RETI DI CALCOLATORI

Autunno 2018

docente: Laura Ricci

laura.ricci@unipi.it

Lezione 12:
TCP:
CONTROLLO DELLA CONGESTIONE
15/11/2018

parte di queste slides sono
ricavati da slides pubblicate in corsi
universitari tenuti da colleghi
italiani e stranieri

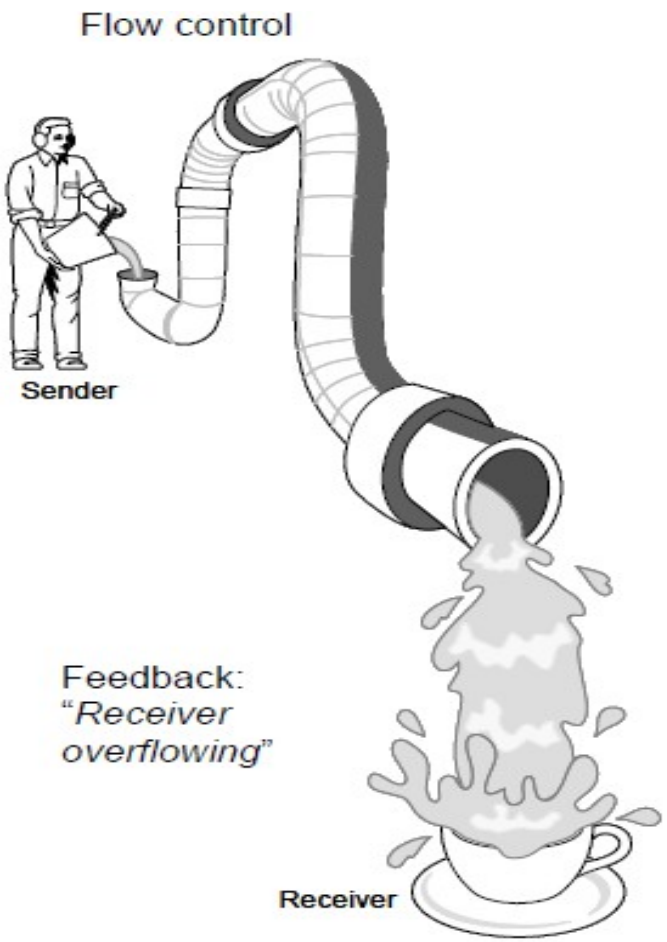
MATERIALE DIDATTICO

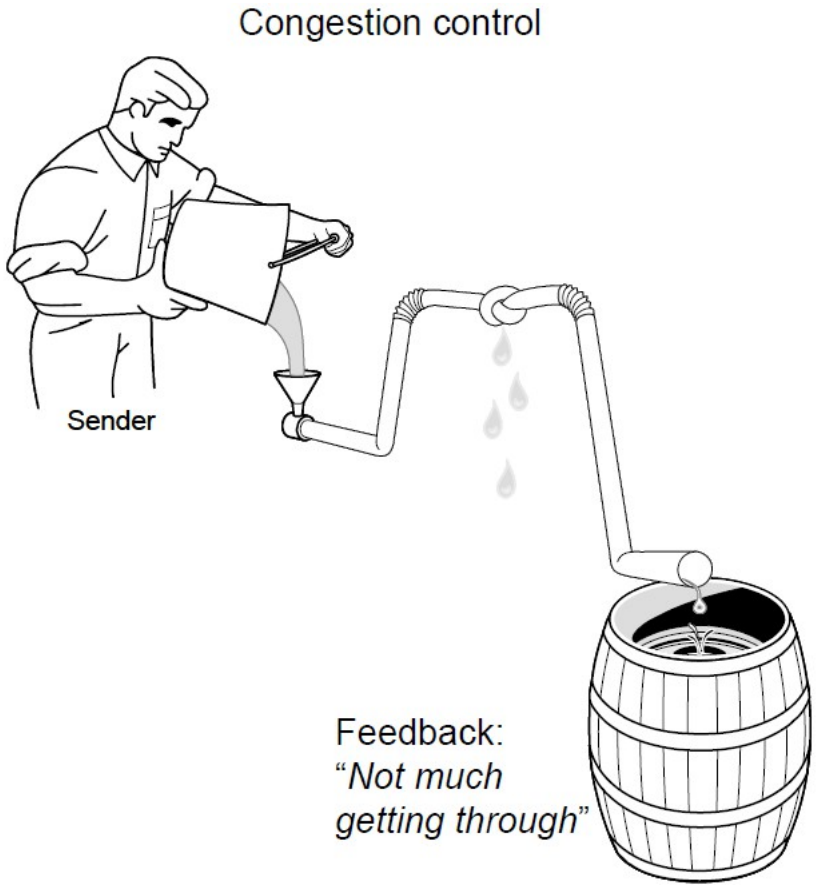
- Fourozan
  - paragrafo 3.4.9
  - paragrafo 3.4.10

IL PROBLEMA DELLA CONGESTIONE

- si manifesta quando il carico sulla rete (numero di pacchetti spediti sulla rete) supera la capacità della rete (numero pacchetti che la rete riesce a gestire)
  - troppe sorgenti che mandano i dati troppo rapidamente
- manifestazioni della congestione della rete:
  - ritardi significativi a causa di lunghi ritardi nei buffer dei router
  - perdita di pacchetti dovuta a buffer overflow nei routers
- controllo della congestione: insieme di meccanismi e tecniche per controllare la congestione della rete e mantenere il traffico al di sotto della capacità della rete

CONTROLLARE FLUSSO E CONGESTIONE





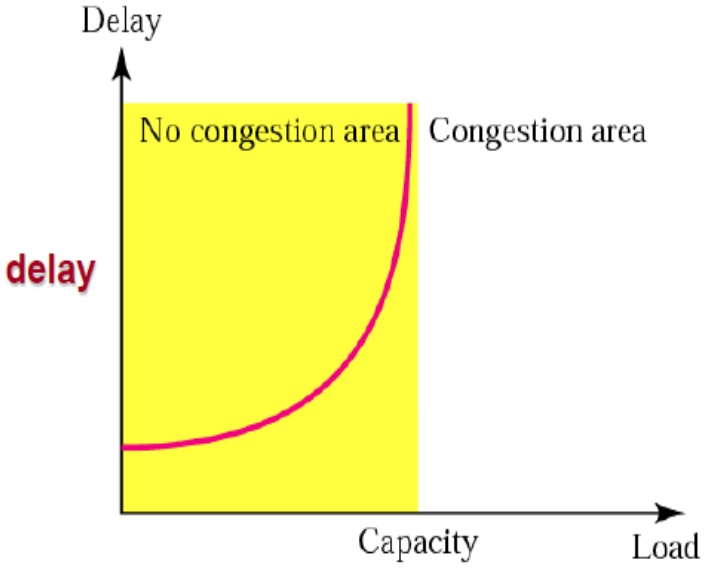
CONTROLLO DI FLUSSO E CONGESTIONE

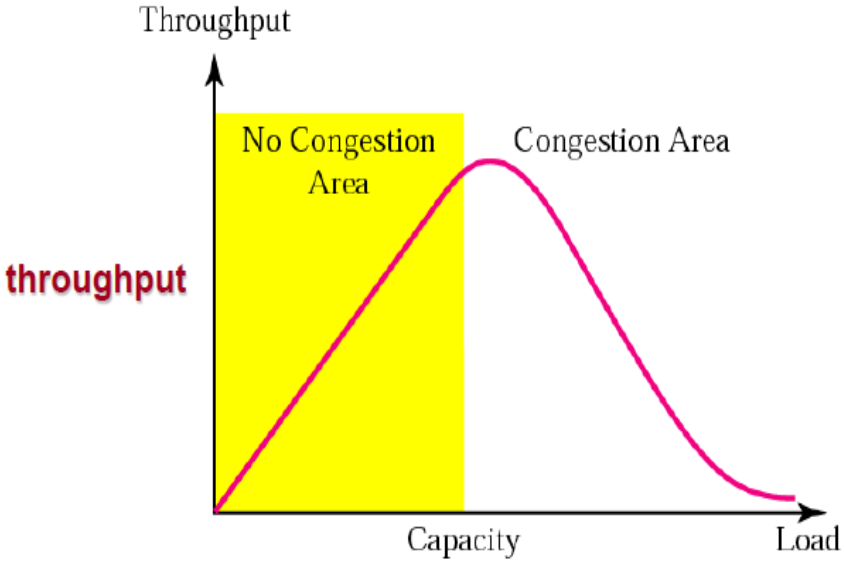
- controllo del flusso:
  - modulare la frequenza di invio del mittente per evitare l'overflow dei buffer di un receiver lento
  - riguarda la trasmissione con un singolo ricevente
- controllo della congestione:
  - modulare la frequenza di invio dei mittenti per evitare di inondare la rete di messaggi
  - "troppe sorgenti trasmettono troppi dati, con una frequenza che la rete non è in grado di gestire"
  - riguarda la rete
- concetti diversi, ma adottano meccanismi simili
  - TCP flow control: notifica receiver window al mittente
  - TCP congestion control: utilizza congestion window
  - TCP window= min{congestion window, receiver window}

AFFRONTARE LA CONGESTIONE: APPROCCI

- ignorare il problema
  - molti pacchetti eliminati e ritrasmessi
  - può casuare "congestion collapse"
- reservations, come nella commutazione di circuito
  - pre-allocazione della banda
  - richiede una fase di negoziazione prima dell'invio dei pacchetti
- pricing
  - non scartare i pacchetti dei "migliori offerenti"
  - richiede un modello di pagamento
- adeguamento dinamico (TCP)
  - ogni mittente inferisce autonomamente il livello di congestione
  - ….ed adatta la frequenza di invio, per "il bene comune"

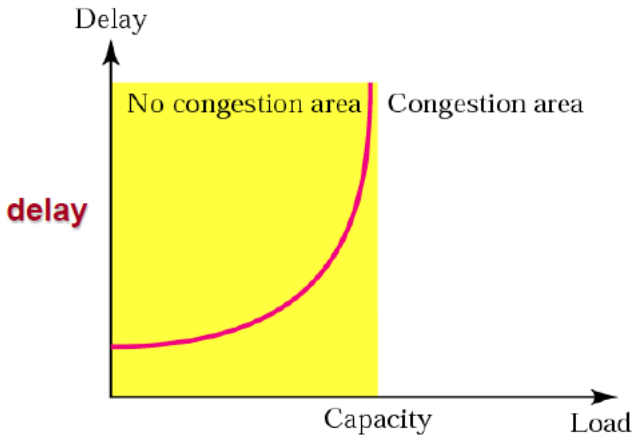
CONTROLLO DI CONGESTIONE

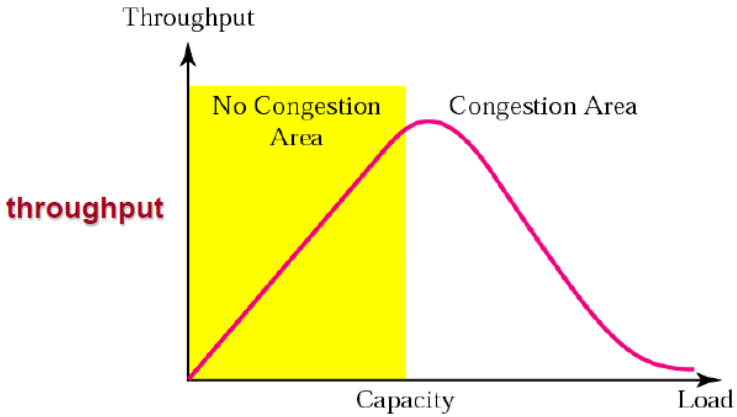




- mano a mano che il carico della rete si avvicina alla sua capacità massima:
  - il ritardo incrementa in modo esponenziale
- mano a mano che il carico della rete si avvicina alla sua capacità massima:
  - il throughput diminuisce in modo esponenziale

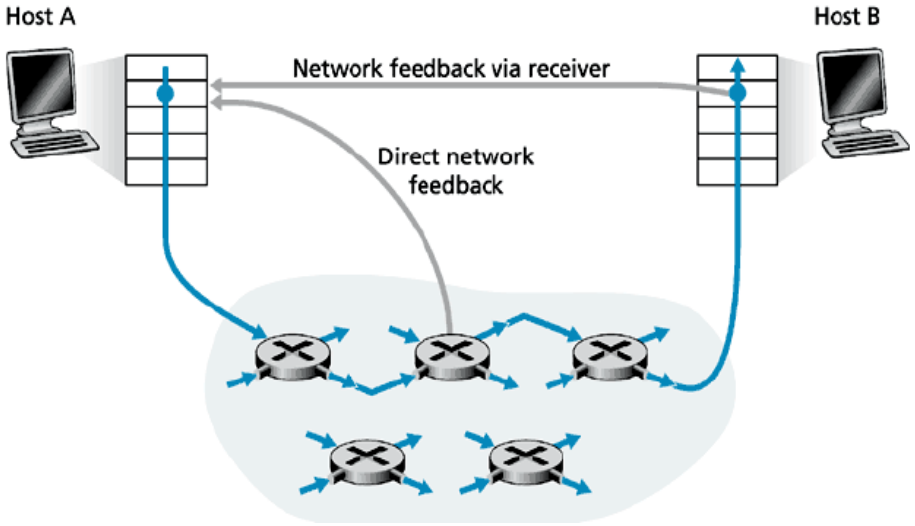
CONTROLLO DI CONGESTIONE





- Si dice che la rete ha un "feedback positivo" sull'aumento del ritardo o la diminuzione del throughput.
- quando un pacchetto è ritardato
  - il mittente, non ricevendo l'acknowledgment, ritrasmette il pacchetto
  - questo rende le code più lunghe ed il ritardo peggiora sempre di più
- quando le code diventano piene ed i router devono scartare alcuni pacchetti.
  - più i pacchetti sono scartati, più vengono ritrasmessi.
  - c'è meno spazio per i pacchetti regolari (non ristrasmessi) nelle code: questo fa diminuire continuamente il throughput

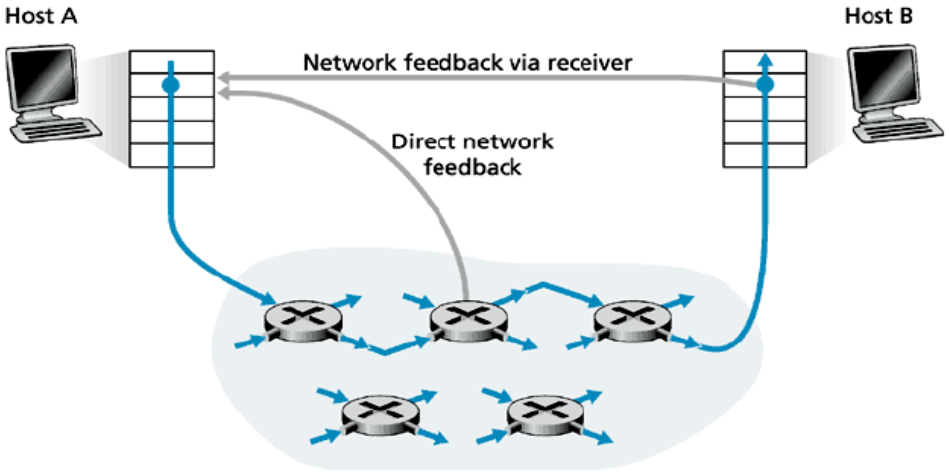
NETWORK ASSISTED CONGESTION CONTROL



explicit congestion control

- i router forniscono un feedback esplicito agli end-system circa lo stato di congestione della rete. Il feedback può essere spedito in entrambe le direzioni
  - dal router direttamente all'end-host mittente
  - un router marca un pacchetto in transito dal server al receiver. Il destinatario, dopo averlo ricevuto avverte il mittente

NETWORK ASSISTED CONGESTION CONTROL



explicit congestion control

- ICMP Source Quench
  - messaggio ICMP inviato da un router all'host sorgente.
  - indica che I Datagram arrivano troppo di frequente rispetto alla capacità di elaborazione
  - l'host deve diminuire la ferquenza di invio dei Datagram
  - non obbligatorio e non utilizzato
- l'approccio dottato attualmente da Internet è diverso: implicit congestion protocol

END TO END CONGESTION PROTOCOL

Implicit congestion control:

- il livello di rete non fornisce alcun supporto esplicito per il controllo della congestione
- gli end systems inferiscono il livello di congestione osservando il comportamento della rete (perdita di pacchetti, ritardi,..)
- TCP window-based congestion protocol
  - perdita di segmenti TCP da una indicazione del livello di congestione
  - TCP diminuisce la dismensione della slideing window, di conseguenza.

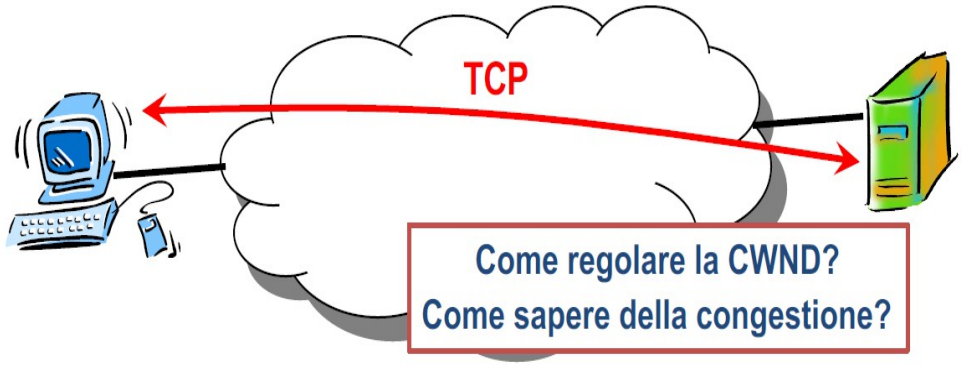
TCP CONGESTION CONTROL: UN PO' DI STORIA

- originariamente: frequenza di invio limitata solamente dal controllo del flusso
  - perdita di pacchetti: i mittenti ritrasmettono i pacchetti
- "congestion collapse" in Ottobre 1986
  - primo episodio osservato di congestione
  - osservato tra Lawrence Berkeley National Laboratory e UC Berkeley (distanti 300 metri e pochi Internet hops) passa da 32K/s a 40 bits/sec
  - cambiamento repentino di ben tre ordini di grandezza
- fenomeno studiato da Van Jacobson, University of California
  - proposto l'algoritmo AIMD (vedi slide successive) tra il 1987 ed il 1988
  - idea: estendere il protocollo TCP esistente basato su sliding-window, ma adattare la dimensione della finestra alla congestione
    - non richiesto alcun aggiornamento a routers o applicazioni!
    - patch di poche linee di codice nella implementazione TCP
    - altro esempio di "keeping things simple" in Internet
    - conseguenza dell'evoluzione di Internet, piuttosto che una pianificazione

L'APPROCCIO PROPOSTO DA JACOBSEN

- il modo più naturale per controllare la frequenza di immissione in rete dei dati per il TCP è quello di regolare la finestra di trasmissione.

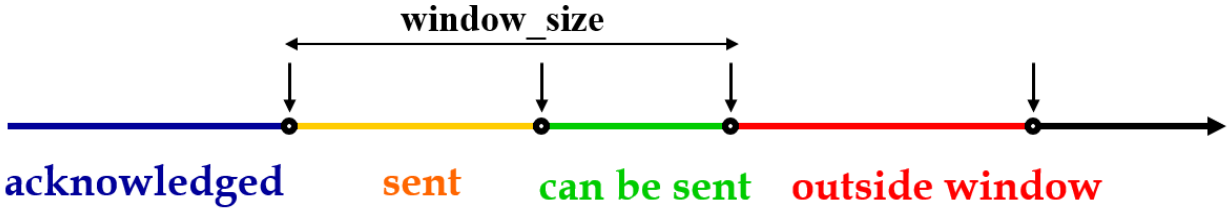
- il trasmettitore
  - mantiene una Congestion Window (CWND) che varia in base agli eventi che osserva
    - ricezione ACK
    - perdita di segmenti segnalata dallo scadere del time out
  - la reazione ad un evento di congestione è quella di ridurre la finestra di invio



ADVERTISED E CONGESTION WINDOW

- congestion Window: CWND
  - numero di bytes che possono essere spediti senza mandare in overflow i routers (approssimazione)
  - calcolata del mittente usano l'algoritmo di controllo di congestione
- flow control window: AdvertisedWindow (RWND)
  - numero di bytes che possono essere spediti senza mandare in overflow il buffer del ricevente
  - determinato dal ricevente e riportato al mittente
- Window del mittente = min{CWND, RWND}
- nel seguito assumeremo, per semplicità che
  - RWND >> CWND
  - E che l' unità di misura di CWND è MSS
  - ...ma tenere a mente che le implementazioni reali mantengono la CWND in bytes

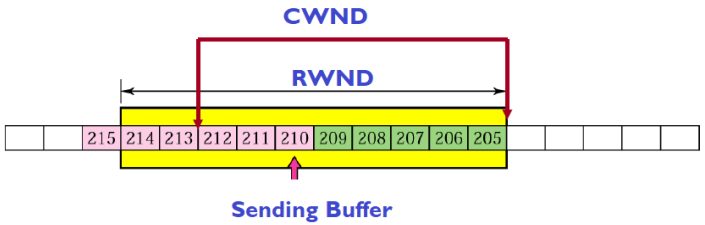
TCP WINDOW SIZE



- La frequenza di trasmissione è definita come la dimensione della finestra
  - variare la dimensione della finestra per controllare la frequenza di spedizione
  - ricalcolata dinamicamente dal mittente, in funzione della congestione percepita dalla rete
- Window size (win) , è il valore più piccolo
  - RWND: receiver window (controllo del flusso)
  - CWND: congestion window (controllo della congestione)

win = min(rwnd, cwnd)

- RWND impostato dal destinatario
- come viene impostato CWND?



