Forgalomszabályozás az Interneten (2)

Sonkoly Balázs
sonkoly@tmit.bme.hu

2015.11.24.

Áttekintés

- TCP példák
  - fontosabb algoritmusok, egy-két illusztráció
- Értsük meg, mit csináltunk
  - matematikai modellek (utólag)
  - probléma megfogalmazása
  - most: optimalizációs feladatként
- TCP javítása (“otthoni kitekintés”)
  - TCP problémái
  - új ötletek, algoritmusok
  - most már a matematikai modellek alapján

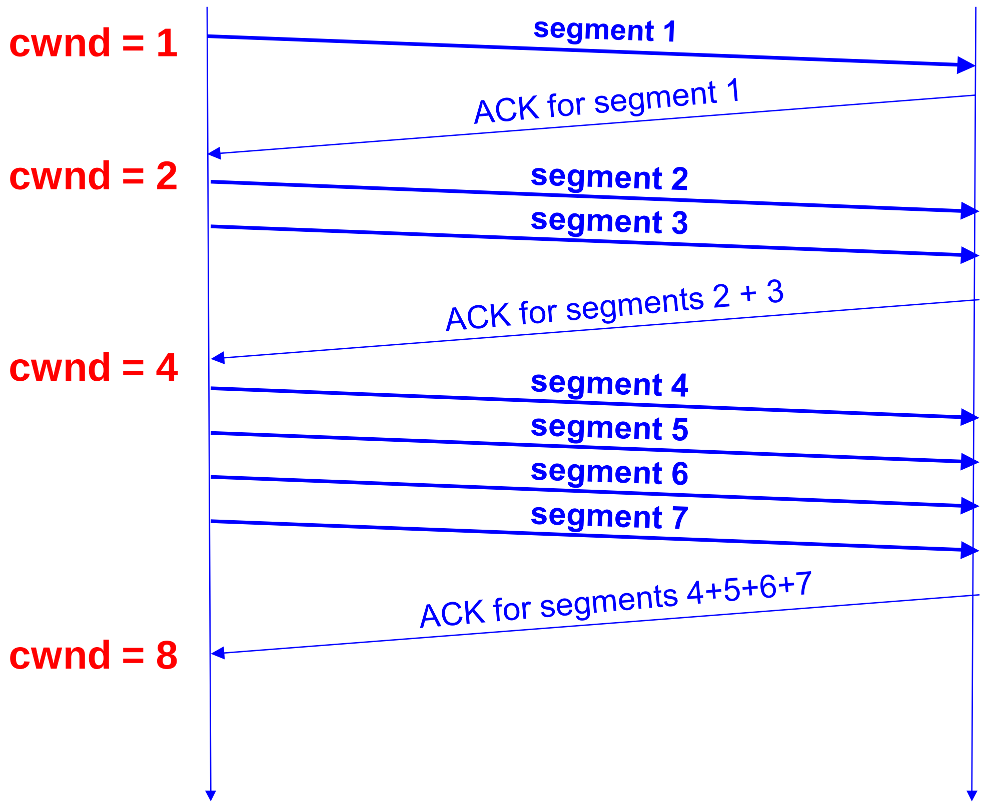
TCP congestion control mechanizmusai

- Csúszóablakos szabályozás
  - flow control és congestion control is ez alapján
  - TCP adó nem tudja, hol a bottleneck: vevőnél vagy a hálózatban
  - congestion window (**`cwnd`**)
    - dinamikusan frissített változó
    - ami meghatározza az adási sebességet
- Négy fő működési fázis
  - (különböző algoritmusok a cwnd szabályozására)
  - Slow start
  - Congestion Avoidance
  - Fast retransmit
  - Fast recovery

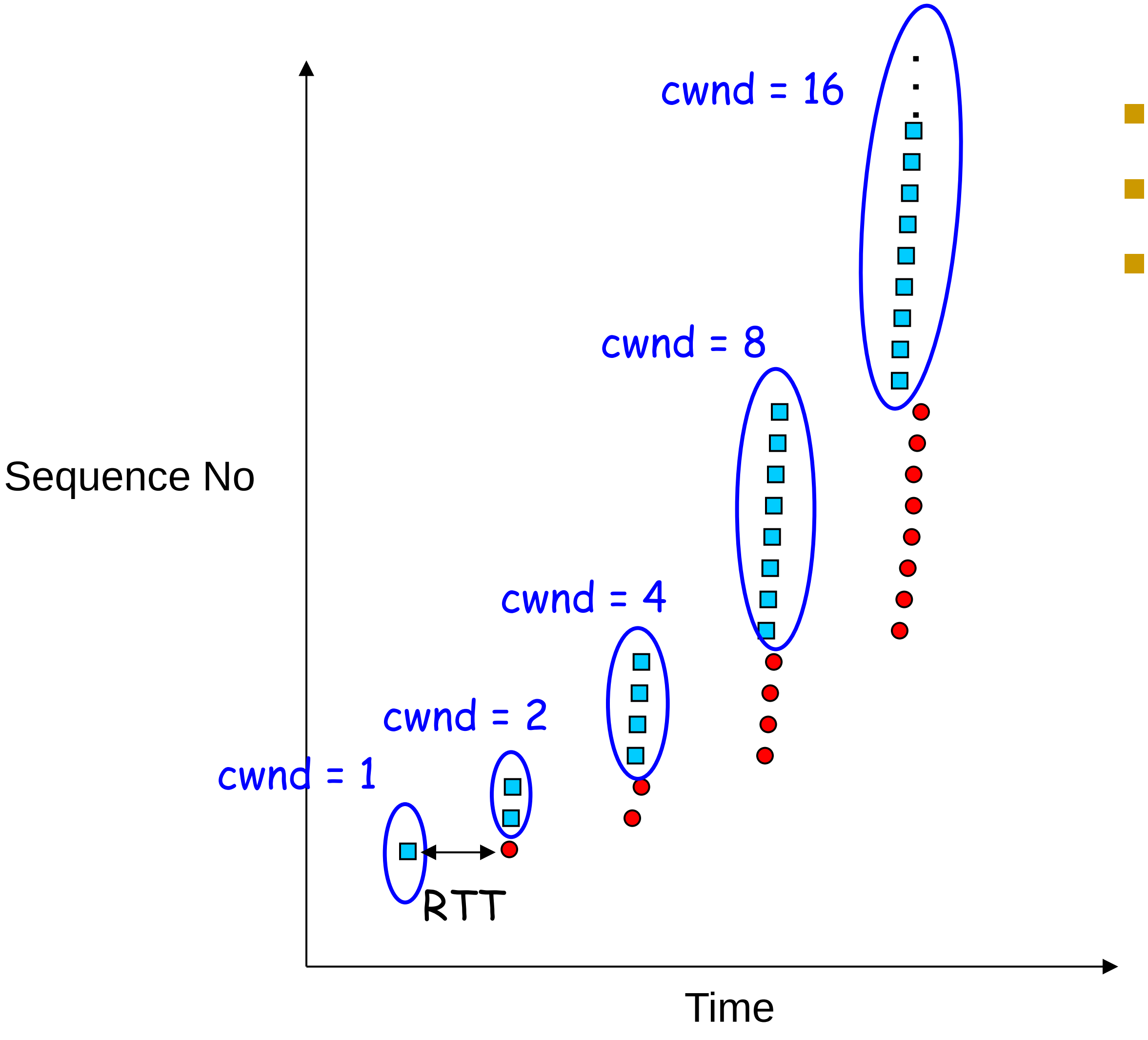
Slow start

- Determine available capacity at first
- TCP transmission is constrained
  - **`awnd = min (adwnd, cwnd)`**
    - allowed window (in segments)
    - advertised window
      - set by receiver
      - unused credit + granted in the most recent ACK
    - congestion window
      - set by sender
- Algorithm
  - set cwnd = 1
  - cwnd++ for each received ACK (~ doubled in one RTT)
  - indication of loss
    - timeout
    - receipt of duplicate ACKs
  - end of slow start
    - loss OR
    - **`cwnd`** exceeds a threshold (**`ssthresh`**)
- Properties
  - exponential growth (not very slow!)
  - but slower growth compared to burst arrival

