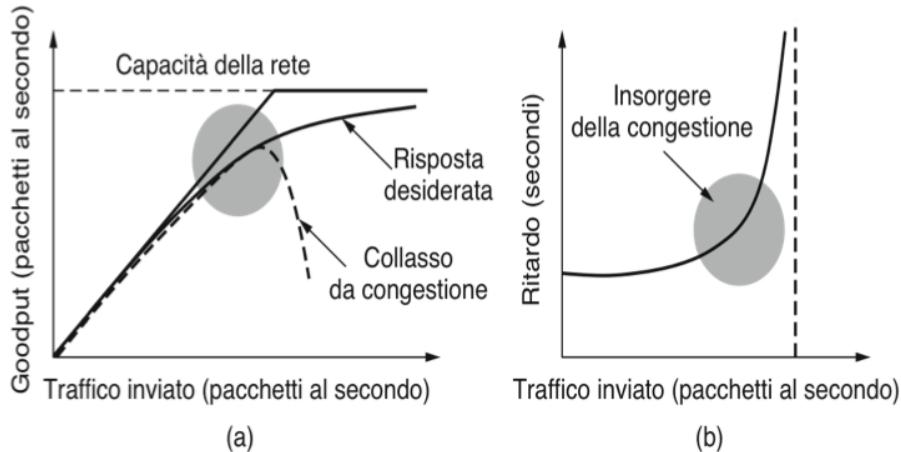


Goodput e ritardo in funzione del traffico inviato



Un sistema ideale ha come curva una bisettrice che poi si appiattisce sul limite della capacità.

La congestione si può gestire in modo:

- **Reattivo:** le sorgenti emetto traffico, questo fa sì che si crea in base all'instradamento a un certo carico sui rami. Per attuare un approccio di congestione reattivo bisogna ideare dei sistemi che si rendono conto di un'eventuale congestione, in quel momento si interviene. Non evita che ci sia la congestione, tiene sottocontrollo il sistema. Agisce se ce ne è bisogno.
- **Proattivo:** lavora a priori per evitare che ci sia congestione. Quando si instaura una connessione si invia un traffico specification (ossia dico come invio il traffico), questo viene distribuito lungo il percorso, se è disponibile, altrimenti si nega la connessione (admission control). Se la connessione avviene, ci sono delle politiche che regolano il traffico che viene prodotto. È come aver stipulato un contratto: la rete offre la capacità e il traffico viene inviato e il policing controlla se viene rispettato il traffic specification.

Tuttavia ad oggi prevale l'approccio reattivo in quanto trovare dei buoni traffic specification è complicato, non si sono ancora trovate soluzioni ottime.

Controllo di congestione reattivo

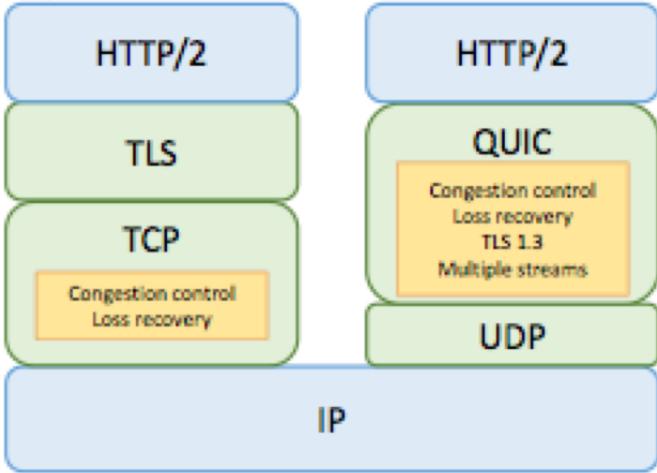
Può seguire due strade:

