

# Optimizing the Internet

Daniel P. Palomar

The Hong Kong University of Science and Technology (HKUST)

ELEC5470/IEDA6100A - Convex Optimization

The Hong Kong University of Science and Technology (HKUST)

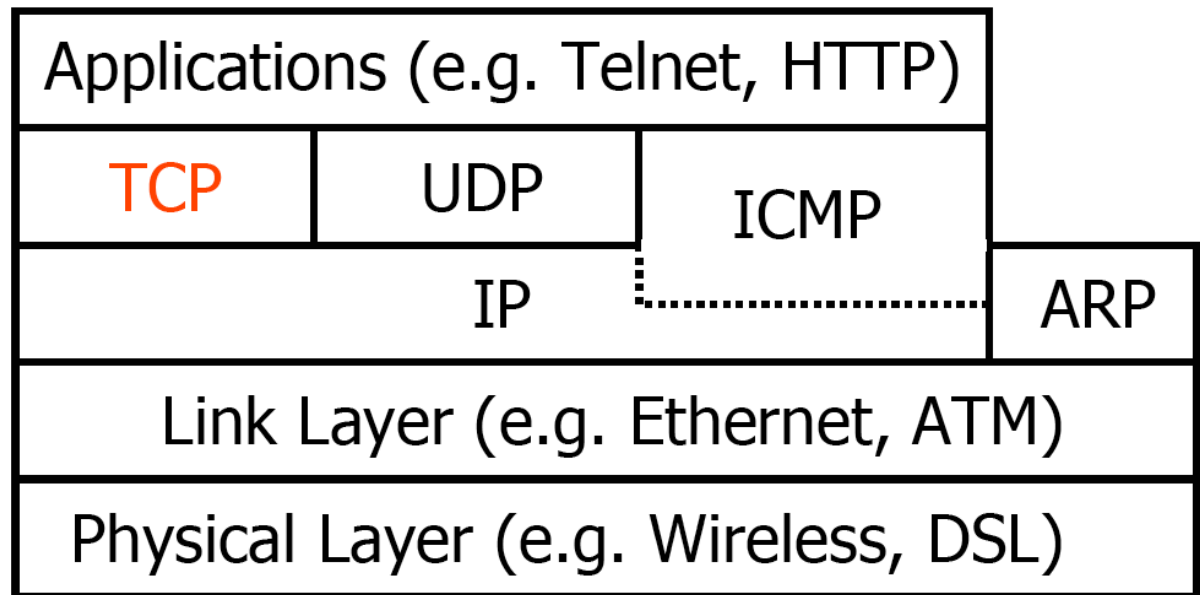
Fall 2020-21

# Outline of Lecture

- Introduction to TCP/IP
- Overview of TCP congestion control
- Steady-state analysis via convex optimization
- Dynamic analysis via convex optimization
- Summary

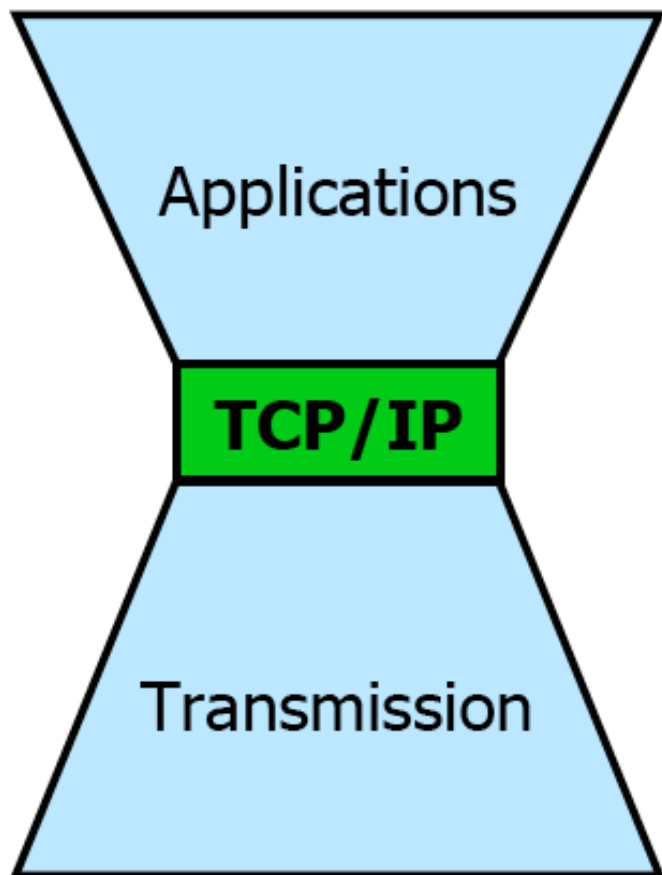
# TCP/IP Protocol Stack

- TCP/IP protocol stack (OSI model):
  - Application layer: ftp, http, etc.
  - Transport layer: adds reliability to network layer. The predominant protocol is TCP (also UDP).
  - Network layer: protocols for routing, IP (Internet Protocol)
  - Data link layer:  
frames over  
the link  
+ error correction
  - Physical layer: bits  
and waveforms



# Success of TCP/IP

WWW, Email, Napster, FTP, ...



Ethernet, ATM, POS, WDM, ...

- TCP/IP has been extremely successful because it is simple and robust:
  - robust against failure
  - robust against technological evolution
  - provides a service to applications (doesn't tell applications what to do)

# TCP

- TCP (Transmission Control Protocol) was introduced in the 1970s for file transfer in the Internet.
- TCP protocol:
  - end-to-end control
  - session initiation and termination
  - in-order recovery of packets
  - flow control/congestion control
  - ...
- Why congestion control?