Slow start – example



Slow start – sequence plot



- data segments
- ACKs
- cwnd doubles every round-trip time

Congestion avoidance

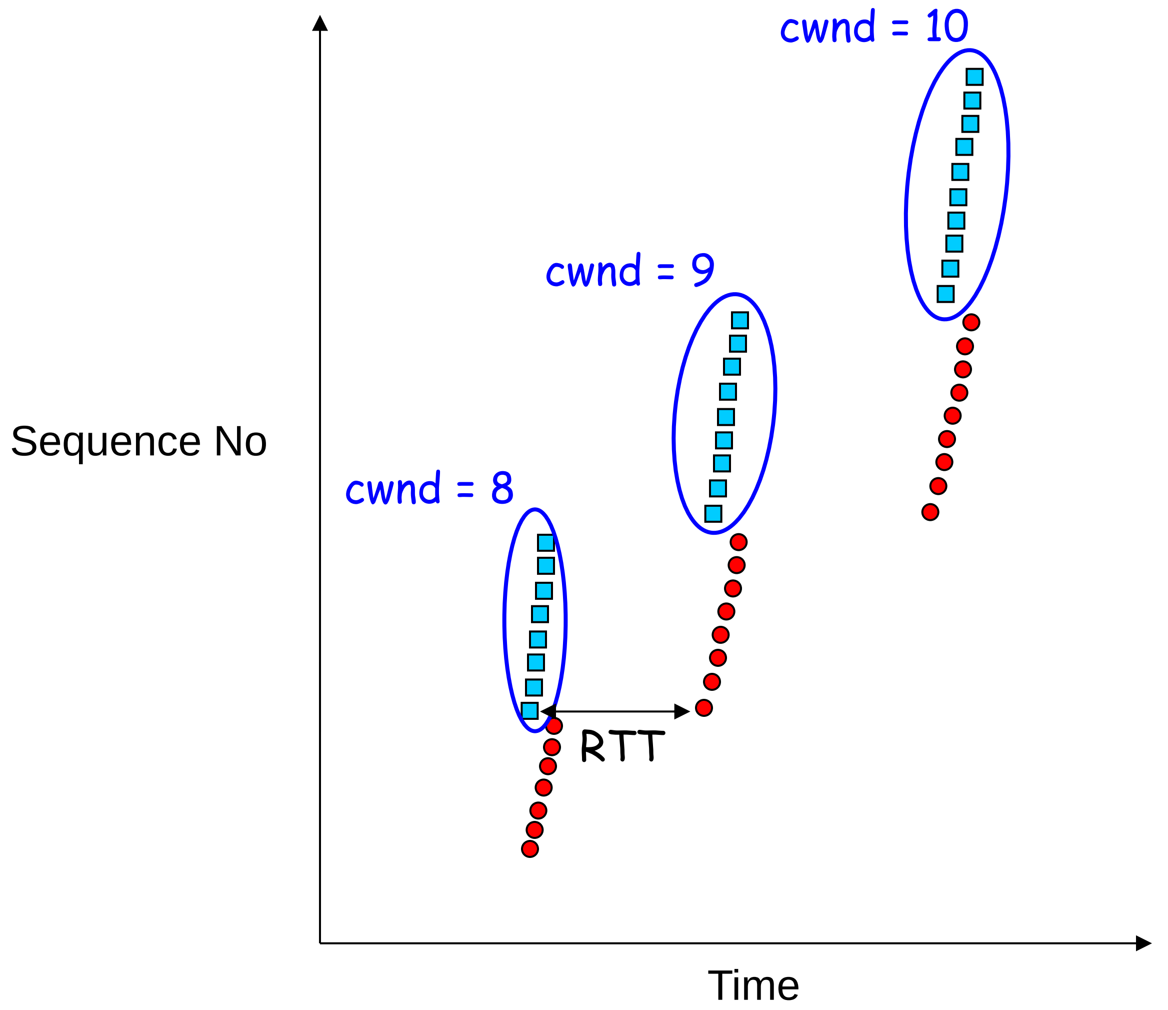
- Easy to drive the network in saturation
- but hard for the network to recover
- Slow start is too aggressive
- Solution: slow start + linear growth in cwnd
- Initialization
  - `cwnd = 1`
  - `ssthresh =` (e.g.) `65,535` bytes (OR arbitrarily high – RFC 2581)
- After timeout
  - `ssthresh = cwnd / 2`
  - `cwnd = 1` $\rightarrow$ slow start until `cwnd == ssthresh`
  - for `cwnd > ssthresh`
    - increase `cwnd` by one for each RTT (***Additive Increase***)
    - in practice:

$$\texttt{cwnd} = \texttt{cwnd} + 1 \quad \Leftarrow \text{ for each RTT}$$

in segments: $$\texttt{cwnd} = \texttt{cwnd} + \frac{1}{\texttt{cwnd}} \quad \Leftarrow \text{ for each ACK}$$

in bytes: $$W = W + \frac{MSS}{\texttt{cwnd}} = W + \frac{MSS^2}{W}$$

Congestion avoidance – sequence plot

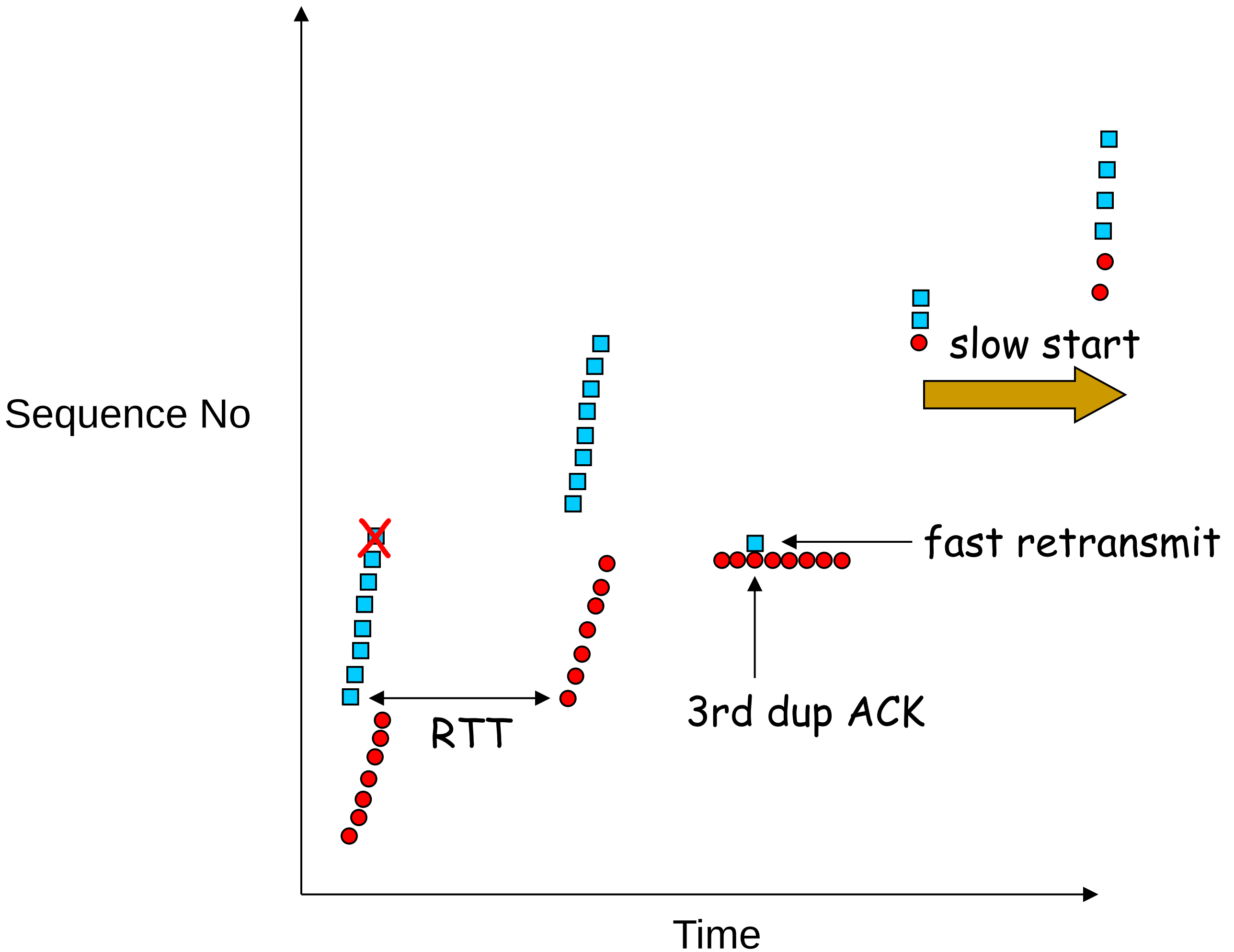


- data segments
- ACKs
- cwnd is increased by 1 for each RTT

Fast retransmit

- After a segment lost TCP may be slow to retransmit
- if this is the only missing segment
  - it delays the whole flow transmission
  - receiver has to wait for the missing segment
- Solution: retransmit packet without waiting for RTO!
- <u>receiver</u>
  - if receives a segment out of order → ACK for the last inordered segment that was received
  - continues repeat this ACK until missing segment arrives
- <u>source</u>
  - when receives a duplicate ACK it means
    1. the segment following the ACKed segment was delayed
       - no action needed
    2. segment was lost
       - retransmission needed
    - test
      - wait for the next ACK
      - 3 dup ACKs → retransmit the segment
- TCP Tahoe (implemented in 4.3 BSD Tahoe, Net/1, ~1988)