CONTROLLO DI CONGESTIONE: IDEE DI BASE

- al momento del rilevamento della congestione
  - l'end host decrementa la frequenza di invio
    - diminuisce la dimensione della finestra

- ma, se le condizioni cambiano?
  - supponiamo sia disponibile più banda, i paccehtti vengono ricevuti normalmente
  - ...sarebbe un peccato non sfruttare la banda disponibile

- quando non rileva la congestione
  - prova ad aumentare la frequenza di invio
  - e vede se tutti i pacchetti vengono consegnati con successo...
  - in quel caso aumenta la dimensione della finestra di invio

TCP CONGESTION CONTROL: AIMD

L'algoritmo AIMD (Additive Increase, Multiplicative Decrease), comprende due fasi principali:

- SlowStart:
  - permette al mittente di scoprire il valore giusto della sliding window, partendo da un valore conservativo ed incrementando la dimensione della finestra
  - simile ad accelerare per trovare la giusta velocità su una autostrada
- Congestion avoidance
  - permette al mittente che ha raggiunto il "valore giusto" della sliding window di rimanere vicino a quel valore, adattandosi alle condizioni della rete
  - simile a mantenere la velocità ideale su una autostrada

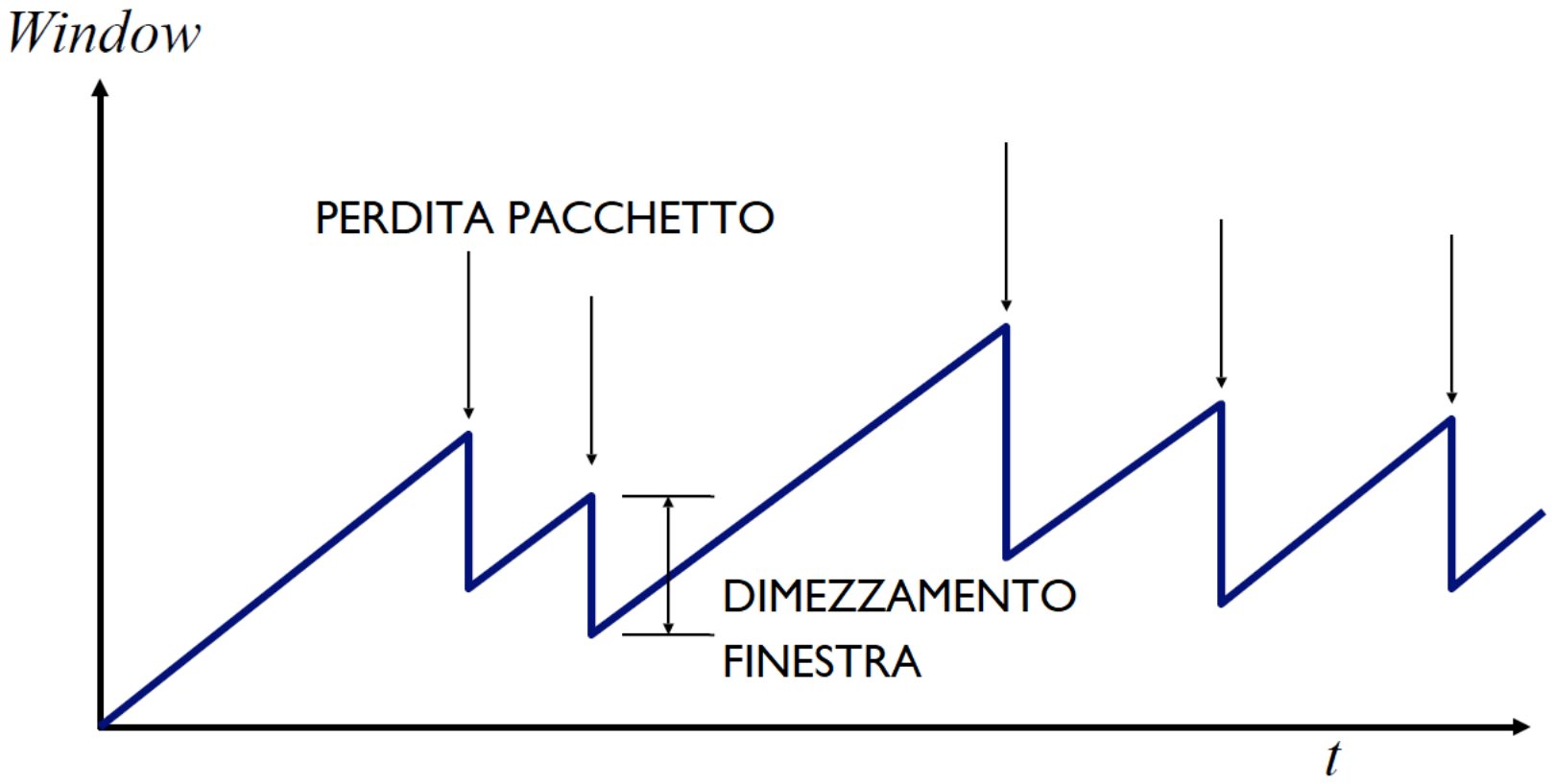
TCP: POLITICHE

- Diverse politiche proposte (RFC 2581)
  - slow start (SS)
  - congestion avoidance (CA)
  - fast re-transmission
  - fast recovery
- TCP Tahoe: SS + CA + fast re-transmission
- TCP Reno: tutte le quattro politiche
- TCP NewReno: come TCP Reno + miglioramento della politica di fast recovery
- altre varianti:
  - TCP SACK
  - TCP Vegas,
  - TCP Westwood, …

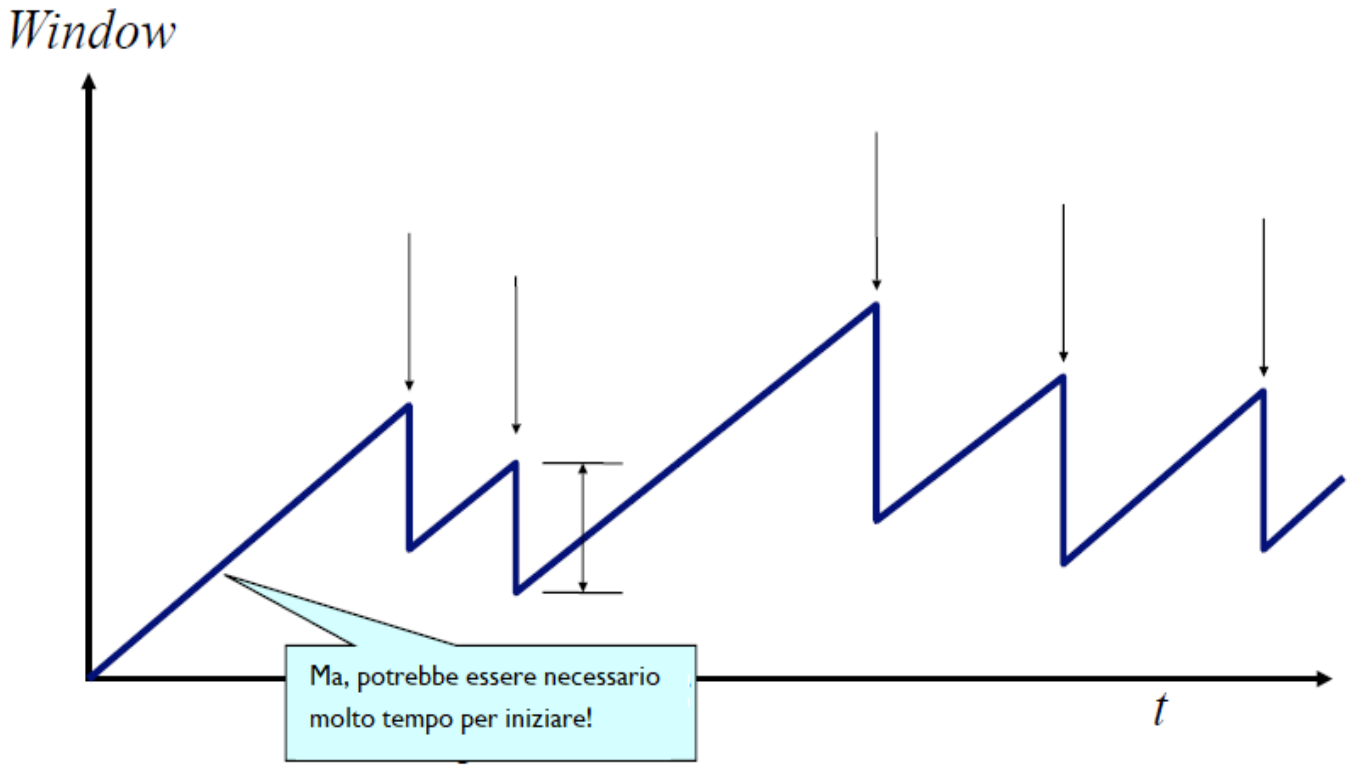
ADDITIVE INCREASE, MULTIPLICATIVE DECREASE

- di quanto incrementare/decrementare la finestra di invio?
  - incrementare in modo lineare, decrementare in modo moltiplicativo
  - condizione necessaria per la stabilità (vedi slides successive)
  - motivo: le conseguenze di una finestra sovra-dimensionata sono peggiori di quelle di una finestra sotto dimensionata
    - finestra sovra dimensionata: eliminazione e ritrasmissione dei pacchetti
    - finestra sotto dimensionata: throughput inferiore al massimo consentito
- Multiplicative decrease
  - quando si verifica la perdita di pacchetti,
    - dimezzare la dimensione della finestra di congestione, oppure
    - riportare drasticamente la dimensione della finestra ad un segmento
- Additive increase/Congestion avoidance
  - quando i pacchetti vengono riscontrati regolarmente, incrementare linearmente la dimensione della finestra

TCP SAWTOOTH



MA COME INIZIALIZZARE UN NUOVO FLUSSO?

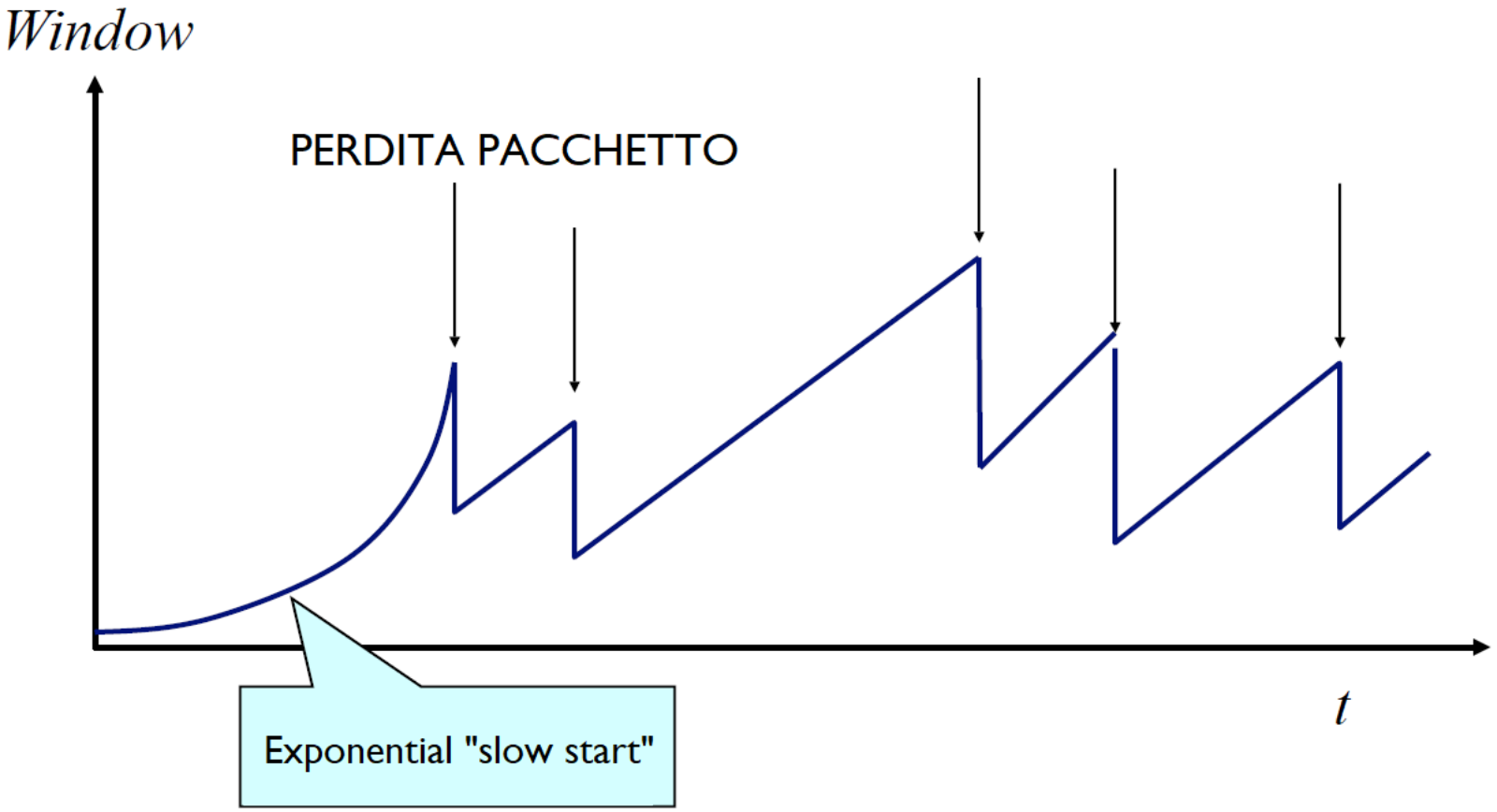


- necessario essere prudenti ed iniziare con una congestion window piccola, per evitare di congestionare la rete
- ma potrebbe richiedere molto tempo prima di raggiungere la frequenza voluta!

CONGESTION CONTROL: SLOW START

- scopo della fase di slow start è stimare la banda disponibile, partendo con valori bassi (safety) ed aumentando rapidamente (efficienza)
- iniziare con una piccola finestra di congestione
  - CWND settata a Max Segment Size (MSS): frequenza iniziale è 1 MSS/RTT
- un incremento lineare potrebbe richiedere un lungo tempo "di accellerazione"
- soluzione: inizialmente incrementare esponenzialmente la frequenza
  - ….fino alla perdita del primo pacchetto
- slow-start in realtà è "fast start": da che deriva il nome "slow start"?
  - TCP progettato inizialmente senza alcun meccanismo di controllo della congestione
    - il mittente utilizza una sliding window statica e spedisce l'intera finestra
  - con "slow start" l'invio è molto "più lento" (anche se veloce, in assoluto), rispetto agli algoritmi precedenti

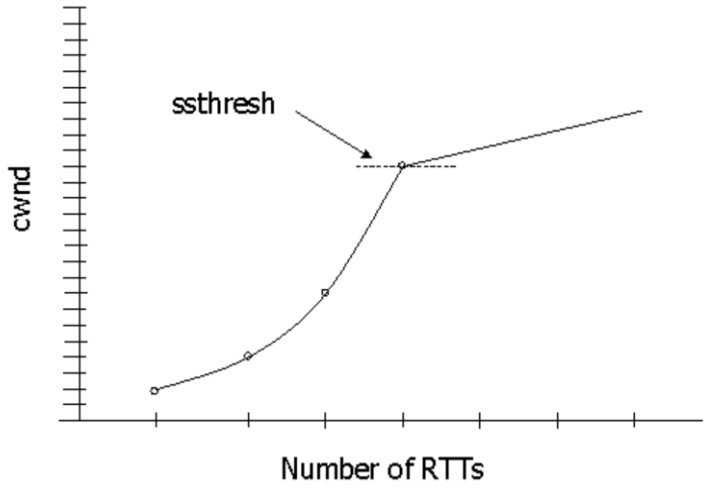
SLOW START E TCP SAWTOOTH



- in uno stato stabile, CWND oscilla attorno alla dimensione ottimale della finestra

SLOW START: SSTHRESH

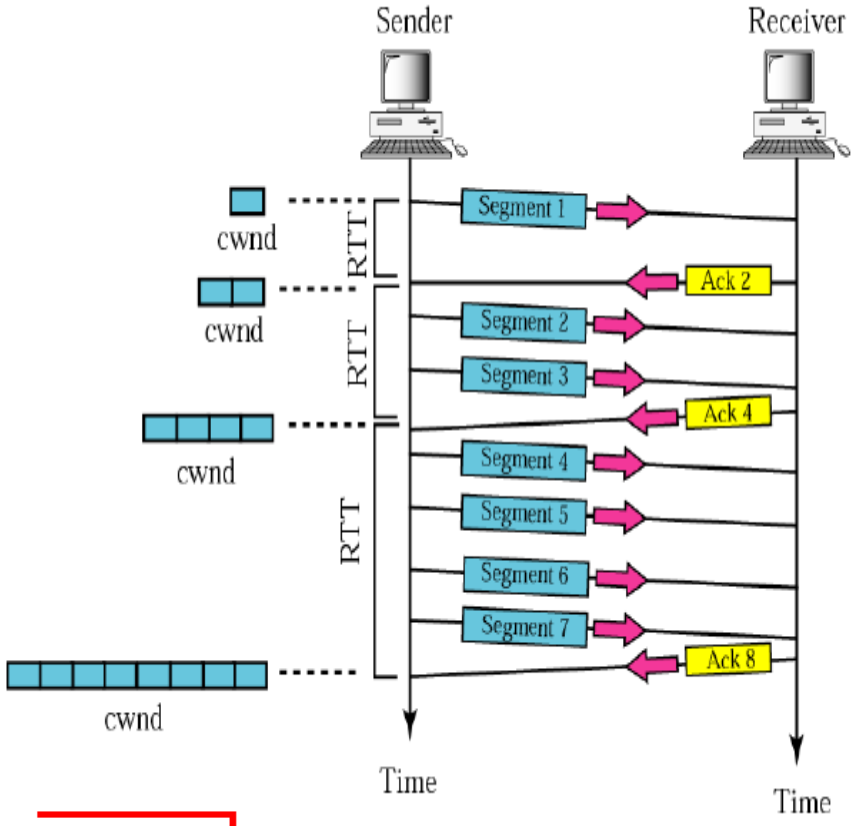
- se non ci sono perdite di pacchetti/timeout, quando interrompere la fase di Slow Start?
- introdotta una "slow start threshold" (ssthresh)
  - inizializzata ad un valore alto
- quando CWND = ssthresh, il mittente passa dalla fase di slow start a quella additiva



SLOW START: IMPLEMENTAZIONE

- creazione connessione
  - inizializza CWND a 1 MSS/RTT
- CWND raddoppiata ad ogni RTT
- implementazione semplice
  - per ogni segmento riscontrato (ACK ricevuto, considerare ACK cumulativi)

    CWND++
  - fino a quando non si verifica la perdita di un segmento o si supera la soglia

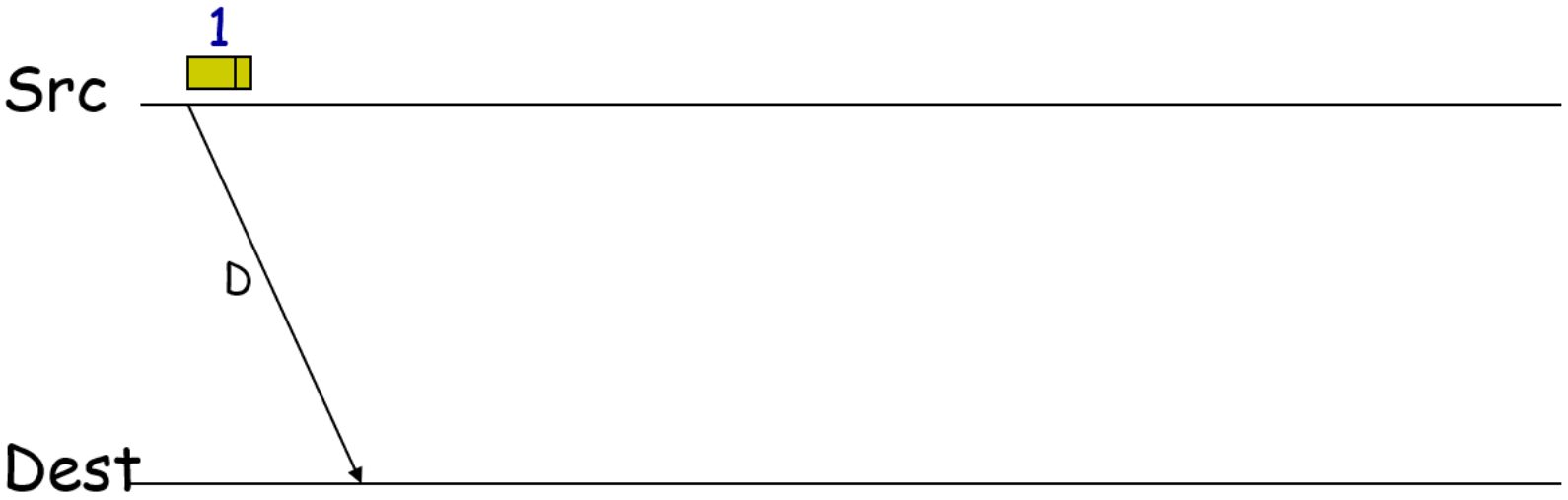


Slowstart algorithm

```
initialize: Congwin = 1 [MSS]
until (loss event OR CongWin > threshold) {
    for each segment ACKed Congwin++ }
```

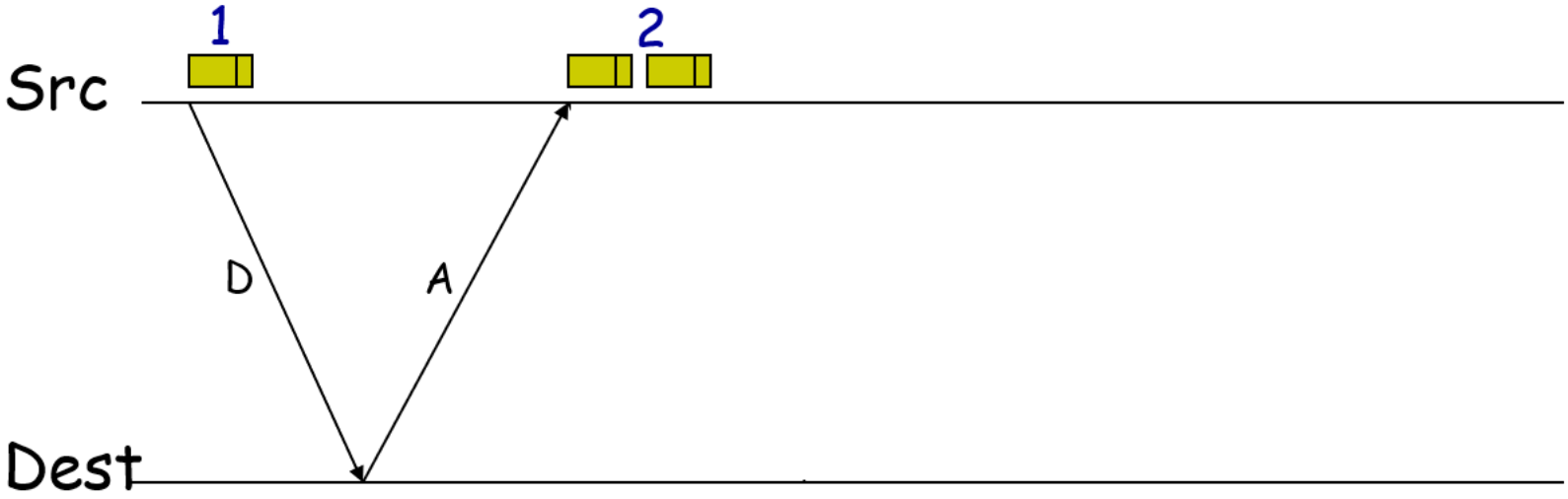
SLOW START: UN ESEMPIO

- per ogni RTT: raddoppia CWND
- implementazione più semplice: per ogni ACK, CWND += 1



