Internet Congestion Control By Network Utility Maximization

Chee Wei Tan

Convex Optimization and its Applications to Computer Science

Outline

- Internet Congestion Control
- TCP Tahoe, TCP Reno and TCP Vegas
- Network Utility Maximization
- FAST TCP

TCP Congestion Control

- Window-based end-to-end flow control, where destination sends ACK for correctly received packets and source updates window size (which is proportional to allowed transmission rate)

- Several versions of TCP mitigate congestion by reducing window sizes. Sources update window sizes and links update (sometimes implicitly) congestion measures that are feed back to sources using the link

Optimization Model: TCP congestion control use distributed algorithms to solve an implicit network utility maximization, where source rates are primal variables updated at sources, and congestion measures are dual variables (shadow prices) updated at links

Why Congestion Control

Oct. 1986, Internet had its first congestion collapse (LBL to UC Berkeley)

- 400 yards, 3 hops, 32 kbps
- throughput dropped by a factor of 1000 to 40 bps

1988, Van Jacobson proposed TCP congestion control

- Window based with ACK mechanism
- End-to-end principle of Internet

Window-based Congestion Control

Limit number of packets in network to window size W

$$\text{Source rate allowed (bps)} = \frac{W \times \text{Message Size}}{\text{RTT}}$$

Too small W: under-utilization of link capacities

Too large W: link congestion occurs

Effects of congestion:

- Packet loss
- Retransmission and reduced throughput
- Congestion may continue after the overload

Basics of Congestion Control

- Goals: achieve high utilization without congestion or unfair sharing
- Receiver control (awnd): set by receiver to avoid overloading receiver buffer
- Network control (cwnd): set by sender to avoid overloading network
- $W = \min(\text{cwnd}, \text{awnd})$
- Congestion window cwnd usually the bottleneck

TCP Tahoe and Reno

- Slow start: start with window size 1 and doubles window size every RTT (increment window size by 1 per ACK)
- Congestion avoidance: pass a threshold, increase window by 1 every RTT (additive increase)
- Detecting a loss (Timeout or three duplicated ACK), retransmit loss packet, back to slow start, cut threshold by half
- Fast recovery in TCP Reno: between detecting loss and receiving ACK for retransmitted packet, temporarily increase window by 1 on receiving each duplicated ACK. After receiving ACK for retransmitted packet, cut window size by half and enters congestion avoidance directly

TCP Vegas

- Corrects oscillatory behavior of TCP Reno
- Obtains queuing delay by subtracting measured RTT from estimated propagation delay
- Sets rate proportional to ratio of propagation delay to queuing delay, proportionality constant between α and β
- Increase window if difference between queuing delay and propagation delay is smaller than α, decrease window if it is larger than β, remain constant otherwise

Queue Buffer Processes

At intermediate links:

- FIFO buffer process updates queuing delay as measure of congestion for Vegas and feeds back to sources
- Drop tail updates packet loss as measure of congestion for Reno and feeds back to sources
- RED: instead of dropping only at full buffer, drops packets with a probability that increases with (exponentially weighted) average queue length (example of Active Queue Management)

Analytic Models

Communication network with L links, each with fixed capacity c_l packets per second, shared by S sources, each using a set L_s of links

R: 0-1 routing matrix with $R_{ls} = 1$ iff $l \in L_s$

Deterministic flow model: $x_s(t)$ at each source s at discrete time t

Aggregate flow on link l:

$$y_l(t) = \sum_i R_{ls} x_s(t - \tau_{ls}^f)$$

where τ_{ls}^f is forward transmission delay

Each link updates congestion measure (shadow price) $p_l(t)$. Each source has access to aggregate price along its route (end-to-end):

$$q_s(t) = \sum_l R_{ls} p_l(t - \tau_{ls}^b)$$

where τ_{ls}^b is backward delay in feedback path

Generic Source and Link Algorithms

Each source updates rate (z_s is a local state variable):

$$\begin{aligned} z_s(t+1) &= F_s(z_s(t), q_s(t), x_s(t)) \\ x_s(t+1) &= G_s(z_s(t), q_s(t), x_s(t)) \end{aligned}$$

Often $x_s(t+1) = G_s(q_s(t), x_s(t))$

Each link updates congestion measure:

$$\begin{aligned} v_l(t+1) &= H_l(y_l(t), v_l(t), p_l(t)) \\ p_l(t+1) &= K_l(y_l(t), v_l(t), p_l(t)) \end{aligned}$$

Notice access only to local state information (distributed)

Network Utility Maximization

Basic problem formulation:

$$\begin{array}{ll} \text{maximize} & \sum_s U_s(x_s) \\ \text{subject to} & Rx \preceq c \\ & x \succeq 0 \end{array}$$

