

TCP Congestion control: basic principles and different implementations

Michele Pagano

Dipartimento di Ingegneria dell'Informazione
Università di Pisa



*15th International Asian School-Seminar
Optimization Problems of complex systems*

26-30 August 2019
Novosibirsk

Outline

- 1 TCP Basics
- 2 Classical TCP Congestion Control
- 3 Alternative congestion control algorithms
- 4 Introduction to TCP modelling
- 5 Conclusions

Outline

- 1 TCP Basics
- 2 Classical TCP Congestion Control
- 3 Alternative congestion control algorithms
- 4 Introduction to TCP modelling
- 5 Conclusions

Transport layer

Services and principles

A transport layer protocol provides for logical communication between application processes running on different hosts

- Application multiplexing and demultiplexing
- Reliability
- Flow Control
- Congestion Control

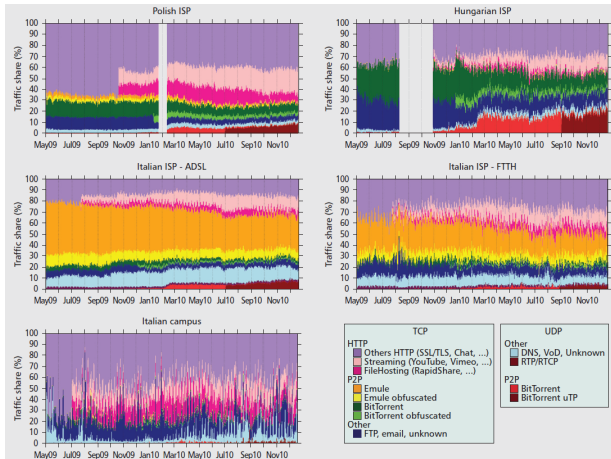
Main protocols

- UDP (User Datagram Protocol)
- TCP (Transmission Control Protocol)
- DCCP (Datagram Congestion Control Protocol)
- SCTP (Stream Control Transmission Protocol)

Transmission Control Protocol (TCP)

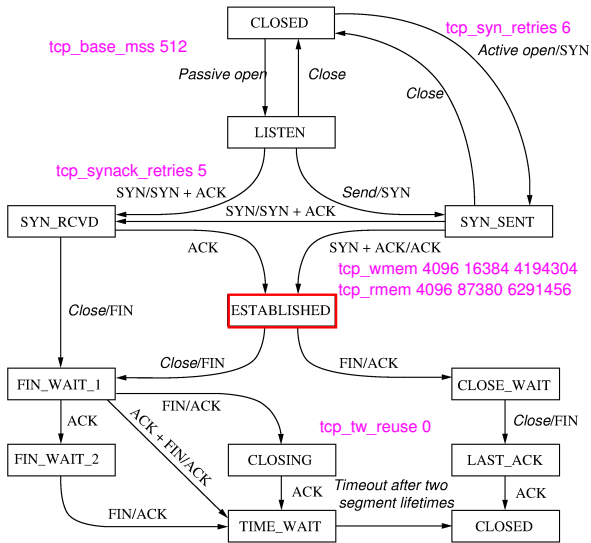
- TCP is based on concepts first described in *V.Cerf, R. Kahn, "A Protocol for Packet Network Intercommunication", IEEE TCOM, May 1974"*
- In IETF world originally defined in **RFC 793 (September 1981)**
- Connection-oriented transport protocol that provides a reliable byte-stream data transfer service between pairs of processes
- Key features:
 - **Full duplex** (piggyback of ACKs)
 - **Connection-oriented** (Establishment and teardown of the connections)
 - **Multiplexing/Demultiplexing** (through Source and Destination Port numbers)
 - **Reliability** (through Sequence Numbers, Checksum, ACKs and timers)
 - **Flow Control** (through Advertized Window)
 - **Congestion Control**, making TCP sensitive to network conditions

TCP vs other Transport layer protocol



Finamore, Mellia, Meo, Munafò, Rossi, “Experiences of Internet Traffic Monitoring with Tstat”, IEEE Network, March/April 2011

TCP State-transition diagram



/proc interfaces

System-wide TCP parameter settings can be accessed by files in the directory `/proc/sys/net/ipv4/`

tcp.h file

Implementation parameters

```
#define
TCP_INIT_CWND
10
```

Outline

- 1 TCP Basics
- 2 Classical TCP Congestion Control
 - Basic principles
 - TCP Reno
- 3 Alternative congestion control algorithms
- 4 Introduction to TCP modelling
- 5 Conclusions



TCP Congestion Control

- TCP congestion control (CC) mechanisms seek to
 - achieve high utilization
 - *control* congestion
 - share bandwidth
- TCP CC introduced in the late 1980s by Van Jacobson
 - in October 1986, the Internet had the first of what became a series of congestion collapses (sudden factor-of-thousand drop in bandwidth)
 - **window-based** mechanism: TCP maintains a state variable **cwnd**, used by the source to limit how much data it is allowed to have in transit at a given time
 - **Slow Start, Congestion Avoidance and Fast Retransmit**
 - *round-trip variance estimation*