- Da estremo a estremo:
- Tra i nodi della rete

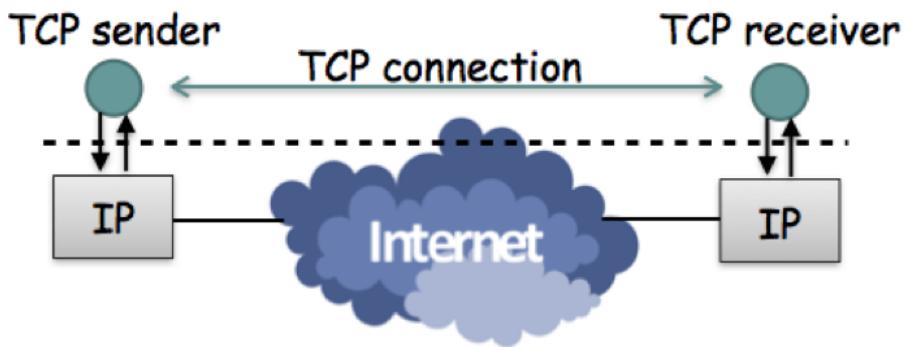
C'è questo dualismo tra i nodi sorgente che mandano il traffico e i nodi di rete che si rendono conto della congestione.

In base al tipo si ha una differenza di strato. Nel caso da estremo a estremo il controllo avviene dallo strato di trasporto in su. Tra i nodi della rete invece a livello di rete.

Quick UDP Internet Connection



Controllo di congestione nel TCP

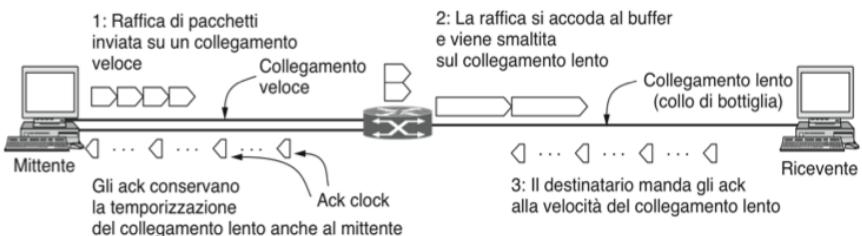


TCP sender e TCP receiver come comunicano?

Il sender prende i suoi dati e li manda al livello inferiore, livello IP. Quest'ultimo interagisce con la rete, saltando da router alla fine arriva alla destinazione. IP si rende conto che è per TCP perché c'è scritto nel campo protocol type del pacchetto, e glie lo manda.

Tra i due quello che può provocare congestione è il sender, per rendersi conto della congestione ci sono ack di avvertenza del receiver.

Una raffica di pacchetti da una sorgente e l'ack clock di ritorno.



Questo adattamento della velocità del bottleneck si ottiene grazie alla finestra. La chiave del controllo della congestione del TCP sta proprio nella finestra.

In TCP si usa il bottleneck model come algoritmo per evitare la congestione. Un pacchetto si dice in volo se è stato mandato ma non è stato ancora mandato un ack di riscontro.

Flight size: numero di pacchetti in volo in un certo istante.

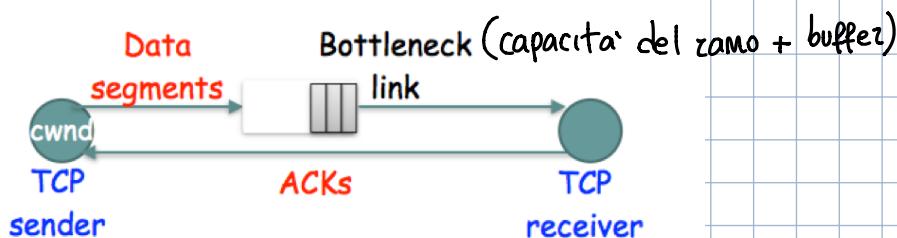
Regole che segue il TCP:

- in ogni istante di tempo il sender TCP rispetta il fatto che il FLIGHTSIZE $\leq W_{tx}$ (il numero di pacchetti in volo non deve superare la finestra di trasmissione). Le finestre vengono contate in pacchetti (quindi la disuguaglianza ha senso).
- La dimensione della finestra cambia nel tempo durante la vita della connessione. È il minimo tra due grandezze: receive window e congestion window (variabile interna del sender che si calcola in base all'algoritmo di controllo di congestione). La finestra può cambiare valore solo quando si riceve un ack:

$$W_{tx} = W = \min \{ rwnd, cwnd \}$$

Supponiamo che la finestra di riduca solamente alla congestion window. Quest'ultima viene aumentata quando si vuole prendere più capacità, viene diminuita quando si rivela la congestione.