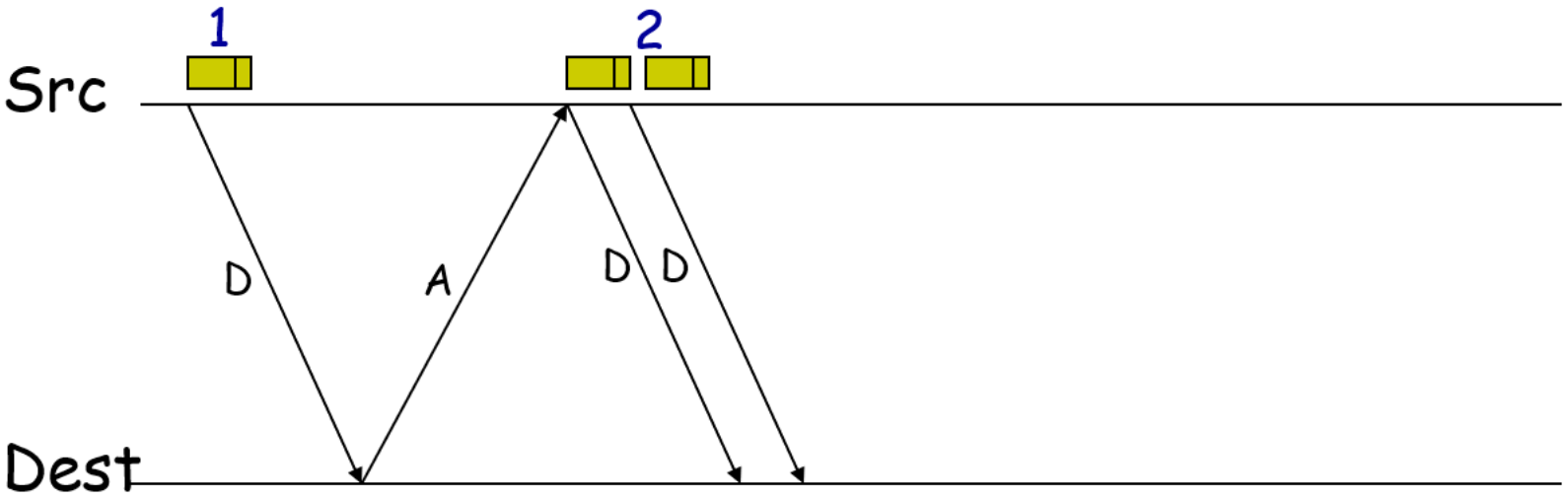
SLOW START: UN ESEMPIO

- per ogni RTT: raddoppia CWND
- implementazione più semplice: per ogni ACK, CWND += 1



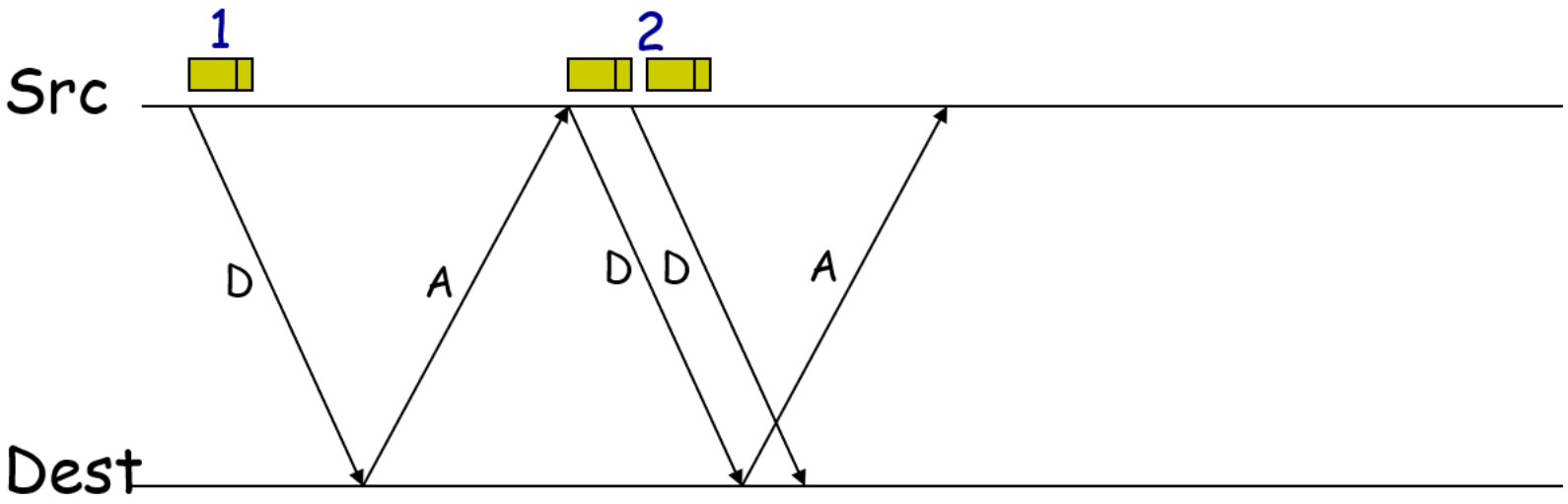
SLOW START

- per ogni RTT: raddoppia CWND
- implementazione più semplice: per ogni ACK, CWND += 1



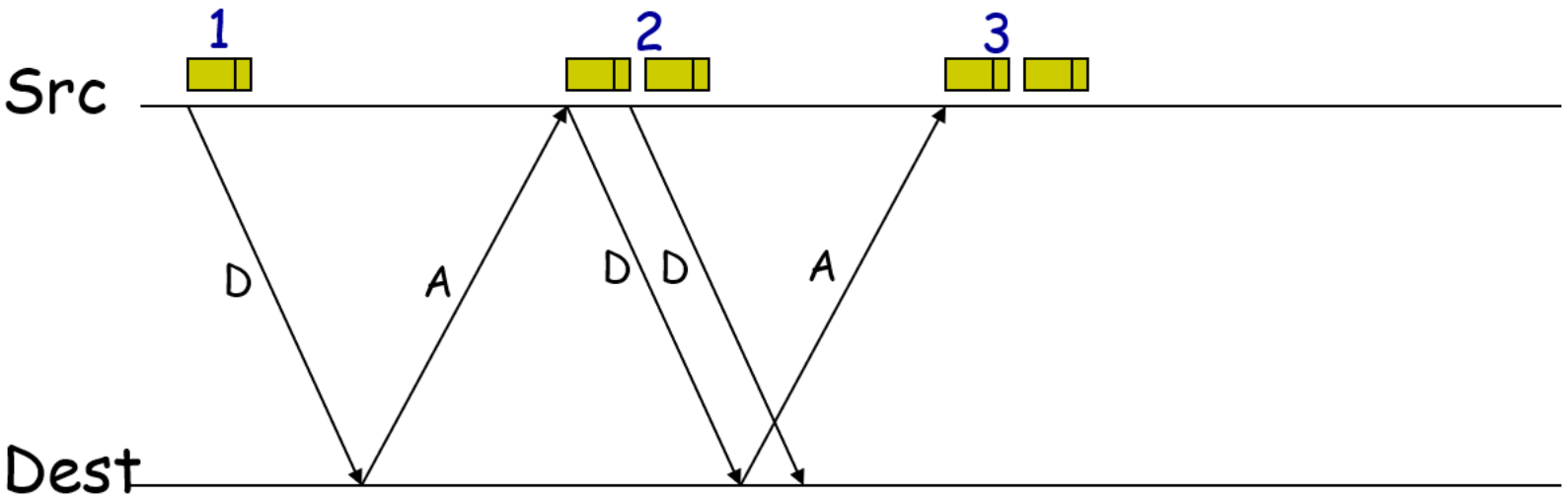
SLOW START: UN ESEMPIO

- per ogni RTT: raddoppia CWND
- implementazione più semplice: per ogni ACK, CWND += 1



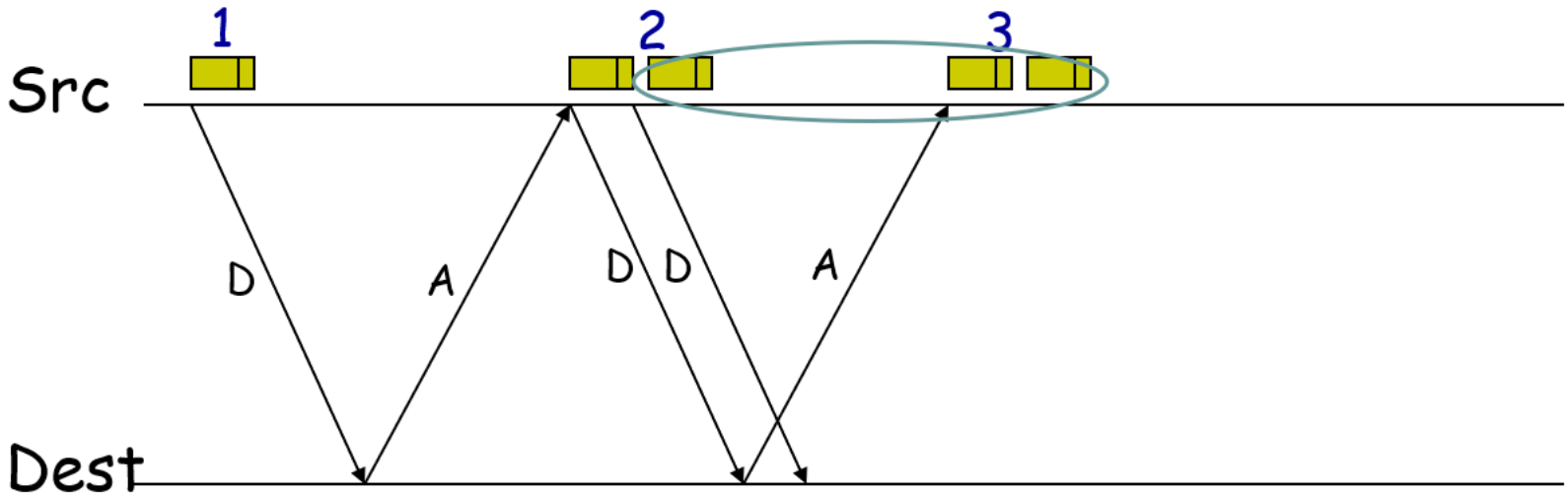
SLOW START: UN ESEMPIO

- per ogni RTT: raddoppia CWND
- implementazione più semplice: per ogni ACK, CWND += 1



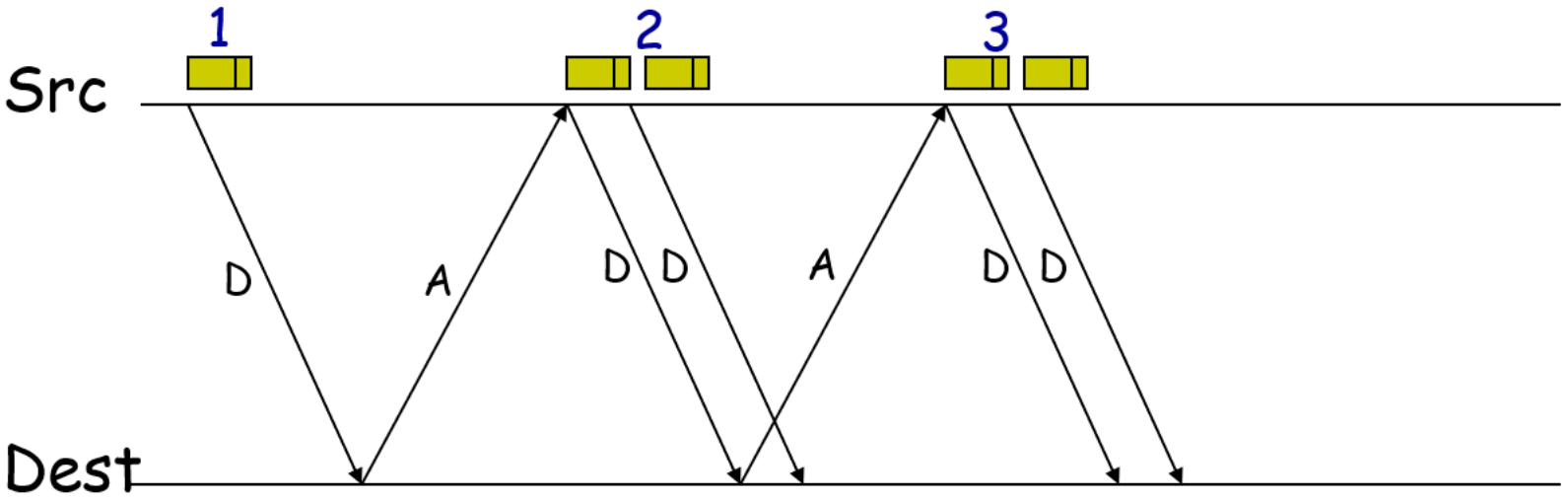
SLOW START: UN ESEMPIO

- per ogni RTT: raddoppia CWND
- implementazione più semplice: per ogni ACK, CWND += 1



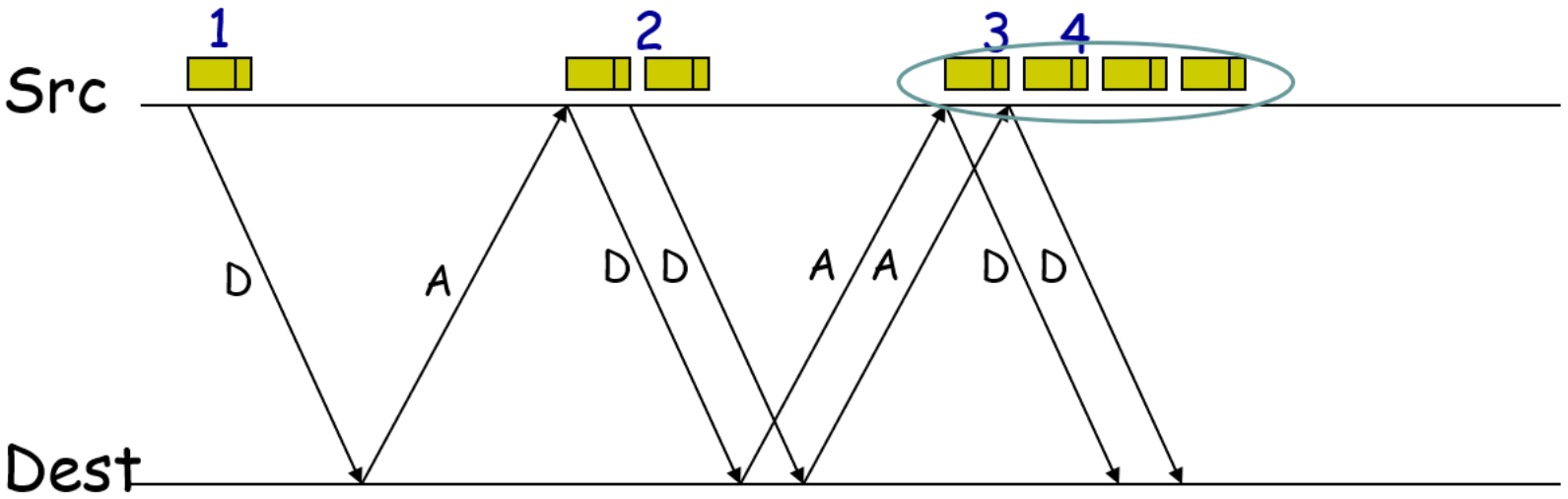
SLOW START: UN ESEMPIO

- per ogni RTT: raddoppia CWND
- implementazione più semplice: per ogni ACK, CWND += 1



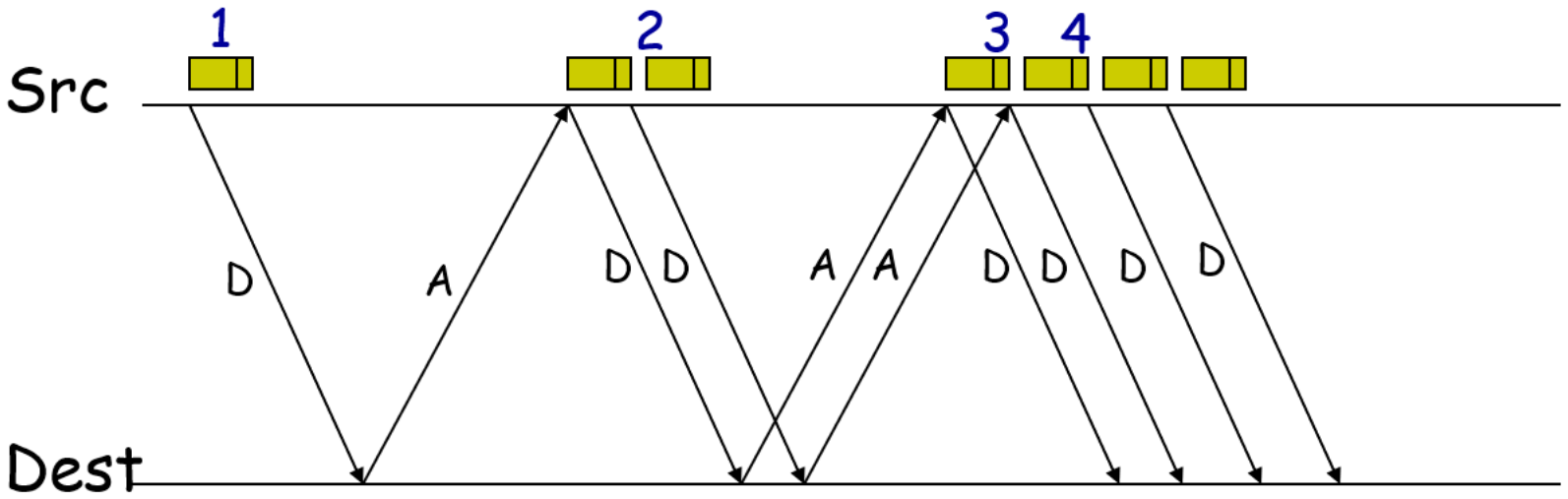
SLOW START: UN ESEMPIO

- per ogni RTT: raddoppia CWND
- implementazione più semplice: per ogni ACK, CWND += 1



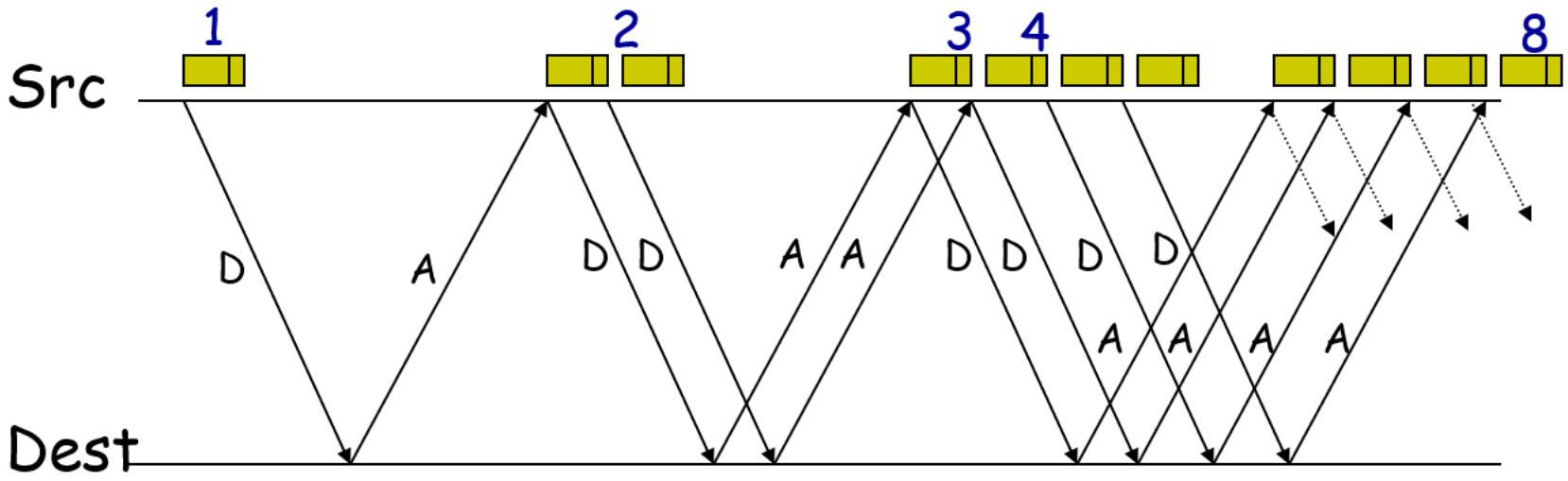
SLOW START: UN ESEMPIO

- per ogni RTT: raddoppia CWND
- implementazione più semplice: per ogni ACK, CWND += 1



SLOW START: UN ESEMPIO

- per ogni RTT: raddoppia CWND
- implementazione più semplice: per ogni ACK, CWND += 1