TCP Tahoe

Van Jacobson, "Congestion Avoidance and Control", ACM SIGCOMM, Computer Communication Review, v. 18, n. 4, Aug. 1988

TCP Congestion Control

- TCP assumes **packet losses are caused by congestion**
 - RFC 2001 - TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms (January 1997): The assumption of the algorithm is that packet loss caused by damage is very small (much less than 1%), therefore the loss of a packet signals congestion somewhere in the network between the source and destination
- The strategy of TCP is to send packets into the network without a reservation and then **to react** to observable events that occur
- Behaviour of **cwnd**
 - no losses \Rightarrow more bandwidth is available \Rightarrow **cwnd** 
 - loss of a packet \Rightarrow network congestion \Rightarrow **cwnd** 
- **Differentiation between major and minor congestion events**
 - Introduction of **Fast Recovery** mechanism
 - High throughput under moderate congestion, especially for large windows

Classical TCP Congestion Control

TCP Reno, 1990

Van Jacobson, “Modified TCP Congestion Avoidance algorithm”, e-mail to the end2end list, Apr. 1990

- *Classical* TCP Congestion Control consists of four different algorithms

- Slow Start (SS)

for each ACK $\text{cwnd} \leftarrow \text{cwnd} + 1$

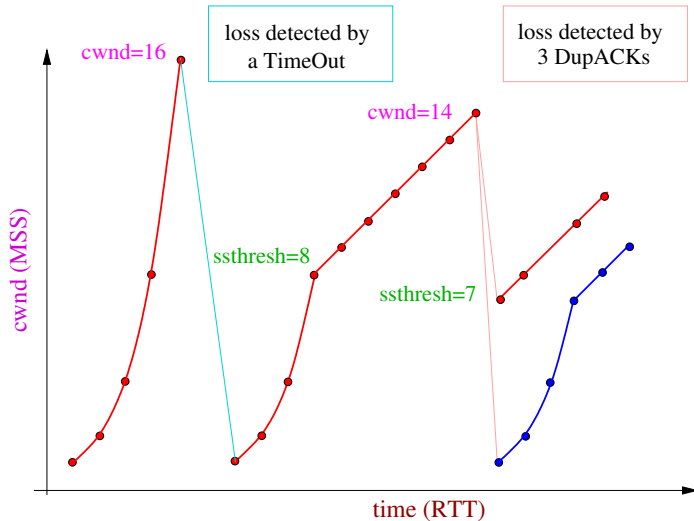
- Congestion Avoidance (CA)

for each ACK $\text{cwnd} \leftarrow \text{cwnd} + 1/\text{cwnd}$

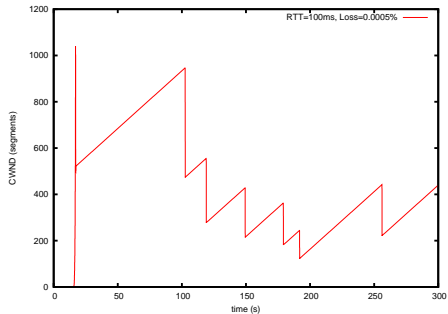
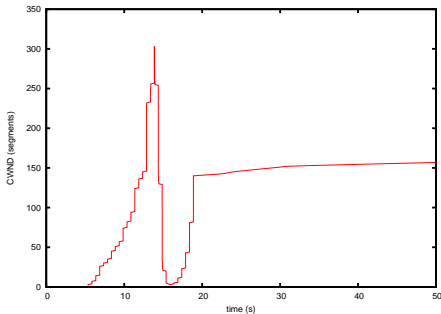
- Fast Retransmit and Fast Recovery

- $\text{ssthresh } s(t)$: slow start threshold determines whether to use SS or CA
 - to achieve high performance, ssthresh should be set close to the Bandwidth-Delay product (BDP)
- Implemented in Linux since 1.3.90 kernel

TCP Reno vs TCP Tahoe



Real cwnd behaviour of TCP Reno



- **cwnd** behaviour at the beginning of a connection
- Typical **AIMD** behaviour (effect of Fast Retransmit/Fast Recovery mechanisms)

Outline

- 1 TCP Basics
- 2 Classical TCP Congestion Control
- 3 Alternative congestion control algorithms**
 - TCP Variants
 - TCP Linux
 - TCP CUBIC
 - TCP BBR
- 4 Introduction to TCP modelling
- 5 Conclusions

TCP Variants

- Congestion indicators
 - Losses
 - Delay
 - Marks
- Standard TCP relies on packet loss as an indicator of network congestion

