

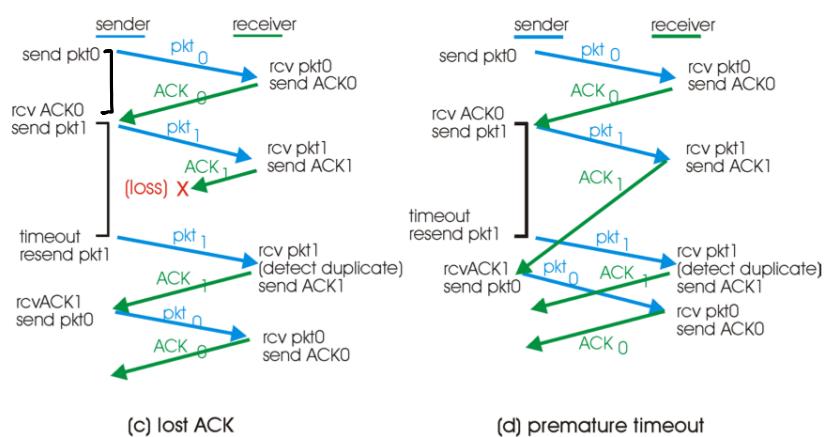
Il principio fondamentale di ARQ implica che nel tentativo di ritrasmettere i dati ottengo altri dati che mi dicono se ho fatto bene o no.

tipi di errore: ci sono dei buchi, dati ripetuti,  
Per correggere tutti questi errori i 4 ingredienti fondamentali da utilizzare sono:

- 1) checksum (codice di rilevazione degli errori)
  - 2) numeri di sequenza (consentono di verificare che la sequenza di dati che ottengo in ricezione è effettivamente ordinata)
  - 3) ACK
  - 4) Timer

Rdt3.0 = stop and wait

## rdt3.0 in action



Il timer si lancia ad ogni pacchetto che si invia e si interrompe prima solo quando si riceve il riscontro. Quando il pacchetto di conferma o inviato si perde il timer finisce e quindi il sender rimanderà lo stesso pacchetto.

È sbagliato dire: non ricevo riscontro. Ha senso solo dire: non ricevo riscontro entro il timer.

$$U = \frac{1}{1 + C \cdot R11/L}$$

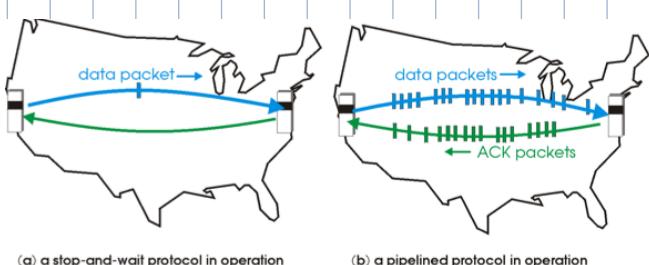
L'efficienza può essere in dipendenza dal prodotto banda ritardo:

Un limite che si riscontra subito nel protocollo è che nonostante abbia tutti i pacchetti pronti ne può mandare solo una alla volta perché deve attendere il riscontro una volta che il receiver lo ha ricevuto, impone che ci sia in volo solo un pacchetto. Quindi c'è una corsa tra due eventi: o ricevo riscontro o scade il timer.

### Introduciamo dei protocolli:

## Pipelined protocols

Consentono di mandare più pacchetti contemporaneamente.



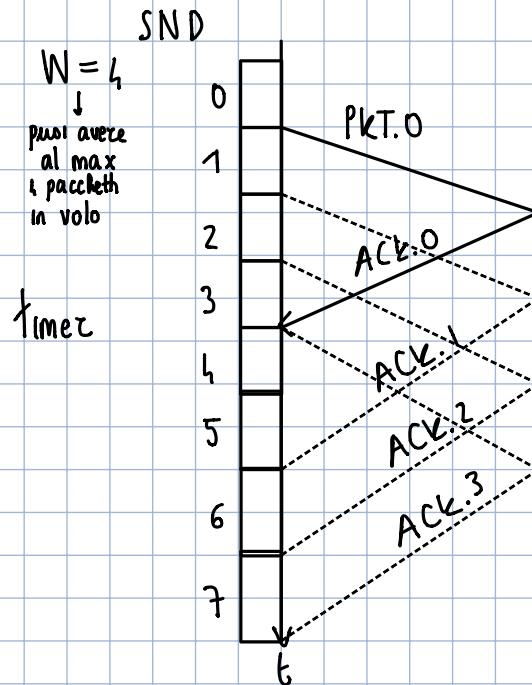
Ci sono due tipi di protocolli pipelined:

- go-Back-N
  - Selective repeat

# Go-back-N

## Lato sender

Si aggiunge una finestra in trasmissione e  $W$  = ampiezza della finestra, ossia il numero di pacchetti che si possono mandare in contemporanea.