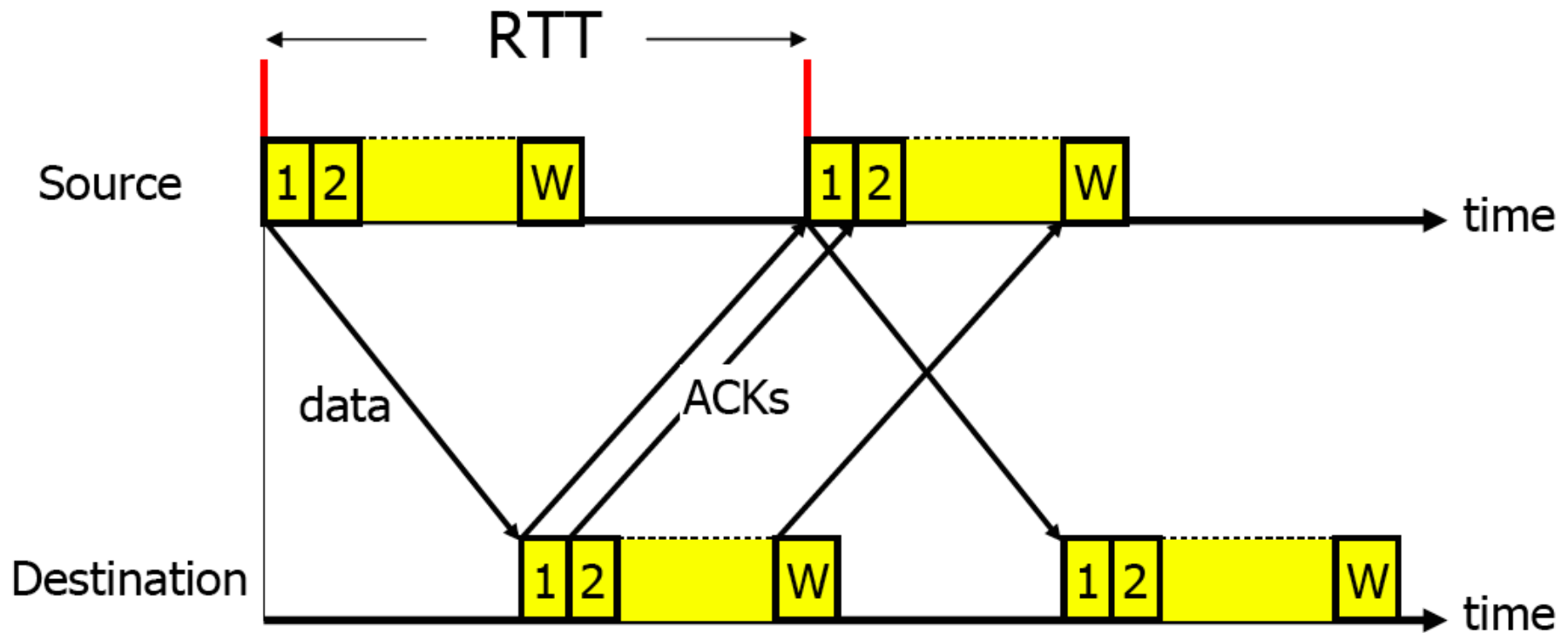
# TCP (II)

- Initially TCP had little to control congestion in the network:
  - If several users started transferring files over a bottleneck link exceeding its capacity, then the packets had to be dropped. This resulted in retransmission of lost packets, making things worse.
  - This phenomenon is known as congestion collapse and occurred many times in the mid-1980s.
- In Oct. 1986, the Internet had its first collapse: throughput dropped by a factor of 1000 from 32 kbps to 10 bps.
- In 1988, Jacobson proposed a congestion control mechanism for TCP, which has been a remarkably successful algorithm for the growth of the Internet.

# TCP Congestion Control

- TCP congestion control:
  - window-based end-to-end flow control, where the destination sends ACK for correctly received packets and the source updates the window size (which is proportional to allowed transmission rate).
  - several instances of TCP congestion control distributively dissolve congestion in bottleneck links by reducing window sizes.
  - sources update window sizes and links update (sometimes implicitly) congestion measures that are fed back to sources using the link.

# Window Flow Control



- $W$  packets per Round-Trip Time (RTT).
- Lost packet detected by missing ACK.



# Evolution of TCP Congestion Control

- With the growth of the Internet over the past decade by several orders of magnitude, there was a need to develop more scalable mechanisms for Internet congestion control (good behavior unaffected by number of nodes, capacities of links, RTT, etc.)
- TCP congestion control algorithms for the Internet were devised based on heuristics, common sense, and trial-and-error in the 1980s-1990s. There was no solid understanding of the convergence and stability properties of the algorithms.
- Mathematical modeling of congestion control based on convex optimization initiated by Kelly and Low in the mid-1990s.

# Cronology of TCP Congestion Control

- Tahoe (Jacobson 1988): slow start, congestion avoidance, fast retransmit.
- Reno (Jacobson 1990): fast recovery.
- Vegas (Brakmo & Peterson 1994): new congestion avoidance.
- RED (Floyd & Jacobson 1993): probabilistic marking.
- REM (Athuraliya & Low 2000): clear buffer, match rate.
- etc.