Bottleneck model



Algoritmo:

```

CWnd <- IW (Initial window, viene inizializzata. Se è troppo grande il server può mandare a raffica tanti pacchetti e crea congestione. Quindi si parte piano per capire la congestione).
ssthresh <- 64 pacchetti e crea congestione. Quindi si parte piano per capire la congestione.

while pkt loss detected == FALSE do
    on ACK reception
        if CWnd < ssthresh then
            CWnd <- CWnd + 1
        else
            CWnd <- CWnd +  $\frac{1}{CWnd}$ 
        end if
    end while
    CWnd <- CWnd * (1 - β)
    ssthresh <- max {2, flightsize / 2}
    GO TO while CYCLE
  
```

la finestra viene incrementata in due modi diversi perché non si sa quale è la grandezza del bottleneck.
La finestra ottima non deve superare la finestra critica (prodotto banda ritardo)

Questo controllo di congestione viene chiamato AIMD. L'approccio è di aumentare mano mano la velocità di trasmissione (ossia la window size), sondando la larghezza di banda utilizzabile, fino a che non si hanno perdite.

- Aumentare cwnd di 1 quando i segmenti cwnd hanno ricevuto riscontro (ACK).
- Diminuzione moltiplicativa: tagliare in due il cwnd dopo la perdita.
- Ma il cwnd scende a 1 MSS se si verifica un timeout.

Andamento a dente di sega:
sondando la larghezza di banda

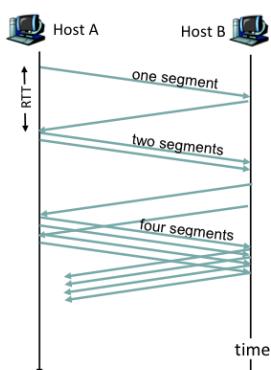


Appena si entra nella fase congestion-avoidance, il valore di cwnd è approssimativamente metà del valore che aveva prima di incontrare la congestione. Il TCP presume che il rischio di congestione sia molto elevato. Quindi, anziché raddoppiare il valore di cwnd ogni RTT sec, il TCP adotta un approccio più prudente ed incrementa il valore di cwnd solo di MSS byte ogni RTT sec. Questa operazione può essere eseguita in vari modi. Il più comune consiste nel far aumentare, al TCP mittente, cwnd di MSS byte (MSS/cwnd) ogni volta che arriva una nuova conferma. Ad esempio, se MSS è 1,460 byte e cwnd è 14,600 byte, allora vengono inviati 10 segmenti in RTT sec. Ogni volta che arriva un ACK (assumendo un ACK per segmento) la finestra di congestione viene allargata di 1/10 di MSS, quindi la finestra di congestione risulterà allargata di 1 MSS dopo che sono stati ricevuti i 10 segmenti ACK.

Ma questa fase di crescita lineare della finestra di congestione (di 1 MSS per RTT) quando dovrebbe terminare? L'algoritmo congestion-avoidance si comporta come se si fosse verificato un timeout. Analogamente alla fase slow start: il valore di cwnd è posto uguale a 1 MSS, e il valore di ssthresh è aggiornato alla metà del valore di cwnd quando si verifica un evento perdita di un pacchetto. Si ricordi che un evento perdita di un pacchetto può essere generato anche dalla ricezione di tre ACK ripetuti. In questo caso, la rete sta continuando a consegnare i pacchetti inviati dal mittente al destinatario (altrimenti il ricevitore non avrebbe ripetuto le conferme). Quindi il comportamento del TCP per questo tipo di evento dovrebbe essere meno drastico di quello dovuto ad un timeout: il TCP dimezza il valore di cwnd (aggiungendo 3 MSS byte, in considerazione dei 3 ACK ripetuti) e registra il valore di ssthresh alla metà del valore di cwnd quando riceve tre conferme per lo stesso segmento.

Slow Start

- When connection begins, $cwnd = 1 \text{ MSS}$
 - Example: if $MSS = 1500 \text{ bytes}$ & $RTT = 200 \text{ msec}$, then initial rate is 60 kbps
- Available bandwidth may be $\gg MSS/RTT$
 - desirable to quickly ramp up to respectable rate
- When connection begins, increase rate exponentially fast until first loss event
 - $cwnd$ is increased by one on every ACK