AIMD: ADDITIVE INCREASE

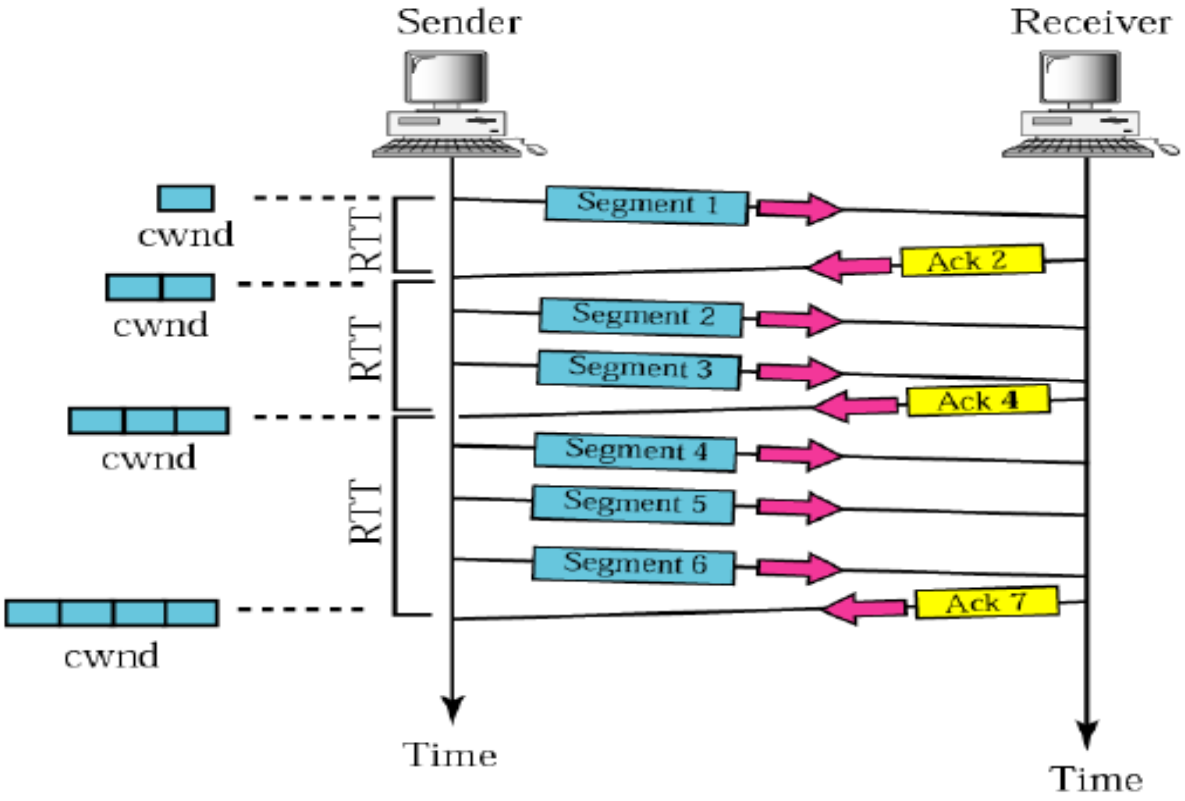
- Slow Start fornisce una stima della banda disponibile
- quando la dimensione della finestra supera la soglia ssthresh
  - entra in esecuzione la fase di additive increase
  - rallenta la fase di "Slow Start"
  - incremento lineare della finestra
- additive increase:
  - CWND è incrementato di 1 per ogni RTT.
  - implementazione semplice: incrementare 1/CWND per ogni ACK

If CWND > ssthresh **then**
each time a segment is acknowledged
CWND += 1/CWND

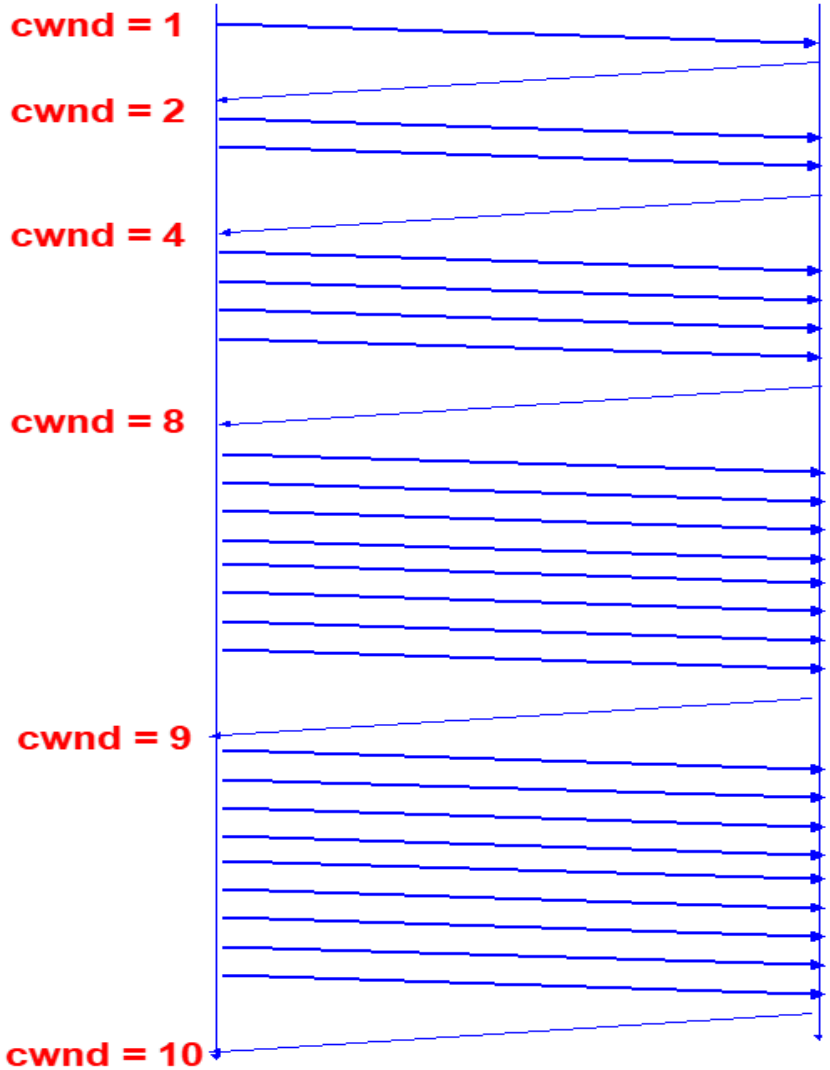
ADDITIVE INCREASE: IMPLEMENTAZIONE

Fase di Additive Increase o Congestion Avoidance

- incrementa la Window di solo un MSS per ogni RTT senza perdite
  - per ogni RTT, CWND = CWND + 1
  - implementazione semplice:
    - per ogni ACK, CWND = CWND+ 1/CWND



SLOW START E CONGESTION AVOIDANCE

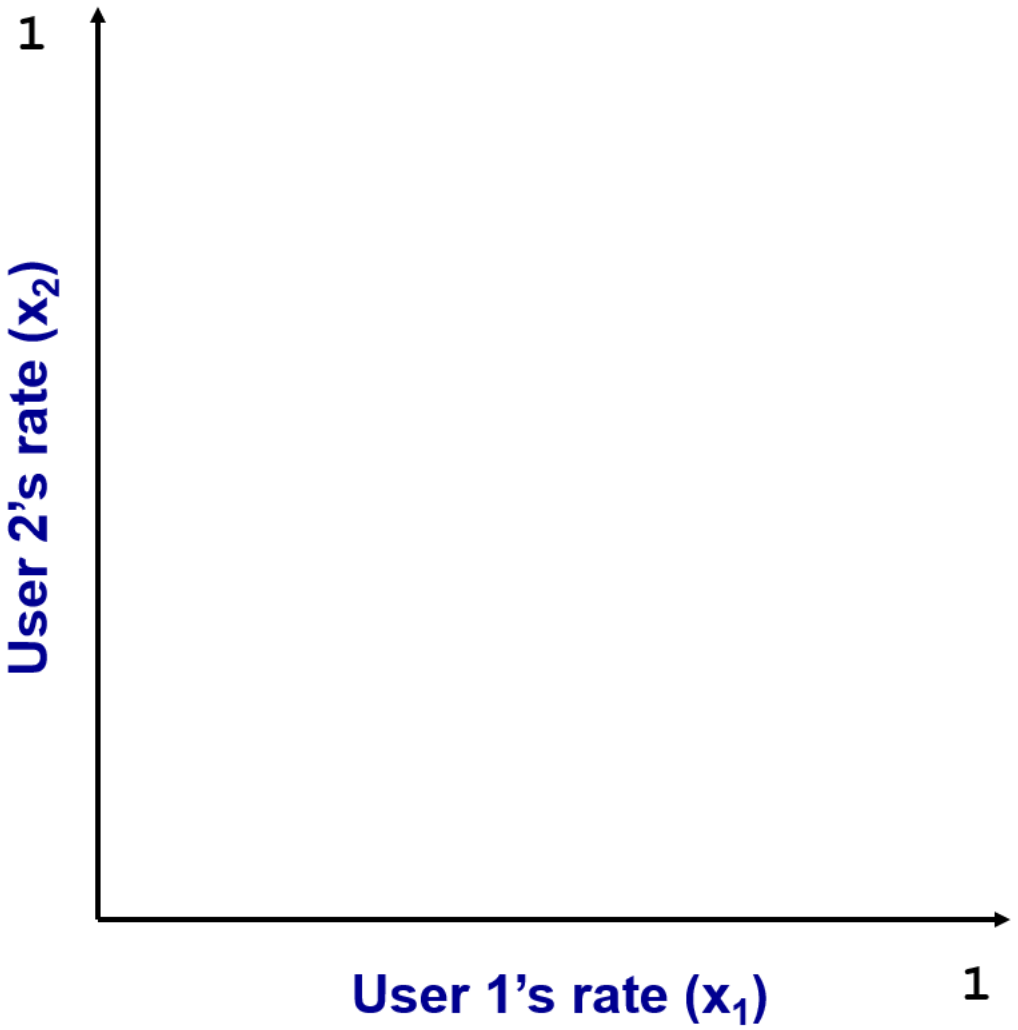


AIMD: PERCHE' QUESTA SCELTA?

- ad ogni RTT sarebbe possibile effettuare
  - un incremento o decremento moltiplicativo CWND a*CWND
  - un incremento o decremento additivo CWND CWND + b
- possibili le seguenti combinazioni:
  - AIMD: incremento prudente, decremento drastico
  - AIAD: incremento prudente, decremento prudente
  - MIMD: incremento drastico, decremento drastico
  - MIAD: incremento drastico, decremento prudente
- una motivazione intuitiva, per AIMD
  - prudenza nell'incremento, e “far marcia indietro” al primo segno negativo
  - è molto facile portare la rete ad un sovraccarico, quindi occorre esseri prudenti
  - ...mentre non è così negativo se in certi intervalli di tempo la rete viene sottoutilizzata

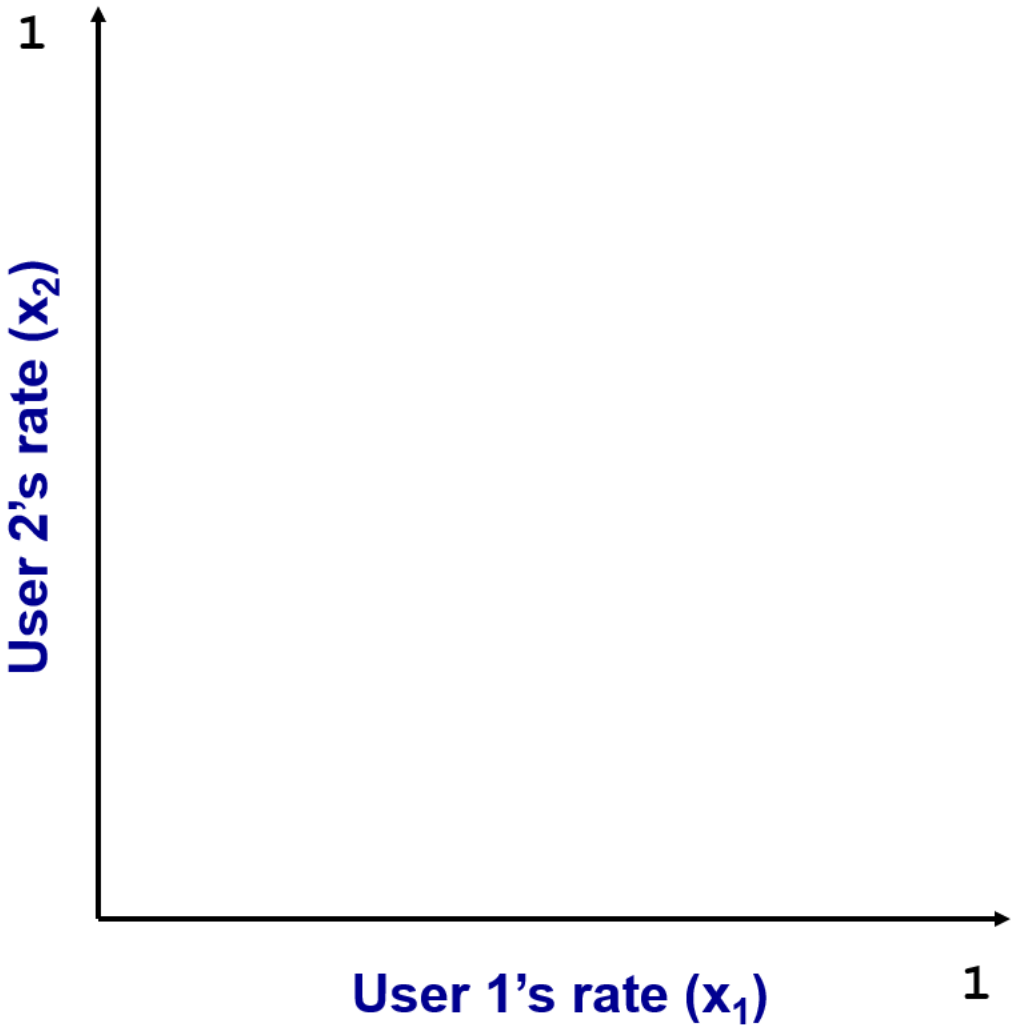
MODELLARE IL CONTROLLO DI CONGESIONE

- Two users
  - rates x_1 and x_2



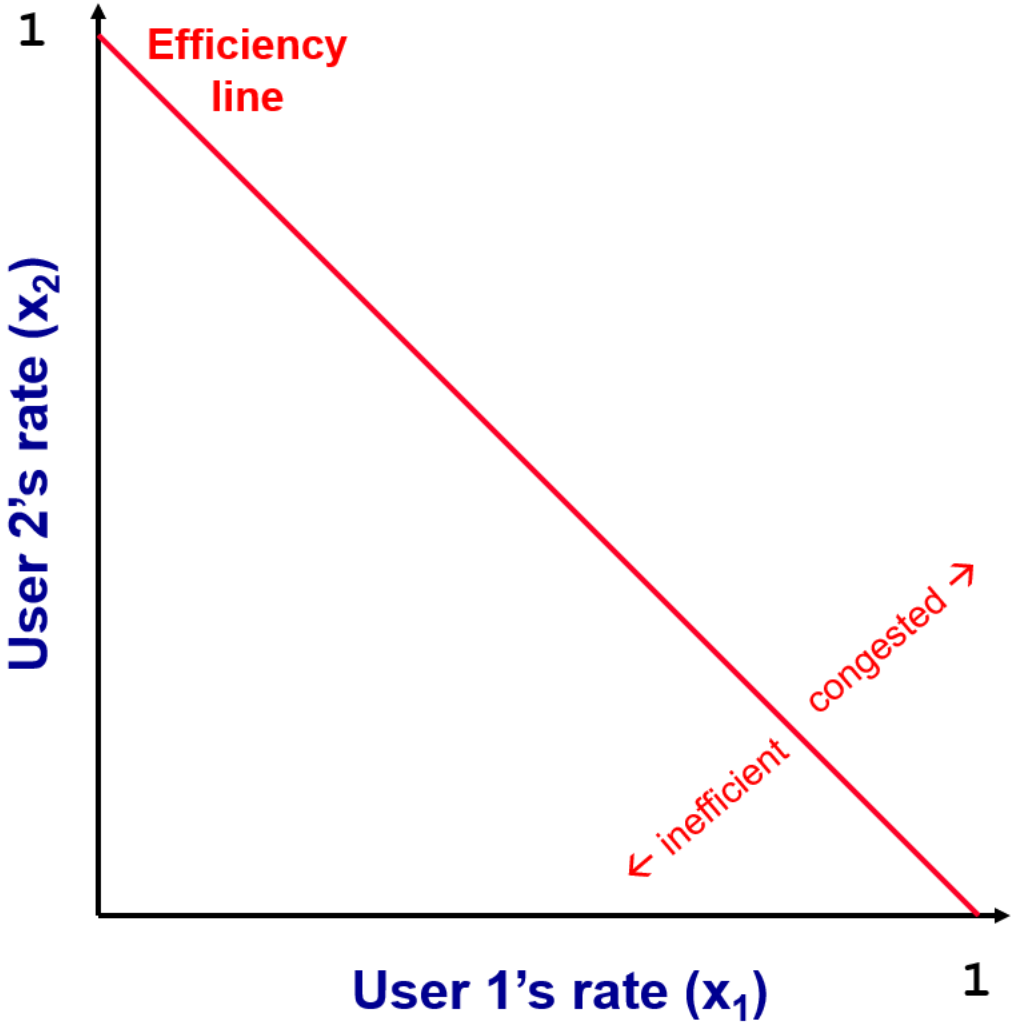
MODELLARE IL CONTROLLO DI CONGESIONE

- Two users
  - rates x_1 and x_2
- Congestion when $x_1+x_2 > 1$



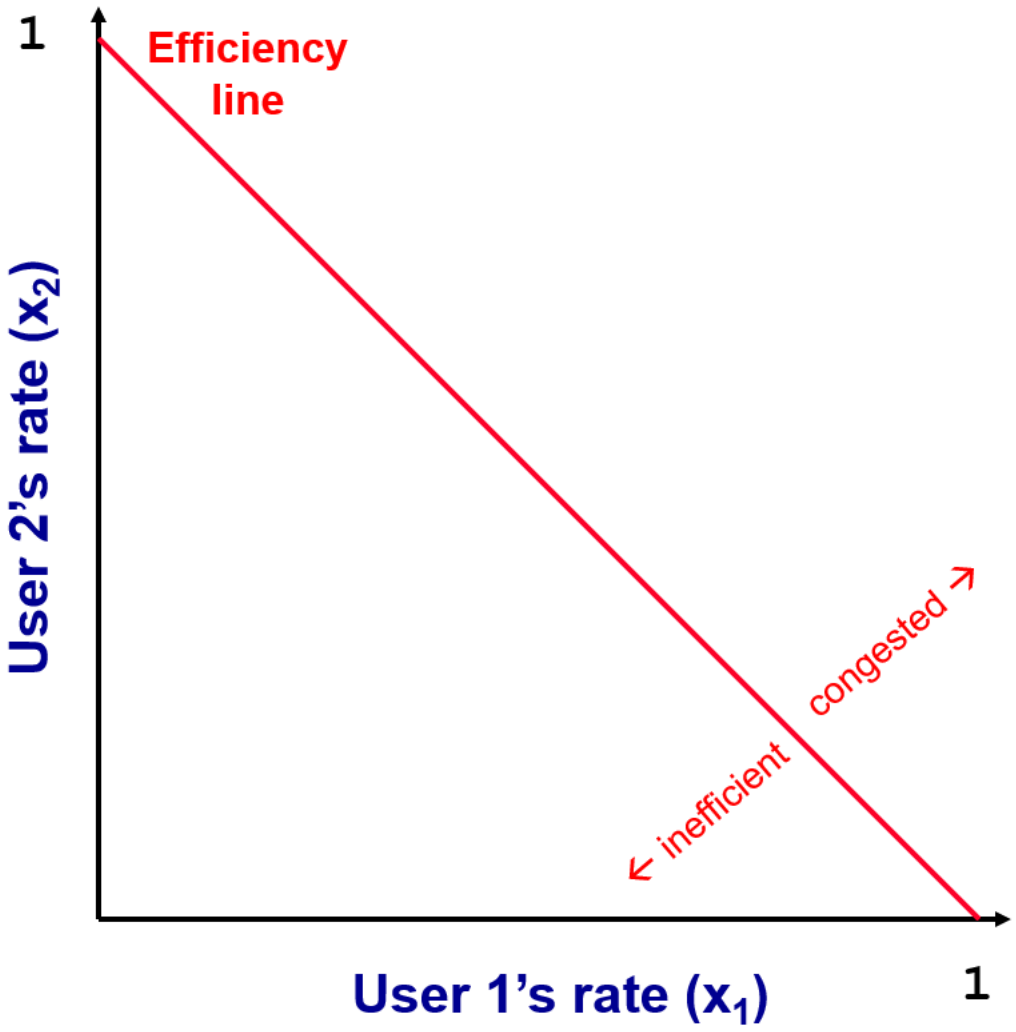
MODELLARE IL CONTROLLO DI CONGESIONE

- Two users
  - rates x_1 and x_2
- Congestion when $x_1+x_2 > 1$
- Unused capacity when $x_1+x_2 < 1$



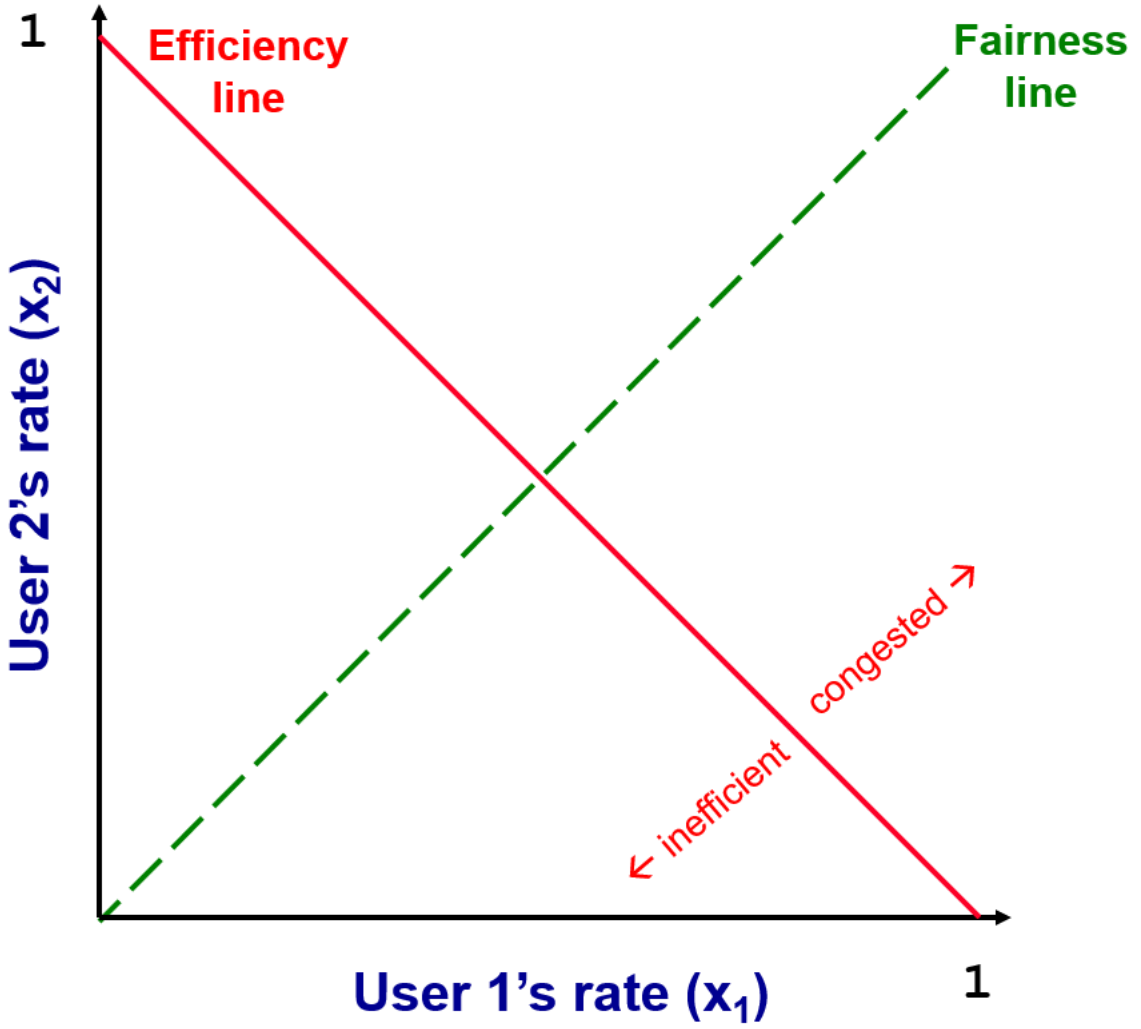
MODELLARE IL CONTROLLO DI CONGESIONE

- Two users
  - rates x_1 and x_2
- Congestion when $x_1+x_2 > 1$
- Unused capacity when $x_1+x_2 < 1$
- Fair when $x_1 = x_2$

