviare pacchetti al medesimo indirizzo IP.

La tabella ARP è una struttura di dati che memorizza le associazioni tra indirizzi IP e indirizzi MAC, e ogni dispositivo di rete mantiene la propria tabella ARP. Le voci nella tabella ARP sono classificate come dinamiche o statiche. Le voci dinamiche vengono aggiunte automaticamente in seguito a una richiesta ARP e hanno un tempo di scadenza, dopo il quale vengono eliminate se non sono più necessarie. Le voci statiche, invece, vengono configurate manualmente dall'amministratore di rete e rimangono nella tabella finché non vengono rimosse manualmente.

ARP è stato introdotto per risolvere il problema della gestione di due livelli di indirizzamento nella rete: gli indirizzi IP a livello di rete e gli indirizzi MAC a livello di collegamento dati. Senza ARP, i dispositivi di rete non sarebbero in grado di risolvere automaticamente gli indirizzi IP nei corrispondenti indirizzi MAC, rendendo impossibile o altamente inefficiente la comunicazione all'interno di una rete locale. Grazie a ARP, i dispositivi possono comunicare in modo rapido ed efficiente, senza la necessità di interventi manuali per la configurazione degli indirizzi.

5. **In riferimento al livello di rete,si spieghi che cosa succede quando un host si connette ad una rete ed ha bisogno di ricever un indirizzo IP (non è necessario andare nei dettagli dei protocolli, è sufficiente descrivere a grandi linee i messaggi scambiati)**
   Quando un host si connette a una rete e ha bisogno di ricevere un indirizzo IP, si attiva una procedura di configurazione dinamica, detta anche plug and play, in cui l'host ottiene automaticamente le informazioni necessarie per comunicare nella rete. Questo processo è solitamente gestito da un server DHCP (Dynamic Host Configuration Protocol).

   (a) Il primo passo è l'invio di un messaggio **DHCP Discover**: il client DHCP crea un datagramma IP contenente il messaggio, con indirizzo IP di destinazione 255.255.255.255 (broadcast) e indirizzo IP

sorgente 0.0.0.0 (che indica "questo host", non ancora configurato). Questo datagramma viene poi passato al livello di collegamento, che lo incapsula in un frame Ethernet broadcast, inviato a tutti i nodi della sottorete.

(b) **Offerta del Server DHCP** : Un server DHCP che riceve il messaggio DHCP Discover risponde con un messaggio DHCP Offer, anch'esso inviato in broadcast, utilizzando di nuovo l'indirizzo IP di destinazione 255.255.255.255. La risposta viene inviata in broadcast perché il client non ha ancora un indirizzo IP assegnato, e potrebbero esserci più server DHCP che rispondono: in questo modo il client può scegliere tra più offerte disponibili. Ogni DHCP Offer contiene:

- l'identificatore di transazione, usato per abbinare la risposta alla richiesta originale,
- l'indirizzo IP offerto,
- mschera di sottorete
- durata del lease,cioè il tempo per cui l'indirizzo può essere usato.

(c) **Richiesta DHCP**: Il client seleziona una delle offerte ricevute e invia un messaggio DHCP Request al server scelto, indicando la volontà di accettare quell'offerta.

(d) **Conferma DHCP**: il server conferma l'assegnazione inviando un messaggio DHCP Acknowledgement, con cui ufficializza l'uso dell'indirizzo IP da parte del client.

A questo punto, il client ha completato il processo di configurazione e può iniziare a utilizzare l'indirizzo IP assegnato per comunicare nella rete. L'indirizzo rimarrà valido per tutta la durata del lease indicata dal server DHCP, al termine della quale il client potrà rinnovare l'assegnazione o richiedere un nuovo indirizzo.

6. **In riferimento al livello di rete, si spieghi che cos'è il Network Address Translation (NAT), mostrando un esempio con le informaioni rilevanti.Si specifichi inoltre per quale motivo tale funzionalità è stata introdotta**
Il Network Address Translation (NAT) è una tecnica utilizzata nel livello di rete, introdotta principalmente per risolvere il problema dell'esaurimento degli indirizzi IPv4 pubblici. Grazie al NAT, una rete privata composta da molti dispositivi può condividere un solo indirizzo IP pubblico per comunicare con l'esterno. In pratica, un intero insieme di host interni — magari decine o centinaia — compare al mondo esterno come se fosse un unico dispositivo, ovvero il router NAT stesso.