Congestion Control vs *Congestion Avoidance*

- Monitoring changes in the flow rate and current RTT to predict congestion before losses occur

Wireless links

- Noisy and fading radio channels are frequent causes of loss
- TCP Reno is not able to distinguish congestion loss from wireless loss (RFC 2001)

TCP Variants

Long-distance (Long) and High-speed (Fat) Networks

- Conservative behavior of TCP Reno in adjusting its cwnd
 - Congestion control parameters depend on current cwnd
 - Queueing delay as a *secondary* congestion signal
 - Impact of multiple losses
-
- Different mechanisms are necessary for congestion control in heterogeneous networks

Compound TCP (CTCP)

- Microsoft algorithm introduced as part of the Windows Vista and Window Server 2008 TCP stack
- Designed to aggressively adjust the sender's congestion window to optimize TCP for connections with large bandwidth-delay products while trying not to harm fairness

Some TCP variants

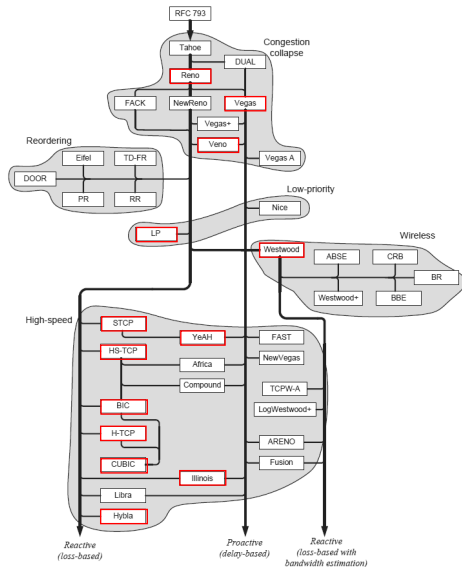
References

Alexander Afanasyev, Neil Tilley, Peter Reiher, and Leonard Kleinrock

“Host-to-Host Congestion Control for TCP”, IEEE Communications Surveys & Tutorials, 2010

Cheng-Yuan Ho, Yaw-Chung Chen, Yi-Cheng Chan, and Cheng-Yun Ho

“Fast retransmit and fast recovery schemes of transport protocols: A survey and taxonomy”, Computer Networks 52 (2008)



Main TCP Linux Variants

CUBIC and Reno (NewReno)

- Loaded into the kernel, via standard kernel module mechanism
- Information available in `/proc/sys/net/ipv4`

Relevant files

- `tcp_congestion_control`
`cubic`
- As root user `modprobe tcp_bic`
- `tcp_available_congestion_control`
`cubic reno bic`
- `tcp_allowed_congestion_control`
`cubic reno`
- `echo bic > /proc/sys/net/ipv4/tcp_congestion_control`
- `tcp_congestion_control`
`bic`

```
ls -a /lib/modules/`uname -r`/kernel/net/ipv4/tcp*
```

kernel 2.6.35-30

- tcp_bic.ko
- tcp_highspeed.ko
- tcp_htcp.ko
- tcp_hybla.ko
- tcp_illinois.ko
- tcp_lp.ko
- tcp_probe.ko
- tcp_scalable.ko
- tcp_vegas.ko
- tcp_veno.ko
- tcp_westwood.ko
- tcp_yeah.ko

kernel 3.19.0-30

- tcp_bic.ko
- tcp_dctcp.ko
- tcp_diag.ko
- tcp_highspeed.ko
- tcp_htcp.ko
- tcp_hybla.ko
- tcp_illinois.ko
- tcp_lp.ko
- tcp_probe.ko
- tcp_scalable.ko
- tcp_vegas.ko
- tcp_veno.ko
- tcp_westwood.ko
- tcp_yeah.ko

ls -a /lib/modules/`uname -r`/kernel/net/ipv4/tcp*

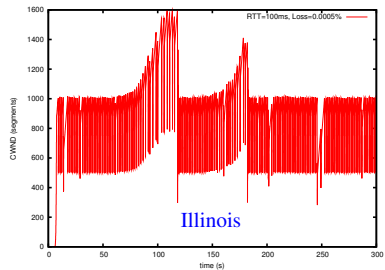
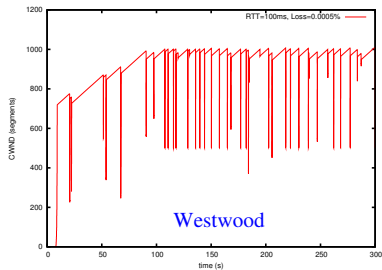
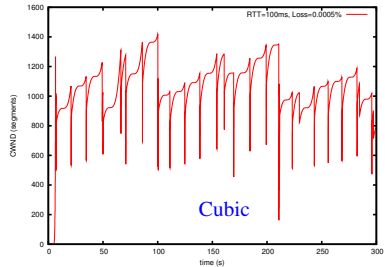
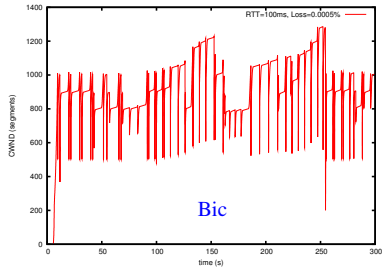
kernel 4.4.0-157