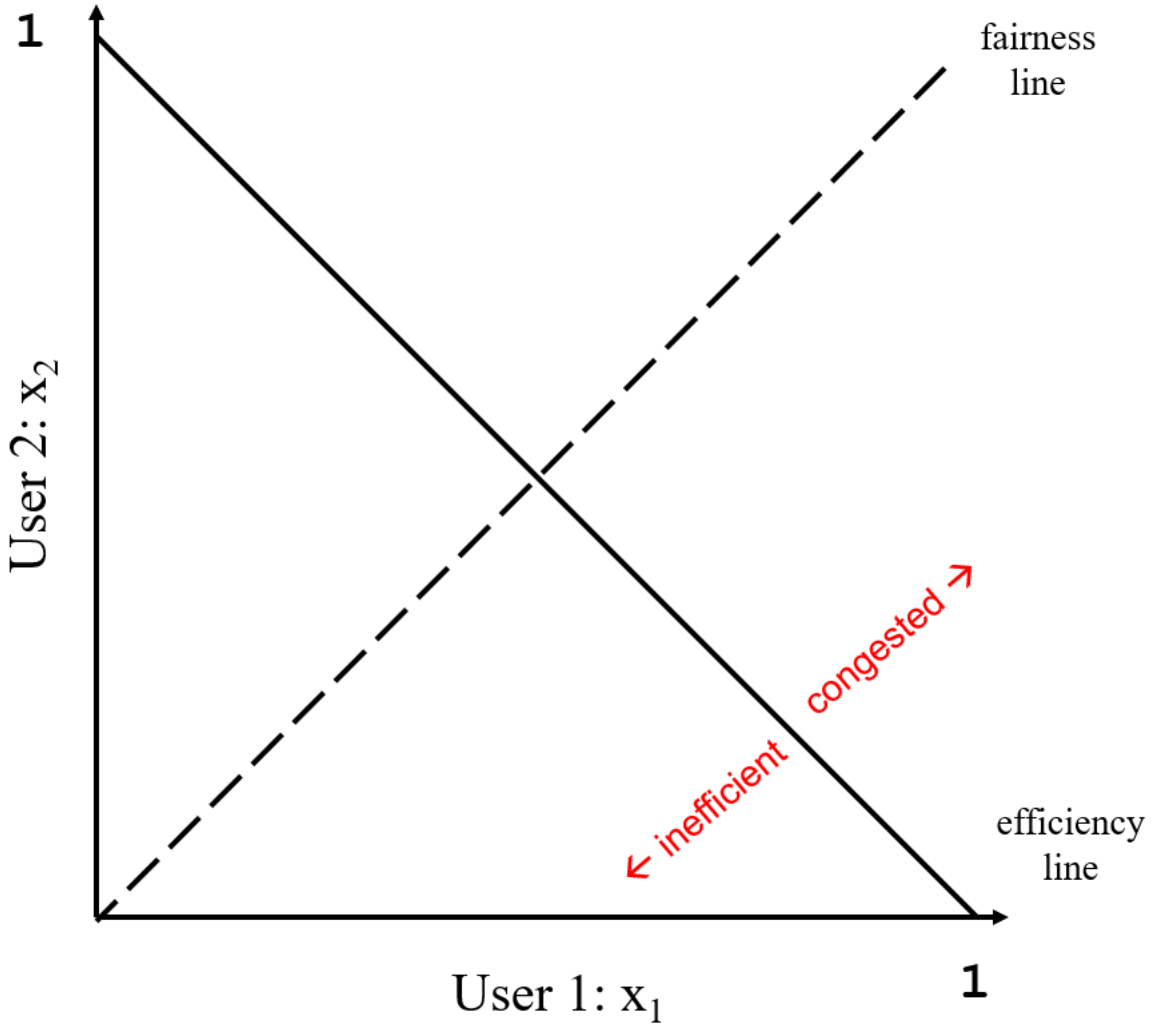


MODELLARE IL CONTROLLO DI CONGESIONE

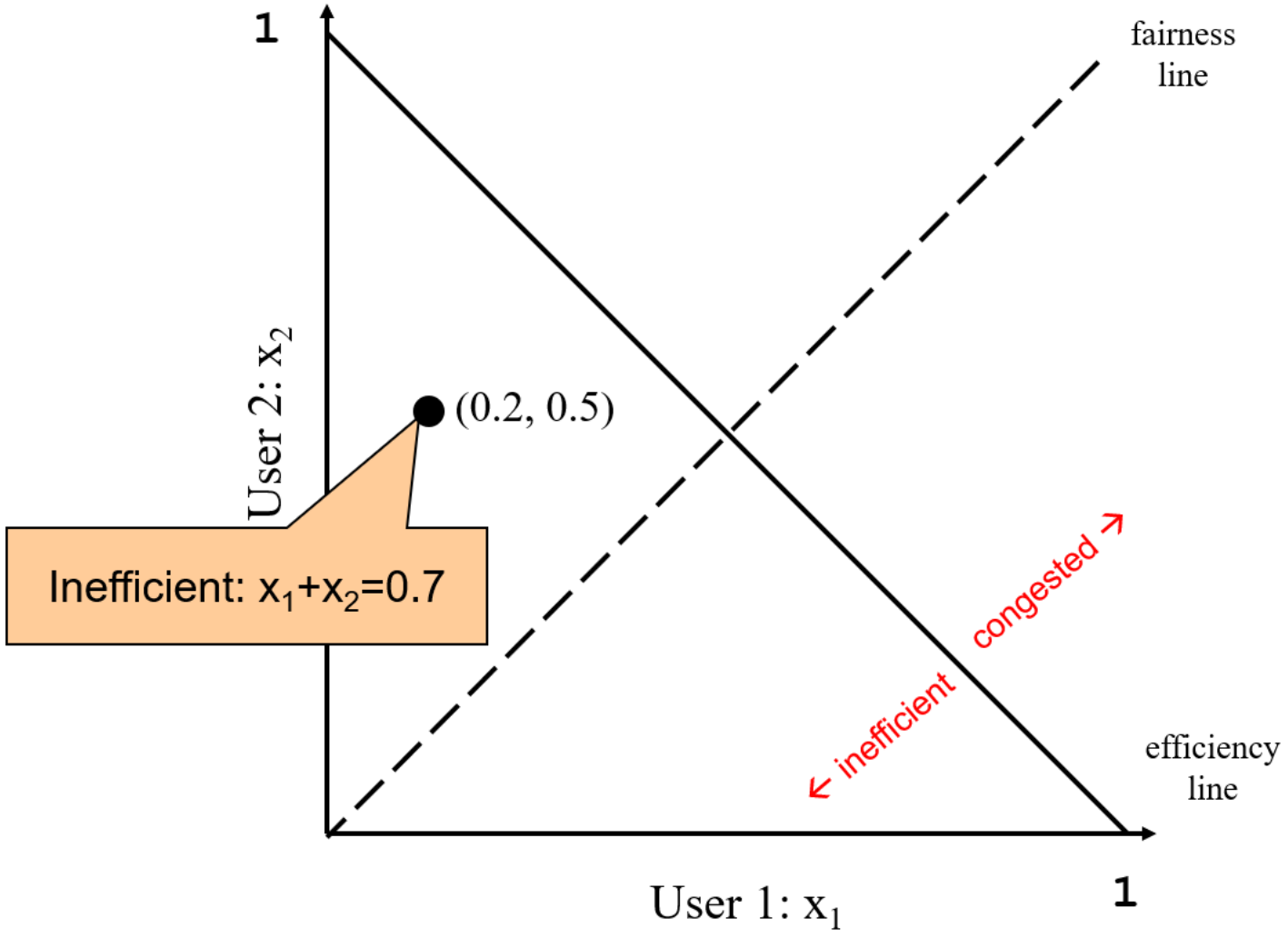
- Two users
  - rates x_1 and x_2
- Congestion when $x_1+x_2 > 1$
- Unused capacity when $x_1+x_2 < 1$
- Fair when $x_1 = x_2$