Possiamo vederla come una sequenza:

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

...

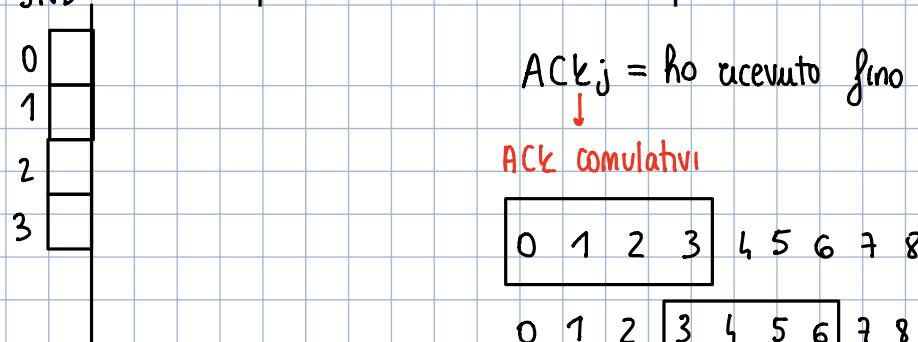
base finestra = seq. più piccola della finestra.  
↓

pacchetto più vecchio  
che deve essere ancora  
riscontrato

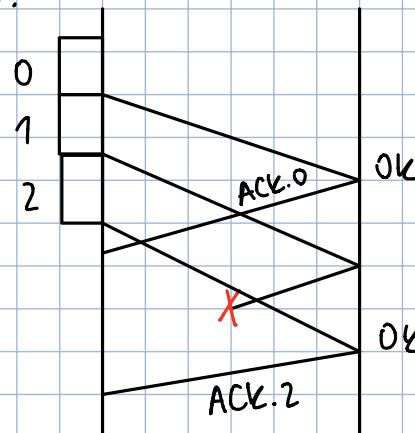
Ogni volta che arriva un riscontro consente di mandare un nuovo pacchetto.

La finestra limita il numero di pacchetti che possono essere in volo, è una sequenza di numeri che va  $[l, \dots, l+W-1]$ .

Poiché ho una finestra si potrebbe cambiare la sequenza dei riscontri:



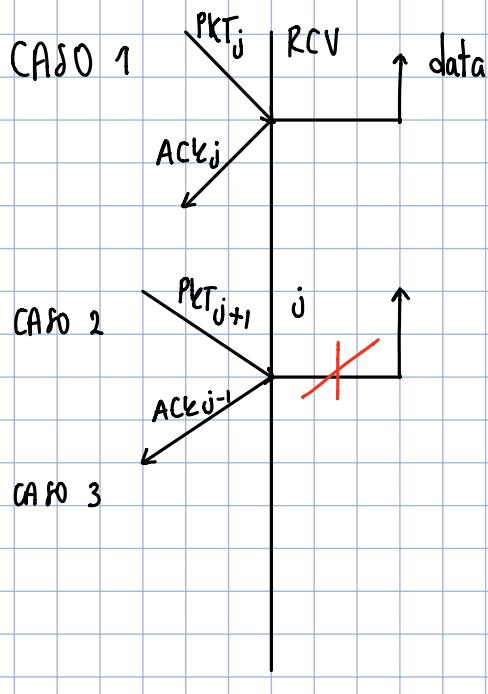
Esempio:



Gli ack comulativi consentono di andare avanti nonostante le perdite.