Objective: total utility (each U_s is smooth, increasing, concave)

Constraint: linear flow constraint

s index of sources and l index of links

Given routing matrix R_{ls}: 1 if flow from source s uses link l, 0 otherwise

x_s: source rate (variables), c_l: link capacity (constants)

Dual-based Distributed Algorithm

Use optimization decomposition theory to write Lagrangian $L(x, p)$:

$$= \sum_s U_s(x_s) + \sum_l p_l \left(c_l - \sum_{s:l \in L(s)} x_s \right)$$

$$= \ \textstyle\sum_s \left[U_s(x_s) - \left(\sum_{l \in L(s)} p_l \right) x_s \right] + \sum_l c_l p_l$$

$$= \ \textstyle\sum_s L_s(x_s, q_s) + \sum_l c_l p_l$$

Dual problem:

$$\begin{array}{ll} \text{minimize} & g(p) = L(x^*(p), p) \\ \text{subject to} & p \succeq 0 \end{array}$$

Dual-based Distributed Algorithm

Source algorithm:

$$x_s^*(q_s) = \text{argmax}\left[U_s(x_s) - q_s x_s\right], \quad \forall s$$

- Selfish net utility maximization locally at source s

Link algorithm (gradient or subgradient based):

$$p_l(t+1) = \left[p_l(t) - \alpha(t)\left(c_l - \sum_{s:l\in L(s)} x_s^*(q_s(t))\right)\right]^+, \quad \forall l$$

- Balancing supply and demand through pricing

Certain choices of step sizes $\alpha(t)$ ($e.g.$, $\alpha(t) = 1/t$) of distributed algorithm guarantee convergence to globally optimal (x^*, p^*)

Reverse Engineering TCP

KKT conditions:

- Primal and dual feasibility
- Lagrangian maximization (selfish net-utility maximization)
- Complementary slackness (generate the right prices to align selfish interest to social welfare maximization)

$p_l^* > 0$ indicates $y_l^* = c_l$ (link saturation) and

$y_l^* < c_l$ indicates $p_l^* = 0$ (buffer clearance)

Now specialize to average model of TCP Reno and TCP Vegas

Focus on congestion avoidance phase and targeted equilibrium state

TCP Reno: Source Algorithm

End-to-end marking probability q_s. Total delay D_s.

Window size w_s. Actual rate $\frac{w_s(t)}{D_s}$.

Net change to the window size:

$$x_s(t)(1-q_s(t)) \cdot \frac{1}{w_s(t)} - x_s(t)q_s(t) \cdot \frac{1}{2} \cdot \frac{4w_s(t)}{3}.$$

Using $x_s = w_s/D_s$, we have

$$x_s(t+1) = \left[x_s(t) + \frac{1-q_s(t)}{D_s^2} - \frac{2}{3}q_s(t)x_s^2(t)\right]^+.$$

TCP Reno: Arctan Utility

Arctan utility : $U_s(x_s) = \frac{\sqrt{3/2}}{D_s} \arctan\left(\sqrt{2/3}x_s D_s\right)$

Why?

At equilibrium ($t \to \infty$),

$$q_s = \frac{3}{2x_s^2 D_s^2 + 3}.$$

By optimality condition, need to check

$$U_s'(x_s) = q_s(x_s).$$

Get the utility function by integrating

TCP Vegas: Source Algorithm

Window size w_s

Propagation delay d_s. Expected rate $\frac{w_s(t)}{d_s}$

Queuing delay q_s and total delay D_s. Actual rate $\frac{w_s(t)}{D_s}$

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

Equilibrium round-trip time and window size satisfy:

$$\frac{w_s^*}{d_s} - \frac{w_s^*}{D_s^*} = \alpha_s$$

TCP Vegas: Log Utility Function

Log utility : $U_s(x_s) = \alpha_s d_s \log x_s$

Why: Complementary slackness condition is satisfied. Need to check

$$U_s'(x_s^*) = \frac{\alpha_s d_s}{x_s^*} = \sum_{l \in s} p_l^*$$

Let b_l^* be equilibrium backlog at link l. Window size equals bandwidth-delay product plus total backlog:

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

Using $x_s = w_s/D_s$, we have

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

KKT condition satisfied if $p_l^* = \frac{b_l^*}{c_l}$ (dual variable is queuing delay)

TCP Vegas: Summary of Algorithm

Primal variable is source rate, updated by source algorithm:

$$\begin{aligned} w_s(t+1) &= [w_s(t) + v_s(t)]^+ \\ v_s(t) &= \frac{1}{d_s + q_s(t)} [\mathbf{1}(x_s(t)q_s(t) < \alpha_s d_s) - \mathbf{1}(x_s(t)q_s(t) > \alpha_s d_s)] \\ x_s(t) &= \frac{w_s(t)}{d_s + q_s(t)} \end{aligned}$$