   L'idea centrale del NAT è che il router, situato al confine tra una rete privata e Internet, traduce gli indirizzi IP e le porte dei pacchetti in transito. Quando un host interno invia un pacchetto verso Internet, il router sostituisce l'indirizzo IP sorgente privato con il suo indirizzo pubblico e

registra questa mappatura in una struttura chiamata tabella NAT.
Esempio NAT:
Immaginiamo una LAN con:

- Host A: 192.168.1.2
- Host B: 192.168.1.3

Il router NAT ha:

- Interfaccia interna: 192.168.1.1
- Interfaccia esterna (IP pubblico): 203.0.113.5

Supponiamo che Host A voglia contattare un server web su Internet (es. 8.8.8.8, porta 80). Host A genera un pacchetto:

- Sorgente: 192.168.1.2:45678
- Destinazione: 8.8.8.8:80

Il router NAT, prima di inoltrarlo verso Internet, effettua la traduzione:
Cambia l'indirizzo sorgente da 192.168.1.2 a 203.0.113.5
Sostituisce la porta sorgente 45678 con una porta libera, ad esempio 50001
Registra queste informazioni nella tabella NAT, che conterrà una riga simile a:203.0.113.5:50001 → 192.168.1.2:45678
Quando il server remoto risponde al pacchetto, indirizza la risposta a:
Destinazione: 203.0.113.5:50001

Il router NAT riceve la risposta, consulta la tabella NAT e inoltra il pacchetto a 192.168.1.2:45678, ripristinando gli indirizzi originali. Il NAT è stato introdotto principalmente per risparmiare indirizzi IPv4 pubblici, ma ha anche effetti collaterali utili:

- Aggiunge un livello di sicurezza, poiché gli host interni non sono direttamente raggiungibili da Internet.
- Permette di riutilizzare liberamente gli indirizzi privati (es. 192.168.x.x) in qualsiasi rete, senza conflitti.

Tuttavia, ha anche dei limiti: ad esempio, complica la gestione delle connessioni in entrata, e può interferire con applicazioni che usano porte dinamiche o protocolli non NAT-friendly.

7. **Si spieghi brevemente la funzionalità di frammentazione dei pacchetti IP, incluso le motivazione e gli apparati che effettuano frammentazione/Deframmentazione e i campi dell'header coinvolti.Nella spiegazione, si mostri un esempio numerico di un pacchetto frammentato**

La frammentazione IP è una funzionalità del protocollo IP che permette di suddividere un pacchetto troppo grande in più frammenti più piccoli, in modo che possano viaggiare su collegamenti di rete con una MTU (Maximum Transmission Unit) inferiore alla dimensione del pacchetto originale. Questo accade perché ogni tipo di rete fisica impone un limite massimo alla dimensione dei pacchetti che può trasportare. Se un pacchetto IP supera questo limite, deve essere frammentato per evitare che venga scartato.

La frammentazione può essere eseguita sia dall'host mittente, se conosce la MTU del percorso, sia da un router intermedio che rileva un collegamento con MTU inferiore alla dimensione del pacchetto. La deframmentazione, invece, è effettuata esclusivamente dall'host destinatario: i router non ricompongono mai i pacchetti frammentati.
A livello di header IP, la frammentazione utilizza tre campi fondamentali:

- **Identification:** un valore comune a tutti i frammenti dello stesso pacchetto, che permette al destinatario di riconoscerli come appartenenti allo stesso messaggio.
- **Flags:** contiene, tra gli altri, il bit MF (More Fragments), che è impostato a 1 in tutti i frammenti tranne l'ultimo, e il bit DF (Don't Fragment), che può essere usato per vietare la frammentazione.
- **Fragment Offset:** indica la posizione del frammento rispetto all'inizio del pacchetto originale, espressa in unità di 8 byte. Serve per riassemblare correttamente i dati nell'ordine originale.

Un'esempio numerico :
Supponiamo di avere un pacchetto IP lungo 4000 byte, di cui 20 byte sono di header IP, quindi 3980 byte di dati. Se il pacchetto deve passare su un collegamento con MTU di 1500 byte, ogni frammento può contenere al massimo 1480 byte di dati (1500 - 20). Viene suddiviso in :
Primo frammento

- Dati: byte 0–1479
- Offset = 0
- MF = 1
- Lunghezza totale = 1500

Secondo frammento

- Dati: byte 1480–2959
- Offset = 1480 / 8 = 185
- MF = 1
- Lunghezza totale = 1500

Terzo frammento

- Dati: byte 2960–3979
- Offset = 2960 / 8 = 370
- MF = 0 (è l'ultimo)
- Lunghezza totale = 1060 (1040 dati + 20 header)

In questo modo, il destinatario, grazie ai campi Identification, Fragment Offset e MF, può ricostruire il pacchetto originale nell'ordine corretto.

In sintesi, la frammentazione IP garantisce l'invio di pacchetti anche su reti con MTU ridotte, ma introduce complessità e rischi: la perdita di un solo frammento rende l'intero pacchetto inutilizzabile. Per questo oggi si tende a preferire meccanismi come il Path MTU Discovery, che cercano di evitare la frammentazione proattivamente.

