A Survey on TCP Performance in a Heterogeneous Network

Chadi BARAKAT

INRIA Sophia Antipolis
2004 route des Lucioles, BP 93
06902 Sophia Antipolis CEDEX

Email: Chadi.Barakat@sophia.inria.fr
http://www.inria.fr/mistral/personnel/Chadi.Barakat

The story

- TCP = Transmission Control Protocol.
- A layer-4 protocol with two main objectives :
  - Make transparent the simple best effort service of the Internet.
  - Control the source traffic to reduce the overload on the network and on the receiver.
- Problem: The Internet has evolved and TCP mechanisms are not able to follow this evolution.
- Intuitive result: A wide research area with many propositions.

Outline

- Background on TCP (until the first work of Jacobson).
- What has changed in the Internet?
- Classification of the new characteristics of Internet paths.
- For every characteristic: TCP Problems and proposed solutions.
- Concluding remarks on the future of TCP.

Original TCP

– Main objectives:

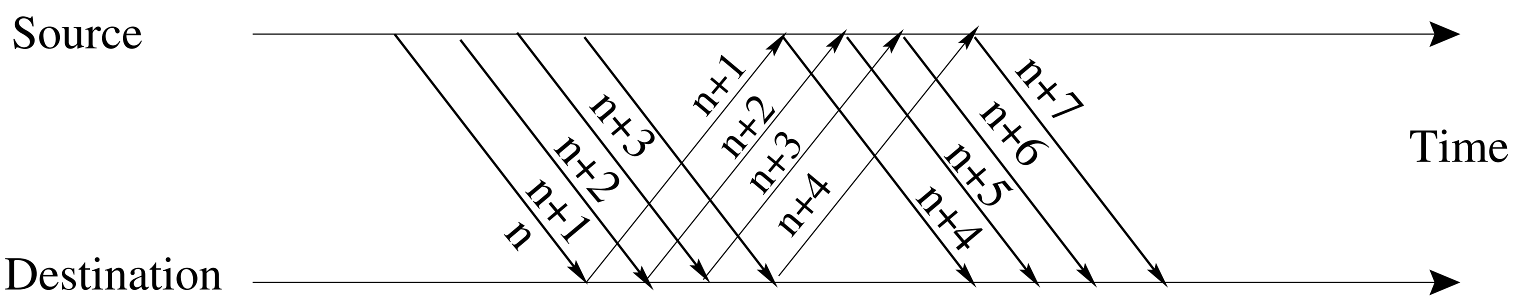
  * Providing a byte-streaming in-order connection-oriented reliable service to the application layer (e.g. Web, Email, Ftp, Telnet, News, ...).
  * Controlling the flow of packets to avoid the overflow of the receiver buffer.

– No consideration of network resources!

Old TCP : Mechanisms

– For flow control:

  * A window W (or $rwnd$) set by the receiver (a 16-bit in the TCP header).
  * Byte sequence numbering with positive cumulative Acknowledgments (ACK).
  * ACK-clocked transmission (when an ACK is received, slide the window and send as many packets as it allows).



Old TCP : Mechanisms

– Retransmission for loss recovery (Go-Back-N).

– Timer for loss detection:

  * Estimate the Round Trip Time using an Exponentially Waited Moving Average algorithm,

$$srtt = \alpha * srtt + (1 - \alpha) * rtt.$$

  * When a new ACK is received, reset the retransmission timer to $\beta srtt$ ($\beta = 2$).
  * Retransmit the lowest-numbered unacknowledged packet upon timeout.

Jacobson's algorithms

- Motivation: The congestion collapse of 1986.
- Idea: Consider the network load in the flow control.
- Implementation:
  * Introduce a congestion window ($cwnd$) that represents the number of packets the source can keep in the network.
  * Take W as the minimum of $rwnd$ and $cwnd$.
  * Change $cwnd$ (increase and decrease) as a function of network conditions.
- Another algorithm for setting the retransmission timer.

TCP Congestion Control

– Main idea :

  * Maintain a certain estimate of the network capacity.
  * Increase the congestion window until a loss is detected (via timeout till this slide).
  * Assume that the network is congested, set the estimate to half the current window and reduce the window.
  * Restart the window increase.

– Two algorithms for the window increase : Slow Start and Congestion Avoidance.

Slow Start

- Increase $cwnd$ quickly (from one segment or a larger value) until the capacity estimate.
- Algorithm: When an ACK is received: cwnd = cwnd + MSS (Maximum Segment Size).
- Result: An exponential window increase (the window doubles every RTT when every data packet is acked).
- The estimate is called the Slow Start Threshold (W_{th} or $ssthresh$).
- At the beginning of the connection, W_{th} is set to a default value usually equal to the receiver window.