- tcp_bic.ko
- tcp_cdg.ko
- tcp_dctcp.ko
- tcp_diag.ko
- tcp_highspeed.ko
- tcp_htcp.ko
- tcp_hybla.ko
- tcp_illinois.ko
- tcp_lp.ko
- tcp_probe.ko
- tcp_scalable.ko
- tcp_vegas.ko
- tcp_veno.ko
- tcp_westwood.ko
- tcp_yeah.ko

kernel 4.18.0-21

- tcp_bbr.ko
- tcp_bic.ko
- tcp_cdg.ko
- tcp_dctcp.ko
- tcp_diag.ko
- tcp_highspeed.ko
- tcp_htcp.ko
- tcp_hybla.ko
- tcp_illinois.ko
- tcp_lp.ko
- tcp_nv.ko
- tcp_scalable.ko
- tcp_vegas.ko
- tcp_veno.ko
- tcp_westwood.ko
- tcp_yeah.ko

Behaviour of some Linux TCP Variants



TCP CUBIC

- Originally proposed in 2005
- Default since 2.6.19 Linux kernel
- It differs from the current TCP standards **only in the congestion control algorithm on the sender side**
 - CUBIC maintains the ACK clocking of Standard TCP
 - No changes to the fast recovery and retransmit of TCP Reno
- It uses a **cubic function** instead of a linear window increase function of the current TCP standards to improve scalability and stability under fast and long-distance networks

References

I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, R. Scheffenegger
“RFC 8312 – CUBIC for Fast Long-Distance Networks”

February 2018

S. Ha, I. Rhee, and L. Xu,

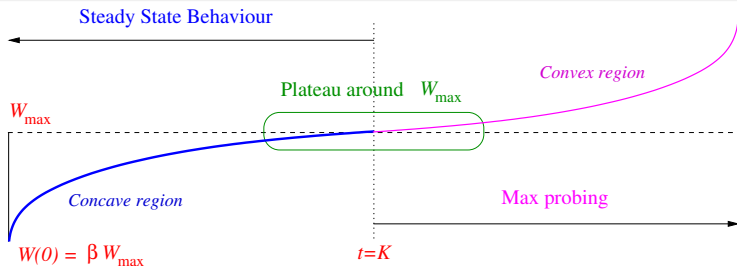
“CUBIC: A New TCP-Friendly High-Speed TCP Variant”

ACM SIGOPS Operating System Review, 2008

Design Principles of CUBIC

- For better network utilization and stability, CUBIC uses both the concave and convex profiles of a cubic function to increase the congestion window size, instead of using just a convex function
 - To be TCP-friendly, CUBIC is designed to behave like Standard TCP in networks with short RTTs and small bandwidth where Standard TCP performs well
 - For RTT-fairness, CUBIC is designed to achieve linear bandwidth sharing among flows with different RTTs
 - CUBIC appropriately sets its multiplicative window decrease factor in order to balance between the scalability and convergence speed
-
- **Congestion event:** a packet loss is detected by **duplicate ACKs** or a network congestion is detected by **ACKs with ECN-Echo flags**
 - CUBIC window growth depends only on the *real time* between two consecutive congestion events, i.e. it becomes *independent of RTTs*

TCP CUBIC window growth – Basic idea



During **congestion avoidance**, after a congestion event

- CUBIC registers the window size W_{\max}
- It performs a multiplicative decrease of congestion window by a factor of β (suggested value: $\beta = 0.7$)
- It starts to increase the window using the **concave profile**
- The **concave growth** continues until W_{\max}
- After that, the cubic function turns into a **convex profile** and the convex window growth begins