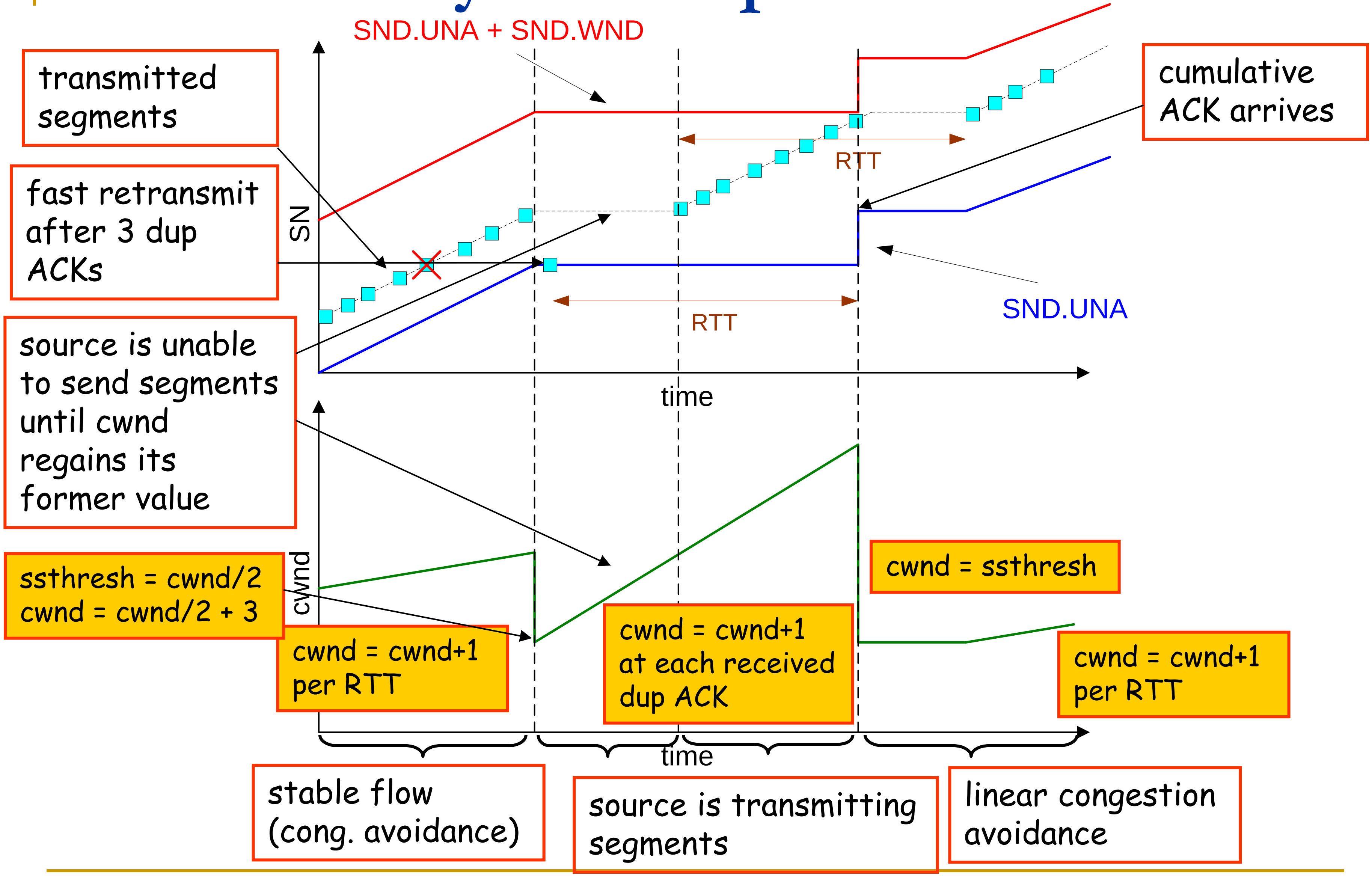
Fast retransmit (TCP Tahoe) – sequence plot



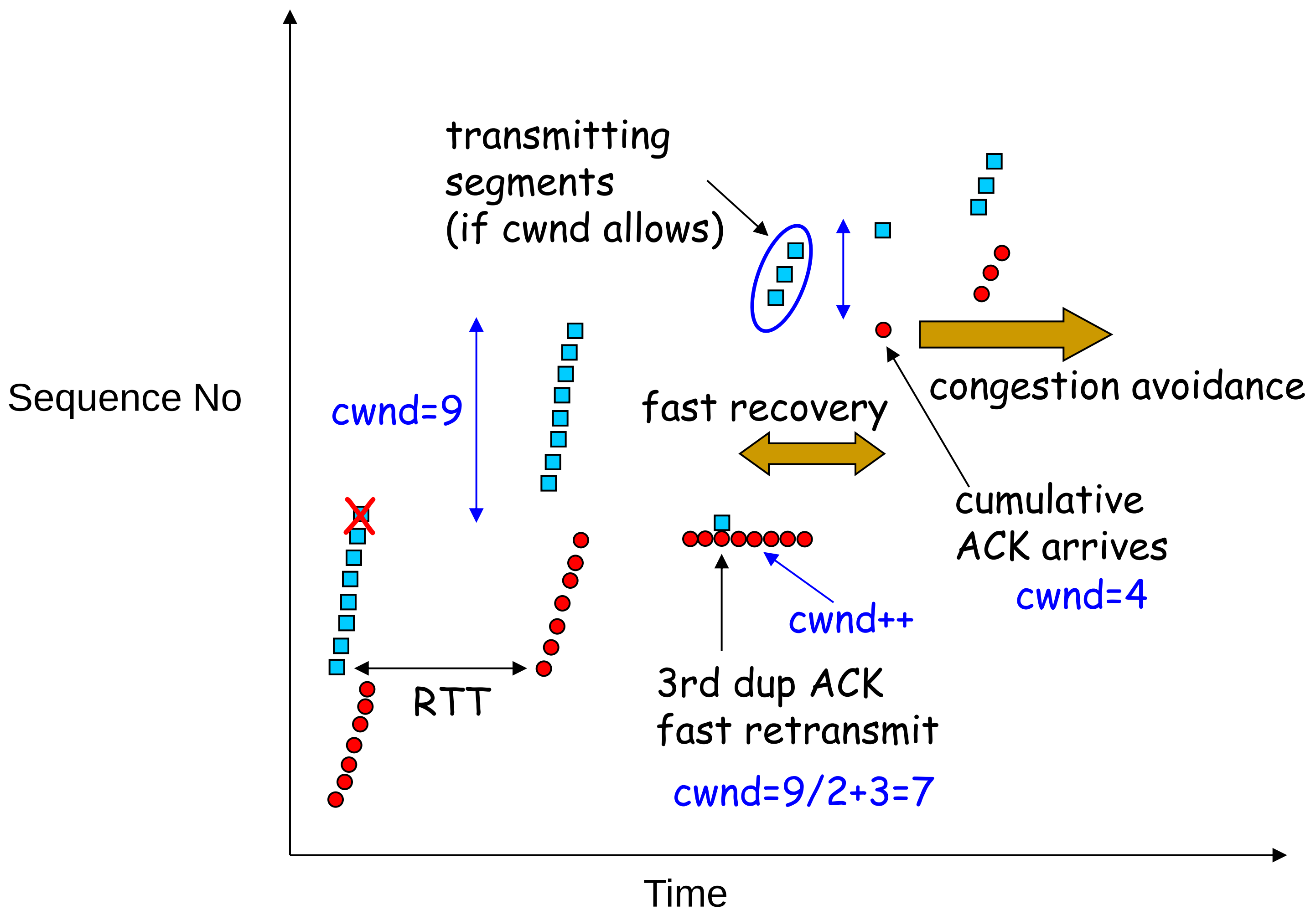
Fast recovery

- Goal: avoid slow start!
- after receiving the third dup ACK
  - `ssthresh = cwnd / 2`
  - retransmit the segment (fast retransmit)
  - `cwnd = ssthresh + 3` (***inflating*** the window)
  - if additional dup ACKs arrives
    - `cwnd = cwnd + 1` (***inflating*** the window)
    - transmit a segment if possible
  - if the next ACK arrives (for new segment)
    - `cwnd = ssthresh` (***deflating*** the window)
- Inflating the window
  - dup ACK means $\rightarrow$ one packet arrived and cached at receiver
  - one new packet can be sent