Slow Start

- Objectives behind Slow Start:
  - Prohibit the source from transmitting a large burst of packets when the ACK clock is stopped (at the beginning of the connection or after an idle period).
  - But at the same time, fill quickly the network capacity.
  - A clear tradeoff between a fast window increase and a low burstiness.
  - Establishing quickly the ACK clock.
  - Estimating quickly the network capacity when it is overestimated.

Congestion Avoidance

– Follows Slow Start when $cwnd$ exceeds $ssthresh$.

– Considered as the steady state of the connection where the bulk of data is transfered.

– The window is increased slowly to probe the network for any extra bandwidth (the network is supposed to be well utilized).

– When an ACK is received: $cwnd = cwnd + MSS * MSS/cwnd$.

– Result: A linear window increase (the window increases by one segment every RTT when every data packet is acked).

Complete window evolution

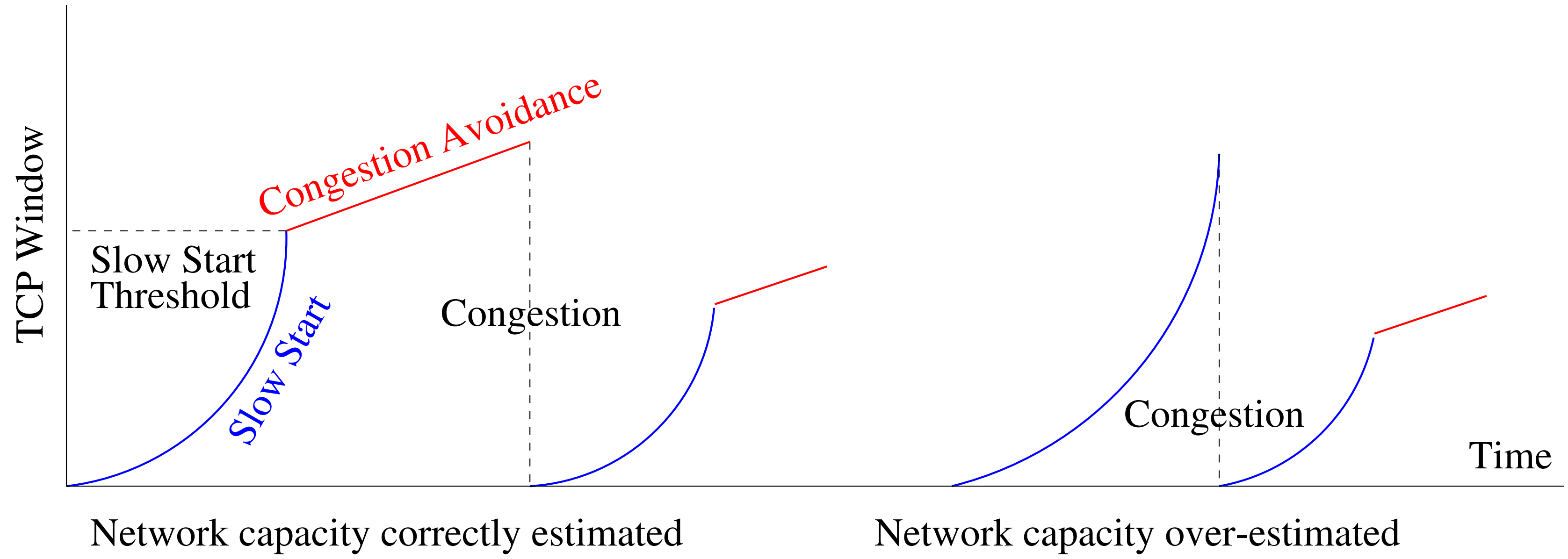
– Losses are detected via timeout. The ACK clock stops and Slow Start is then required.

– Upon timeout: Set $ssthresh$ to $min(rwnd, cwnd)/2$, set $cwnd$ to one packet, and call Slow Start until the new value of $ssthresh$.

– Window evolution at large time scale:

  ∗ An additive-increase multiplicative-decrease strategy with a transitory phase to reach again the linear phase after every window reduction.

Complete window evolution



Refinement of the retransmission timer setting

– Introduction of the variance of RTT ($rttvar$):
$$rttvar = (1 - \beta) * rttvar + \beta * |srtt - rtt|$$
$$srtt = (1 - \alpha) * srtt + \alpha * rtt$$
$$rto = srtt + K * rttvar$$
$$\alpha = 1/8, \beta = 1/4, K = 4$$

– Karn algorithm and exponential timer back-off:

  * Don't use retransmitted packets for RTT estimation.
  * Double the timer interval in case of successive timeouts.

TCP and the new Internet

– The previous TCP has served the Internet for years.

