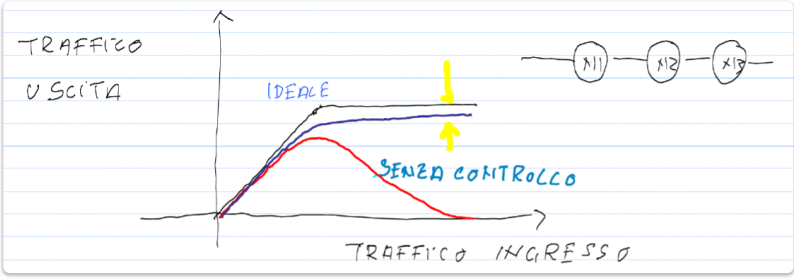
CONTROLLO DELLA CONGESTIONE

Nel contesto delle reti la congestione causa ritardi nel ricevere pacchetti/informazioni che partono da una certa sorgente

- Tempo di viaggio più lungo



- La curva ideale prevede:
  - all'inizio una crescita direttamente proporzionale tra traffico di ingresso e traffico in uscita
  - dopo un certo punto diventa orizzontale: la rete ha raggiunto il massimo $rate$ di trasferimento, quindi il rapporto tra traffico in ingresso e in uscita è costante
- La curva $rossa$ è quella relativa all'andamento del traffico se non introduciamo nessuna regola di controllo $\rightarrow$ per un certo periodo si ha un andamento (quasi) ideale, ma dopo un certo limite (punto di rottura) il collegamento è completamente congestionato - aumentare il traffico di ingresso aumenta l'affollamento del collegamento e causa addirittura l'effetto contrario: diminuisce il $rate$ di uscita (fino ad arrivare al collasso: non passa niente)
- La curva blu intermedia è il caso a cui dobbiamo auspicare, applicando un controllo adatto
  - Va tenuto conto del gap che si viene a creare con la curva ideale, e dobbiamo cercare di limitarlo

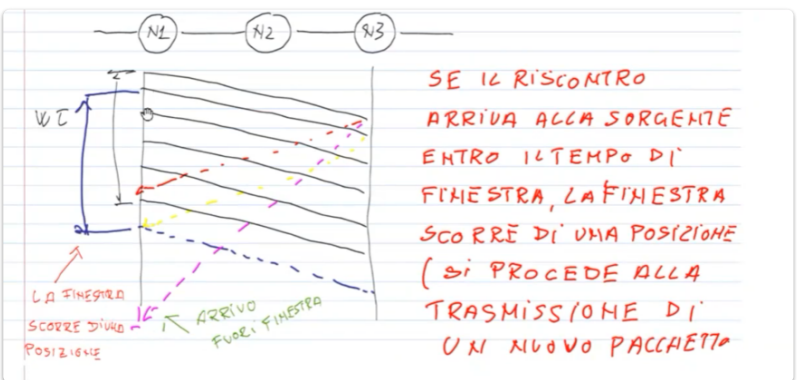
METODI DI CONTROLLO

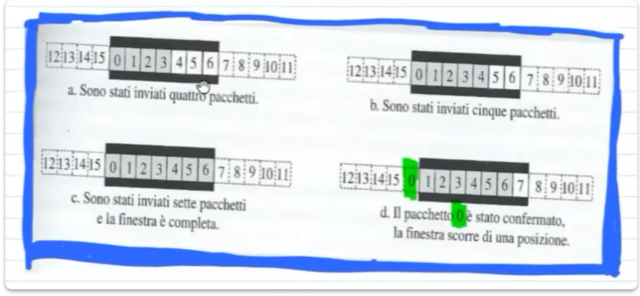
Due metodi:

- **Reattivi**: si attivano quando c'è congestione
- **Preventivi**: attivi da subito per evitare la congestione