# NUM Perspective

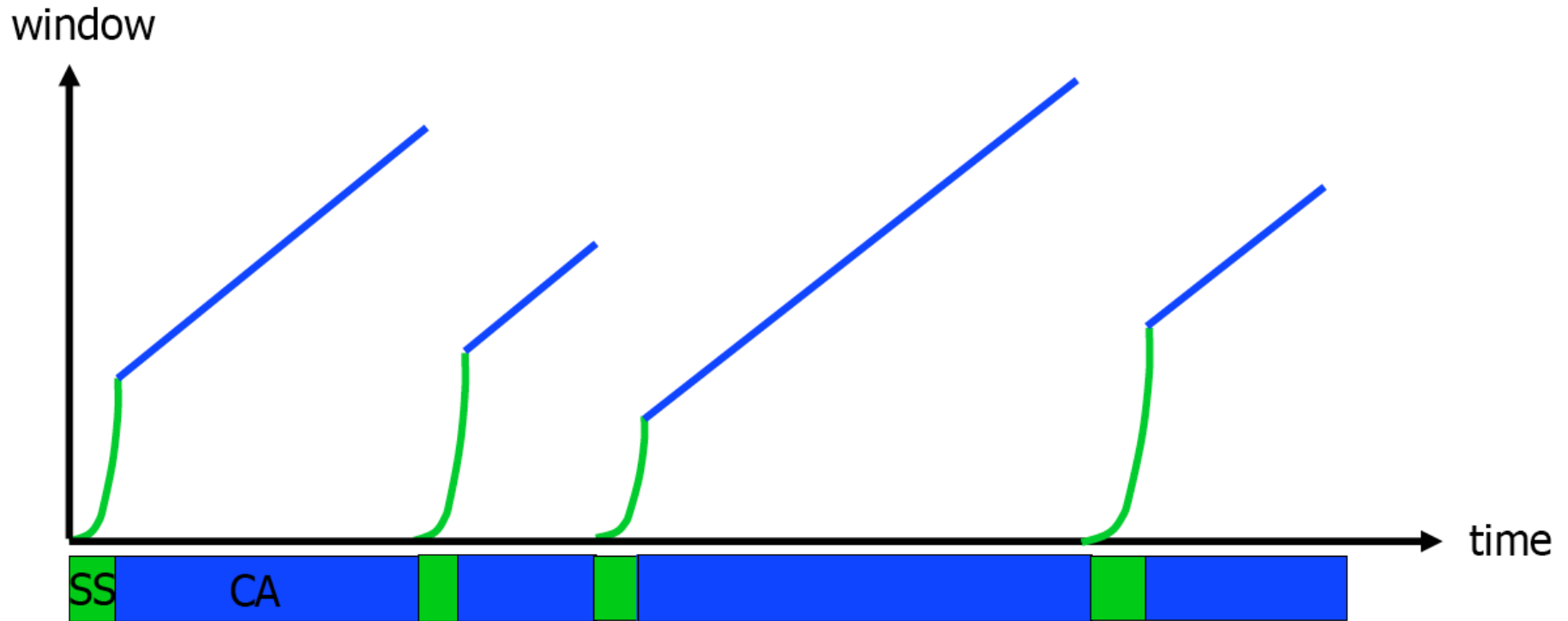
- Optimization-theoretic perspective of TCP congestion control:
  - a TCP congestion control algorithm carries out a distributed algorithm to solve an implicit global convex optimization problem, a **Network Utility Maximization (NUM)**
  - source rates are **primal variables** updated at the sources
  - congestion measures are **dual variables** (prices) updated at the links.

# Overview of TCP Congestion Control

- TCP uses a window-based flow control to pace the transmission of packets.
- Each source maintains a “window size” variable that limits the maximum number of packets that can be unacknowledged.
- Two features:
  1. the algorithm is “self-clocking”: TCP automatically slows down the source when the network becomes congested.
  2. the window size variable determines the source rate: one window is worth the packets sent every roundtrip time.

# TCP Tahoe

- TCP Tahoe (Jacobson 1988):



SS: Slow Start  
CA: Congestion Avoidance

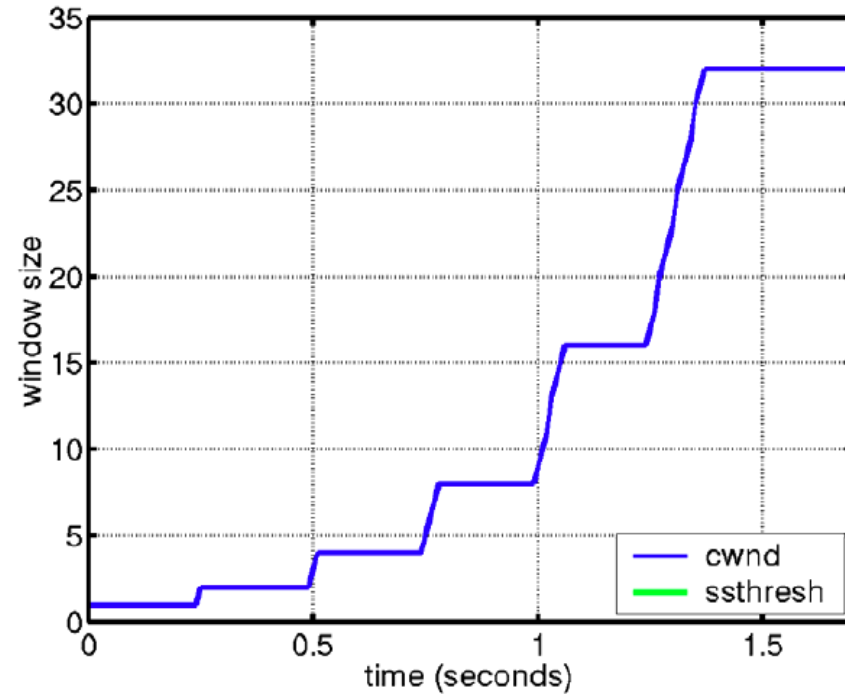
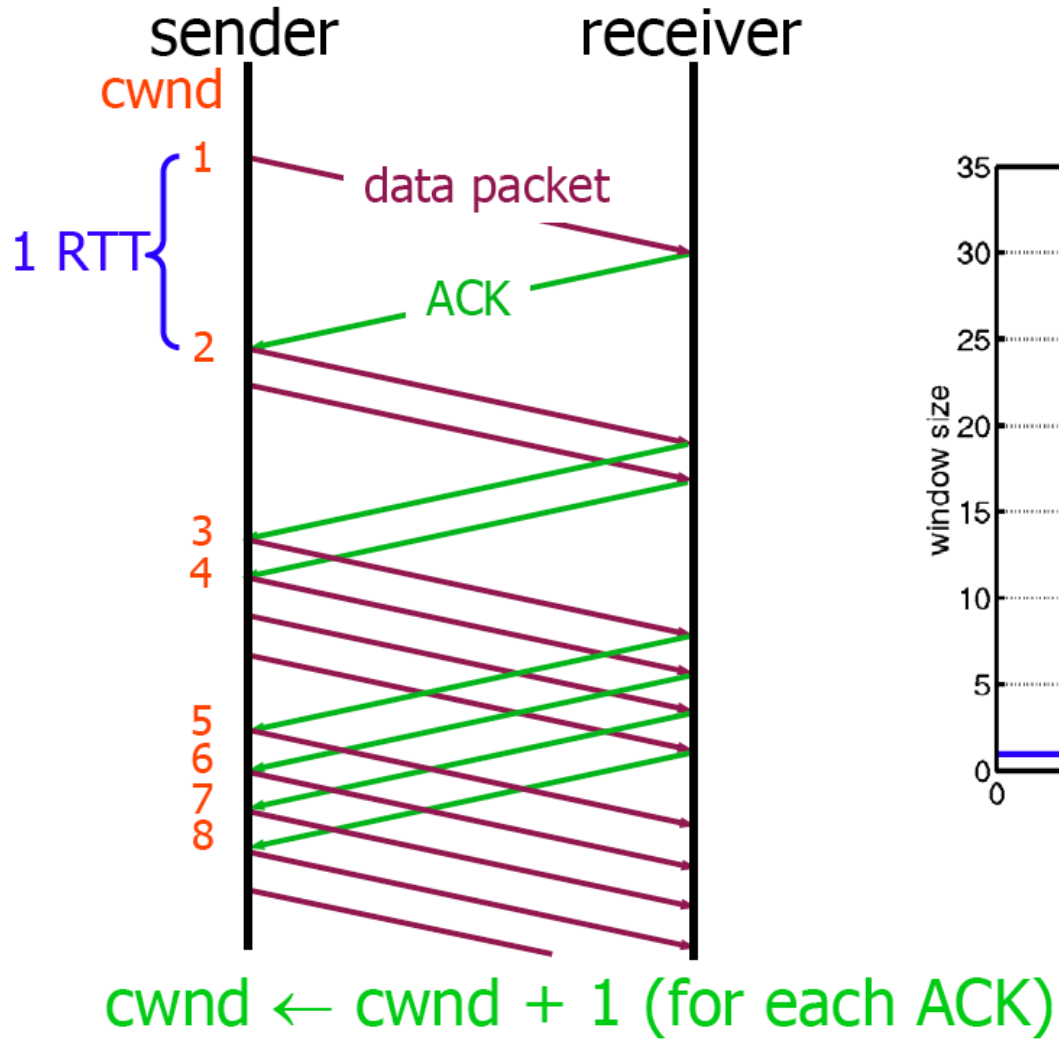
# TCP Tahoe: Algorithm