Dual variable is queuing delay, updated by link algorithm:

$$p_l(t+1) = \left[p_l(t) + \frac{1}{c_l}(y_l(t) - c_l) \right]^+$$

Equilibrium: $x_s^* = \frac{\alpha_s d_s}{q_s^*}$

TCP Reno and Vegas

TCP Reno (with Drop Tail or RED):

- Source utility: arctan
- Link price: packet loss

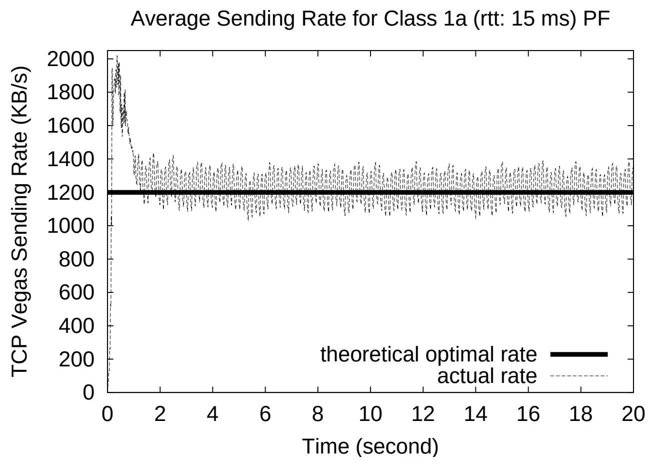
TCP Vegas (with FIFO)

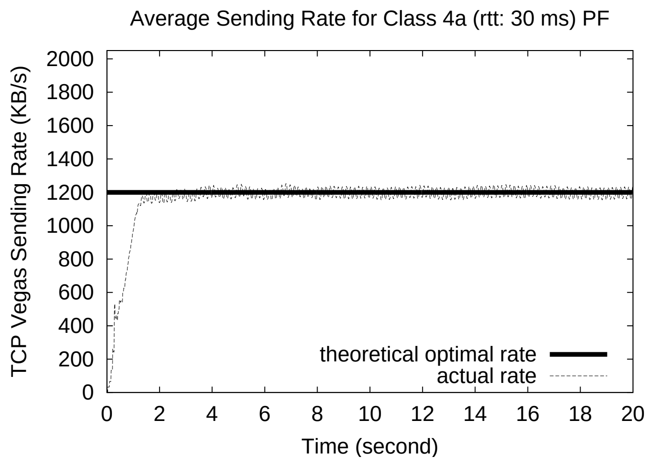
- Source utility: weighted log
- Link price: queuing delay

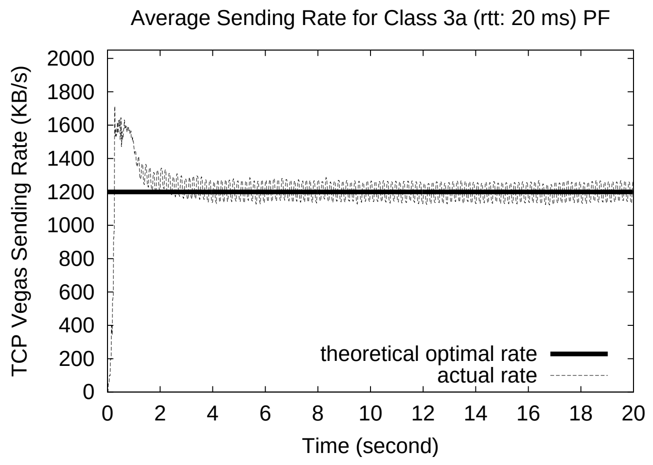
Implications: Delay, Loss, Fairness

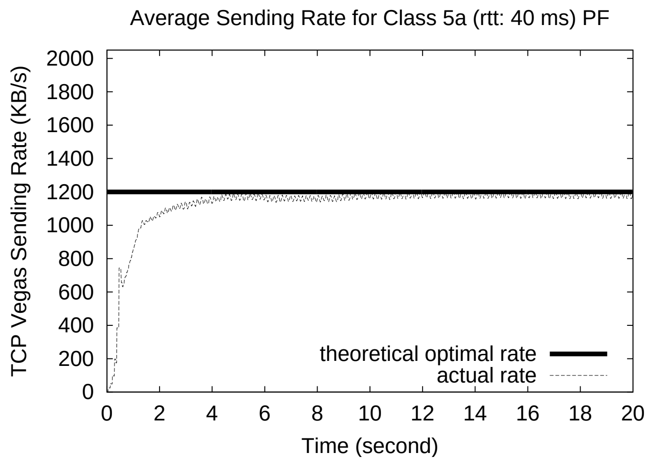
- TCP Reno: equilibrium loss probability is independent of link algorithms and buffer sizes. Increasing buffer sizes alone does not decrease equilibrium loss probability (buffer just fills up)
- TCP Reno: discriminates against connections with large propagation delays
- TCP Vegas: bandwidth-queuing delay product equals number of packets buffered in the network $x_s^* q_s^* = \alpha_s d_s$
- TCP Vegas: achieves proportional fairness
- TCP Vegas: gradient method for updating dual variable. Converges with the right scaling (γ small enough)