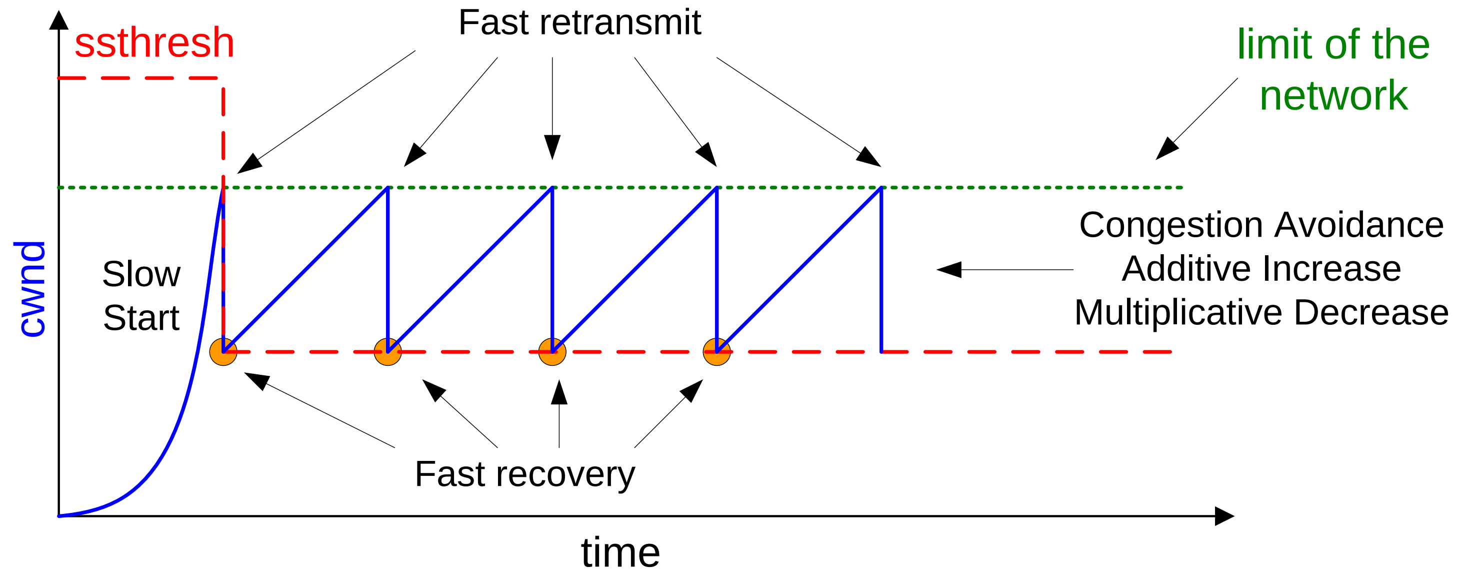
Fast recovery – example



Fast recovery (TCP Reno) – sequence plot

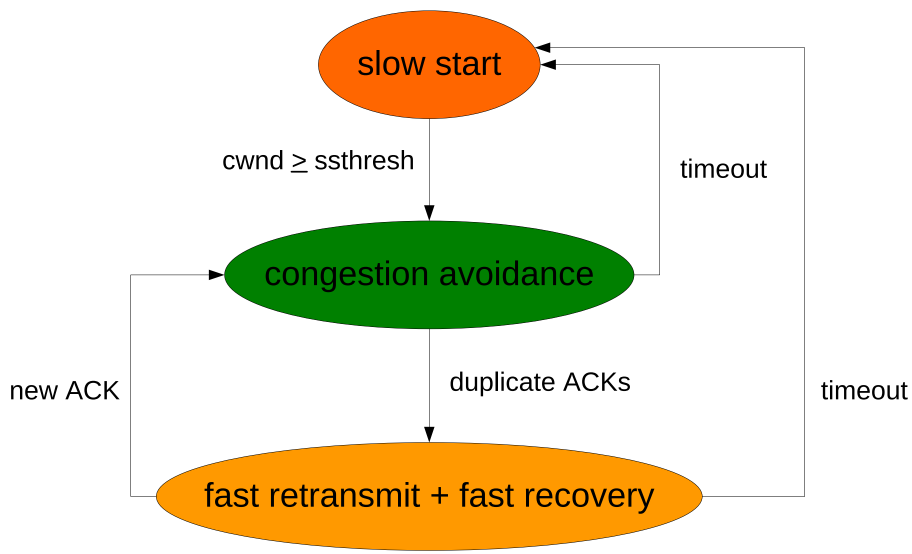


Fast recovery – TCP Reno



- TCP Reno
  - implemented in 4.3 BSD Reno, Net/2, ~1990
  - Slow start
  - Congestion avoidance: AIMD (Additive Increase Multiplicative Decrease)
  - Fast retransmit
  - Fast recovery
- Problem
  - multiple losses from a single window??

Summary of the algorithm (TCP Reno)

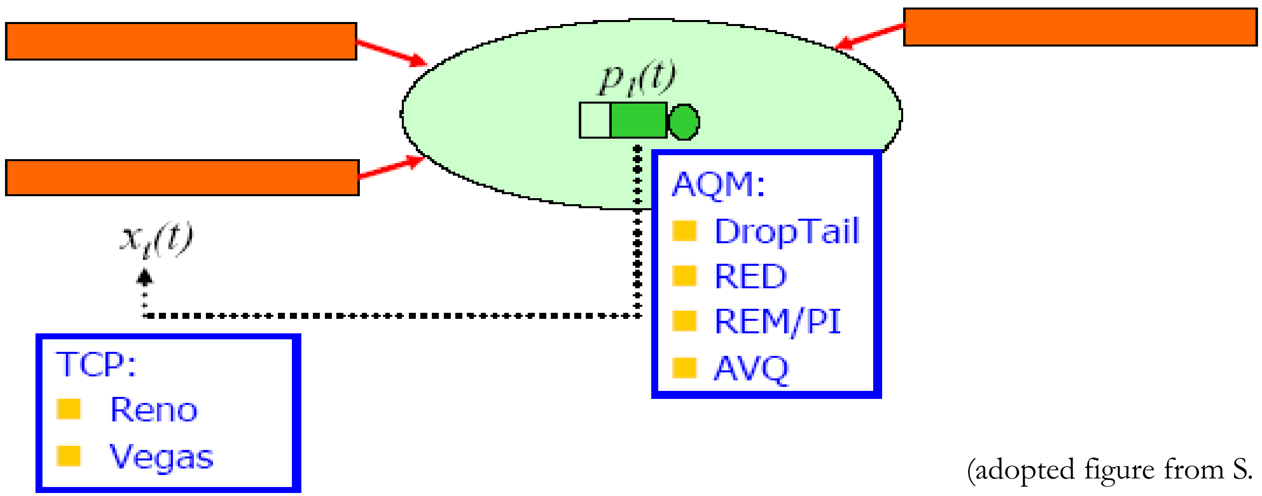


- Initialization
  - cwnd = 1 (segment)
  - ssthresh = 65,535 bytes
- TCP sender sends segment: effwnd
  - maxwnd = min(cwnd, adwnd)
  - effwnd = maxwnd – (lastbytesent – lastbyteacked)
- Congestion avoidance
  - cwnd = cwnd + 1 for each RTT
  - cwnd = cwnd + 1/cwnd for each ACK
  - if congestion:
    - ssthresh = max(2, min(cwnd, adwnd)/2)
- Slow start
  - cwnd = 1
  - cwnd = cwnd + 1 for each ACK
  - if cwnd > ssthresh → congestion avoidance
- Fast recovery
  - cwnd = ssthresh + 3
  - if additional dup ACKs
    - cwnd = cwnd + 1
    - transmit segment if effwnd > 0
  - if new ACK
    - cwnd = ssthresh
    - congestion avoidance

Áttekintés

- TCP példák
  - fontosabb algoritmusok, egy-két illusztráció
- Értsük meg, mit csináltunk
  - matematikai modellek (utólag)
  - probléma megfogalmazása
  - most: optimalizációs feladatként
- TCP javítása ("otthoni kitekintés")
  - TCP problémái
  - új ötletek, algoritmusok
  - most már a matematikai modellek alapján

TCP & AQM



(adopted figure from S. H. Low et al.)

- Transport protocols
  - sending rate (**$x_i(t)$**)
  - input of the system
- Active Queue Management (AQM)
  - congestion measure (**$p_l(t)$**) (e.g., loss rate, delay,...)
  - feedback signal

TCP & AQM

TCP – AQM model:

$$\begin{aligned} x(t+1) &= F(p(t), x(t)) \\ p(t+1) &= G(p(t), x(t)) \end{aligned}$$

- **Equilibrium**
  - performance (throughput, loss, delay,...)
  - fairness
  - utility
- Duality theory (optimization)
  - sending rates → primal variables
  - cong. measure → dual variables
  - flow / cong. control → optimization

- **Dynamics**
  - local stability
  - global stability
- Control theory
  - feedback system
  - distributed
  - delayed

Optimization approach

- Three fields (S.H.Low):
  - formulate the problem as an optimization problem
  - interpret a given technique as an optimizer algorithm for an optimization problem (*reverse engineering*)
  - extend the underlying theory using optimization theoretic techniques (theorems, analytic proofs)

Goals

- **F. P. Kelly, A. Maulloo, D. Tan,** "*Rate control for communication networks: shadow prices, proportional fairness and stability*", Journal of the Operational Research Society, 49 (1998)
- Sources that can modify their sending rates according to the available bandwidth within the network → ***elastic traffic***
- e.g., TCP
- key questions:
  - How should the available bandwidth be shared between competing streams of elastic traffic?
  - How could a "fair" algorithm be implemented in a large-scale network?
- tractable analytical framework is necessary
  - for analyzing the fairness characteristics
  - stability & convergence of the system, etc.

Basic model

- Flow / congestion control – resource allocation problem

Notations

J	:	set of resources (e.g., links)
C_j	:	finite capacity of resource $j \in J$ (e.g., link bandwidth)
$C = (C_j, j \in J)$	:	capacity vector
r	:	route, non-empty subset of J (associated with a user)
R	:	set of possible routes
$A = (A_{jr}, j \in J, r \in R)$	:	0/1 matrix: $A_{jr} = 1$ if $j \in r$ (resource j lies on route r), $A_{jr} = 0$ otherwise
x_r	:	rate allocated to user r
$x = (x_r, r \in R)$	:	rate vector
$U_r(x_r)$	:	utility function of user r
$U = (U_r(.), r \in R)$	:	utility functions

- Utility functions
  - the utility (benefit) of the rate allocation to user
  - increasing, strictly concave, additive, continuously differentiable over the range $x_r \geq 0$
  - e.g., $U_r = \log x_r$

Basic model

- System optimal rates solve the following optimization problem:

$SYSTEM(U, A, C):$

$$\max_{x \geq 0} \sum_{r \in R} U_r(x_r)$$

subject to

$$Ax \leq C$$

- Constraint: capacity constraint
- Different utility functions lead to different resource allocations
- Simple model, analytically tractable
- Unique solution if utility functions are strictly concave

- Goal: fair resource allocation
- What does "**fairness**" mean??