TCP CUBIC window growth function

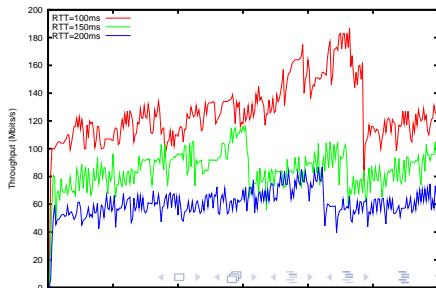
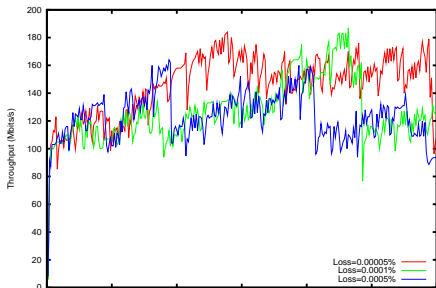
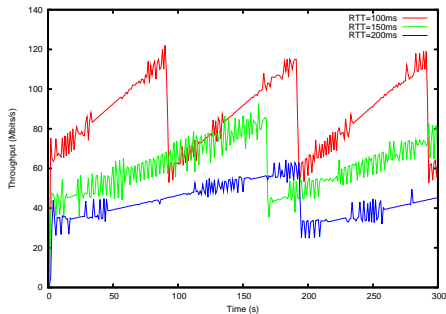
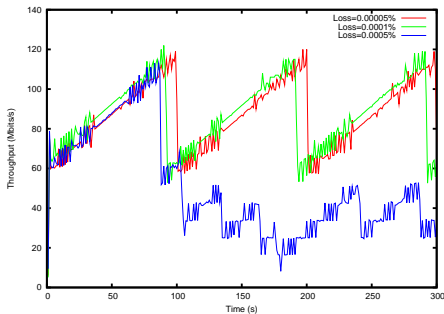
$$W(t) \triangleq W_{\text{CUBIC}}(t) = C(t - K)^3 + W_{\text{max}}$$

- W_{max} is the window size just before the window is reduced in the last congestion event
- C is a constant fixed to determine the aggressiveness of window increase in high BDP networks (suggested value: $C = 0.4$)
- t is the elapsed time from the beginning of the current congestion avoidance
- K is the time that the above function takes to increase the current window size to W_{max} if there are no further congestion events

$$K = \sqrt[3]{\frac{W_{\text{max}}(1 - \beta)}{C}}$$

- The initial implementation in Linux uses the **bisection method**
- The use of the **Newton-Raphson method** improved the running time by more than 10 times on average (1032 clocks vs. 79 clocks) and reduced the variance in running times

TCP Reno (top) vs. TCP CUBIC (bottom)

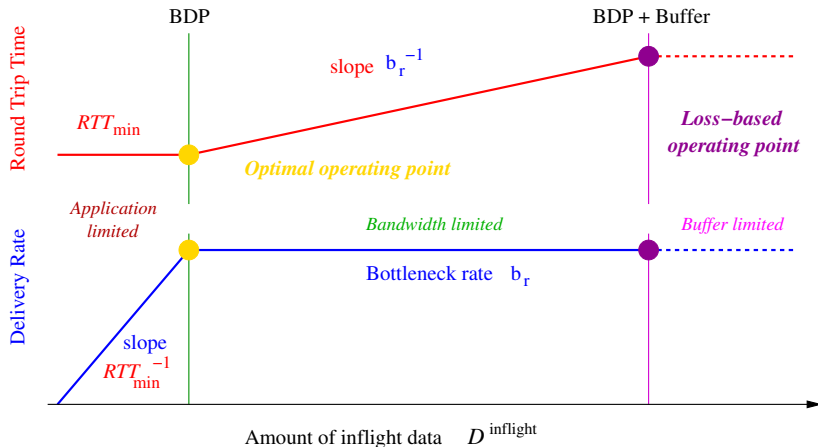


Basic principles of TCP BBR

N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson
“**BBR: Congestion-Based Congestion Control**”, ACM Queue,
Oct. 2016

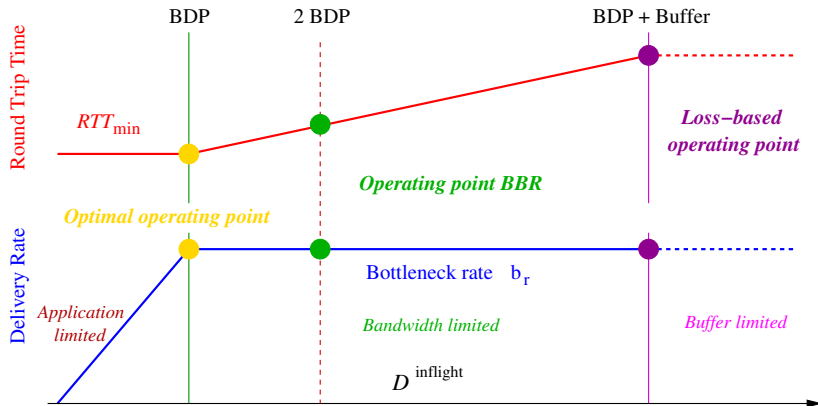
- As NICs evolved from Mbps to Gbps and memory chips from KB to GB, the *relationship between packet loss and congestion became more tenuous*
- BBR was proposed by a team from Google as a *new “Congestion-based” congestion control mechanism*
 - TCP Vegas and New Vegas presaged many elements of BBR
- BBR is *neither delay-based nor loss-based*
- BBR *ignores* packet loss as congestion signal
- BBR also does not explicitly react to congestion, whereas *cwnd-based approaches* often use a multiplicative decrease strategy to reduce D^{inflight}
- A BBR sender controls its transmission rate s_r with the help of pacing and an estimated data rate b_r
⇒ **Rate-based congestion control**