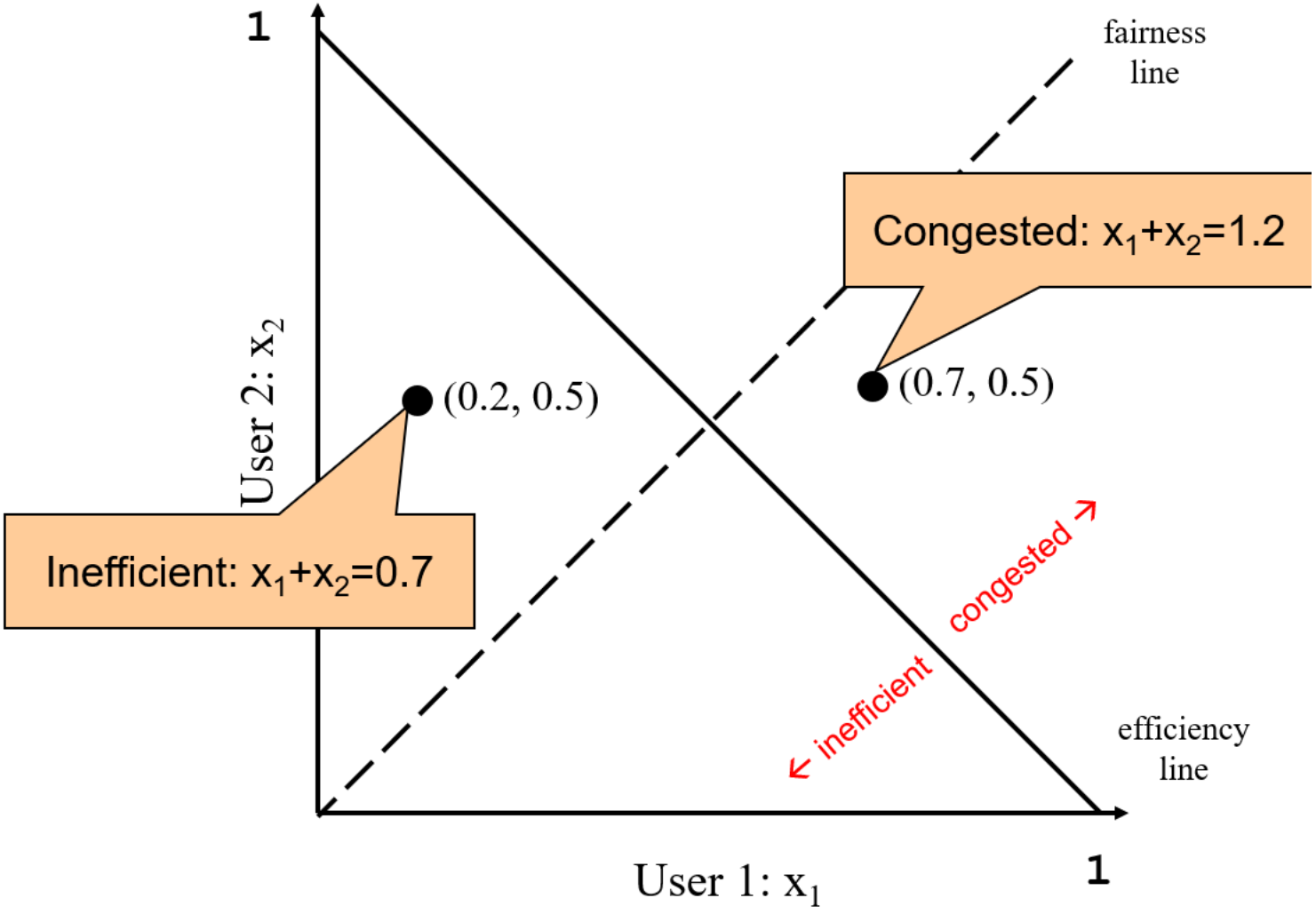
MODELLARE IL CONTROLLO DI CONGESIONE



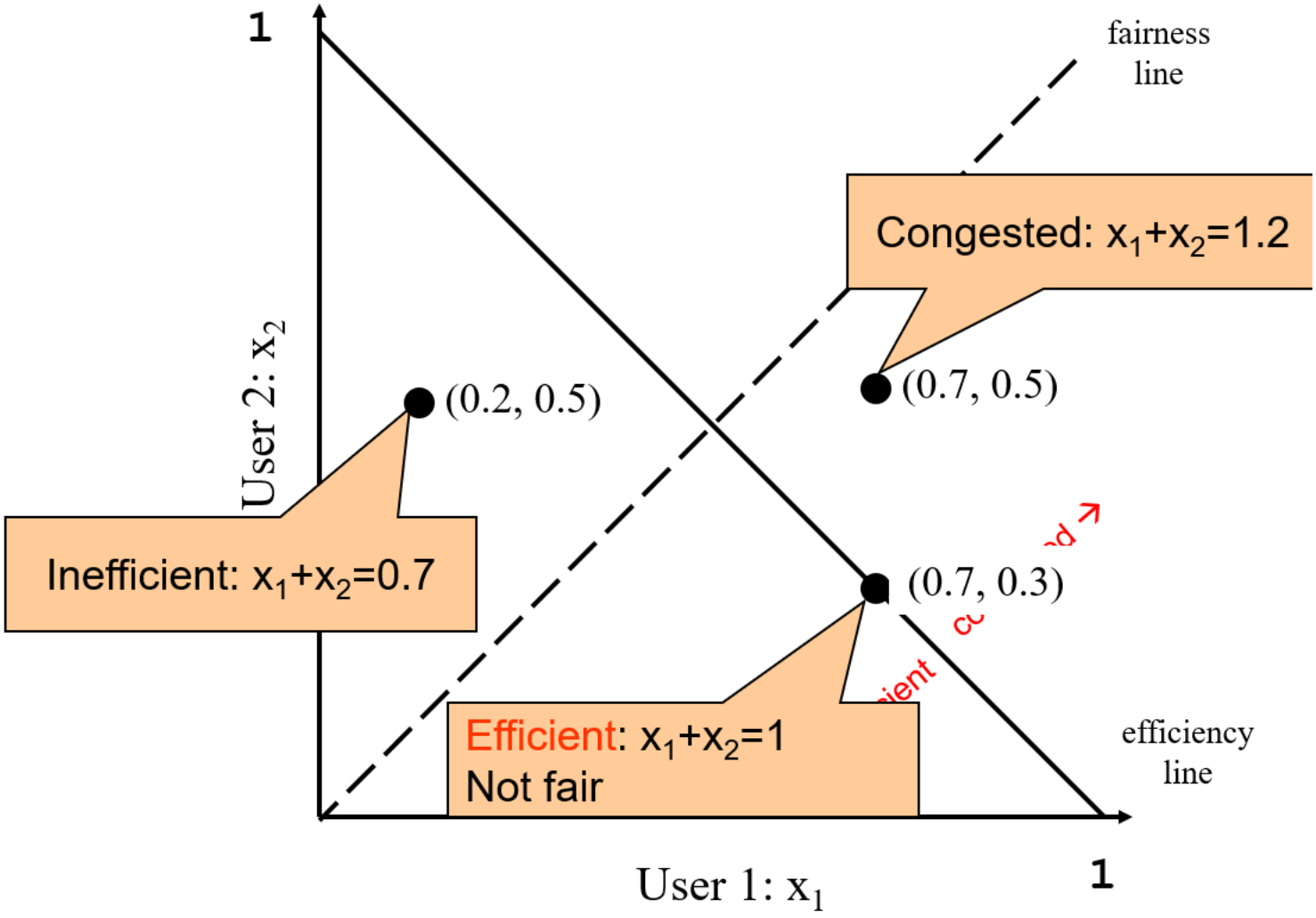
MODELLARE IL CONTROLLO DI CONGESIONE



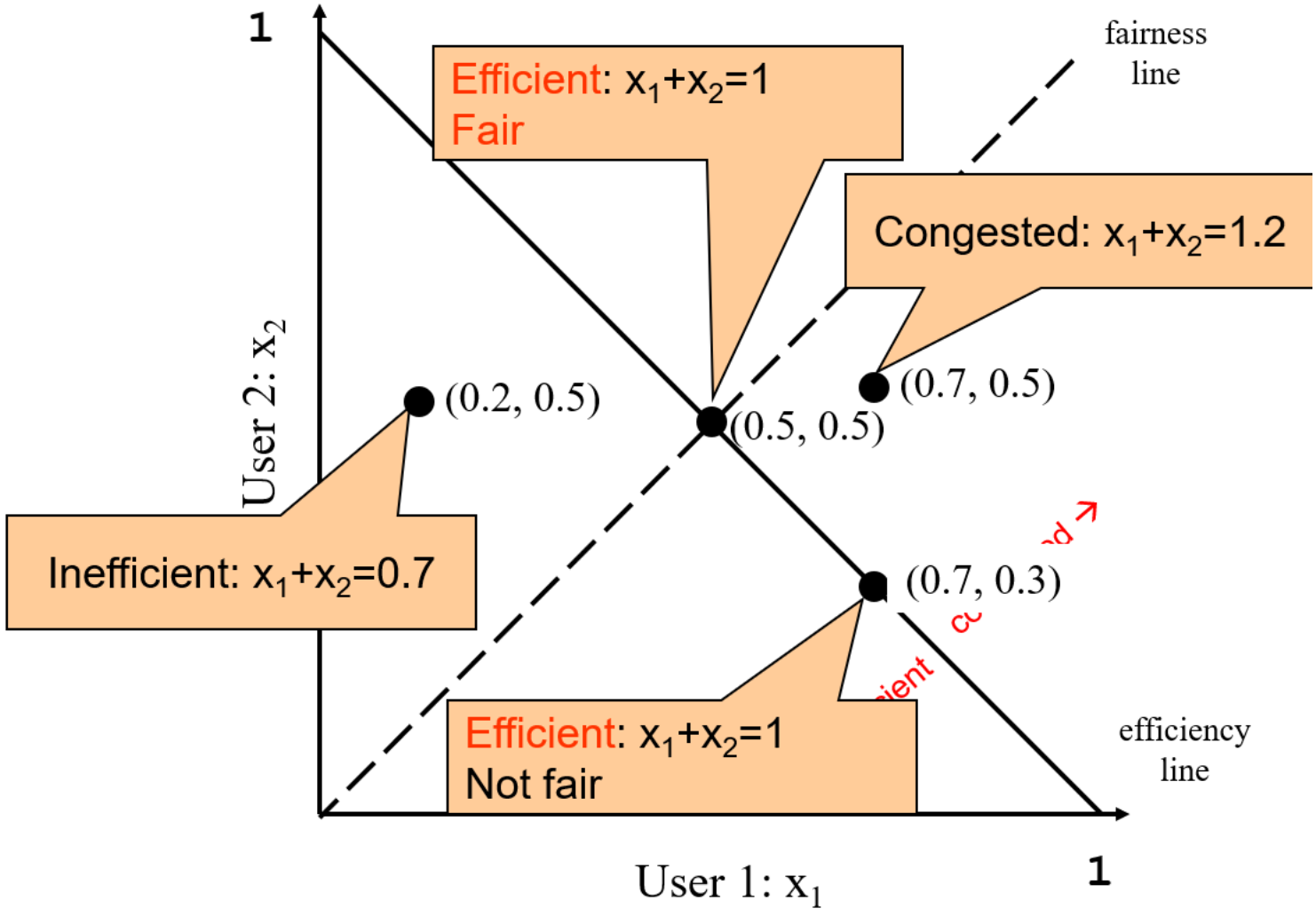
MODELLARE IL CONTROLLO DI CONGESIONE



MODELLARE IL CONTROLLO DI CONGESIONE

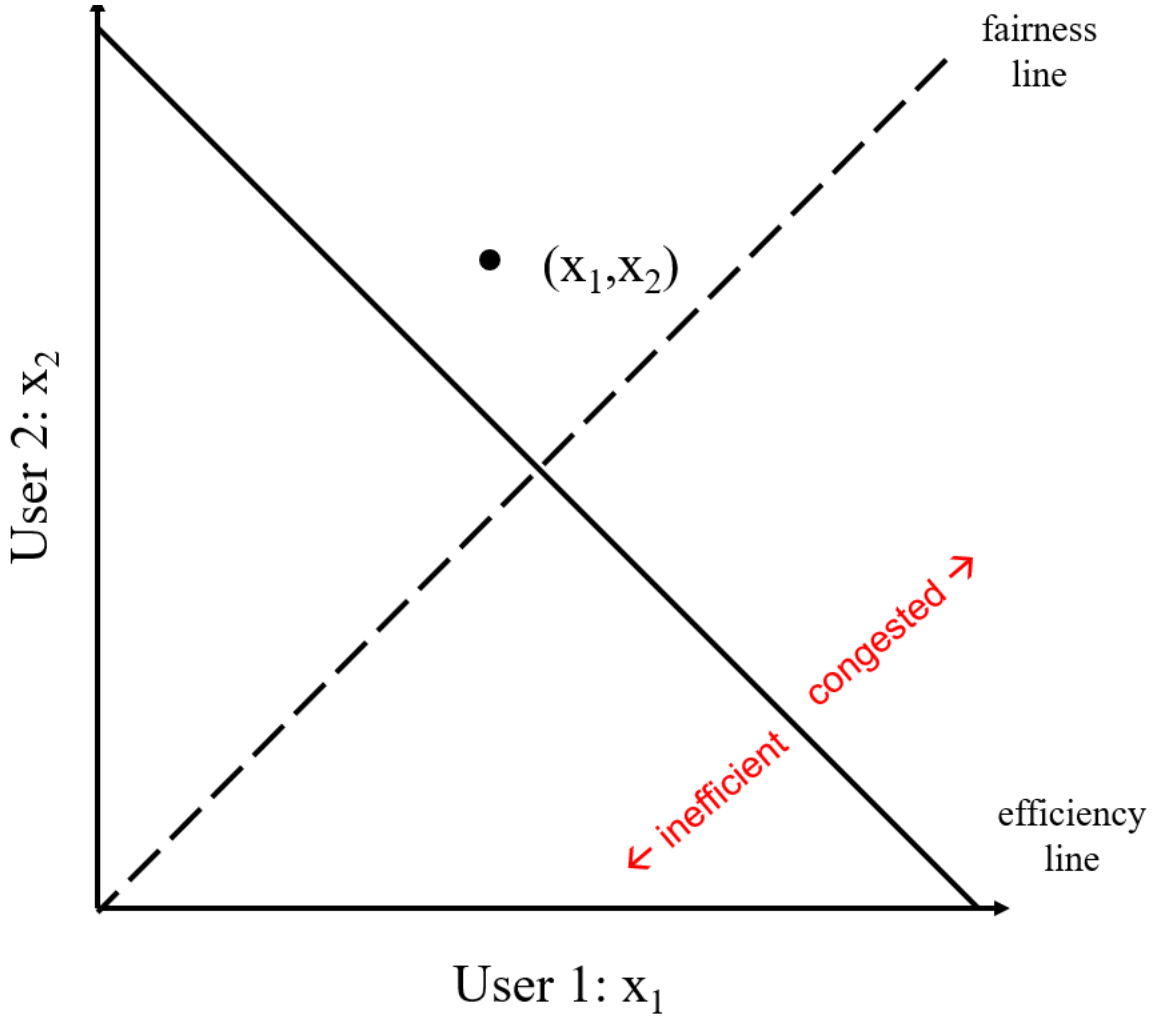


MODELLARE IL CONTROLLO DI CONGESIONE



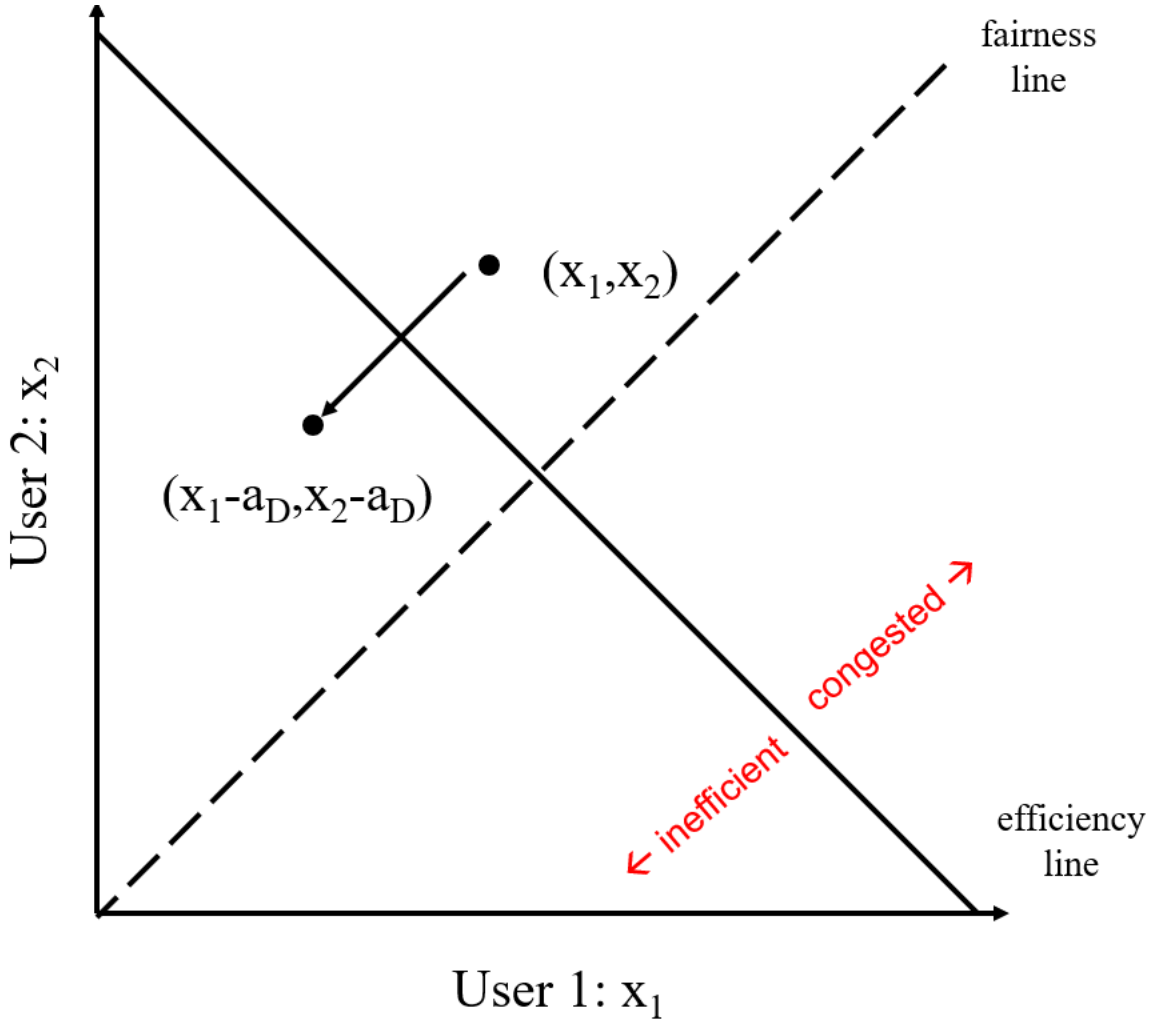
ADDITIVE INCREASE ADDITIVE DECREASE

- Increase: $x + a_I$
- Decrease: $x - a_D$



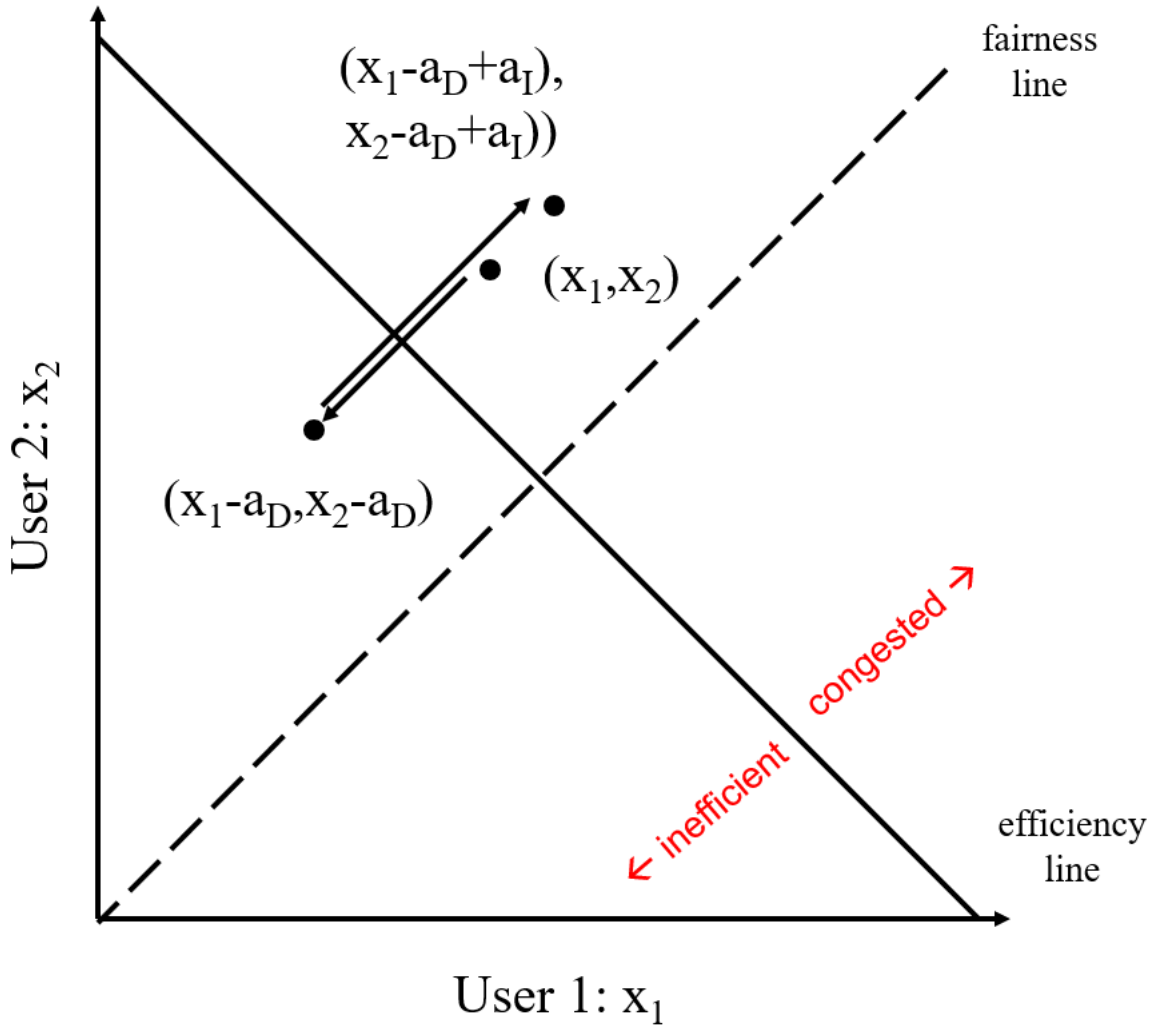
ADDITIVE INCREASE ADDITIVE DECREASE

- Increase: $x + a_I$
- Decrease: $x - a_D$



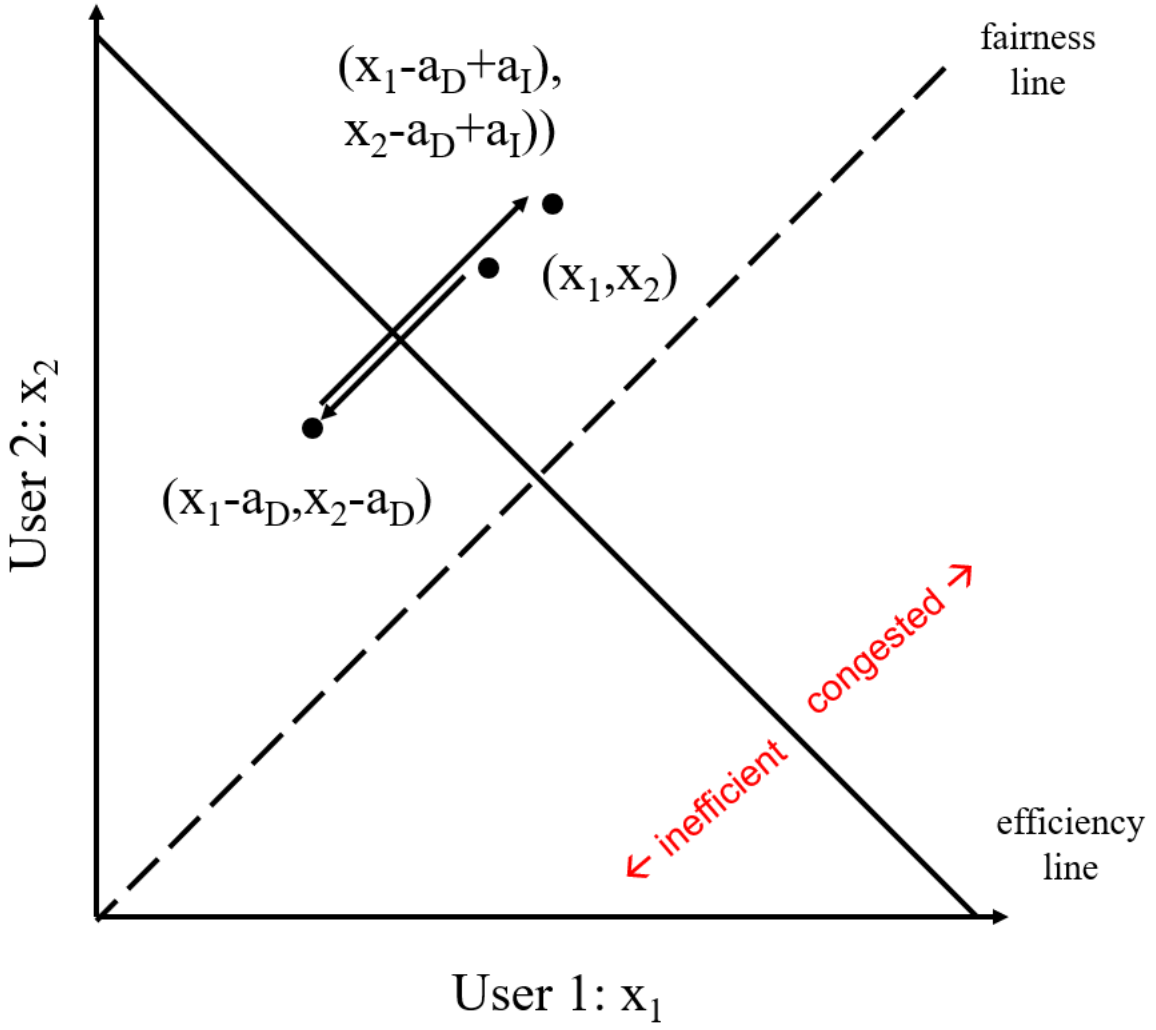
ADDITIVE INCREASE ADDITIVE DECREASE

- Increase: $x + a_I$
- Decrease: $x - a_D$



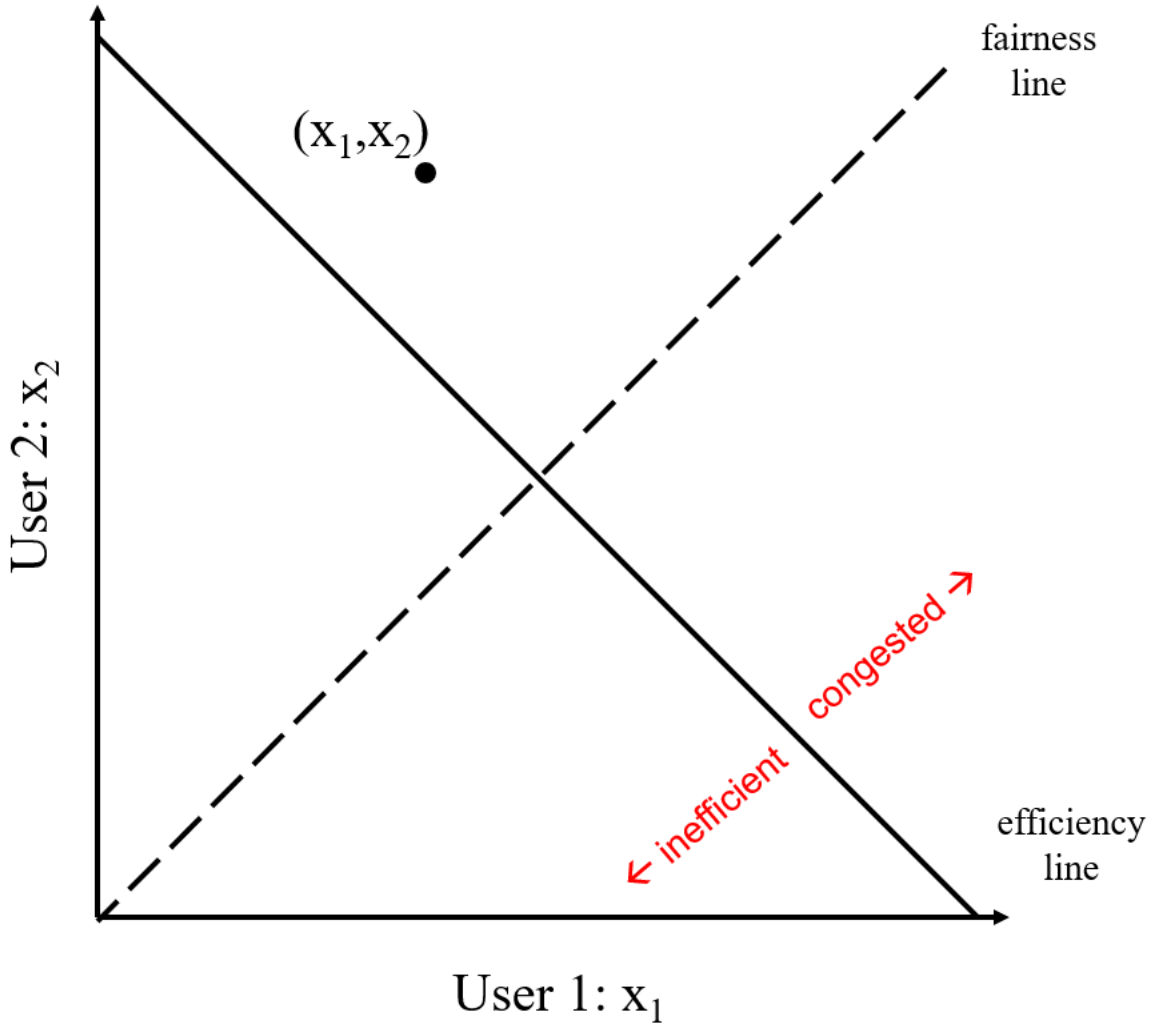
ADDITIVE INCREASE ADDITIVE DECREASE

- Increase: $x + a_I$
- Decrease: $x - a_D$
- Does not converge to fairness



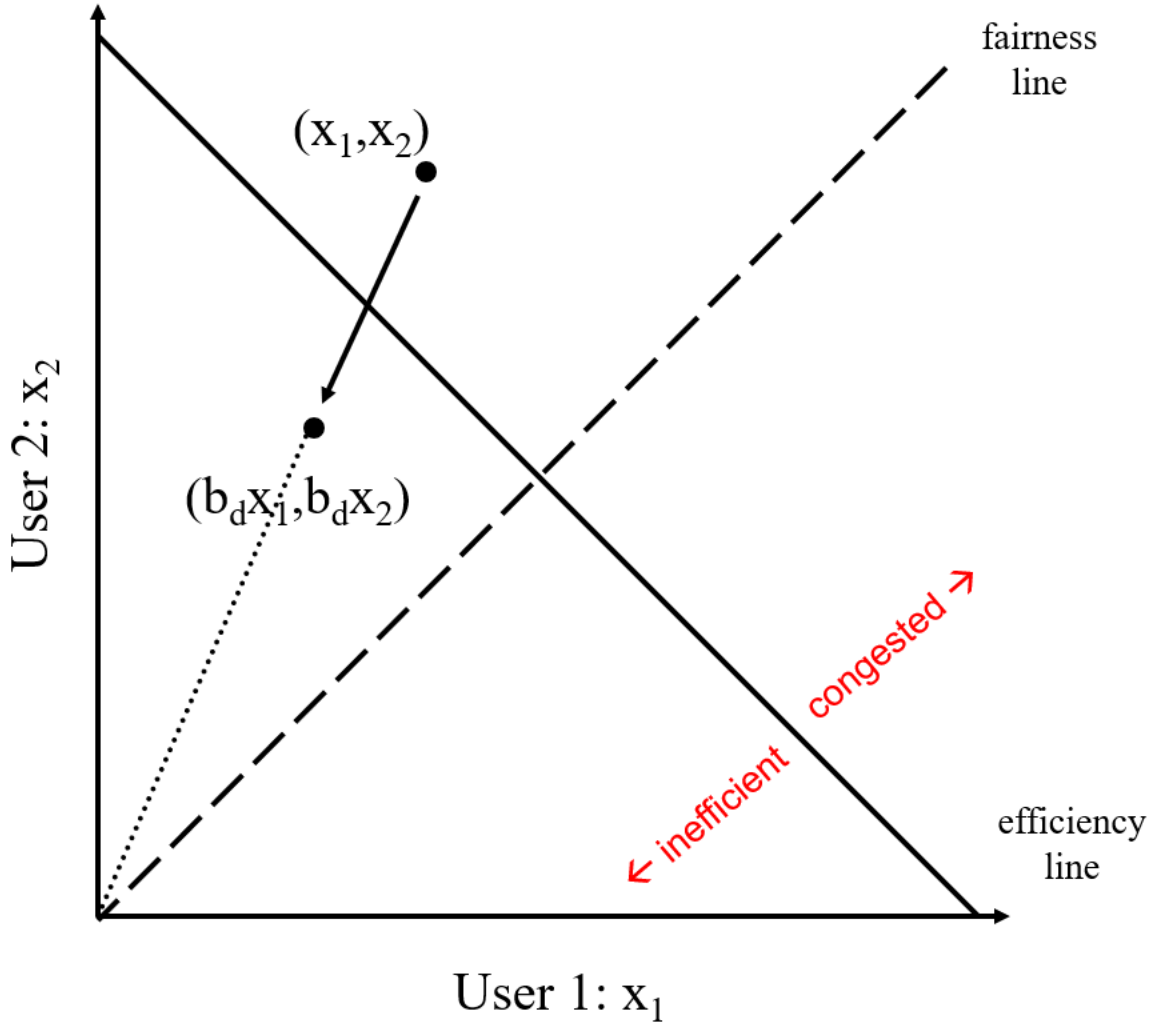
MULTIPLICATIVE INCREASE MULTIPLICATIVE DECREASE

- Increase: $x*b_I$
- Decrease: $x*b_D$



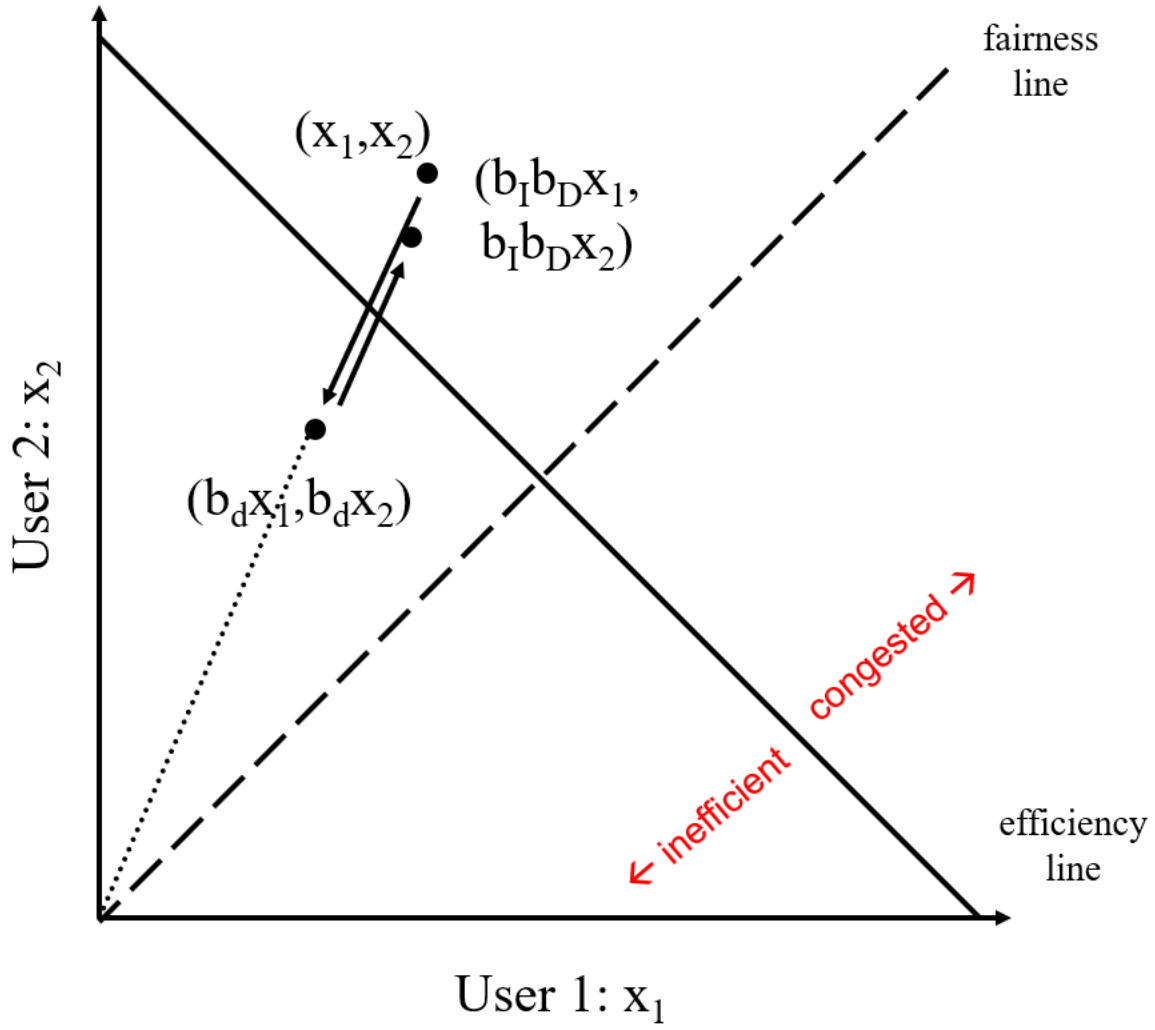
MULTIPLICATIVE INCREASE MULTIPLICATIVE DECREASE

- Increase: $x*b_I$
- Decrease: $x*b_D$



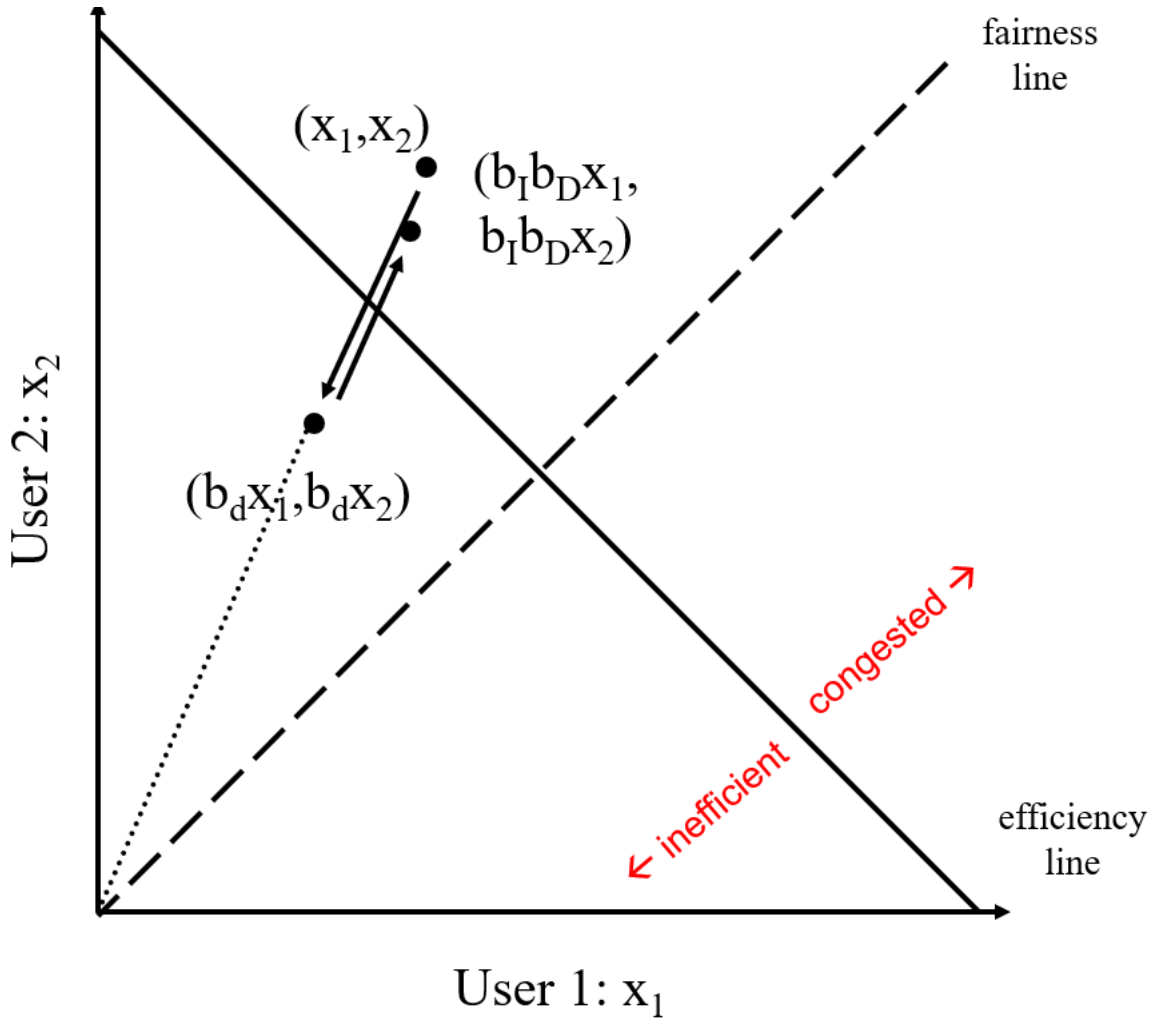
MULTIPLICATIVE INCREASE MULTIPLICATIVE DECREASE

- Increase: $x*b_I$
- Decrease: $x*b_D$