- TCP Tahoe (relies on packet loss):

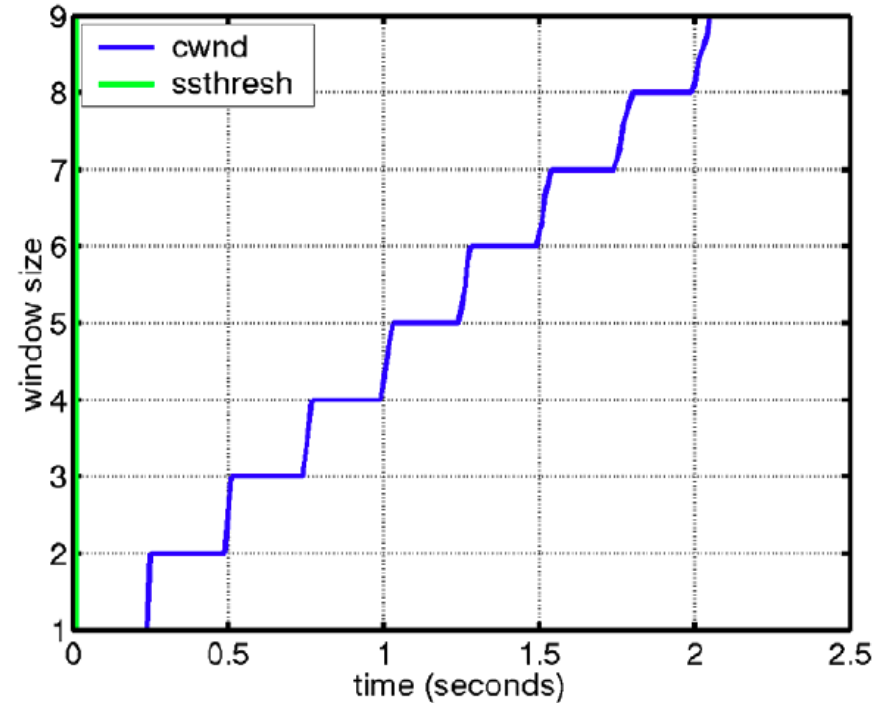
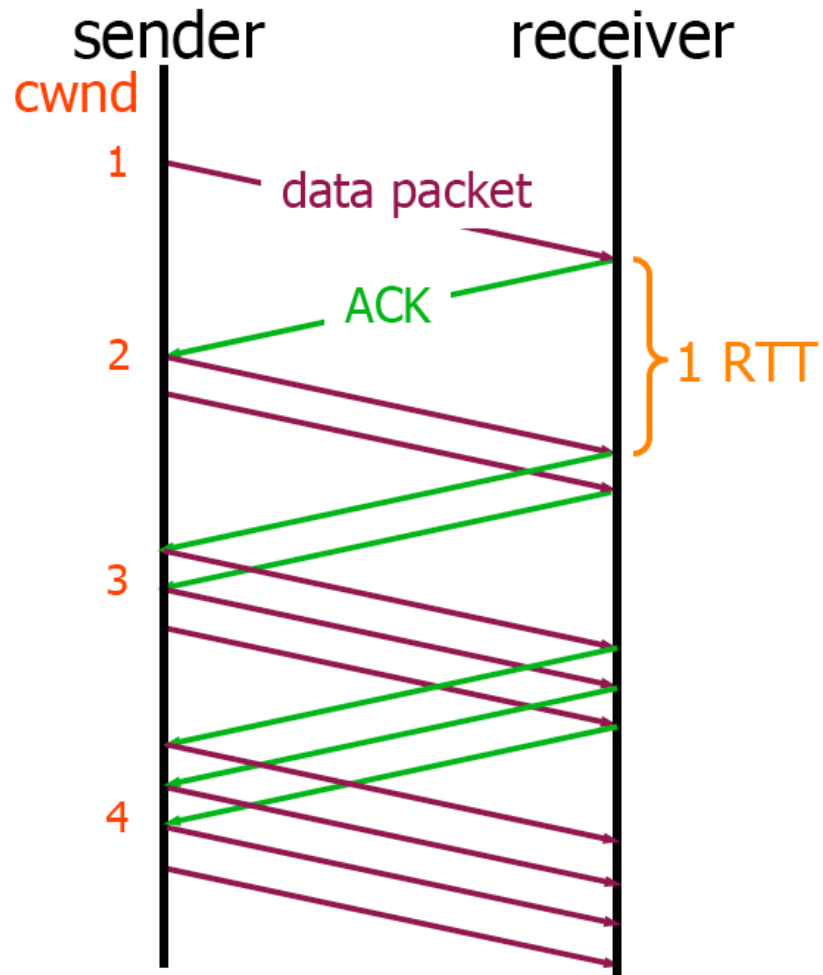
**Slow-start phase:** start with a window of  $\text{cwnd} = 1$ , increase the window size by 1 for every ACK received  $\text{cwnd} = \text{cwnd} + 1$  (this doubles the window every RTT: exponential growth). If window reaches a threshold,  $\text{cwnd} \geq \text{ssthresh}$ , enter next phase.