## Lato receiver

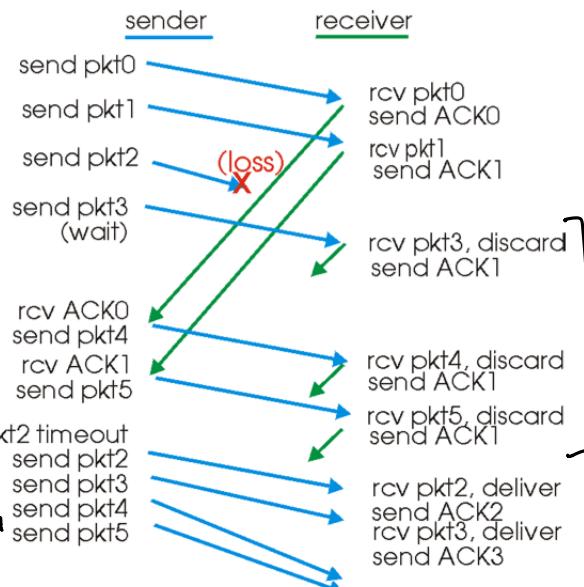
Si comporta come nello stop and wait:



- 1) Verifica checksum
- 2) # seq.

## GBN in action

oltre a rimandare il pack 2 perché il timer è scaduto rimanda anche tutti i pacchetti di cui ancora non ha ricevuto conferma



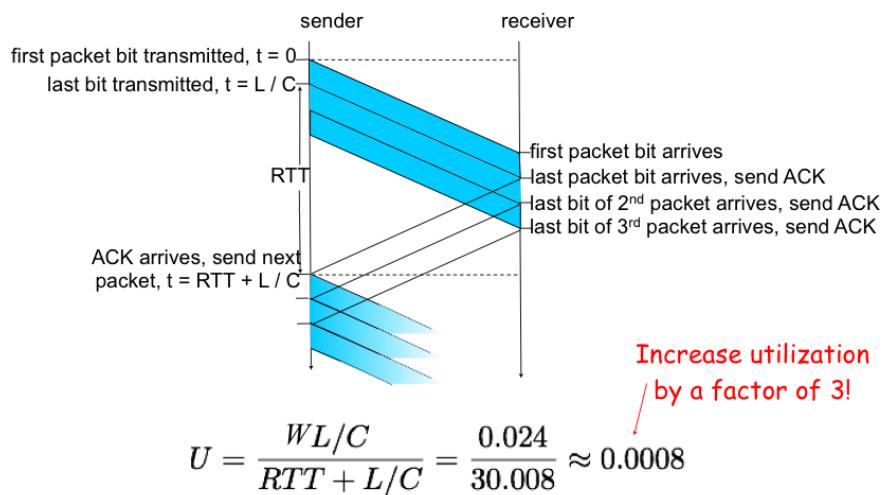
3,4,5 li ha scartati perché 2 non è stato ricevuto.

questa è una perdita di efficienza in quanto magari quei pacchetti sono stati ricevuti.

Tuttavia dal lato del sender se il pacchetto 2 non è stato ricevuto (non vede che ncc. nel RCV) rimanda anche gli altri perché potrebbero essere stati scartati in quanto il RCV aspettava il 2.

Il vantaggio è che il ricevitore è semplice da programmare.

## Pipelining: increased utilization



$$U = \frac{WL/C}{RTT + L/C} = \frac{W}{1 + C \cdot RTT/L}$$

ma  $W$  potrebbe essere  $t \cdot c$   
 $U > 1$ , che non ha senso

$$\Downarrow U = \min \left\{ 1, \frac{W}{1 + C \cdot RTT/L} \right\}$$

$1_c \leq W \frac{L}{C} \Rightarrow$  i pacchetti vengono mandati in sequenza

$$RTT + \frac{L}{C} \leq W \frac{L}{C} \Rightarrow W \geq \left\lceil \frac{C \cdot RTT}{L} + 1 \right\rceil = W_{\text{crit}}$$

Il prodotto banda ritardo è il valore della finestra che dà la max efficienza.

Si può irrobustire ancora di più il protocollo evitando il rimando di pacchetti con potenziali errori.