Different notions of fairness

- max-min fairness
  - a set of rates is max-min fair if no rate may be increased without simultaneously decreasing another rate which is already smaller
  - for a single bottleneck → equal share of the resource for each flow
- proportional fairness
  - rate allocation x is proportionally fair if
    - x is feasible ($x \geq 0$ and $Ax \leq C$)
    - for any other feasible vector x^*, the aggregate of proportional changes is zero or negative

$$\sum_{r \in R} \frac{x_r^* - x_r}{x_r} \leq 0$$

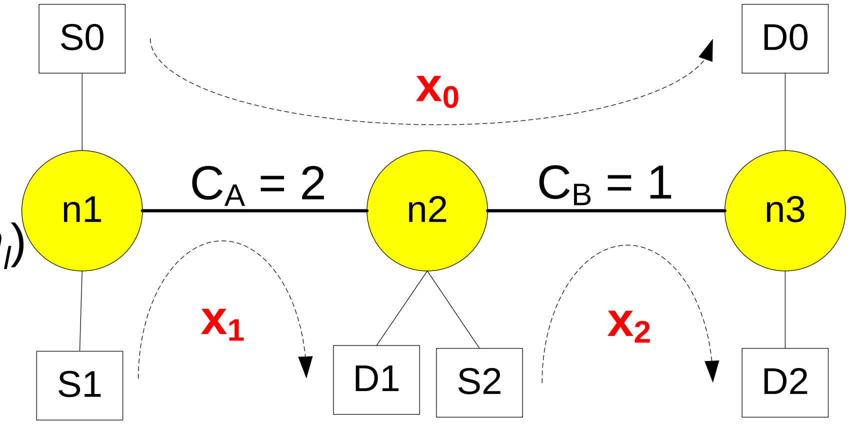
- weighted proportional fairness

$$\sum_{r \in R} w_r \frac{x_r^* - x_r}{x_r} \leq 0$$

  - similar, but a weight is also used
  - weight: e.g., the no. of sub-users → non-cooperative context
  - $w_r x_r$ aggregate rate

Example: max-min fairness

- max-min fairness
  - algorithm to find $\{ x_r \}$
  - 1. divide c_l capacities equally among all the flows sharing the link ($f_l = c_l / n_l$)
  - find the smallest rate for all routes $z_r = min \{f_l : l \text{ in route } r\}$
  - find the smallest of these rates ($z_{min} = min\ z_r$)
  - for all sources: if $z_r == z_{min}$ => allocate $x_r = z_{min}$ (max-min rate)
  - 2. reduce all c_l with the allocated max-min rates
  - if we have unallocated capacity => goto 1.



  - example:
    - $f_A = C_A / 2 = 1$
    - $f_B = C_B / 2 = 0.5$
    - $z_0 = \min(1, 0.5) = 0.5$
    - $z_1 = 1$, $z_2 = 0.5$
    - $z_{min} = 0.5$
    - allocate x_0 , $x_2 = z_{min}$
    - $C_A = C_A - 0.5$, $C_B = C_B - (2*0.5)$
    - $f_A = C_A / 1 = 1.5$
    - remaining capacity on A → x_1
    - $x_0 = 0.5$ and $x_1 = 1.5$ and $x_2 = 0.5$

Example: proportional fairness

- <u>proportional fairness</u>
  - utility function ($w_r = 1$)

$$U_r(x_r) = \log x_r$$

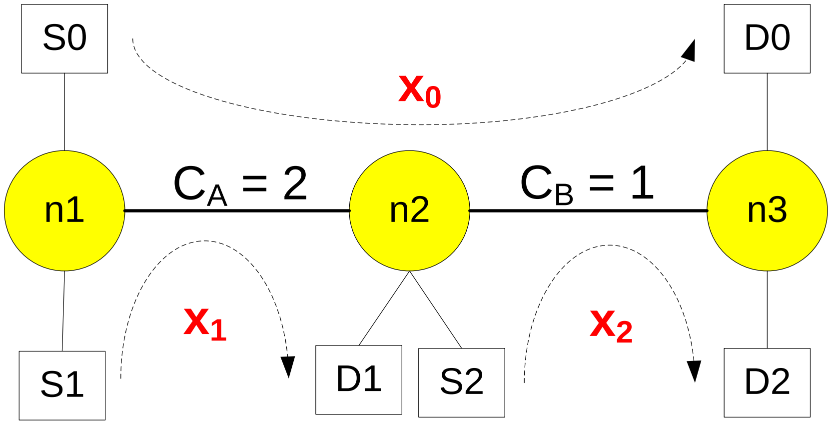
  - resource allocation problem:

$$\log x_0 + \log x_1 + \log x_2$$

  - subject to

$$\begin{aligned} x_0 + x_1 &\leq 2 \\ x_0 + x_2 &\leq 1 \end{aligned}$$

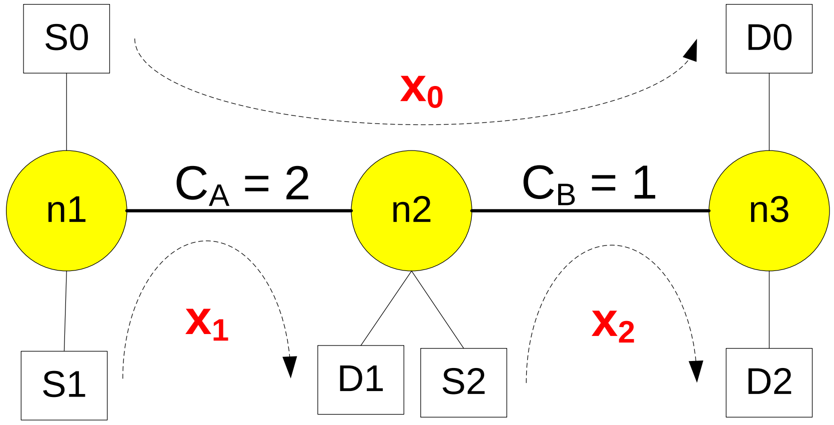
  - and constraints would be satisfied with equality (max utilization)



  - Lagrange multipliers technique can be used for
    - finding local maxima/minima of a function
    - subject to equality constraints

Example: proportional fairness

- <u>proportional fairness</u>
  - to solve the optimization problem
  - use *Lagrange multiplier technique*
  - λ_A and λ_B Lagrange multipliers
  - corresponding to the capacity constraints
  - Lagrangian for the problem:



$$L(x,\lambda) = \log x_0 + \log x_1 + \log x_2 - \lambda_A(x_0+x_1) - \lambda_B(x_0+x_2)$$

  - x: vector of allocated rates, λ: vector of Lagrange multipliers
  - setting the partial derivatives to zero: $\dfrac{\partial L}{\partial x_r} = 0$
  - we get:

$$x_0 = \frac{1}{\lambda_A+\lambda_B}, \quad x_1 = \frac{1}{\lambda_A}, \quad x_2 = \frac{1}{\lambda_B}$$

$$\begin{aligned} x_0 + x_1 &= 2 \\ x_0 + x_2 &= 1 \end{aligned}$$

  - thus:

$$\hat{x}_0 = \frac{\sqrt{3}+1}{3+2\sqrt{3}} \approx 0.42, \quad \hat{x}_1 = \frac{\sqrt{3}+1}{\sqrt{3}} \approx 1.57, \quad \hat{x}_2 = \frac{1}{\sqrt{3}} \approx 0.57$$

Basic model

- System optimal rates solve the following optimization problem:

$SYSTEM(U, A, C):$

$$\max_{x \geq 0} \sum_{r \in R} U_r(x_r)$$

subject to

$$Ax \leq C$$

- What is the problem?

- Utility functions are not known by the network
- Solution: decomposition of the problem
- SYSTEM → USER, NETWORK

Basic model (pricing interpretation)

- User:
- choose (bid) an amount to pay per unit time (w_r)
- receives x_r rate proportional to w_r
- $x_r := w_r / \lambda_r$
- where λ_r → charge per unit flow for user r (price for a unit of BW)

$USER_r(U_r; \lambda_r)$:

$$\max_{w_r \geq 0} \left\{ U_r \left(\frac{w_r}{\lambda_r} \right) - w_r \right\}$$

- Network:
- suppose that knows vector w and attempts to maximize the following

$NETWORK(A, C; w)$:

$$\max_{x \geq 0} \sum_{r \in R} w_r \log x_r$$

subject to

$$Ax \leq C$$

Basic model (pricing interpretation)

- For this system always exist vectors *λ, w, x*
- where *w* solves the *USER* problems
- *x* solves the *NETWORK* problem
- and *x* is a unique solution of *SYSTEM*
- What are the fairness properties of this system?
- How can we get these solutions?

Fairness of the basic model

- If $w_r = 1$ for all r, then x solves *NETWORK(A, C; w)* if and only if it is <u>proportionally fair</u>
- the utility functions are constructed with a goal to provide this
- For general w_r values → x solves *NETWORK(A, C; w)* if and only if it is <u>weighted proportionally fair</u>
- If for a fixed set of users and arbitrary parameters, the network solves *NETWORK(A, C; w)* then
- the resulting x solve a variant of the original problem *SYSTEM(U, A, C)*
- with a weighted objective function: $\sum_r \alpha_r U_r(x_r)$
- where

$$\alpha_r = \frac{w_r}{x_r U_r'(x_r)}$$

Kitekintés (1)

HOGYAN OLDJUK MEG?

Solution for *Network(A, C; w)*

- Lagrangian for *NETWORK(A, C; w)*

$$L(x, z; \mu) = \sum_{r \in R} w_r \log x_r + \mu^T (C - Ax - z)$$

- z ≥ 0 slack variables (turn the inequality into equation)
- μ Lagrange multipliers (*shadow prices*)
- taking the partial derivatives

$$\frac{\partial L}{\partial x_r} = \frac{w_r}{x_r} - \sum_{j \in r} \mu_j$$

- setting them to zero
- we get the unique optimum of the primal problem

$$x_r = \frac{w_r}{\sum_{j \in r} \mu_j}$$

Dual problem of *Network(A, C; w)*

- The dual problem can also be established:

$DUAL(A, C; w):$

$$\max_{\mu \geq 0} \left\{ \sum_{r \in R} w_r \log \left(\sum_{j \in J} \mu_j \right) - \sum_{j \in J} \mu_j C_j \right\}$$

- *NETWORK(A, C; w)* and *DUAL(A, C; w)*
- mathematically tractable, but
- difficult to implement in a centralized manner in large-scale networks
- decentralized, distributed algorithms are necessary!!