Congestion control operating points



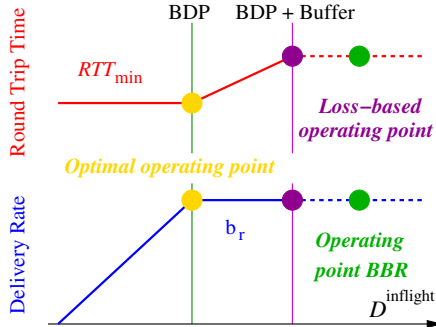
D^{inflight} – Data in flight, i.e. data sent but not yet ACKed

Operating point with multiple flows - large buffers



$$\sum \widehat{b_{ri}} > b_r \Rightarrow \text{Increasing delays and RTT unfairness}$$

Operating point with multiple flows - small buffers

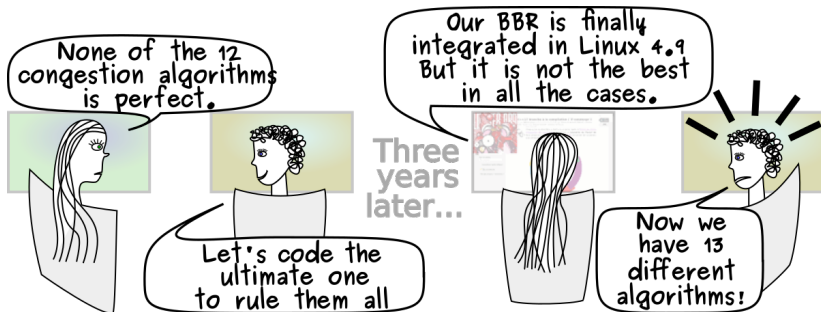


Packet loss is caused by congestion

- Massive amount of packet losses
- Unfairness to flows with loss-based congestion control

- BBR has no explicit mechanism to let multiple BBR flows converge to a fair share
- BBR has neither an explicit congestion detection mechanism nor an explicit reaction to congestion
- Investigation of BBR behavior with Active Queue Management mechanisms (BBR does not react to packet loss as congestion signal)

TCP BBR v2



BBR v2 – A Model-based Congestion Control

N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, V. Vasiliev, P. Jha, Y. Seung, M. Mathis, V. Jacobson, IETF 104, Prague, **March 2019**
<https://groups.google.com/d/forum/bbr-dev>

Outline

- 1 TCP Basics
- 2 Classical TCP Congestion Control
- 3 Alternative congestion control algorithms
- 4 Introduction to TCP modelling**
 - Overview on TCP modelling
 - Deterministic models
 - Stochastic model of TCP Reno
- 5 Conclusions

TCP models

Single connection models

- Assume the knowledge of network characteristics, such as mean RTT and loss probability
- This class can be further divided into models for **short-lived** and **long-lived** connections
 - A small number of long flows (**elephants**) carries the most (e.g. 80%) of Internet traffic
 - Widespread diffusion of web-browsing, characterized by small (around 30 KB on average) and frequent data transfer (**mice**)

Models of interaction with AQM

- Derive the performance of TCP and network statistics
- Introduce a sub-model of TCP and a sub-model of IP network protocol and solve through fixed-point procedures

Single source models: long-lived connections

General aim of the models

Simple analytic expression for the send rate of a *saturated* **TCP Reno sender** (i.e., a flow with an unlimited amount of data to send) as a function of loss rate and average RTT

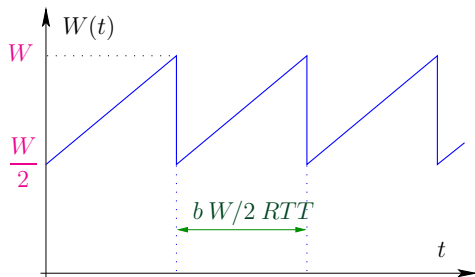
$1/\sqrt{p}$ expression for TCP Reno

Underlying assumptions

- Steady state analysis
- Loss rate and RTT are independent of the window size
- No ACK loss
- **TCP Reno**
 - Congestion avoidance (slow-start phase is neglected)
 - Fast Retransmit and Fast Recovery
 - **Delayed ACK**: many TCP receiver implementations send one **cumulative ACK** for 2 consecutive packets received ($b = 2$)

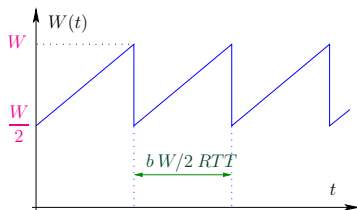
Simple deterministic model of TCP Reno

- TCP source running over a lossy path with sufficient bandwidth and **sufficiently low competing traffic** \Rightarrow the queuing delay contribution to RTT is *negligible*
- Assume that the link introduces **one drop after the successful delivery of $1/p$ consecutive packets**