**Congestion avoidance phase:** increase the window by its reciprocal,  $\text{cwnd} = \text{cwnd} + 1/\text{cwnd}$ , for every ACK received (this increases the window by one every RTT: linear growth).

# TCP Tahoe: Slow Start



# TCP Tahoe: Congestion Avoidance

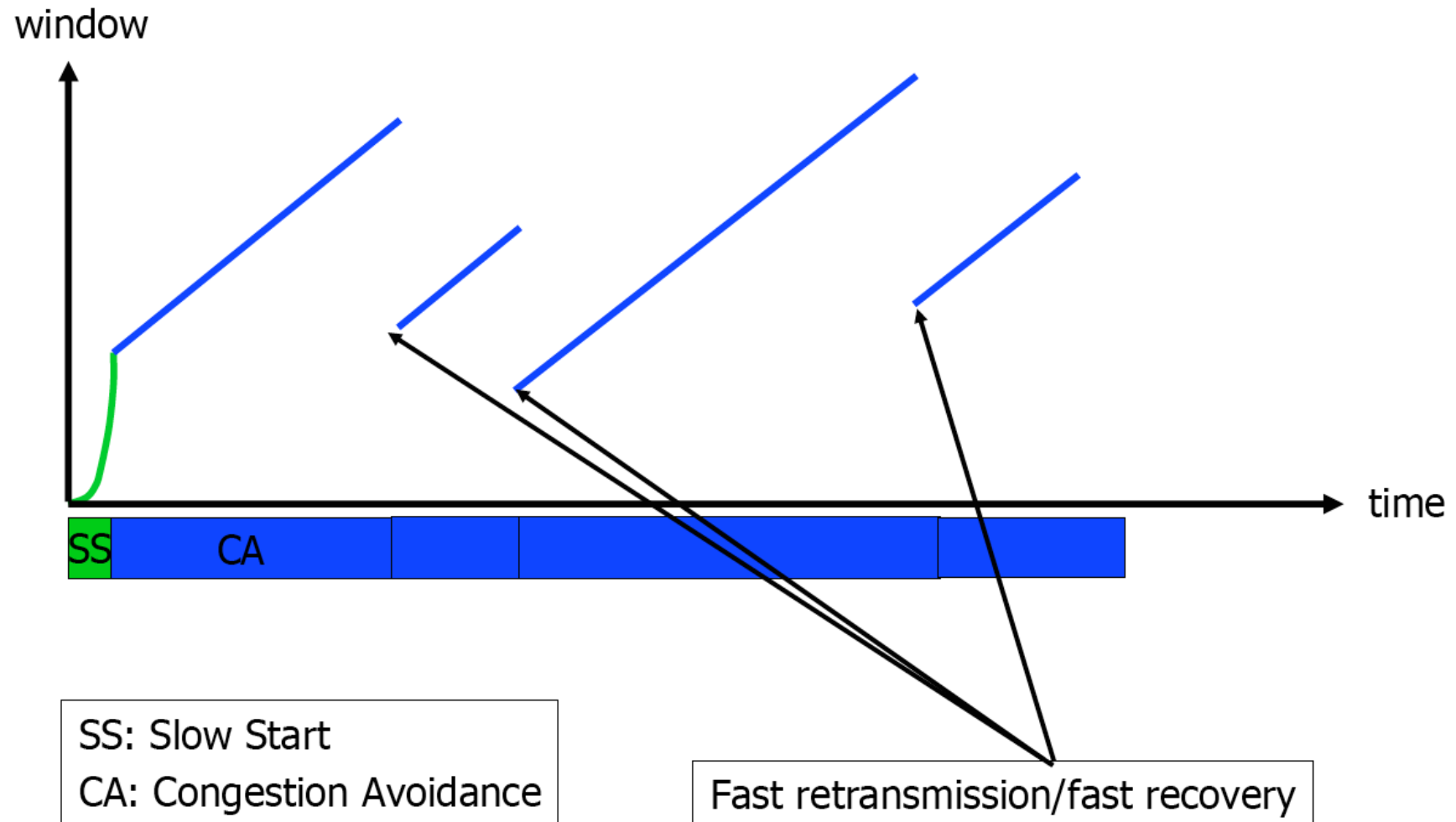


$cwnd \leftarrow cwnd + 1$  (for each cwnd ACKS)