Primal algorithm

- Decentralized algorithm to implement solutions to relaxations of the problem *NETWORK(A, C; w)*
- algorithms at the sources and at the links

$$
\begin{aligned}
\dot{x}_r(t) &= \kappa \left(w_r - x_r(t) \sum_{j \in r} \mu_j(t) \right) \\
\mu_j(t) &= p_j \left(\sum_{s: j \in s} x_s(t) \right)
\end{aligned}
$$

- static link law, $p_j(y)$: → steady-state queues!
  - price charged by resource *j* when the total flow trough resource *j* is *y*
  - or rate of continuous stream of feedback signals → congestion indicator
- dynamic sources (differential equation, first order dynamics)
  - source rate is adjusted → to equalize aggregate cost of the flow with a target value (w_r)
  - steady increase at rate proportional to w_r (Additive Increase)
  - multiplicative decrease at rate proportional to the stream of feedback signals

Dual algorithm

- Decentralized algorithm to implement solutions to relaxations of the problem *DUAL(A, C; w)*
- algorithms at the sources and at the links

$$\dot{\mu}_j(t) = \kappa \left(\sum_{r:j\in r} x_r(t) - q_j(\mu_j(t)) \right)$$

$$x_r(t) = \frac{w_r}{\sum_{k\in r} \mu_k(t)}$$

- static source control
  - based on shadow prices
  - simple function
- dynamic links (differential equation, first order dynamics)
  - shadow prices are adjusted gradually
  - $q_j(\eta)$: the flow trough resource *j* that generates a price of η at resource *j*
  - buffers at the links!

Kitekintés (2)

TCP RENO EBBEN A KERETRENDSZERBEN

A model of TCP Reno

- So far: theoretical models with possible implementations
- How should a protocol be designed in an analytically tractable optimization framework?
- Now: How could TCP (Reno) be placed in this framework?
- dynamics of congestion window in Congestion Avoidance phase can be modeled by “similar” differential equation
- utility function (implicitly) applied by TCP Reno can be determined (*reverse engineering*)
- e.g., a continuous-time approximation of TCP at the flow-level can be established

A model of TCP Reno

Notations

$W_r(t)$	:	congestion window size of flow r at time t
T_r	:	round-trip time of flow r (now constant)
$x_r(t) = \dfrac{W_r(t)}{T_r}$	:	transmission rate of flow r at time t
$q_r(t)$	:	fraction of packets lost at time t
$\beta = \frac{1}{2}$	:	multiplicative decrease factor

- dynamics of congestion window:

$$\dot{W}_r(t) = \frac{x_r(t-T_r)(1-q_r(t))}{W_r(t)} - \beta x_r(t-T_r) q_r(t) W_r(t)$$

- first term: Additive Increase
  - *W* is increased by *1/W* for one ACK
  - "positive" ACKs arrived at a rate proportional to
    - $(1-q_r(t))$ → not lost packets
    - $x_r(t-T_r)$ → sending rate at one RTT earlier
- second term: Multiplicative Decrease (cont'd)

A model of TCP Reno

$$\dot{W}_r(t) = \frac{x_r(t-T_r)(1-q_r(t))}{W_r(t)} - \beta x_r(t-T_r)q_r(t)W_r(t)$$

- second term: Multiplicative Decrease
  - *W* is decreased by $\beta W_r(t)$ (for Reno → halving)
  - at a rate proportional to
    - $q_r(t)$ → lost packets (loss ratio)
    - $x_r(t-T_r)$ → sending rate at one RTT earlier
- sending rate can also be modeled (substituting *W* in terms of *x*)

$$\dot{x}_r(t) = \frac{x_r(t-T_r)(1-q_r(t))}{T_r^2 x_r(t)} - \beta x_r(t-T_r)q_r(t)x_r(t)$$

- equilibrium value of *x* at the equilibrium loss probability (*q*)

$$\hat{x}_r = \sqrt{\frac{1-\hat{q}_r}{\beta \hat{q}_r}}\frac{1}{T_r}$$ for small values of *q*: $$\hat{x}_r \propto \frac{1}{T_r\sqrt{\hat{q}_r}}$$

- which is the well-known steady-state model!

A model of TCP Reno

- If $T_r = 0$ → simplified model

$$\begin{aligned}\dot{x}_r(t) &= \frac{1-q_r(t)}{T_r^2} - \beta x_r^2(t) q_r(t) \\ &= \left(\beta x_r^2(t) + \frac{1}{T_r^2}\right)\left(\frac{1}{\beta T_r^2 x_r^2(t)+1} - q_r(t)\right)\end{aligned}$$

- this is similar to the general model
- gain and utility function can be identified

$$U_r(x_r) = \frac{\arctan(x_r T_r \sqrt{\beta})}{\sqrt{\beta} T_r}$$

- further simplification: *q* is very small → *1-q ≈ 1*

$$\dot{x}_r(t) = \frac{1}{T_r^2} - \beta x_r^2(t) q_r(t)$$

- which yields: $U_r(x_r) = -\dfrac{1}{x_r T_r}$

Áttekintés

- TCP példák
  - fontosabb algoritmusok, egy-két illusztráció
- Értsük meg, mit csináltunk
  - matematikai modellek (utólag)
  - probléma megfogalmazása
  - most: optimalizációs feladatként
- TCP javítása ("otthoni kitekintés")
  - TCP problémái
  - új ötletek, algoritmusok
  - most már a matematikai modellek alapján

Drawbacks of TCP (1)

- TCP Reno is *inefficient* in
  - high speed
  - wide area networks
  - together
    - high bandwidth-delay product networks
    - BDP = C * RTT
  - AI is too slow / conservative
  - MD is too harsh
  - very very very ... low loss rate should be necessary...

Drawbacks of TCP (2)

- TCP Reno is *inefficient* in
  - wireless networks
  - all packet losses considered as congestion signals
  - random losses can be occurred on the radio interface (Layer 2)
  - halving the window is a spurious reaction

Drawbacks of TCP (3)

- TCP Reno is *inefficient* in
  - wireless networks
  - handover
    - between base stations (horizontal)
    - or different technologies (vertical)
  - RTO expires → timeout
  - restarting with slow-start...

Drawbacks of TCP (4)

- RTT unfairness
  - AIMD is fair if the flows meet the same RTTs
  - heterogeneous RTTs?
    - the flow with the shorter RTT possesses more BW
    - shorter “update period”
- oscillation (sawtooth)
  - can be a problem
- traffic phase effect
  - synchronized losses
  - global synchronization
- ...

Overview

- Transport protocols, TCP (summary)
- Congestion control
  - in general
  - approaches
- Drawbacks of TCP
- New proposals
  - loss-based versions
  - delay-based versions
  - combined versions
  - measurement-based versions

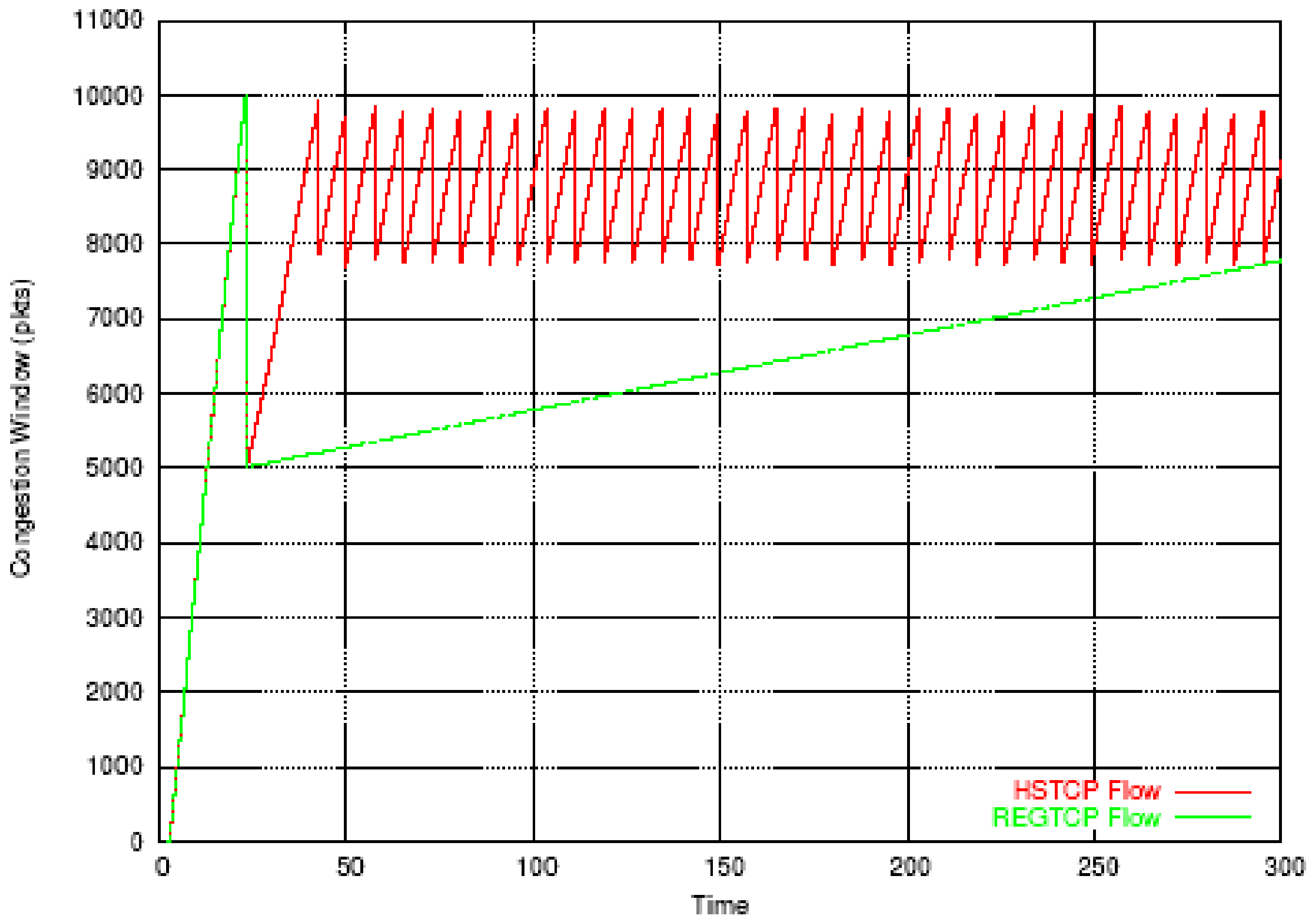
New proposals

- Lot of proposals for high BDP networks
- “high speed TCP protocols”
- loss-based versions
- delay-based versions
- combined versions
- proposals with measurements