4 Capitolo 6 : LIVELLO FISICO

1. **Si descriva il problema del "terminale nascosto" (hidden terminal problem) nelle Wireless LAN e la soluzione adottata dallo standard 802.11**
   Nelle reti wireless LAN, il problema del terminale nascosto si verifica quando due stazioni, pur comunicando con lo stesso Access Point o destinatario, non riescono a rilevarsi reciprocamente. Questo accade a causa della limitata portata del segnale o della presenza di ostacoli fisici. Ad esempio, i terminali A e C possono entrambi comunicare con il terminale B, ma non percepiscono le rispettive trasmissioni. Di conseguenza, A e C potrebbero iniziare a trasmettere contemporaneamente verso B, provocando una collisione a livello del destinatario, anche se ciascuno aveva valutato il canale come libero.

   Per affrontare questo problema, lo standard IEEE 802.11 introduce il meccanismo di controllo RTS/CTS (Request to Send / Clear to Send):

   - Quando una stazione vuole trasmettere, invia un pacchetto RTS al destinatario, indicando l'intenzione di trasmettere e la durata prevista della comunicazione.
   - Se il destinatario riceve correttamente l'RTS e il canale è libero, risponde con un pacchetto CTS
   - Tutti i terminali che ricevono l'RTS o il CTS aggiornano il loro Network Allocation Vector (NAV) e sospendono le trasmissioni per il tempo specificato, evitando interferenze.

   In questo modo, anche terminali nascosti che non "sentono" direttamente l'RTS, ma percepiscono il CTS, vengono avvisati di astenersi dalla trasmissione, prevenendo collisioni e aumentando l'efficienza della comunicazione.

2. **Si spieghi che cosa si intende, quando si parla delle funzionalità del livello 2, per framing e si descriva una delle possibili tecniche con un semplice esempio** Nel modello OSI (Open Systems Interconnection), il livello 2, noto come livello di collegamento dati, ha il compito di fornire un mezzo affidabile di trasmissione dei dati tra due nodi direttamente connessi, gestendo l'affidabilità della comunicazione sul canale fisico. Una delle funzionalità fondamentali di questo livello è il framing. Il framing è il processo mediante il quale i dati provenienti dal livello 3 (rete) vengono suddivisi in frame. Un frame è una unità di trasmissione che contiene non solo i dati effettivi, ma anche informazioni di controllo necessarie per una corretta trasmissione. Le informazioni di controllo all'interno di un frame possono includere:

   - **Indirizzo del mittente e destinatario**: permette di indirizzare correttamente i frame al destinatario previsto.
   - **Delimitatori**: segnano l'inizio e la fine del frame, per evitare che i dati vengano letti in modo errato.

- **Controllo di errore**: come il CRC (Cyclic Redundancy Check), che consente al destinatario di verificare l'integrità del frame ricevuto.

Il framing è essenziale per organizzare i dati in modo che possano essere trasmessi correttamente e in modo ordinato tra due dispositivi nella stessa rete locale. Una delle tecniche comunemente usate per il framing è il byte stuffing. Questa tecnica viene utilizzata per garantire che i delimitatori (che segnano l'inizio e la fine di un frame) non appaiano nei dati veri e propri.

In pratica, il **byte stuffing** prevede l'inserimento di un byte di escape (ESC) ogni volta che un delimitatore appare all'interno dei dati. In questo modo, il ricevente può distinguere tra i delimitatori reali e quelli che fanno parte del contenuto del frame. Immaginiamo che il delimitatore di inizio e fine del frame sia il byte 01111110 (rappresentato come FLAG). Se all'interno dei dati da trasmettere appare un byte identico al delimitatore, questo viene "mascherato" aggiungendo un byte ESC prima del byte stesso.

- Dato da trasmettere: ... 01111110 ...
- Dato trasmesso (dopo byte stuffing): ... ESC 01111110 ...

In questo modo, anche se il byte 01111110 appare nei dati, il ricevente sarà in grado di capire che si tratta di dati e non di un delimitatore. Quando il ricevente legge il byte ESC, sa che il byte successivo fa parte dei dati e non deve essere interpretato come delimitatore.

3. **Si scriva lo pseudo-codice, corredato da commenti, dell'algoritmo CSMA nella variante Collision Detection (CSMA-CD). Si indichi inoltre il motivo che ha portato all'introduzione di tale variante.**
Il Pseudo-codice dell'algoritmo CSMA-CD è il segunete :
1.Se ha una trama da trasmettere
Ascolta il canale
2.Se il canale è libero :
Trasmette la trama
Altrimenti:
Aspetta che il canale si liberi e poi trasmette.
3. Durante la trasmissione:
Se c'è collisione:
Interrompi immediatamente la trasmissione.
Invia un segnale di jam.
Attendi un tempo casuale (backoff esponenziale).
Torna al punto 1.