# TCP Reno

- TCP Reno (Jacobson 1990):



# TCP Reno: Algorithm

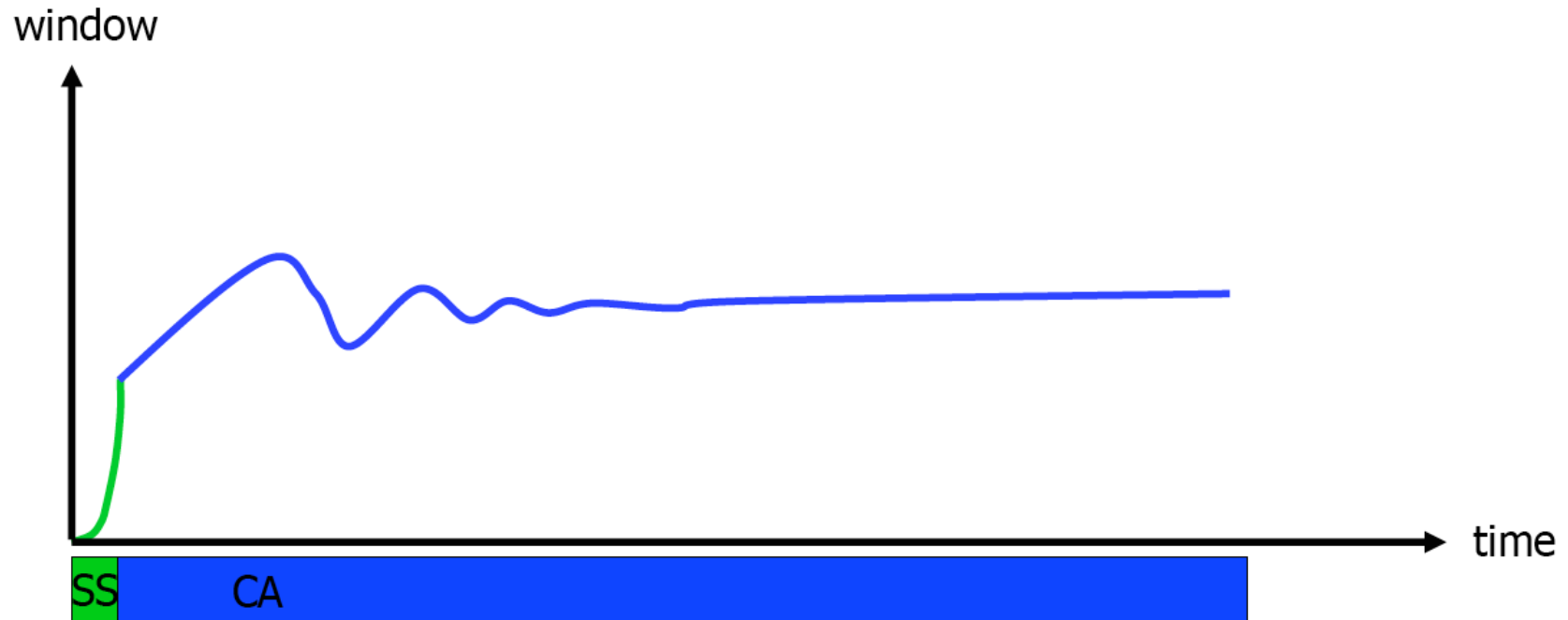
- TCP Reno (relies on packet loss):

**Slow-start phase:** start with a window of 1, increase the window size by 1 for every ACK received (this doubles the window every RTT: exponential growth). If window reaches a threshold enter next phase. If a packet loss is detected, halve threshold and set window to 1.

**Congestion avoidance phase:** increase the window by its reciprocal for every ACK received (this increases the window by one every RTT: linear growth). If a packet loss is detected, halve threshold, set window to some given value, and enter slow-start phase again.

# TCP Vegas

- TCP Vegas (Brakmo & Peterson 1994)



- Converges, no retransmission
- ... provided buffer is large enough

# TCP Vegas: Algorithm

- Window update for TCP Vegas (relies on queuing delay) in the congestion avoidance phase:

$$w_s(t+1) = \begin{cases} w_s(t) + \frac{1}{D_s(t)} & \text{if } \frac{w_s(t)}{d_s} - \frac{w_s(t)}{D_s(t)} < \alpha_s \\ w_s(t) - \frac{1}{D_s(t)} & \text{if } \frac{w_s(t)}{d_s} - \frac{w_s(t)}{D_s(t)} > \alpha_s \\ w_s(t) & \text{else} \end{cases}$$

where

- $D_s(t)$  is the RTT (Round Trip Time)
- $d_s$  is the propagation time (minimum  $D_s(t)$ )
- $w_s(t)/d_s$  is the expected rate
- $w_s(t)/D_s(t)$  is the actual rate

## TCP Vegas: Algorithm (II)

- Observe that
  - the difference  $\frac{w_s(t)}{d_s} - \frac{w_s(t)}{D_s(t)}$  should be kept between  $\alpha_s$  and  $\beta_s$  (for simplicity we will assume  $\alpha_s = \beta_s$ )
  - $w_s(t) - d_s x_s(t) =$  total backlog buffered in the path of  $s$  ( $x_s(t) = w_s(t)/D_s(t)$ ).
- Therefore, we can think of a source incrementing/decrementing its window according to whether the total backlog is smaller/larger than  $\alpha_s d_s$ .

# TCP Vegas: Algorithm (II)

- We will now show that:
  1. Steady-state analysis: the objective of TCP Vegas is to maximize the aggregate source utility subject to capacity constraints of network resources.
  2. Dynamic analysis: the TCP Vegas algorithm is a dual method to solve the maximization problem.