- Periodic evolution of **cwnd**
 - W : maximum value of **cwnd** reached at the equilibrium
 - **cwnd** is backed off to $W/2$ after each loss, starting a new congestion avoidance phase

Simple deterministic model – Analysis



- Duration of a cycle

$$T_{\text{cycle}} = RTT \cdot b \frac{W}{2}$$

- Total number of segments delivered within each cycle

$$b \frac{W}{4} \cdot \left(\frac{W}{2} + W \right) = b \frac{3W^2}{8} = 1/p$$

- Maximum size of **cwnd**

$$W = \sqrt{\frac{8}{3pb}}$$

Simple deterministic model – Main results

Mean throughput

$$\mathcal{B} = \frac{A_{\text{cycle}}}{T_{\text{cycle}}} = \frac{MSS \cdot b^{\frac{3}{8}} W^2}{RTT \cdot \frac{b}{2} W} = \sqrt{\frac{3}{2b}} \cdot \frac{MSS}{RTT \sqrt{p}}$$

- The throughput is proportional to the packet size
- The throughput is inversely proportional to RTT (unfair behavior) and to the square root of loss probability
- Slightly different values of the proportionality constant in other similar models

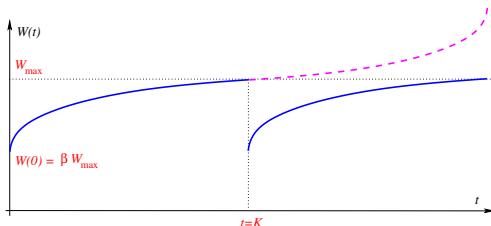
Limitations

- The timeout mechanisms is not taken into account
- Optimistic estimate of the bandwidth of a TCP connection
- Accurate in the range of small loss probabilities
- Not suitable to determine performance of TCP over slow-speed line (few packets in transit)

Simple deterministic model of TCP CUBIC

With a **deterministic loss model** where the number of packets between two successive packet losses is always **1/p**

- CUBIC always operates with the **concave window profile**
- **cwnd** has a periodic evolution



$$W(t) \triangleq W_{\text{CUBIC}}(t) = C(t - K)^3 + W_{\max} \quad K = \sqrt[3]{\frac{W_{\max}(1 - \beta)}{C}}$$

Simple deterministic model of TCP CUBIC – Analysis

- Average send rate (in segments per RTT)

$$\mathbb{E} W = \frac{1}{K} \int_0^K W(t) dt = W_{\max} \left(\frac{3 + \beta}{4} \right)$$

- Total number of segments delivered within each cycle

$$W_{\max} \frac{3 + \beta}{4} \cdot \frac{K}{RTT} = 1/p$$

- Maximum size of **cwnd**

$$W_{\max} = \sqrt[4]{\frac{C}{1 - \beta} \left(\frac{4}{3 + \beta} \right)^3 \left(\frac{RTT}{p} \right)^3}$$

- Average **cwnd** size

$$\mathbb{E} W_{\text{CUBIC}} = \sqrt[4]{\frac{C(3 + \beta)}{4(1 - \beta)} \left(\frac{RTT}{p} \right)^3}$$

PFTK model - general features

Jitendra Padhye, Victor Firoiu, Donald F. Towsley, and James F. Kurose, *Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation*, IEEE/ACM TON, April 2000

- The congestion avoidance mechanism is modelled in terms of rounds
 - A round starts with transmission of W packets, where W is the current size of the TCP congestion window
 - No other packets are sent until the first ACK is received for one of these packets
- The duration of a round is equal to the RTT and is assumed to be independent of the window size
- The time needed to send all the packets in a window is smaller than the RTT
- In the absence of loss, the window size increases linearly in time, with a slope of $1/b$ packets per RTT

PFTK model - losses

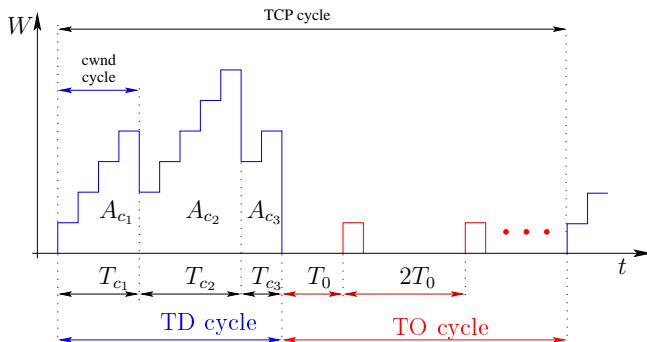
- Packet loss can be detected in one of two ways
 - by the reception at the TCP sender of triple-duplicate acknowledgments — **TD (triple-duplicate) loss indication**
 - via time-outs – **TO loss indication**