## Selective Repeat

I 5 elementi fondamentali sono:

- 1)Checksum
- 2)numero di sequenza
- 3)ACK
- 4)timer
- 5)finestra (in trasmissione e ricezione)

SND 0 1 2 3 4 5 6 7 8

$W_{tx} = \text{dim. della finestra in tx (tx window)} [l, \dots, l+W-1]$

RCV 0 1 2 3 4 5 6 7 8

In questo caso se ricevo un pacchetto  $\neq 1$ , lo butto

Dobbiamo anche il ricevitore di una finestra: receiver window

$W_{Rx}$  = dimensione della finestra in zx

RCV

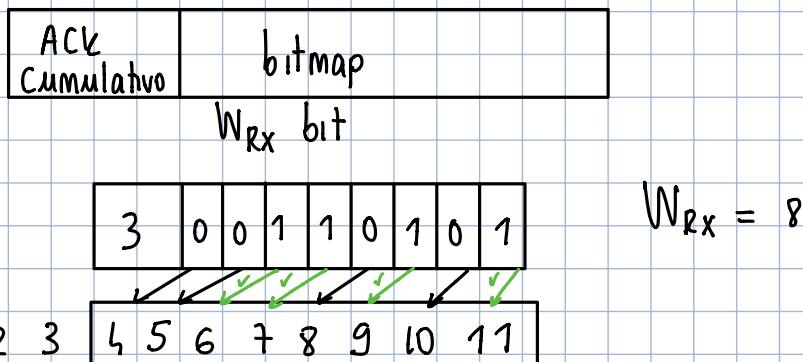


(Ad esempio se ricevo il pacchetto 1 che è all'interno della sequenza, non lo butto, ma nemmeno lo mando. Lo metto in un buffer in attesa di completare la sequenza)

La finestra slitta quando ricevo la sequenza.

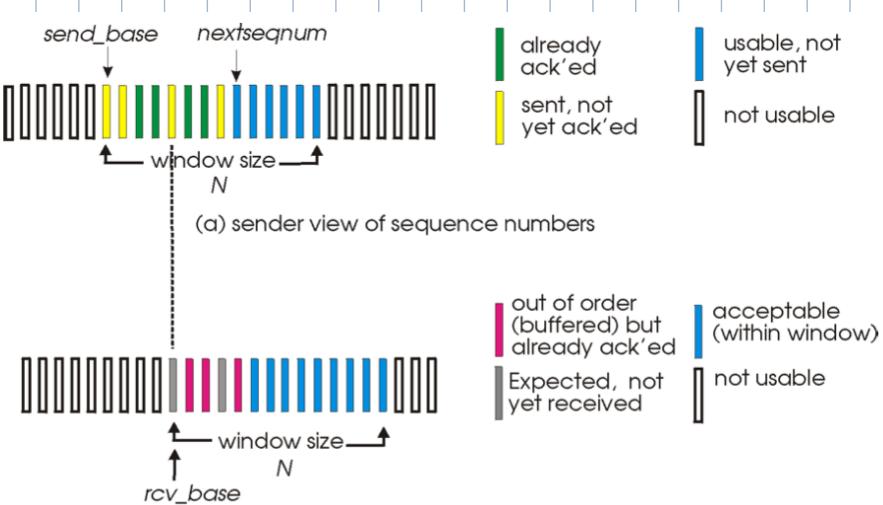
I riscontri non possono essere più cumulativi.

Oppure potrei gestire i riscontri con dei dati più complessi:



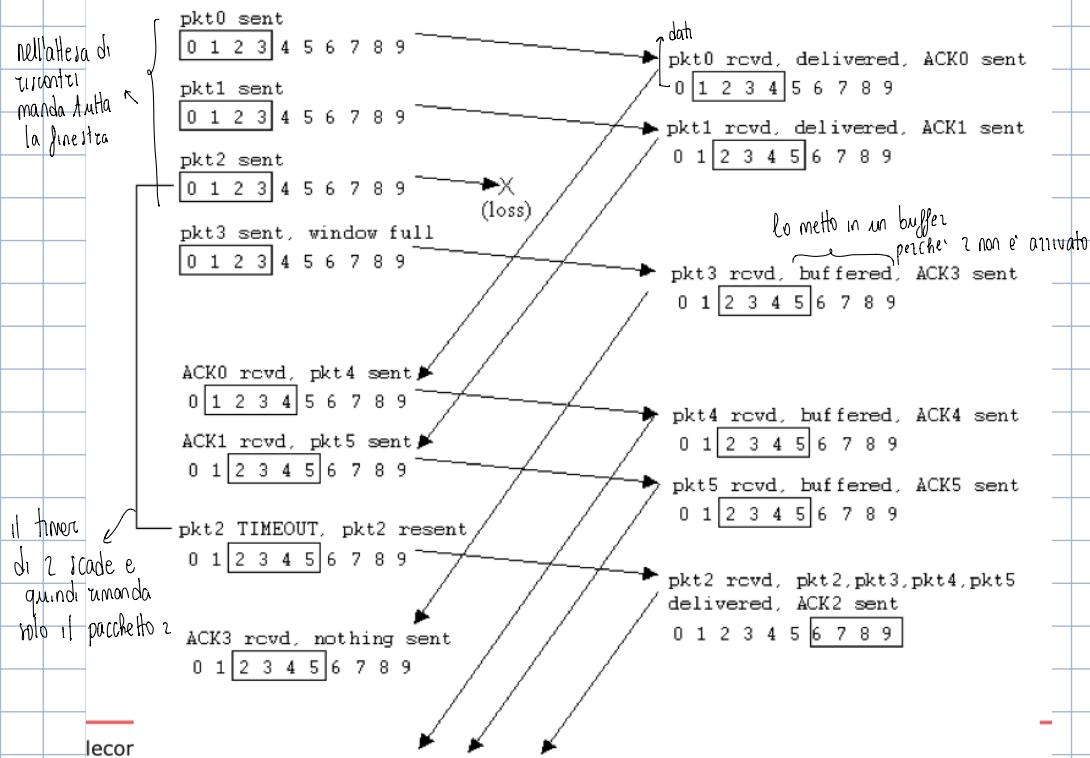
L'ACK cumulativo ci dice che fino al pacchetto 3 è stato ricevuto tutto, mentre la bit map ci dice attraverso gli 0 e 1 cosa gli manca (0) e cosa ha già (1). Quindi mando un riscontro selettivo, delle sole cose che mancano.