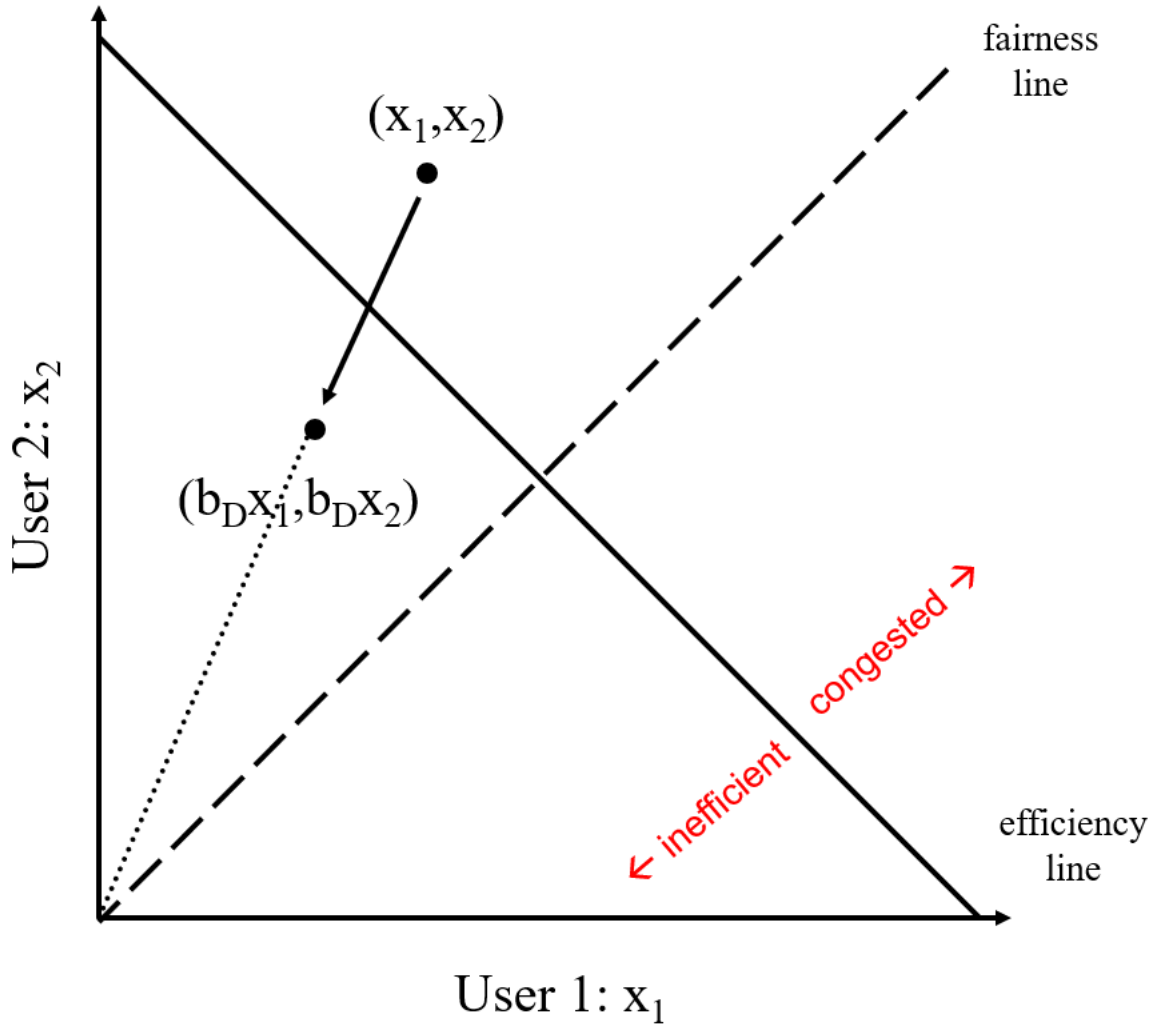
MULTIPLICATIVE INCREASE MULTIPLICATIVE DECREASE

- Increase: $x*b_I$
- Decrease: $x*b_D$
- Does not converge to fairness



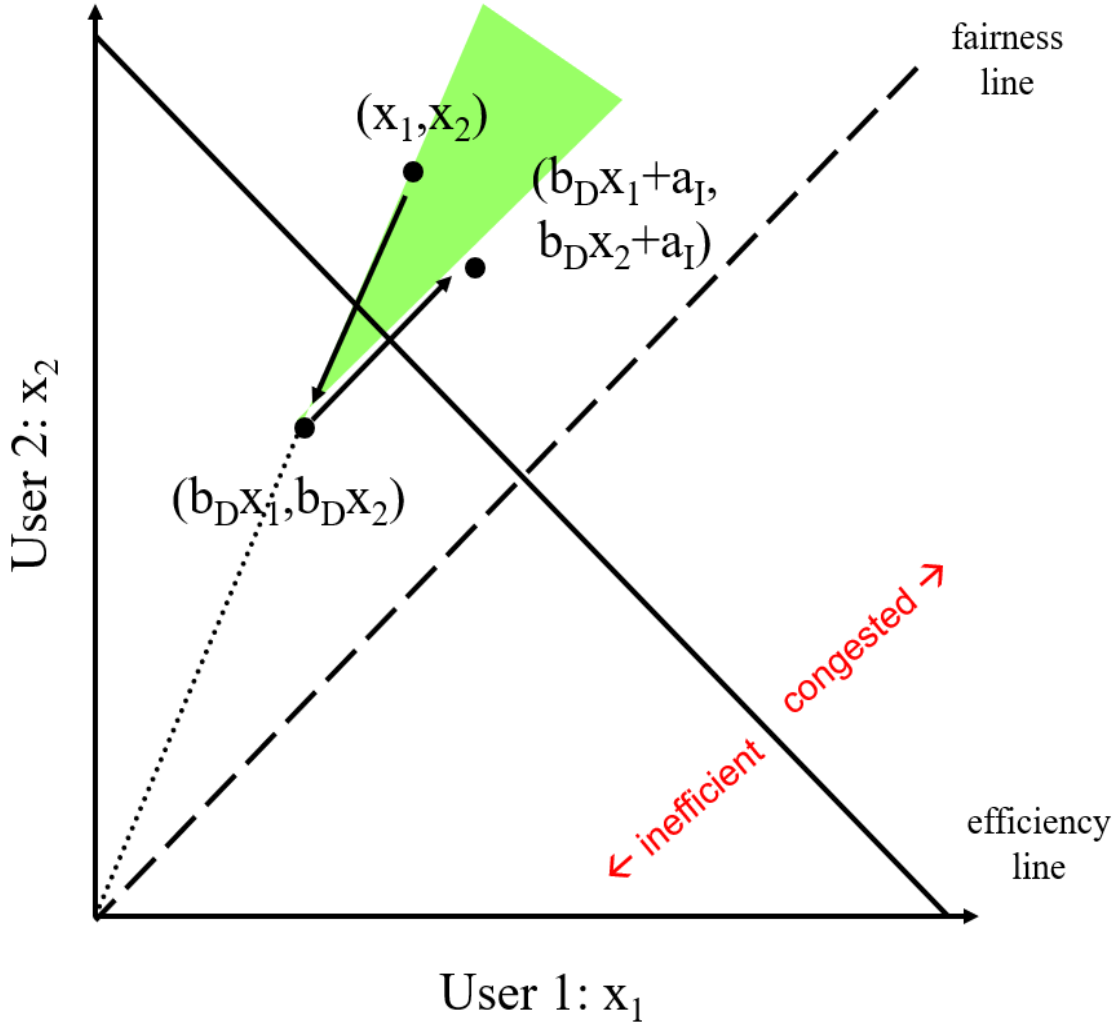
ADDITIVE INCREASE MULTIPLICATIVE DECREASE

- Increase: $x+a_I$
- Decrease: $x*b_D$



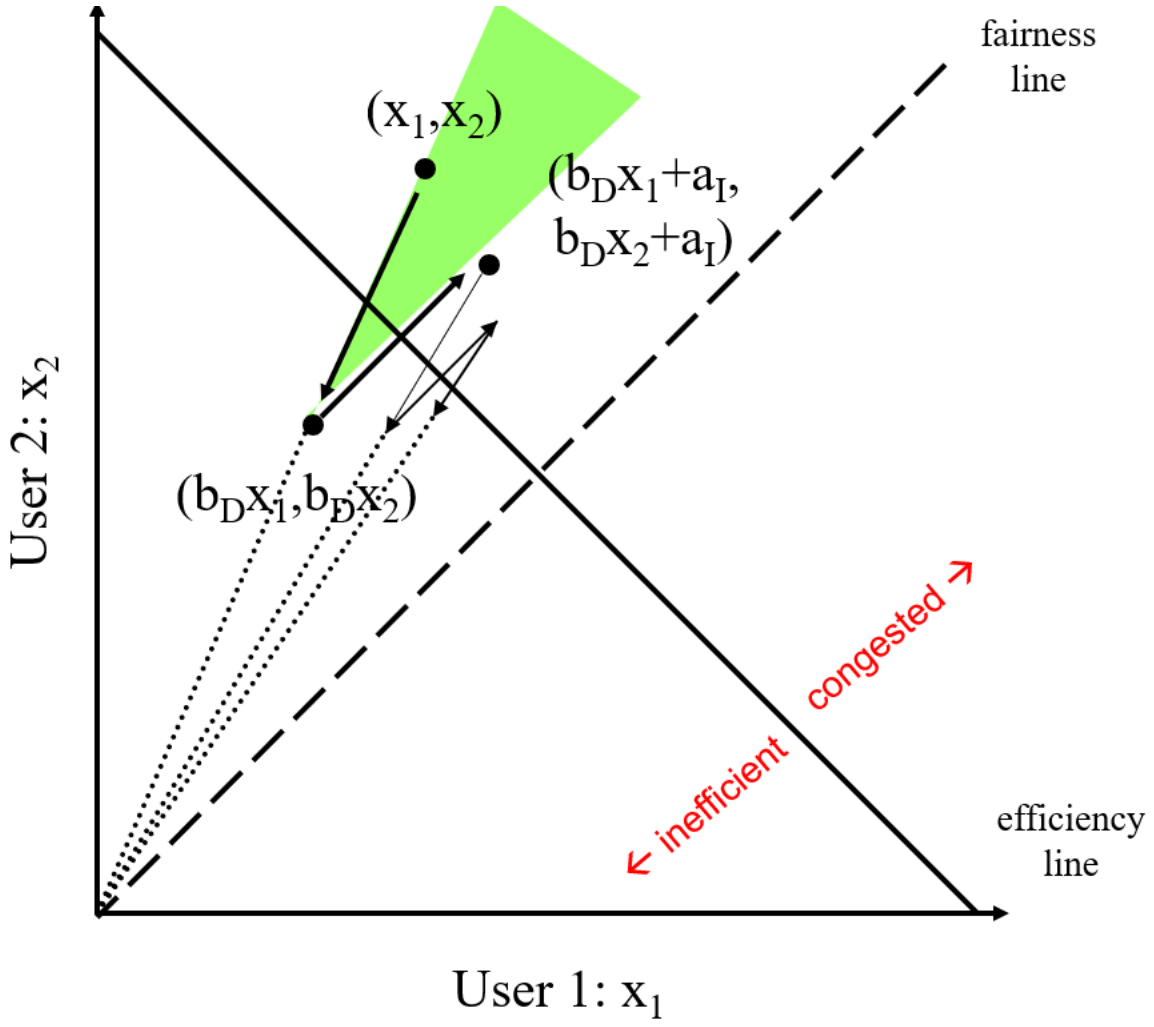
MULTIPLICATIVE INCREASE MULTIPLICATIVE DECREASE

- Increase: $x+a_I$
- Decrease: $x*b_D$



MULTIPLICATIVE INCREASE MULTIPLICATIVE DECREASE

- Increase: $x+a_I$
- Decrease: $x*b_D$
- Converges to fairness



CONGESTION CONTROL: IMPLEMENTAZIONE

- Stato nel mittente
  - CWND (inizializzata ad un valore piccolo)
  - ssthresh (inizializzata ad una costante grande)
  - DupACKcount : contatore di ACK duplicati
  - timer

- Eventi
  - ricezione ACK
  - ricezione ACK duplicato
  - timeout

EVENTO: ACK (NON DUPLICATO)

If CWND < ssthresh

- CWND += 1

Slow start phase

Else

- CWND = CWND + 1/CWND

"Congestion Avoidance" phase (additive increase)

COME REAGIRE ALLA PERDITA DEI PACCHETTI

- **Timeout**
  - un pacchetto è perso e la perdita viene individuata dallo scadere del timeout
  - anche i pacchetti successivi sono probabilmente persi, altrimenti sarebbe arrivato un ACK
  - la congestione "è seria"
    - ricominciare con un valore molto basso di CWND e ritornare alla fase di slow start.