# Steady-State Analysis (I)

- When the algorithm converges, the equilibrium windows  $w_s^*$  and the associated equilibrium round-trip times  $D_s^*$  satisfy:

$$\frac{w_s^*}{d_s} - \frac{w_s^*}{D_s^*} = \alpha_s \quad \text{for all } s.$$

- Consider now the following NUM:

$$\begin{array}{ll} \underset{\mathbf{x} \geq 0}{\text{maximize}} & \sum_s U_s(x_s) \\ \text{subject to} & \sum_{s:l \in \mathcal{L}(s)} x_s \leq c_l \quad \forall l \end{array}$$

with utilities:  $U_s(x_s) = \alpha_s d_s \log x_s$ .

## Steady-State Analysis (II)

- Since the NUM problem is convex, we know that a rate vector  $\mathbf{x}^*$  is optimal if and only if we can find Lagrange multipliers so that the KKT conditions are satisfied.
- The Lagrangian and its gradient are

$$L(\mathbf{x}, \boldsymbol{\lambda}) = \sum_s U_s(x_s) + \sum_l \lambda_l \left( c_l - \sum_{s:l \in \mathcal{L}(s)} x_s \right)$$

$$\nabla_{x_s} L = U'_s(x_s) - \sum_{l \in \mathcal{L}(s)} \lambda_l = \frac{\alpha_s d_s}{x_s} - \sum_{l \in \mathcal{L}(s)} \lambda_l$$



## Steady-State Analysis (III)

- KKT conditions:

$$\sum_s x_s \leq c_l, \quad \lambda_s \geq 0$$

$$\frac{\alpha_s d_s}{x_s} = \sum_{l \in \mathcal{L}(s)} \lambda_l$$

$$\lambda_s \left( \sum_s x_s - c_l \right) = 0$$

- Let's verify that, indeed, the equilibrium point satisfies the KKT conditions for some  $\lambda$ .

## Steady-State Analysis (IV)

- The rate at equilibrium is  $x_s^* = \frac{w_s^*}{D_s^*}$ .
- Let  $b_l^*$  be the equilibrium backlog at link  $l$ . The fraction of  $b_l^*$  that belongs to source  $s$  is  $b_l^* \frac{x_s^*}{c_l}$  where  $c_l$  is the link capacity.
- Hence, the source  $s$  maintains a backlog of  $\sum_{l \in \mathcal{L}(s)} b_l^* \frac{x_s^*}{c_l}$  along its path in equilibrium.
- On the other hand, recall that the expression for the total backlog buffered along the path of source  $s$  is  $w_s(t) - d_s x_s(t)$ .

## Steady-State Analysis (V)

- Thus, we can write

$$w_s^* - d_s x_s^* = \sum_{l \in \mathcal{L}(s)} b_l^* \frac{x_s^*}{c_l}$$

and then

$$\alpha_s = \frac{w_s^*}{d_s} - \frac{w_s^*}{D_s^*} = \frac{1}{d_s} (w_s^* - d_s x_s^*) = \frac{1}{d_s} \left( \sum_{l \in \mathcal{L}(s)} b_l^* \frac{x_s^*}{c_l} \right).$$

- If we now denote  $\lambda_l^* = b_l^*/c_l$ , we can rewrite the above as

$$\frac{\alpha_s d_s}{x_s^*} = \sum_{l \in \mathcal{L}(s)} \lambda_l^*$$

which is one of the KKT conditions.

## Steady-State Analysis (VI)

- Regarding the primal and dual feasibility KKT conditions, clearly  $\lambda_l^* \geq 0$  (by definition) and  $\sum_s x_s^* \leq c_l$  (otherwise we would be magically transmitting at a rate higher than capacity).
- The only remaining condition to verify is the complementary slackness. Since the backlog  $b_l^* = 0$  at link  $l$  if the aggregate rate is strictly less than the capacity (implying  $\lambda_l^* = 0$ ), we have

$$\lambda_s^* \left( \sum_s x_s - c_l \right) = 0.$$

## Steady-State Analysis (VII)

- Summarizing the steady-state analysis, we have seen that an equilibrium point of TCP Vegas satisfies the KKT conditions and, therefore, is an optimal solution of the following NUM:

$$\begin{array}{ll} \underset{\mathbf{x} \geq 0}{\text{maximize}} & \sum_s U_s(x_s) \\ \text{subject to} & \sum_{s:l \in \mathcal{L}(s)} x_s \leq c_l \quad \forall l \end{array}$$

with utilities  $U_s(x_s) = \alpha_s d_s \log x_s$ .

- The following analysis will deal not just with the final equilibrium point of the algorithm but with the dynamics of the updates in the TCP Vegas algorithm.

# Dynamic Analysis (I)

- Let's now derive the dual algorithm to solve the NUM.
- The Lagrangian is

$$\begin{aligned} L(\mathbf{x}, \boldsymbol{\lambda}) &= \sum_s U_s(x_s) + \sum_l \lambda_l \left( c_l - \sum_{s:l \in \mathcal{L}(s)} x_s \right) \\ &= \sum_s \left( U_s(x_s) - x_s \sum_{l \in \mathcal{L}(s)} \lambda_l \right) + \sum_l \lambda_l c_l \end{aligned}$$

where  $\lambda^s = \sum_{l \in \mathcal{L}(s)} \lambda_l$ .