L'introduzione di CSMA-CD è stata necessaria per minimizzare il tempo sprecato durante una collisione. Nei protocolli pure Aloha e slotted Aloha, il nodo trasmette l'intero pacchetto anche se si verifica una collisione, causando una grave inefficienza. Con CSMA, il nodo aspetta che il canale

sia libero prima di trasmettere, ma se due nodi iniziano quasi contemporaneamente, si ha comunque una collisione.

CSMA-CD migliora ulteriormente:

- il nodo continua a monitorare il canale mentre trasmette,
- rileva immediatamente una collisione,
- interrompe subito la trasmissione,
- invoca un meccanismo di backoff per evitare un'altra collisione immediata.

Questa strategia riduce drasticamente:

- la quantità di dati corrotti,
- il tempo sprecato,
- l'inefficienza della rete, soprattutto in ambienti ad alta contesa come Ethernet.

4. **Si dia una definzione di periodo di vulnerabilità per una rete ad accesso multiplo con mezzo condiviso e si mostri come essa possa essere ricavata nel caso di protocollo ALOHA,supponendo che nella rete le stazioni generino trame della stessa lunghezza pari a L (byte)**

   Il **periodo di vulnerabilità** in una rete ad accesso multiplo è l'intervallo di tempo durante il quale una trasmissione è esposta al rischio di collisione da parte di altre stazioni.

   Nel **protocollo ALOHA semplice**, ogni stazione trasmette una trama non appena questa è pronta, senza alcun tipo di coordinamento o sincronizzazione con le altre stazioni.

   Supponendo che tutte le trame abbiano lunghezza fissa L byte e che la velocità di trasmissione sia R bit/sec, il **tempo di trasmissione** di una trama è:

   $$T_{\text{trama}} = \frac{L \times 8}{R}$$

   Poiché una collisione può verificarsi se un'altra stazione inizia a trasmettere entro un intervallo pari a T_{trama} prima o dopo l'inizio della trasmissione corrente, il **periodo di vulnerabilità** risulta essere:

   $$\text{Periodo di vulnerabilità} = 2 \times T_{\text{trama}}$$

   Questa ampiezza temporale spiega la bassa efficienza del protocollo ALOHA, che può arrivare a un massimo teorico di circa il 18.4%.

Algorithm 1 Protocollo ALOHA (pseudocodice)

```
1: while vero do
2:   if c'è un frame da trasmettere then
3:     Trasmetti il frame in broadcast
4:     if si verifica una collisione then
5:       if estrazione casuale ≤ p then
6:         Ritrasmetti immediatamente il frame
7:       else
8:         Attendi il tempo di trasmissione del frame
9:       end if
10:    else
11:      Attendi il tempo di trasmissione del frame
12:    end if
13:    Attendi il tempo di ritrasmissione del frame
14:    if estrazione casuale ≤ p then
15:      Ritorna al punto 1
16:    else
17:      Rimani inattivo
18:    end if
19:  end if
20: end while
```

5. **Descrivere in pseudo codice i passaggi di ALOHA**

   Descrizione:
   Il protocollo ALOHA è un metodo di accesso casuale utilizzato per la trasmissione di dati su un canale condiviso. Quando un nodo ha un frame da trasmettere, lo invia immediatamente in broadcast. Se più nodi trasmettono contemporaneamente, si verifica una collisione. In tal caso, ogni nodo coinvolto decide in modo probabilistico (con probabilità p) se ritrasmettere subito il frame o attendere. Dopo ogni trasmissione o collisione, si aspetta un tempo prestabilito, dopodiché si decide ancora con probabilità p se tentare nuovamente l'invio o rimanere inattivi. Questo approccio probabilistico serve a ridurre il rischio di collisioni ripetute e a gestire l'accesso al canale in modo distribuito.

6. **Descrivere attraverso pseudo codice gli algoritmi CSMA puro, no-persistent, p-persistent.**

Algorithm 2 CSMA Puro

```
1: while la stazione ha dati da inviare do
2:     if canale libero then
3:         Trasmetti la trama
4:     else
5:         Attendi un tempo casuale prima di ritentare
6:     end if
7: end while
```

Algorithm 3 CSMA No-Persistent

```
1: while la stazione ha dati da inviare do
2:     if canale libero then
3:         Trasmetti la trama
4:     else
5:         Attendi un tempo casuale e riprova
6:     end if
7: end while
```

Algorithm 4 CSMA p-Persistent

```
1: while la stazione ha dati da inviare do
2:     if canale libero then
3:         Con probabilità p
4:         Trasmetti la trama
5:     else
6:         Attendi un tempo casuale e riprova
7:     end if
8: end while
```