- Duplicate ACK: ricevuto tre volte di seguito un ACK pe ril solito segmento:
  - il pacchetto n è perso, ma i pacchetti n+1, n+2, etc. sono stati ricevuti
    - il ricevente invia ACK duplicati per il pacchetto n, all'arrivo di ogni pacchetto non in sequenza
  - la congestione è "meno seria" perchè almeno 3 pacchetti hanno raggiunto il destinatario
    - impostare la dimensione della finestra alla metà della dimensione attuare e passare alla fase di Congestion Avoidance

EVENTO: TIME OUT

- On Timeout
  - ssthresh ← CWND/2
  - CWND ← 1

Continua con la fase di Slow Start

EVENTO: ACK DUPLICATO

Comportamenti diversi a seconda della versione di TCP

Soluzione 1

- dupACKcount ++
- If dupACKcount = 3 /* fast retransmit */
  - ssthresh = CWND/2
  - CWND = CWND/2

Continua con la fase AIMD

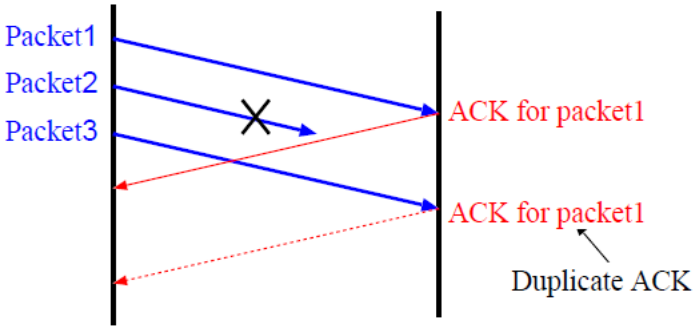
Soluzione 2

- dupACKcount ++
- If dupACKcount = 3 /* fast retransmit */
  - ssthresh = CWND/2
  - CWND ← 1

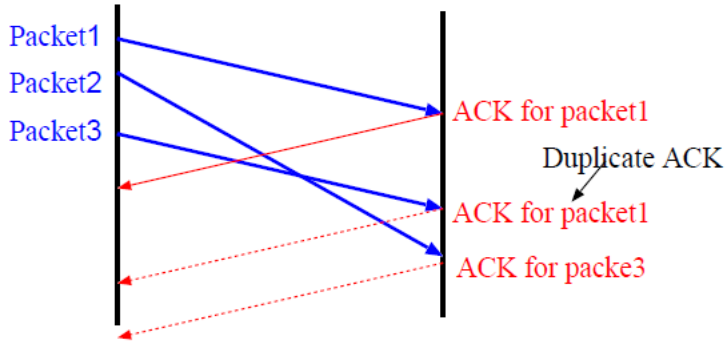
Continua con la fase di Slow Start

FAST RETRASMIT

- Ritrasmissione dei pacchetti senza aspettare lo scadere del time out
- Fast retransmit utilizza "duplicate ACK" per attivare la ritrasmissione dei pacchetti
- Duplicate ACK
  - ACKs che sono già stati ricevuti
  - Generati da perdita o riordinamento di pacchetti



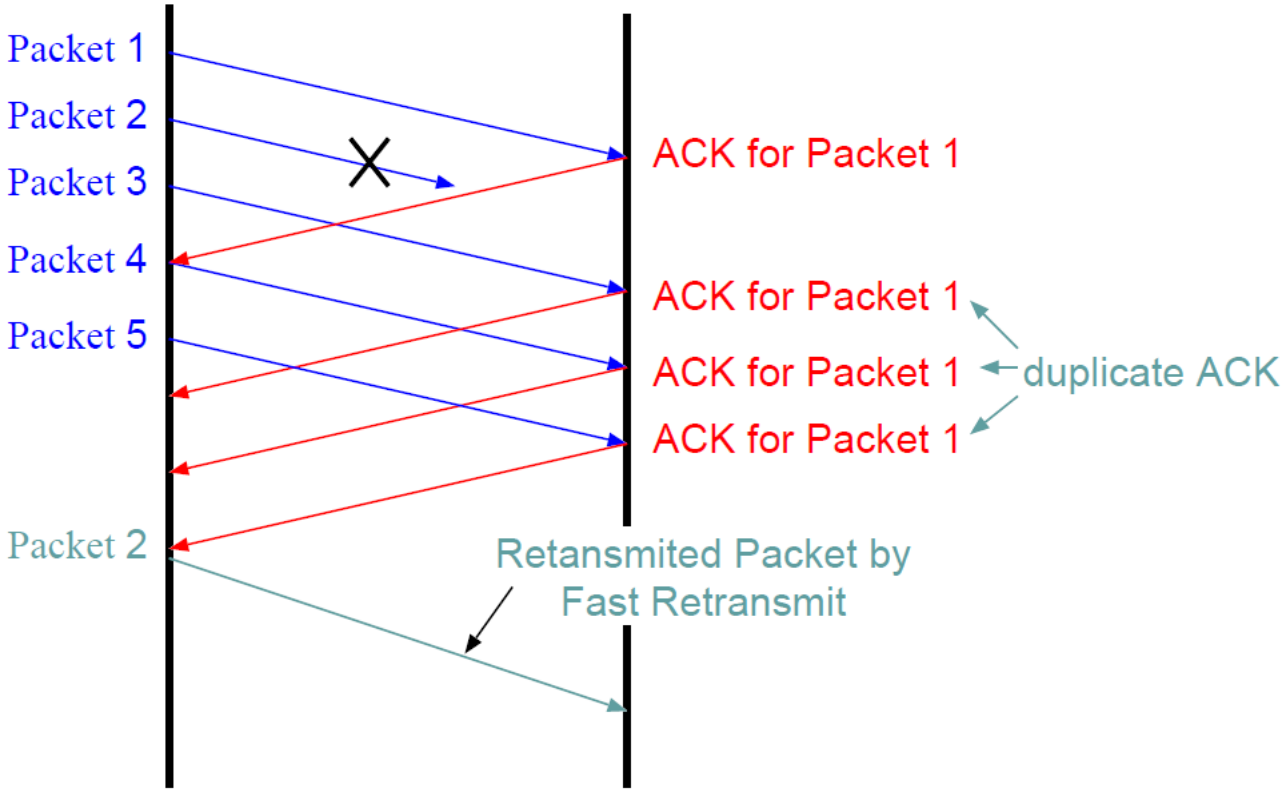
duplicate ACK generated by packet loss



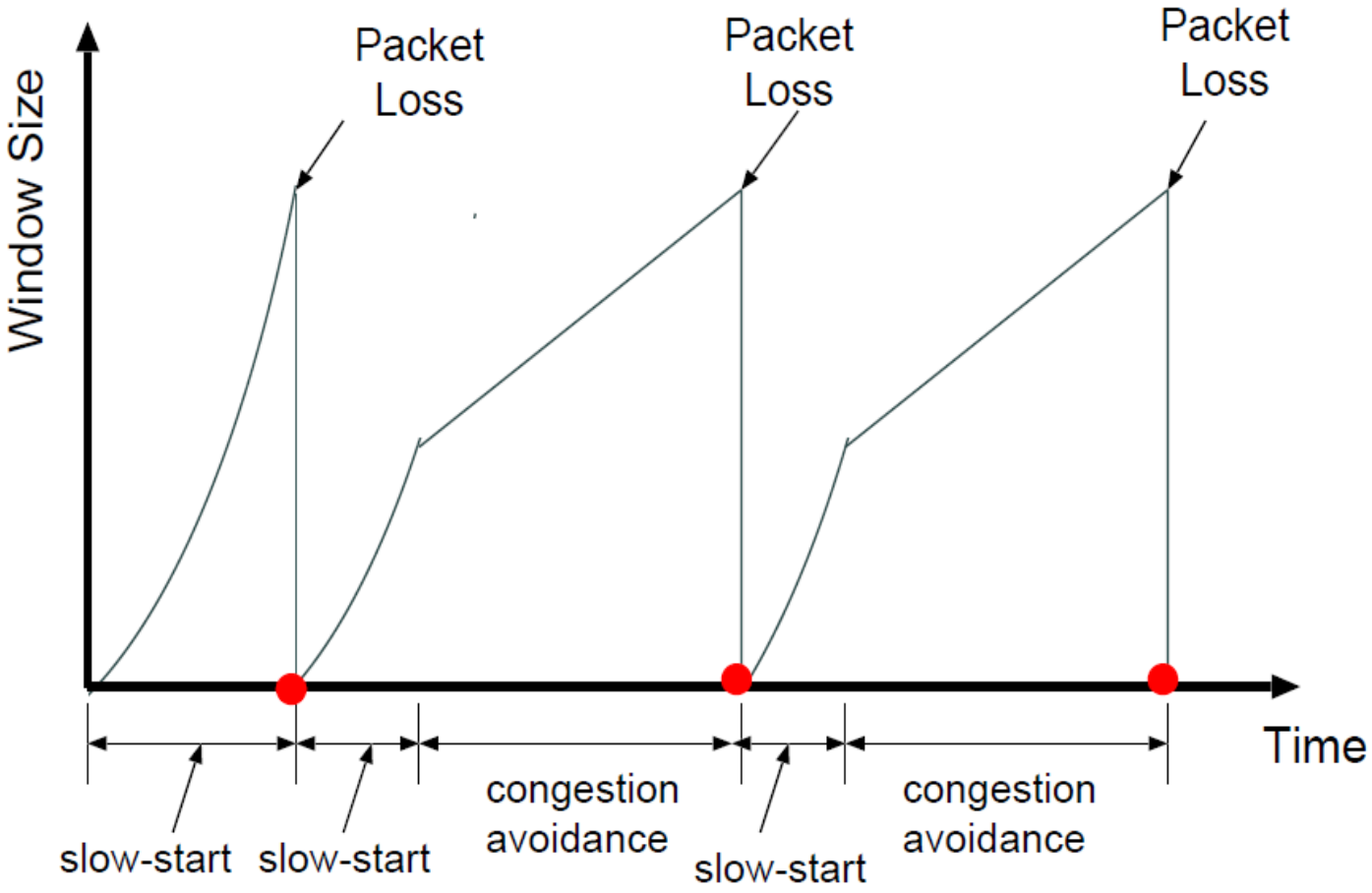
duplicate ACK generated by packet disorder

FAST RETRASMIT

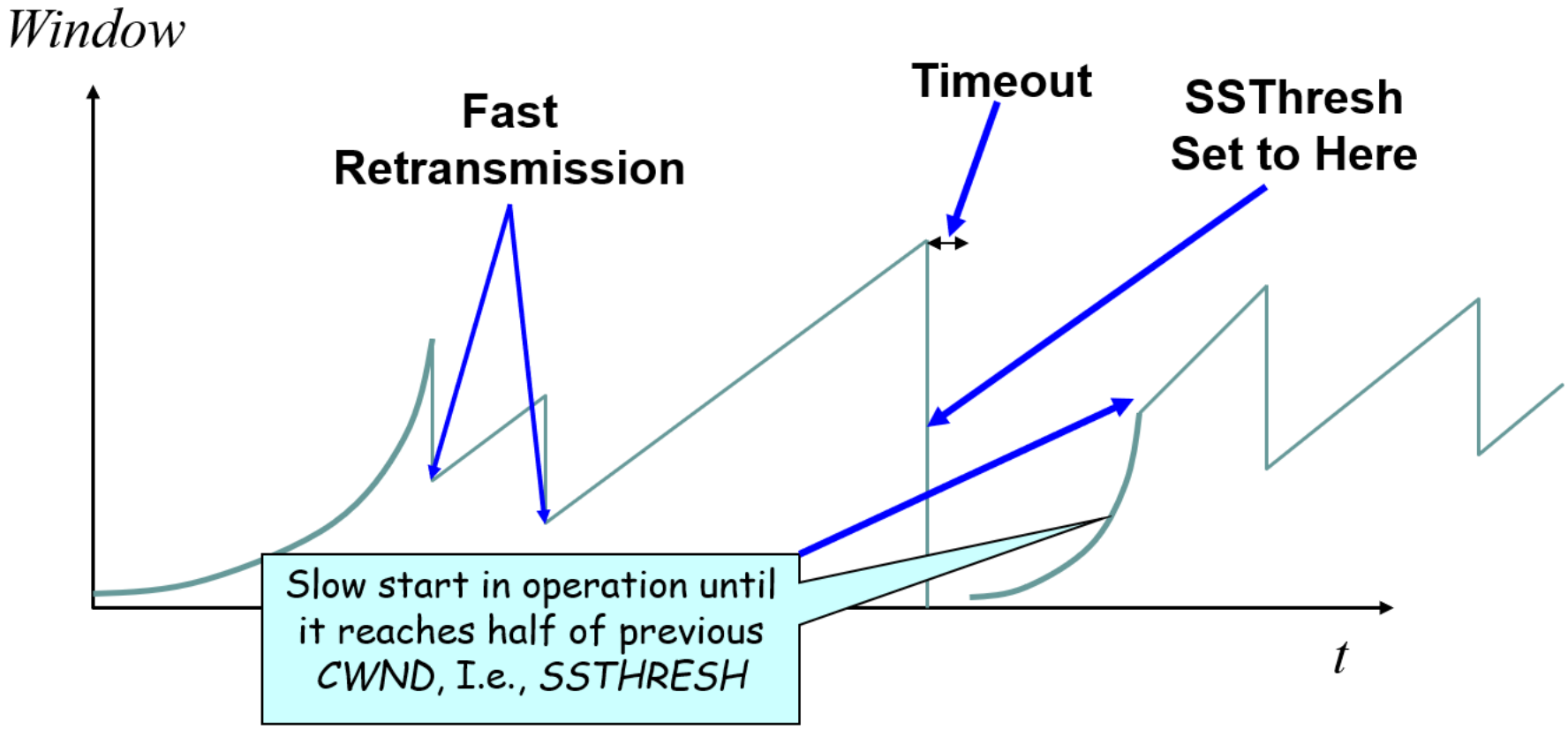
- TCP non è in grado di determinare se gli ACK duplicati sono generati dalla perdita di pacchetti oppure dal riordinamento dei pacchetti
- Ma il TCP assume che 3 ACK duplicati successivi sono causati da perdita di pacchetti



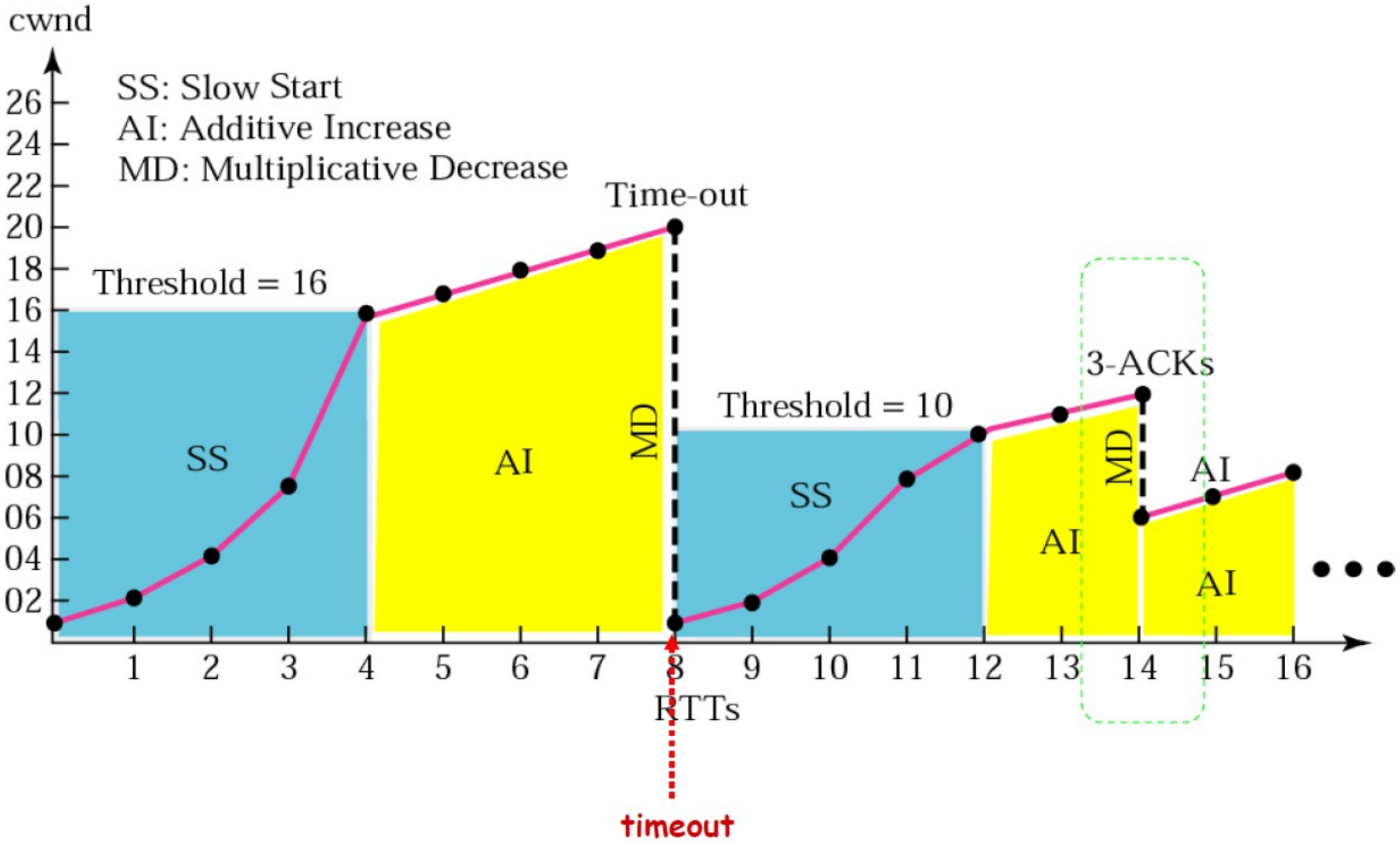
TCP TAHOE


Window Size
Packet Loss
Packet Loss
Packet Loss
Time
slow-start
slow-start
congestion avoidance
slow-start
congestion avoidance

CONGESTION AVOIDANCE: TCP RENO (SEMPLIFICATO)



CONGESTION AVOIDANCE: TCP RENO (SEMPLIFICATO



TCP RENO: FAST RECOVERY

- Fast Recovery introdotto per accelerare il recovery da una perdita isolata
  - congestion avoidance risulta troppo lento, in questo caso.
- Considerare una connessione TCP con:
  - CWND=10 pacchetti
  - l'ultimo ACK ricevuto è per il pacchetto #101
    - il ricevente si aspetta che il prossimo pacchetto abbia numero di sequenza = 101
  - nel frattempo, il mittente può inviare altri pacchetti
- 10 pacchetti [101,102,103,...110] in transito
  - il pacchetto 101 è perso
  - quali ACK vengono generati?
  - e come risponde il mittente?

PERDITA ISOLATA DI PACCHETTI

- ACK 101 (due to 102) cwnd=10 dupACK#1 (no xmit)
- ACK 101 (due to 103) cwnd=10 dupACK#2 (no xmit)
- ACK 101 (due to 104) cwnd=10 dupACK#3 (no xmit)
- RETRANSMIT 101 ssthresh=5 cwnd= 5
- ACK 101 (due to 105) cwnd=5 + 1/5 (no xmit)
- ACK 101 (due to 106) cwnd=5 + 2/5 (no xmit)
- ACK 101 (due to 107) cwnd=5 + 3/5 (no xmit)
- ACK 101 (due to 108) cwnd=5 + 4/5 (no xmit)
- ACK 101 (due to 109) cwnd=5 + 5/5 (no xmit)
- ACK 101 (due to 110) cwnd=6 + 1/5 (no xmit)
- ACK 111 (due to 101) ← only now can we transmit new packets

TCP RENO: FAST RECOVERY

- Idea di base: dare al mittente un "credito temporaneo" per ogni ACK duplicato, in modo da tenere la pipeline piena

- if dupACKcount = 3

  ssthresh = cwnd/2

  cwnd = ssthresh + 3

- nella fase di fast recovery
  - cwnd = cwnd + , per ogni ACK duplicato addizionale
  - come per slow start

- si esce dalla fase di fast recovery dopo aver ricevuto un nuovo ACK
  - set cwnd = ssthresh

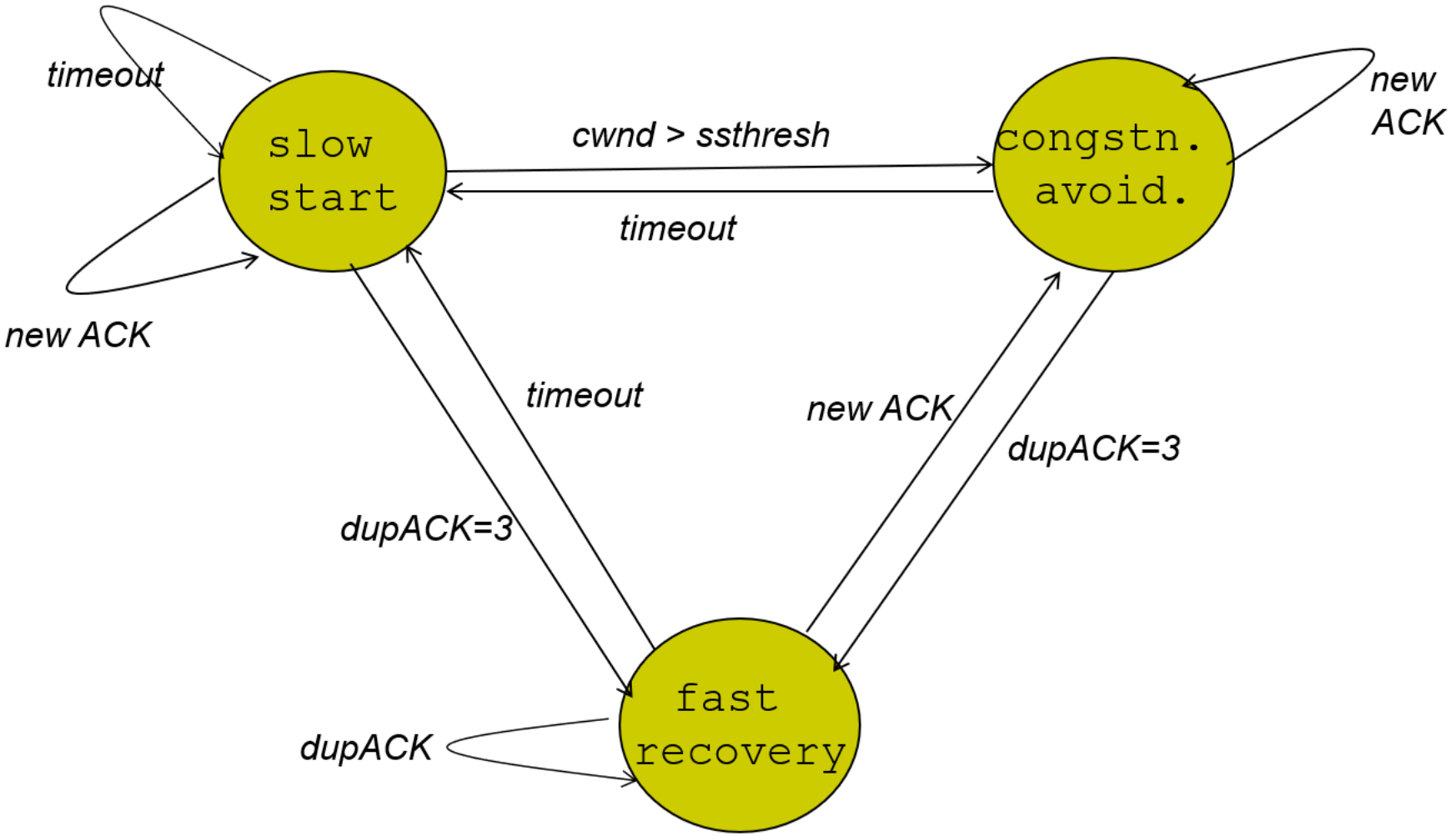
- invio pacchetti anche dopo ACK duplicato

FAST RECOVERY (TCP RENO)

stesso scenario precedente

- ACK 101 (due to 102) cwnd=10 dup#1
- ACK 101 (due to 103) cwnd=10 dup#2
- ACK 101 (due to 104) cwnd=10 dup#3
- REXMIT 101 ssthresh=5 cwnd= 8 (5+3)
- ACK 101 (due to 105) cwnd= 9 (no xmit)
- ACK 101 (due to 106) cwnd=10 (no xmit)
- ACK 101 (due to 107) cwnd=11 (xmit 111)
- ACK 101 (due to 108) cwnd=12 (xmit 112)
- ACK 101 (due to 109) cwnd=13 (xmit 113)
- ACK 101 (due to 110) cwnd=14 (xmit 114)
- ACK 111 (due to 101) cwnd = 5 (xmit 115) ← exiting fast recovery
- Packets 111-114 already in flight
- ACK 112 (due to 111) cwnd = 5 + 1/5 ← back in congestion avoidance

TCP RENO: DIAGRAMMA DEGLI STATI



VERSIONI DI CONTROLLO CONGESTIONE

- Tahoe
  - implementato in 4.3BSD UNIX Tahoe, (circa 1988)
  - Slow Start e Congestion Avoidance
  - Fast retrasmit: rinvio e cwnd =1 alla ricezione di 3 dupACK
- Reno
  - implemented in 4.3BSD UNIX Reno, (circa 1990)
  - Slow Start + Congestion Avoidance +Fast Recovery (prima versione)
  - cwnd =1 alla ricezione del timeout
  - cwnd = cwnd/2 fast recovery: alla ricezione di 3 dupACK
- TCP newReno
  - nessuna implementazione di riferimento
  - TCP-Reno + algoritmo di Fast-recovery (seconda versione)
- TCP-SACK
  - utilizza selective acknowledgements