## Dynamic Analysis (II)

- The dual function is then

$$g(\boldsymbol{\lambda}) = \max_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}) = \sum_s \max_{x_s} [U_s(x_s) - x_s \lambda^s] + \sum_l \lambda_l c_l$$

- Therefore, the dual method consists of the master problem:

$$\text{minimize}_{\boldsymbol{\lambda} \geq 0} \quad \sum_s g_s^*(\lambda^s) + \sum_l \lambda_l c_l$$

and the subproblems

$$g_s^*(\lambda^s) = \max_{x_s} U_s(x_s) - x_s \lambda^s \quad \text{for all } s.$$

## Dynamic Analysis (III)

- A simple way to solve the master problem is with a gradient/subgradient projection method:

$$\lambda_l(t+1) = [\lambda_l(t) - \gamma\theta_l \nabla_{\lambda_l} g(\boldsymbol{\lambda}(t))]^+$$

where  $\gamma\theta_l$  denotes the stepsize for the  $l$ th element,  $[\cdot]^+ = \max(0, \cdot)$ , and the gradient/subgradient of the dual function is

$$\nabla_{\lambda_l} g(\boldsymbol{\lambda}(t)) = c_l - \sum_{s:l \in \mathcal{L}(s)} x_s^*(\lambda^s).$$



## Dynamic Analysis (IV)

- The solution to the subproblems for the particular choice of the utilities  $U_s(x_s) = \alpha_s d_s \log x_s$  is

$$x_s^*(\lambda^s) = \frac{\alpha_s d_s}{\lambda^s}.$$

- The gradient update can then be written as

$$\lambda_l(t+1) = \left[ \lambda_l(t) + \gamma \theta_l \left( \sum_{s:l \in \mathcal{L}(s)} \frac{\alpha_s d_s}{\lambda^s} - c_l \right) \right]^+.$$

- For sufficiently small  $\gamma$  the algorithm will converge to a primal-dual optimal point  $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ .

## Dynamic Analysis (V)

- Let's now go back to the TCP Vegas algorithm.
- The evolution of the buffer occupancy  $b_l(t)$  at link  $l$  is

$$b_l(t + 1) = \left[ b_l(t) + \sum_{s:l \in \mathcal{L}(s)} x_s(t) - c_l \right]^+$$

or

$$\frac{b_l(t + 1)}{c_l} = \left[ \frac{b_l(t)}{c_l} + \frac{1}{c_l} \left( \sum_{s:l \in \mathcal{L}(s)} x_s(t) - c_l \right) \right]^+ .$$

## Dynamic Analysis (VI)

- Recalling that we had interpreted  $b_l(t)/c_l$  as the Lagrange multiplier  $\lambda_l(t)$ , we can rewrite the previous expression as

$$\lambda_l(t+1) = \left[ \lambda_l(t) + \frac{1}{c_l} \left( \sum_{s:l \in \mathcal{L}(s)} x_s(t) - c_l \right) \right]^+$$

which is exactly the gradient update that we previously derived for the master dual problem:

$$\lambda_l(t+1) = \left[ \lambda_l(t) + \gamma \theta_l \left( \sum_{s:l \in \mathcal{L}(s)} x_s^*(t) - c_l \right) \right]^+$$

with stepsize  $\gamma = 1$  and scaling factor  $\theta_l = 1/c_l$ .

## Dynamic Analysis (VII)

- Thus, the dual update based on the gradient projection algorithm coincides with the TCP Vegas algorithm with the difference that the source rates  $x_s(t)$  are updated differently:
  - in the dual method, we use the optimum (one-shot update)

$$x_s^*(t) = \frac{\alpha_s d_s}{\lambda^s(t)}$$

- in TCP Vegas algorithm, the window  $w_s(t)$  is updated based on whether

$$w_s(t) - x_s(t)d_s < \alpha_s d_s \quad \text{or} \quad w_s(t) - x_s(t)d_s > \alpha_s d_s.$$

## Dynamic Analysis (VIII)

- Interestingly, recalling how this quantity is related to the backlog:

$$w_s(t) - x_s(t)d_s = \sum_l \frac{x_s(t)}{c_l} b_l(t) = x_s(t) \sum_l \lambda_l(t) = x_s(t) \lambda^s(t),$$

the conditions for the update become

$$x_s(t) < \frac{\alpha_s d_s}{\lambda^s(t)} \quad \text{or} \quad x_s(t) > \frac{\alpha_s d_s}{\lambda^s(t)}.$$

- Observe now that, at equilibrium, the previous TCP Vegas update will imply  $x_s(t) = \frac{\alpha_s d_s}{\lambda^s(t)}$ , which coincides with the one-shot update of the dual-based algorithm  $x_s^*(t) = \frac{\alpha_s d_s}{\lambda^s(t)}$ .

# Different Congestion Control and Utilities

- We have seen how TCP Vegas algorithm can be interpreted as an approximate dual-based gradient method corresponding to a NUM with a particular choice of utilities.
- TCP Reno:
  - source utility:  $\arctan$
  - link price: packet loss
- TCP Vegas
  - source utility: weighted log
  - link price: queuing delay

# Summary (I)

- TCP Reno/Vegas algorithms for the Internet were devised based on heuristics, common sense, and trial-and-error in the 1980s-1990s. There was no solid understanding of the convergence and stability properties of the algorithms.
- More recently, in the early 21st century, it was realized that one can reinterpret those algorithms as approximate dual-based gradient methods solving implicitly a NUM problem.

## Summary (II)

- In particular,
  - a careful inspection of the KKT conditions shows that a steady-state solution of the TCP Vegas algorithm solves a NUM with logarithmic utilities
  - the window update in TCP Vegas is in fact an approximation of a dual-based gradient projection method solving the NUM.
- Convex optimization has been used to reverse-engineer and understand existing protocols.
- Even more than that, convex optimization is being used to devise better protocols for the Internet.



# References

- Steven H. Low and David E. Lapsley, “Optimization Flow Control, I: Basic Algorithm and Convergence,” *IEEE/ACM Trans. on Networking*, vol. 7, no. 6, Dec. 1999.
- Steven H. Low, Larry L. Peterson, and Liming Wang, “Understanding Vegas: a duality model,” *Journal of the ACM*, vol. 49, no. 2, March 2002.
- Steven H. Low, Fernando Paganini, and John C. Doyle, “Internet Congestion Control,” *IEEE Control Systems Magazine*, Feb. 2002.
- Steven H. Low, “A Duality Model of TCP and Queue Management Algorithms,” *IEEE/ACM Trans. on Networking*, Oct. 2003.