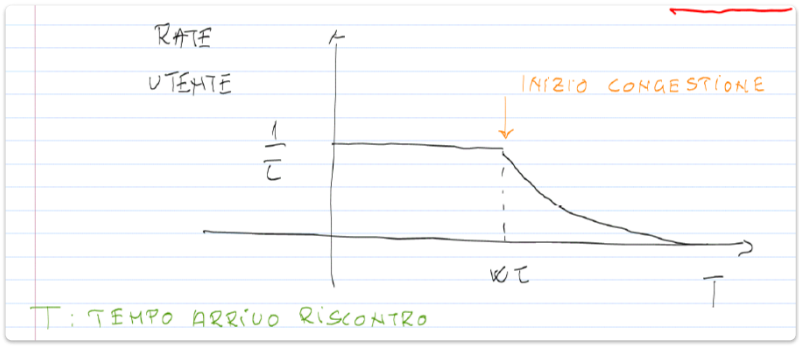
METODO REATTIVO: SLIDING WINDOW

- Utilizzato nel protocollo IP
- Metodo *credit based*: al trasmettitore gli viene concessa la possibilità di inviare una certa quantità di pacchetti, che rappresenta il credito (w)
  - Ogni volta che viene inviato un pacchetto spende 1 di credito
  - Si possono inviare più pacchetti, quindi se ne manda tutti e w per sfruttare tutto il credito
  - Il riscontro dell'invio avvenuto deve arrivare entro un tempo $w\tau$, con τ tempo di trasmissione di un pacchetto
  - Una volta che è stato controllato un pacchetto, la finestra scorre e si invia quindi un nuovo pacchetto
- Se il riscontro avviene fuori tempo di finestra (ovvero impiega un tempo maggiore di $w\tau$), il processo si ferma fin quando tale pacchetto non arriva.
  - Una volta che arriva riparte
  - Ci si ferma perché se c'è stato un ritardo è probabilmente a causa di congestioni interne alla rete





- 😖: meccanismo non user oriented - non garantisce all'utente un rate minimo (il rate è variabile)
  Infatti, ecco com'è il rate:




```
- gestione (implicita delle congestioni)
```

COME SI IMPLEMENTA IN UNA RETE TCP/IP

Le reti devono essere intelligenti e autonome:

- A priori non si può sapere lo stato di congestione della rete, e di conseguenza il valore w che la finestra deve avere affinché il collegamento non abbia congestione (mantenere cioè $rate$ adatto)
  - La rete quindi lo deve scoprire da solo - processo di autoapprendimento -
    - Tale meccanismo si chiama *slow-start*

SLOW START

Obiettivo: trovare una dimensione ottima per la finestra

- Tenendo conto delle capacità dei dispositivi collegati (quindi tenere un $rate$ adatto, non esagerato)
- All'inizio del collegamento si sceglie la dimensione della finestra *più piccola possibile*, ad esempio 1. Se riesce a essere trasmessa senza problemi, si raddoppia la dimensione - e così via
  - Eventi indesiderati quando aumenta la finestra:
    1. Si può saturare la capacità del dispositivo ricevente (limite fisico collegamento)
    2. Si verifica la congestione: il riscontro del pacchetto non arriva
       - Ci si adegua quindi a ridurla o eliminarla: il metodo $slow\ start$ riprogramma una nuova sequenza in modo autonomo impostando come valore della finestra la *metà* di quella che ha causato la congestione
       - Si riparte e si continua allo stesso modo. Può accadere che:
         1. Congestione - quindi si dimezza nuovamente la finestra
         2. Si raggiunge il valore impostato e la congestione non è stata evidenziata
            - Il dispositivo allora, cerca di aggiustare un po' le misure perché si rende conto che dimezzare la finestra forse è stato un po' esagerato ($*$)

CONGESTION AVOIDANCE

- ($*$) Meccanismo conservativo che consente alla sorgente di aumentare le dimensioni della finestra con un incremento *lineare* (non esponenziale)

Una volta aumentata la dimensione della finestra può capitare che (tre possibilità):

1. il collegamento finisce - no congestione! top
2. si raggiunge il limite di capacità del collegamento - no congestione! top
3. la congestione nonostante l'incremento moderato si presenta lo stesso
   - in questo caso si rieffettua lo $slow\ start$ e si riparte

CONGESTION AVOIDANCE FAST RECOVERY

Alternativa al precedente: ideata perché spesso la congestione è temporanea, cioè dipendente da valori che cambiano in fretta da un istante all'altro
Quindi prima di prendere una decisione sul taglio della dimensione della finestra ci si prendere un attimo per riflettere

- FAST RECOVERY -

- Introduce un $warning$, del tipo: "ok la congestione c'è (il pacchetto non è arrivato), però se arrivano riscontri di pacchetti nella stessa batteria w allora si continua ad andare avanti linearmente (di 1)"
  - Il ricevitore continua a chiedere il pacchetto mancante

Può accadere che:

- Scade il $time\text{-}out$
  - Allora c'è congestione: si riparte con lo $slow\text{-}start$
- Se entro 3 pacchetti non ricevuti in sequenza arriva il riscontro del pacchetto che si sta effettuando allora la procedura riprende normalmente!