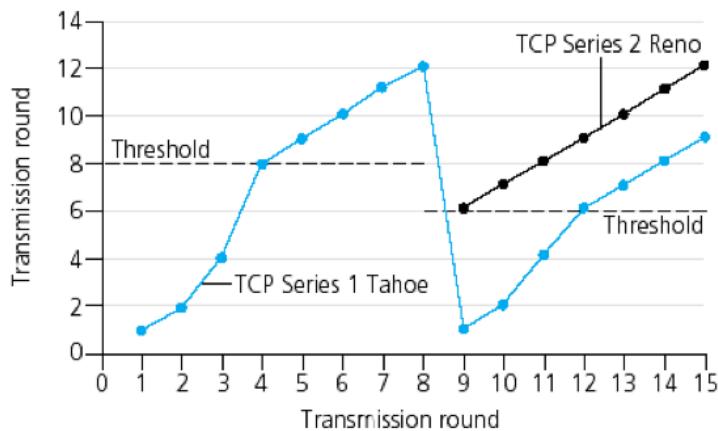


- Quando la connessione comincia, $cwnd = 1 \text{ MSS}$
 - Esempio: se $MSS = 1500 \text{ byte}$ e $RTT = 200 \text{ msec}$, allora il tasso iniziale è 60 kbps.
- La larghezza di banda disponibile può essere MSS/RTT
 - Auspicabile per accelerare rapidamente fino al tasso accettabile
- Quando la connessione comincia, aumentare il tasso esponenzialmente veloce fino al primo evento di perdita
 - $cwnd$ è aumentato di uno su ogni ACK

Quando si inizia a trasmettere su una connessione TCP, si usa un valore iniziale di cwnd molto piccolo, ad esempio 1 MSS (Maximum Segment Size), che corrisponde ad una velocità di trasmissione, approssimativa, di MSS/RTT . Ad esempio se $MSS = 500 \text{ byte}$ e $RTT = 200 \text{ msec}$, la velocità iniziale di trasmissione è di circa 20 kbps. Siccome la banda disponibile al mittente potrebbe essere un po' di più di MSS/RTT , il TCP mittente prova a calcolare anche il margine a disposizione, in tempi rapidi. Così, nella fase Slow Start, il valore di cwnd inizia da 1 MSS e cresce di 1 MSS ogni volta che viene confermato un segmento trasmesso. Ad esempio, il TCP invia il primo segmento in rete ed aspetta di ricevere una conferma. Quando arriva questa conferma, il TCP mittente allarga la finestra di congestione di 1 MSS e manda due segmenti di MSS byte. Se anche questi segmenti sono confermati, il mittente allarga la finestra di congestione di 1 MSS per ognuno dei segmenti confermati, allargando la finestra di congestione a 4 MSS, e così via. Questo processo Slow Start provoca un raddoppio della velocità di trasmissione ogni RTT sec. Cioè il TCP inizia con una velocità bassa, ma poi aumenta la velocità di trasmissione in modo esponenziale.

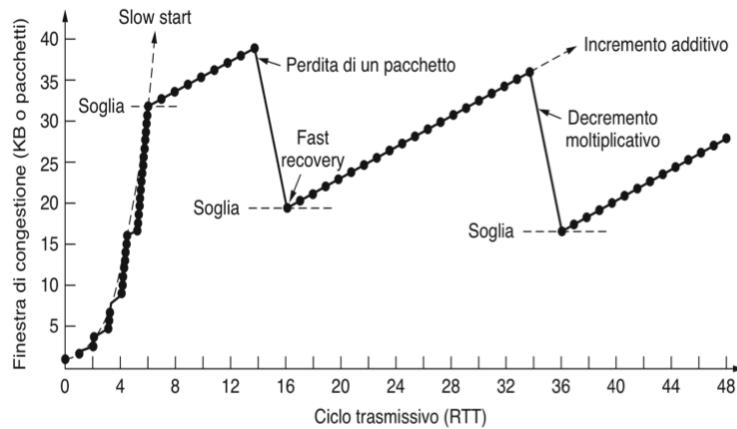
allargando la finestra di congestione a 4 MSS, e così via. Questo processo Slow Start provoca un raddoppio della velocità di trasmissione ogni RTT sec. Cioè il TCP inizia con una velocità bassa, ma poi aumenta la velocità di trasmissione in modo esponenziale.