Occorre comunque dare un timer al ricevitore per completare la sequenza perché non può rimanere per sempre in attesa.



Quando viene riscontrato il `send_base` la finestra scivola in avanti e mi fermo al primo pacchetto non riscontrato.

## Esempio:



Finora abbiamo considerato sequenze illimitate, dentro la PDU c'è un campo di controllo (bit che servono a distinguere da PDU che portano dati, quelli che portano riscontro ecc), poi ci sarà un numero di sequenza, payload, FCS

da questo dipende la grandezza delle finestre

CONTROL	N seq	payload	FCS
---------	-------	---------	-----

se ha  $h$  bit, # bit di N seq =  $h$   
 $h$  ha una lunghezza di sequenza =  $2^h - 1$

quando esaurisce tutte le combinazioni  
 ripete dal n° di sequenza 0.

0000	→ 0
0001	→ 1
0010	→ 2
0011	→ 3
0100	→ 4
0101	→ 5
0110	→ 6
0111	→ 7
1000	→ 8
1001	→ 9
1010	→ 10
1011	→ 11
1100	→ 12
1101	→ 13
1110	→ 14
1111	→ 15

Il numero di sequenza dipende dalla quantità di dati che dobbiamo mandare.

Esempio:

1 Gbyte

$$L = 1000 \text{ byte}$$

$$1 \text{ GB} = 10^9 \text{ byte}$$

$$1 \text{ pkt} \rightarrow 10^3 \text{ byte}$$

$$\# \text{pkt} = \frac{10^9}{10^3} = 10^6 \text{ pkt}$$

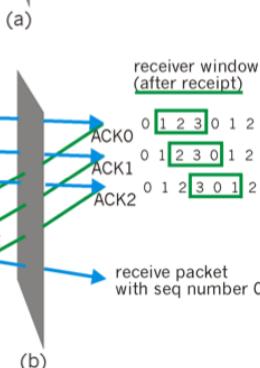
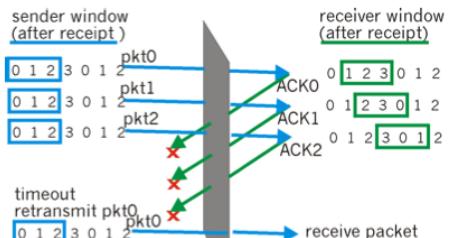
$$\text{bit } N_{seq} = \lceil \log_2 10^6 \rceil = \lceil 6 \log_2 10 \rceil = \lceil$$

## Sequence numbers and window size

Example:

- seq #'s: 0, 1, 2, 3
- window size=3
- receiver sees no difference in two scenarios!
- incorrectly passes duplicate data as new in (a)

Q: what relationship between seq # size and window size?