HighSpeed TCP (1)

- RFC 3649: HighSpeed TCP for Large Congestion Windows (Sally Floyd, 2003)
- loss-based protocol
- minor changes
- modified AIMD method
  - the increase and decrease factors depend on the current value of the congestion window
  - “scalable” method
  - in case of heavy congestion (high loss rate) → TCP Reno-like behavior
  - otherwise → more aggressive control (when the cwnd is high)

HighSpeed TCP (2)



Scalable TCP (1)

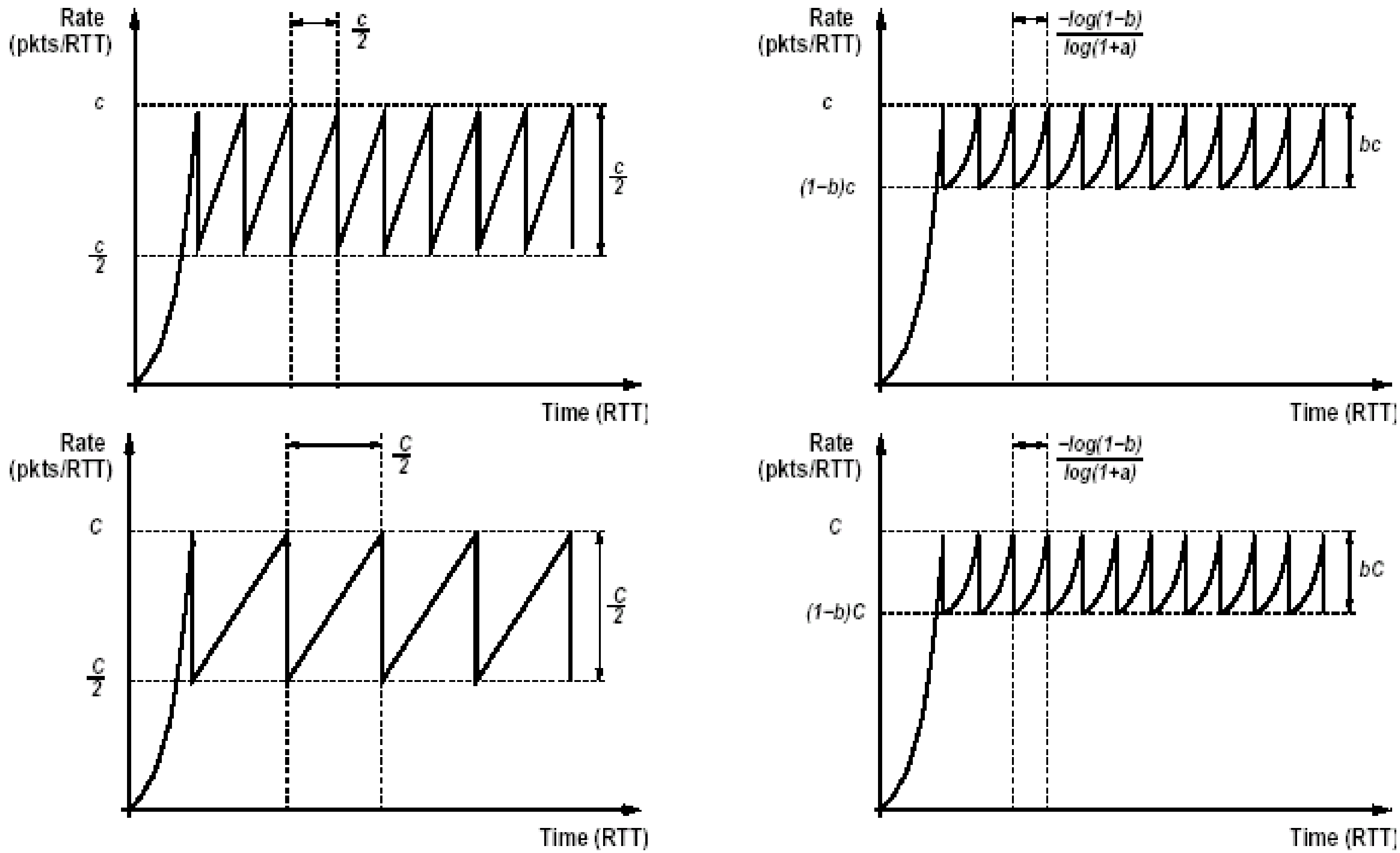
- Proposed by Tom Kelly, 2003
- loss-based protocol
- MIMD mechanism
  - scalable solution
  - probing time does not depend on capacity
  - depends only on RTT
  - aggressive behavior
  - fairness?
  - in case of synchronized losses MIMD can not guarantee the fair behavior
  - large number of flows → no synchronization

Scalable TCP (2)

TCP Reno ↔ Scalable TCP

Small capacity

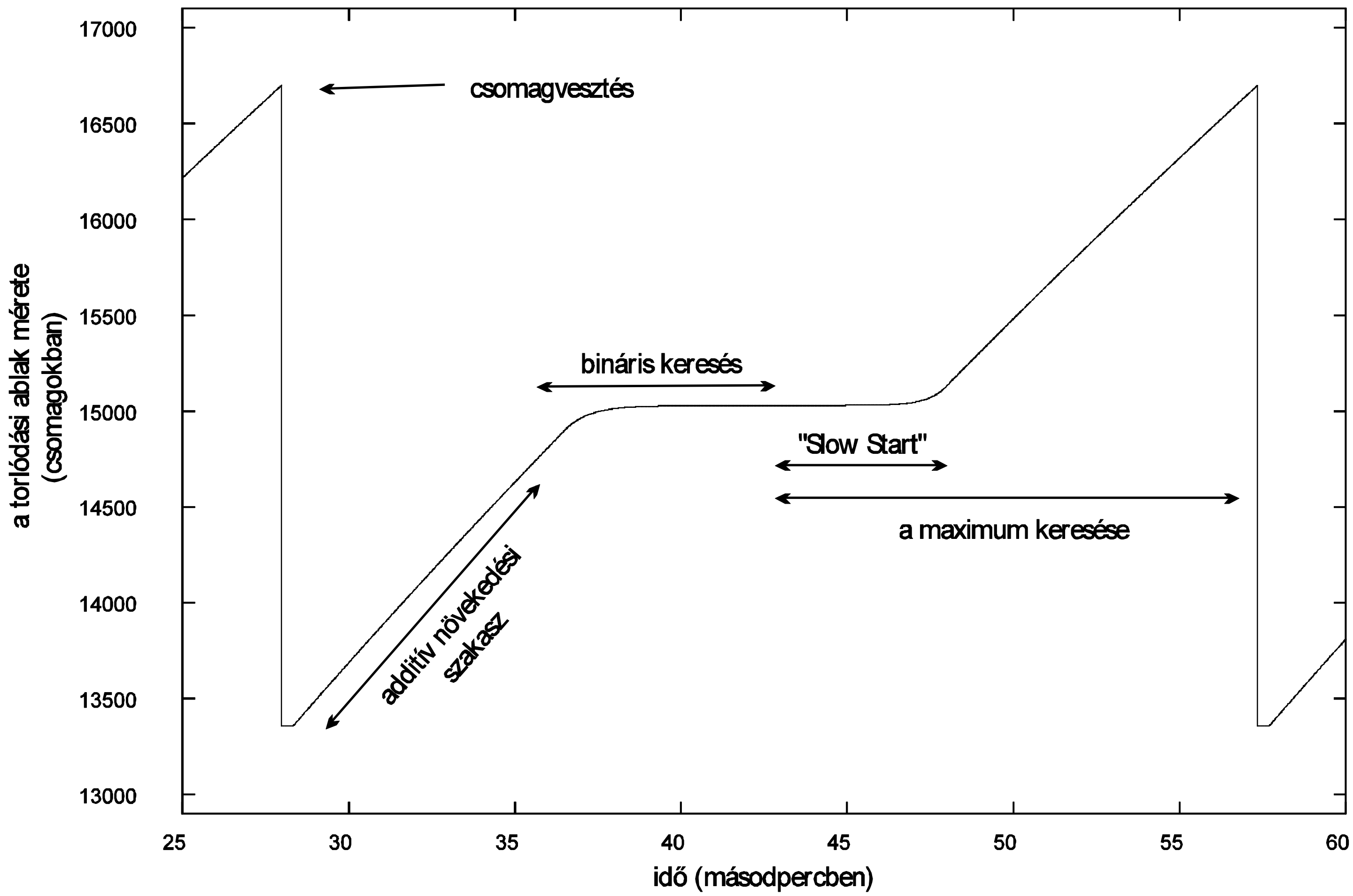
Large capacity



BIC TCP (1)