Slow start threshold (ssthresh)



collegato al valore della variabile ssthresh. Poiché quando viene riconosciuta la congestione, il valore di cwnd potrebbe essere rischioso continuare a raddoppiare cwnd quando raggiunge o supera il valore di ssthresh. Così, quando il valore di cwnd diventa uguale a ssthresh, la fase slow start termina e il TCP passa nella fase congestion avoidance. In questa fase, il TCP incrementa cwnd con precauzione. L'ultimo modo in cui slow start termina si ha quando vengono ricevuti tre ACK ripetuti, nel qual caso, il TCP esegue una ritrasmissione ed entra nella fase fast recovery.

CC in TCP Reno



| State | Event | TCP Sender Action | Commentary |
|---------------------------|---|---|---|
| Slow Start (SS) | ACK receipt for previously unacked data | cwnd = cwnd + MSS, If (cwnd > ssthresh) set state to "CA" | Resulting in a doubling of CongWin every RTT |
| Congestion Avoidance (CA) | ACK receipt for previously unacked data | cwnd = cwnd+MSS/cwnd | Additive increase, resulting in increase of CongWin by 1 MSS every RTT |
| SS or CA | Loss event detected by triple duplicate ACK | ssthresh = cwnd/2, cwnd = ssthresh, Set state to "CA" | Fast recovery, multiplicative decrease. cwnd will not drop below 1 MSS. |
| SS or CA | Timeout | ssthresh = cwnd/2, cwnd = 1 MSS, Set state to "SS" | Enter slow start |
| SS or CA | Duplicate ACK | Increment duplicate ACK count for segment being acked | cwnd and ssthresh not changed |

Ma quando dovrebbe finire questa crescita esponenziale? L'algoritmo Slow start prevede tre modi: Primo, se si verifica un evento perdita di un pacchetto (forse provocato dalla congestione) segnalato dal timeout, il TCP mittente porta il valore di cwnd a 1 e ricomincia la fase di Slow Start. In questo caso però il TCP assegna un valore anche ad una nuova variabile: ssthresh=cwnd/2 (ssthresh è l'abbreviazione di "slow start threshold"), cioè a metà del valore di larghezza della finestra di congestione, quando viene rilevata una possibile congestione. Il secondo modo in cui slow start può terminare la crescita esponenziale della velocità di trasmissione è direttamente

La portata del TCP

- Qual'è la portata media di un TCP?

Ignore slow start:

$$TH = \frac{W_{media}}{RTT_{media}}$$

Sia W_{loss} la dimensione della finestra quando avviene una perdita

$$- W_{loss} = C \times T + B$$

↑ ↑ dimensione buffer collo di bottiglia
 capacità collo di bottiglia = RTT_{base}

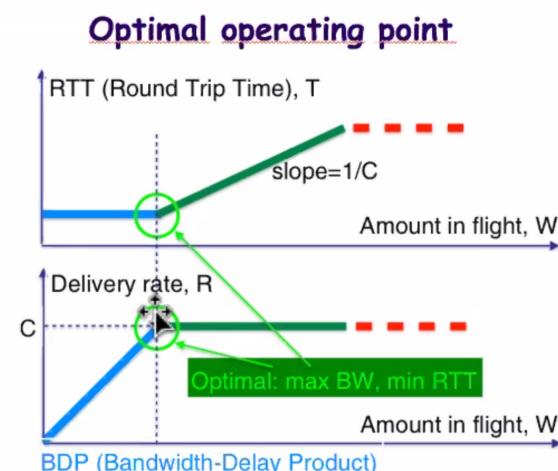
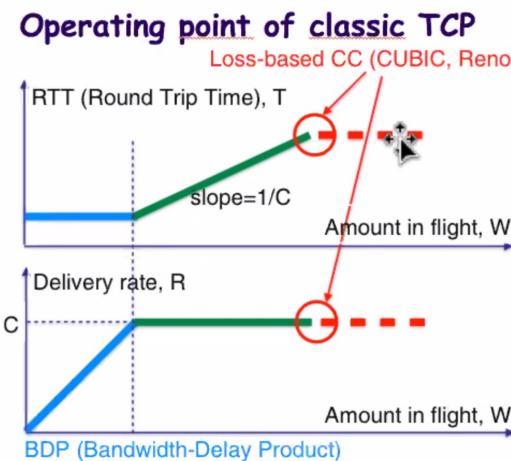
- Appena dopo la perdita la finestra scende a $W_{IN} = \frac{W_{LOSS}}{2}$

$$\cdot W_{\text{media}} = \frac{W_{\text{IN}} + W_{\text{loss}}}{2} = \frac{3 W_{\text{loss}}}{4}$$