dimensione max delle sequenze che mi posso permettere

$$SR \quad W_{Rx} + W_{Tx} \leq M$$

$$M = 2^b$$

$$GBN \quad 1 + W_{Tx} \leq M \quad 0, 1, \dots, M-1$$

$$SandW \quad 1 + 1 \leq W \quad 01$$

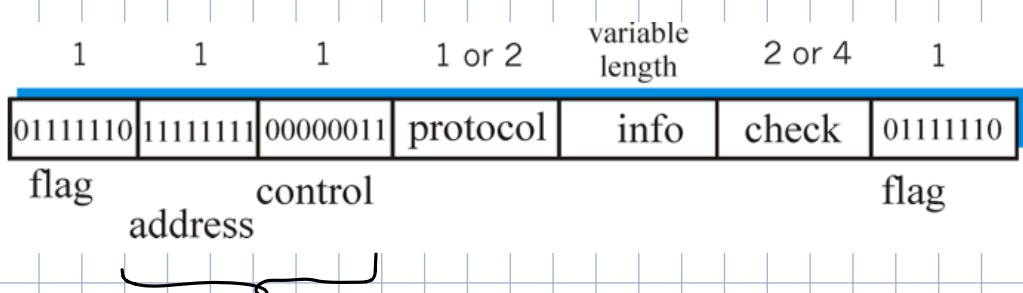
Bisogna bilanciare quanti bit dedico al numero di sequenza e quanto è la dimensione max delle sequenze che mi posso permettere.

Un altro protocollo data link è il PPP, quando non c'è il MAC.

## Point to point Data link control

È un protocollo molto semplice che fa packet framing, ossia delimitazione. Fa soltanto rivelazione degli errori, se sono sbagliati li butta.

PDU di PPP:



non ci sono n° di sequenza perché non c'è recupero

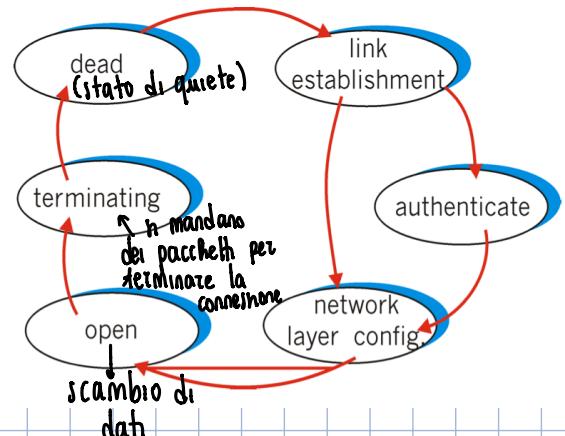
sono fissi (erano stati pensati per sviluppi futuri ma sono rimasti congelati)

PPP si occupa di alzare la connessione e di abbatterla.

Il PPP è un protocollo di data link che non effettua recupero degli errori mediante ritrasmissione, non implementa quindi il meccanismo ARQ.

Quello che fa è delimitazione orientata al byte, introduce flag e poi fa byte stuffing.

Anche il PPP ha pacchetti dedicati allo scopo di gestione.

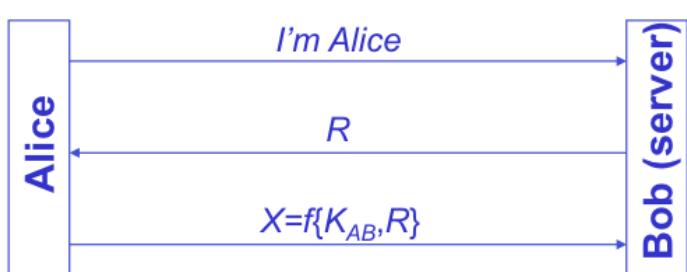


(Rivedere spiegazione grafo)

PPP contiene il protocollo di configurazione di IP.

L'autenticazione (fornire una prova che l'identità è autentica) si può fare in due modi:

- PAP: password authentication protocol
- CHAP: challenge handshake authentication protocol



Alice, un client, si presenta al server rivendicando la sua identità.

Per confermare l'identità il server ha un segreto in comune con Alice, come ad esempio la password. Tuttavia la password è facilmente corruttibile, il CHAP è il protocollo sviluppato di PAP. Alice e Bob si mandano una funzione di risposta (X), è una sfida-risposta.

Chi ascolta può rendersi conto che X è la risposta, senza sapere il segreto, che può essere una password se questa rimane costante (per questo si tende a farle complesse per diminuire la probabilità di scovarla).