- Binary Increase TCP
- proposed by I. Rhee et al., 2004
- default TCP protocol of Linux kernels 2.6.7 → 2.6.19
- loss-based protocol
- improve RTT fairness!
- mechanisms
  - binary search increase
  - additive increase
  - slow-start
  - max-probing
  - fast convergence

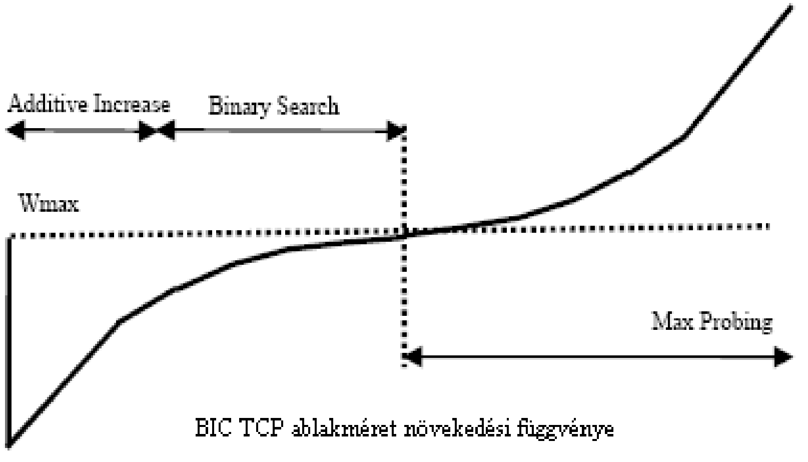
BIC TCP (2)

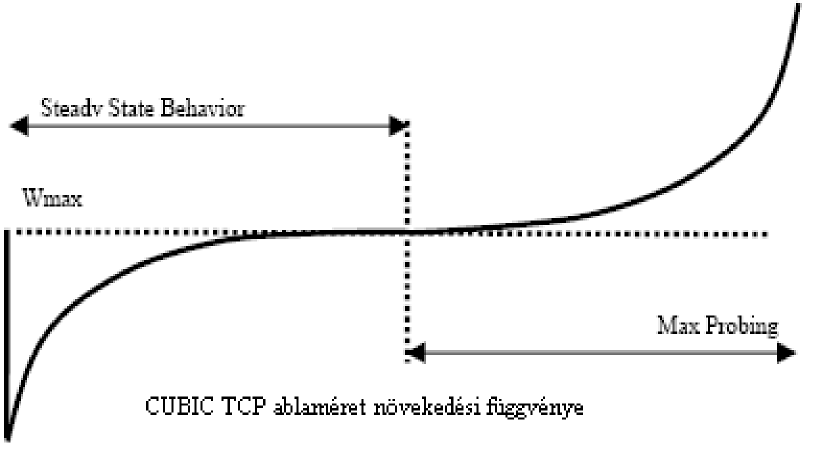


17000
16500
16000
15500
15000
14500
14000
13500
13000
a torlódási ablak mérete (csomagokban)
25
30
35
40
45
50
55
60
idő (másodpercben)
csomagvesztés
bináris keresés
"Slow Start"
a maximum keresése
additív növekedési szakasz

CUBIC

- Refinement of BIC TCP
- proposed by I. Rhee et al., 2005
- default TCP protocol of current Linux kernels
- similar cwnd curves
- the BIC function is approximated by cubic functions





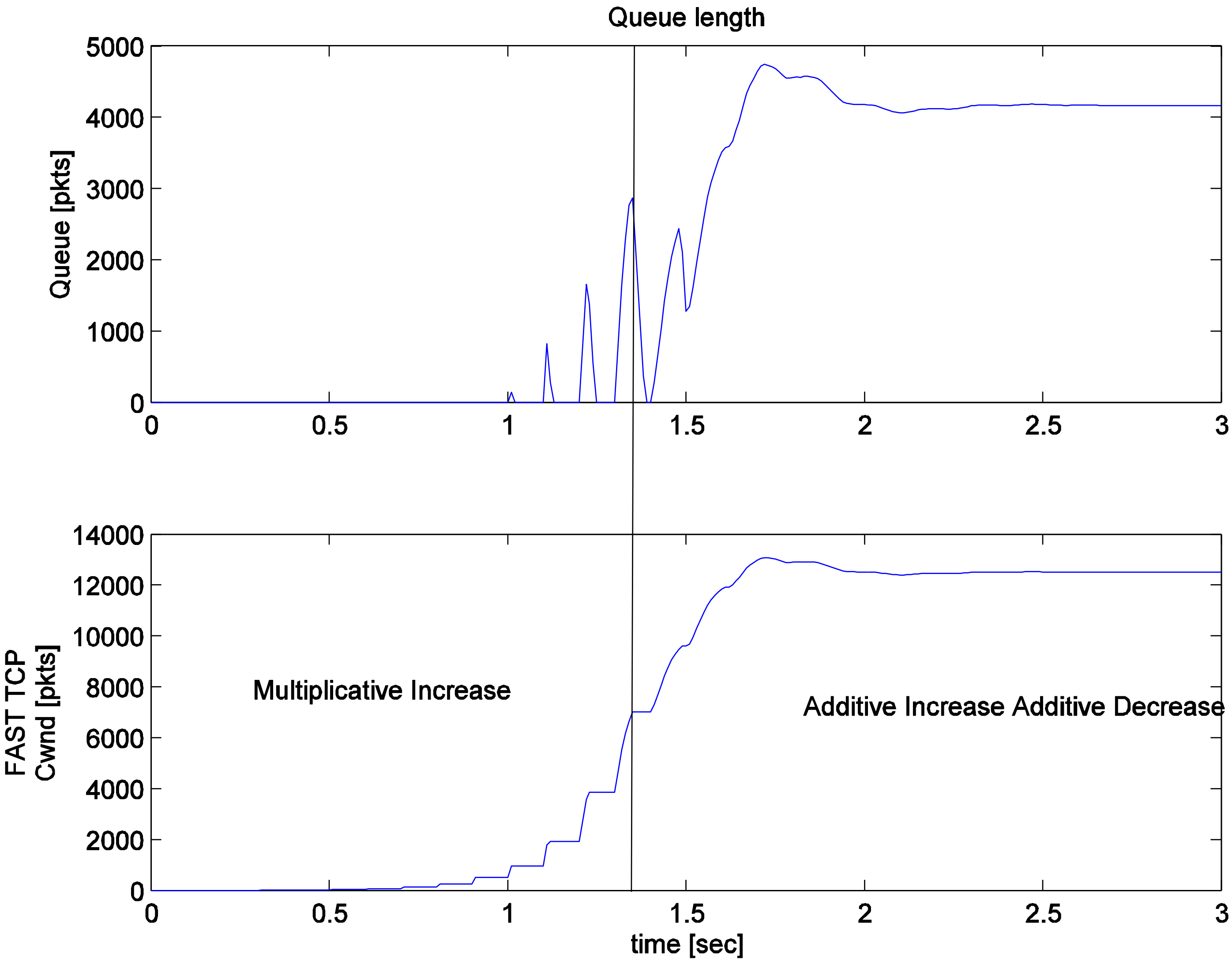
TCP Vegas

- Delay-based protocol (pioneer)
- proposed by Brakmo et al., 1994
- congestion measure
  - queueing delay
  - estimated based on RTT measurements
  - current RTT ↔ base RTT (without queueing)
- control mechanism
  - goal: keep the number of “my” packets in the bottleneck queue between two parameters (alpha, beta)
  - if delay is too small → cwnd++
  - if delay is too high → cwnd--
  - otherwise: nothing to do

FAST TCP (1)

- A fast version of TCP Vegas
- with a lot of extension
- proposed by Low et al., 2004
- now it is a delay/loss-based protocol
- congestion measures
  - delay (fine adjustment)
  - loss (window halving)
- used mechanisms
  - MI (far from the equilibrium)
  - MD (in case of loss event)
  - AIAD (fine cwnd adjustment based on queueing delay)

FAST TCP (2)


Queue length
Queue [pkts]
5000
4000
3000
2000
1000
0
0
0.5
1
1.5
2
2.5
3
FAST TCP
Cwnd [pkts]
14000
12000
10000
8000
6000
4000
2000
0
Multiplicative Increase
Additive Increase Additive Decrease
time [sec]

Compound TCP

- Combined delay/loss-based protocol
- proposed by K. Tan et al., 2006
- TCP protocol of MS Windows Vista and Windows Server 2008
- TCP Reno + TCP Vegas
- goals
  - good utilization in high BDP networks
  - fair behavior with other protocols
  - efficient behavior in case of small buffers

TCP Westwood

- Operation based on accurate bandwidth estimation
- proposed by Wang, Yamada, Sanadidi, Gerla, 2005
- cwnd and ssthresh are set based on eligible rate estimation
- a lot of versions of Westwood
- a lot of estimation methods

Other versions

- TCP Libra (RTT fairness)
- H-TCP (Hamilton TCP)
- LTCP (macroscopic/microscopic control)
- combined delay/loss-based protocols
  - TCP Africa
  - YeAH-TCP
  - TCP-Illinois
  - TCP-Adaptive Reno
- …

Áttekintés

- TCP példák
  - fontosabb algoritmusok, egy-két illusztráció
- Értsük meg, mit csináltunk
  - matematikai modellek (utólag)
  - probléma megfogalmazása
  - most: optimalizációs feladatként
- TCP javítása (“otthoni kitekintés”)
  - TCP problémái
  - új ötletek, algoritmusok
  - most már a matematikai modellek alapján