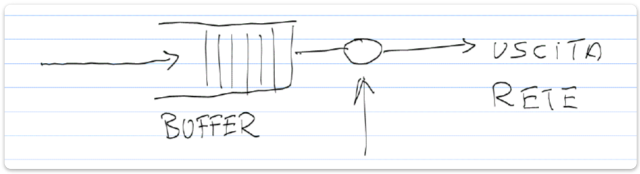
> *Si agisce quindi in maniera più prudente e si dà al trasmettitore la chance di continuare a trasmettere nella speranza che l'effetto di congestioni si vanifichi entro poco*

METODI PREVENTIVI

- I metodi preventivi permettono di fare un controllo del traffico

LEACKY-BUCKET

- Leacky-Bucket (secchio forato)

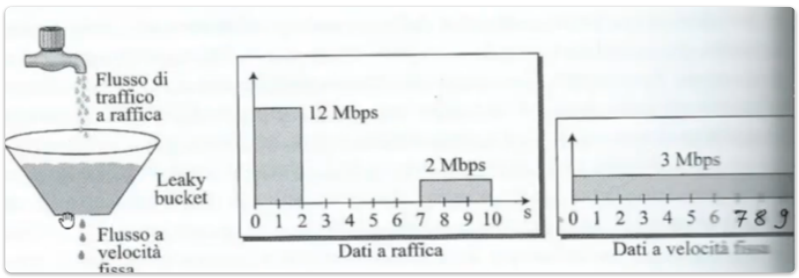


Il nodo sorgente viene schematizzato con un BUFFER dove vengono memorizzati i pacchetti da trasmettere su una determinata porta e con un $trasmettitore$ che è il dispositivo fisico che invia effettivamente il pacchetto

- Se vogliamo controllare il flusso di ingresso come prevedono i metodi preventivi, si può andare a concedere l'abilitazione alla trasmissione con un $rate$ pari a una volta ogni τ secondi, ovvero:

$$transmission\text{-}rate := \frac{1}{\tau}$$

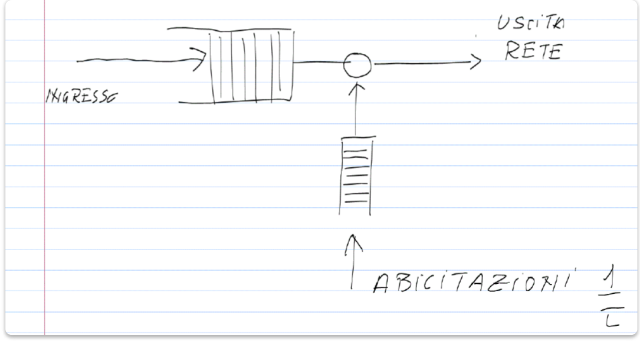
  - Un pacchetto ogni τ secondi cioè
- Il meccanismo permette di controllare in modo esatto il rate massimo!!
  - Coi metodi visti prima non conoscevamo il rate massimo a priori
  - Tuttavia questo metodo non consente di sfruttare il collegamento al meglio: controlla il rate massimo ma non il rate medio!



- Si può aprire quanto si vuole la cannella, ma il flusso che esce è al massimo pari al foro di uscita (imbuto)!

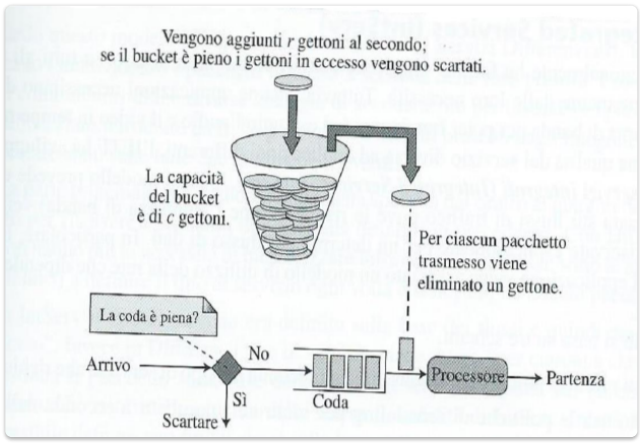
TOKEN BUCKET

- Cestino dei permessi
  Consente di non perdere le opportunità di accesso se queste arrivano quando non ci sono pacchetti da trasmettere
  - Permette di conservare le richieste di inoltro se non ci sono altre operazioni da fare



- Si inseriscono i pacchetti da conservare dentro un buffer, da cui si leggono comunque con lo stesso rate di invio, ovvero $\frac{1}{\tau}$

😃: si ha un controllo perfetto sul rate medio! Non si possono inviare sicuramente più di un pacchetto ogni τ secondi
😳: non si ha più controllo sul rate massimo



- si gestiscono meglio i picchi di traffico: si limita le probabilità di ritardo