Simple bursty loss model

- A packet is lost in a round *independently* of any packets lost in other rounds
- Packet losses are correlated among the back-to-back transmissions within a round: if a packet is lost, *all remaining packets* transmitted until the end of that round are also lost
- **p** : probability that a packet is lost, given that either it is the first packet in its round or the preceding packet in its round is not lost



Congestion window dynamics



- Mean throughput

$$\mathcal{B} = \frac{\text{pkts in TD cycle} + \text{pkts in TO cycle}}{\text{duration of TD cycle} + \text{duration of TO cycle}}$$

Sketch of analysis

Compute average number of **cwnd cycles** per **TD cycle**

- enumerate all possible events leading to **TD ACK**
- assume independence between consecutive **cwnd cycles**
- average size of **cwnd** at the end of a **cwnd cycles**

Compute average length of **TO cycles**

- probability that a loss is detected by a **TO**
- the number of timeouts in a **TO cycle** has a geometric distribution
- exponential backoff of **TOs**

Impact of window limitation

- The receiver advertizes a maximum buffer size which determines a maximum congestion window size, W_m
- During a period without loss indications, the window size can grow up to W_m , but will not grow further beyond this value

Basic notations

- W is the average window size W_i at the end of a **cwnd cycle**

$$W = \mathbb{E} W_i = \sqrt{\frac{8(1-p)}{3bp} + \frac{(3b-2)^2}{9b^2}} - \frac{3b-2}{3b}$$

$$W \rightarrow \sqrt{\frac{8}{3bp}} \quad \text{as } p \rightarrow 0$$

- $\hat{Q}(w)$ is the probability that a loss in a window of size w is a **TO**

$$\hat{Q}(w) = \min \left(1, \frac{(1 - (1-p)^3)(1 + (1-p)^3(1 - (1-p)^{w-3}))}{1 - (1-p)^w} \right)$$

- $f(p)$ take into account the exponential back-off of **TO** duration

$$f(p) = 1 + p + 2p^2 + 4p^3 + 8p^4 + 16p^5 + 32p^6$$

PFTK model – Mean throughput

- Case $W < W_m$

$$\mathcal{B} = \frac{\frac{1-p}{p} + W + \hat{Q}(W) \frac{1}{1-p}}{RTT \left(\frac{b}{2} W + b + 1 \right) + \hat{Q}(W) T_0 \frac{f(p)}{1-p}}$$

$$\mathcal{B} \approx \frac{1}{RTT \sqrt{\frac{2bp}{3}} + T_0 \min \left(1, 3 \sqrt{\frac{3bp}{8}} \right) p} \quad \text{as } p \rightarrow 0$$

- Case $W \geq W_m$

Impact of window limitation $\Rightarrow \mathbb{E}[W] \approx W_m$

$$\mathcal{B} = \frac{\frac{1-p}{p} + W_m + \hat{Q}(W_m) \frac{1}{1-p}}{RTT \left(\frac{b}{8} W_m + \frac{1-p}{p W_m} + \frac{b+6}{4} \right) + \hat{Q}(W_m) T_0 \frac{f(p)}{1-p}}$$

Outline

- 1 TCP Basics
- 2 Classical TCP Congestion Control
- 3 Alternative congestion control algorithms
- 4 Introduction to TCP modelling
- 5 Conclusions**

References

Latest RFC on TCP Congestion Control

M. Allman, V. Paxson, and E. Blanton, “**RFC 5681 — TCP Congestion Control**”, September 2009. It obsoletes RFC 2581, which in turn obsoleted RFC 2001

Some recent RFCs

C. Raiciu, M. Handley, D. Wischik, “**RFC 6356 — Coupled Congestion Control for Multipath Transport Protocols**”, October 2011

T. Henderson, S. Floyd, A. Gurtov, Y. Nishida, “**RFC 6582 — The NewReno Modification to TCP’s Fast Recovery Algorithm**”, April 2012

M. Duke, R. Braden, W. Eddy, E. Blanton, A. Zimmermann, “**RFC 7414 — A Roadmap for TCP Specification Documents**”, February 2015

I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, R. Scheffenegger, “**RFC 8312 — CUBIC for Fast Long-Distance Networks**”, February 2018

Some our works on TCP

C. Callegari, S. Giordano, M. Pagano, and T. Pepe, “**Behavior analysis of TCP Linux variants**”, Computer Networks 56 (2012)

C. Callegari, S. Giordano, M. Pagano, and T. Pepe, “**A survey of congestion control mechanisms in Linux TCP**”, Springer-Verlag CCIS 279

M. Pagano and R. Secchi, “**An Introduction to Modelling and Performance Evaluation for TCP Networks**”, Network Performance Engineering, 2011