Numerical Example: Single Bottleneck



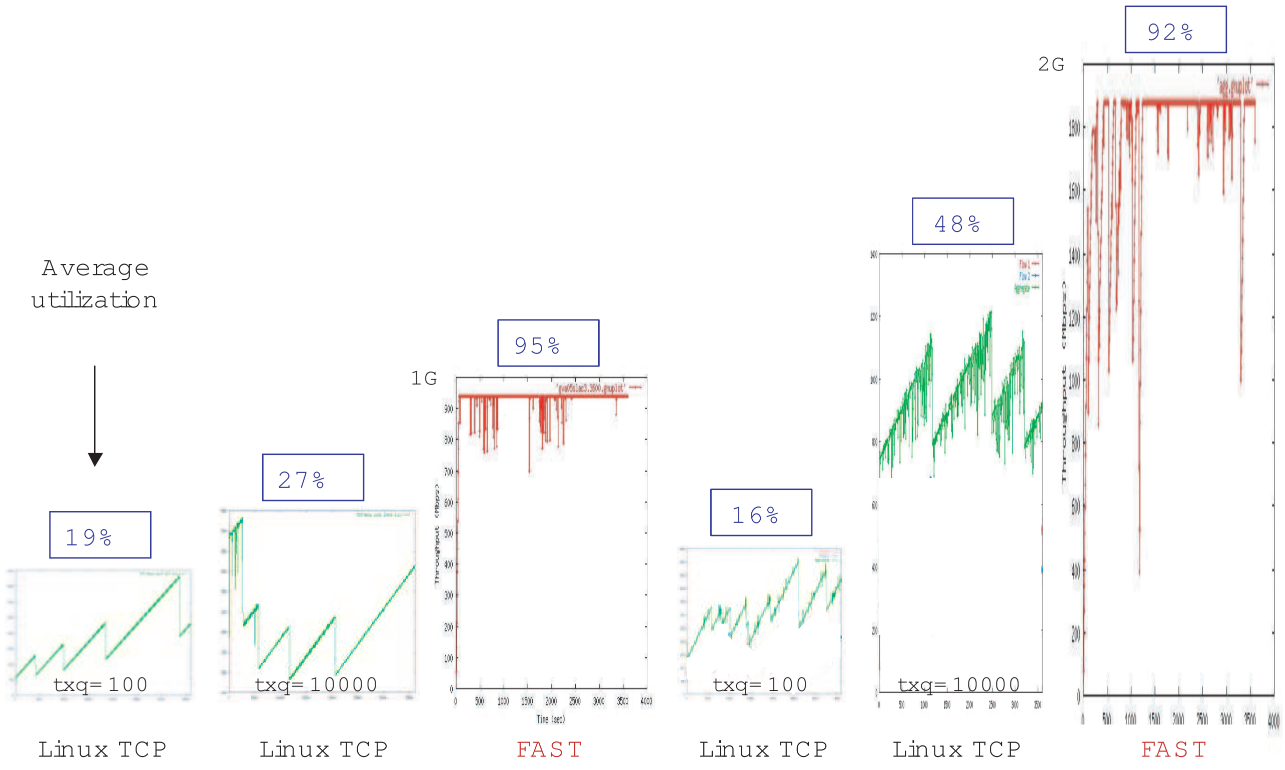






Successful Knowledge Transfer: FAST TCP

FAST TCP commercialized in 2006, increased bandwidth utilization efficiency in real networks from 20% to 90%, sold to Akamai in 2012



D. X. Wei, C. Jin, S. H. Low and S. Hegde, *FAST TCP: motivation, architecture, algorithms, performance*, IEEE/ACM Transactions on Networking, 2007

Summary

- Internet TCP congestion control as network optimizer: the use of *optimization criteria* and *optimization decomposition theory* to replace the prevalent "trial and error" approach to design

- Stability of TCP dual decomposition algorithm: In-class demo

Reading assignment:

- S. H. Low and D. E. Lapsley, Optimization flow control. I. Basic algorithm and convergence, IEEE/ACM Trans. Networking, 7(6), 1999

- S. Shakkottai and R. Srikant, Network Optimization and Control, Foundations and Trends in Networking, Now Publishers, 2007