- l'RTT quando avviene la perdita e' $T + \frac{B}{C}$

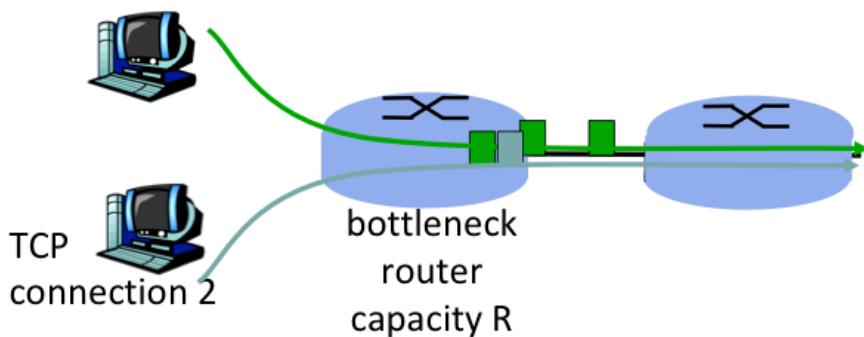
- l'RTT appena avuta la perdita, quando la finestra di congestione e' dimezzata, e' T (se il buffer e' vuota)
 - $RTT_{media} = \frac{(T+T+\frac{B}{c})}{2} = T + \frac{B}{2c}$

$$TH = \frac{\bar{W}}{\bar{R}\pi} = \frac{0.75(CT + B)}{T + \frac{B}{2C}} = 1.5 \frac{1 + \frac{B}{CT}}{2 + \frac{B}{CT}}$$



Equità dell'output

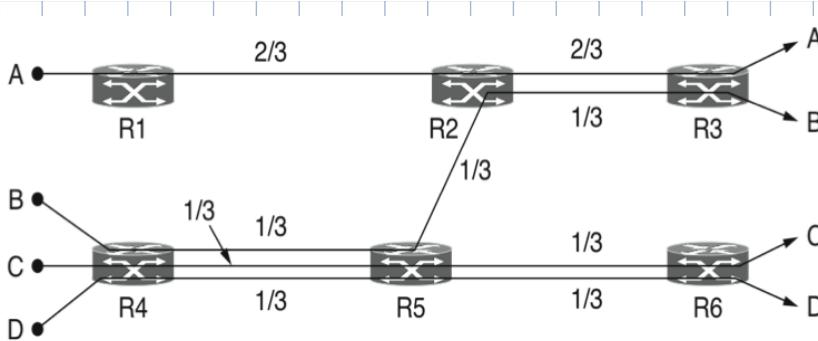
TCP connection 1



- Equità di produzione: stessi valori di portata
- Ci sono molte definizioni differenti per fairness (equità)
 - Equità proporzionale
 - Equità max-min

Si supponga di avere K connessioni TCP, ognuna con un diverso percorso tra sistemi terminali, ma tutte costrette ad attraversare uno stesso canale che ha una capacità R bps. Si suppone che su ogni connessione stia avvenendo un trasferimento di grandi quantità di dati e che sul canale in comune non ci siano pacchetti UDP. Un meccanismo di controllo della congestione è detto equo se la velocità media di ogni connessione è ripartita in misura uguale R/K. Cioè ogni connessione ottiene la stessa frazione di banda condivisa.

Si consideri il caso più semplice di due connessioni TCP che condividono un canale con capacità R. Si assuma che le due connessioni abbiano gli stessi MSS e RTT (quindi hanno la finestra di congestione della stessa larghezza e, di conseguenza, lo stesso flusso), inoltre i due host hanno una grande quantità di dati da trasmettere e sul canale non passano dati di altre connessioni. Si ignori la fase slow-start del TCP. Se il TCP deve ripartire la banda in misura uguale tra le due connessioni, allora la somma dei due flussi dovrebbe essere uguale a R.



Viene impostato un problema di ottimizzazione per la condivisione equa delle risorse limitate:

$$\max_x \sum_{j=1}^N U_j(x_j)$$

$$Ax \leq c \quad x \geq 0$$

Ossia è quella che massimizza l'utilità sociale.

Il TCP è come un algoritmo distribuito al gradiente che cerca un ottimo in un problema di questo tipo.

È un algoritmo stabile sotto certe condizioni.