– The problems started to appear with the increase in the Internet heterogeneity:

  * High speed links (e.g. optic fibers at hundreds of Gb/s).
  * Low speed links (e.g. dial-up modem lines, small antenna satellite uplinks).
  * Long delay paths (mainly due to satellite links).
  * Variable delay paths (e.g. satellite constellations, mobile networks).
  * Lossy links (e.g. noisy wireless and satellite links).
  * Asymmetric paths (e.g. direct broadcast satellite networks, cable networks).

– This new medium is completely different from the one TCP is tuned to and TCP is no longer able to provide good service for up-layer applications.

[RFC2488, PS97]

TCP and satellites

– All of the new characteristics of Internet paths figure in a satellite environment:

  * The delay is pronounced on a GEO link (250ms).
  * Losses and variable delays will be pronounced in a LEO constellation due to mobility and handover.
  * High data rate and asymmetry can be found in the two environments.

– The satellite environment has served as an area to summarize the different problems of TCP.

Organization of the study

- The path of a TCP connection is characterized by its:
  * Bandwidth-Delay Product (BDP)
  * Round Trip Time (RTT)
  * Loss rate
  * Asymmetric degree
- For every quantity, we present:
  * Its negative effect on TCP performance.
  * A study of the proposed mitigations (advantages and drawbacks).

Problems with a large BDP

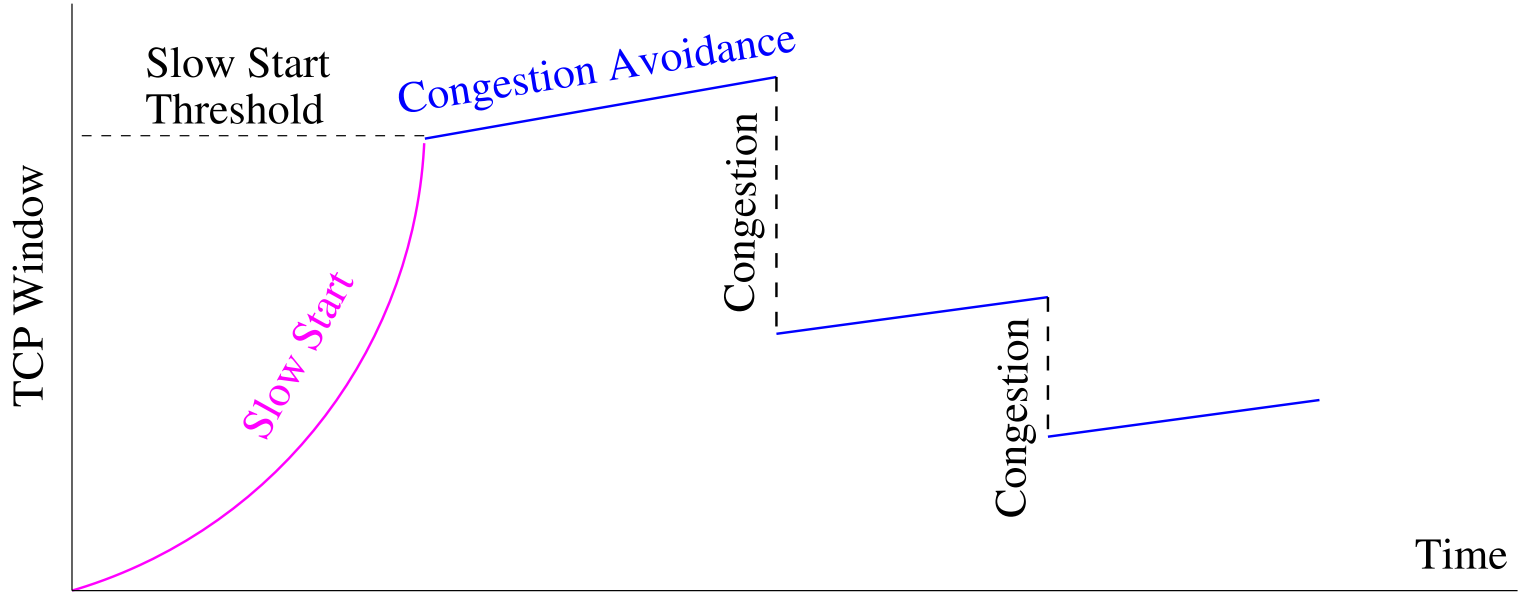
- Problem of window size: Large windows are needed to fully utilize the available bandwidth.

- Problem of loss recovery:

  * Waiting a timeout and slow starting after every loss detection yield poor performance:
    · Coarse granularity of the retransmission timer (500ms).
    · Long time taken by Slow Start due to a large $ssthresh$.
    · Unnecessary retransmissions in a Slow Start based loss recovery.
  * At large windows, many packets can be lost from a window but many packets can be correctly received (an information to exploit).

Problems with a large BDP

* Ideal recovery: Detect and retransmit quickly the losses, reduce the window once for all the losses in a window and enter directly Congestion Avoidance without the need for Slow Start.



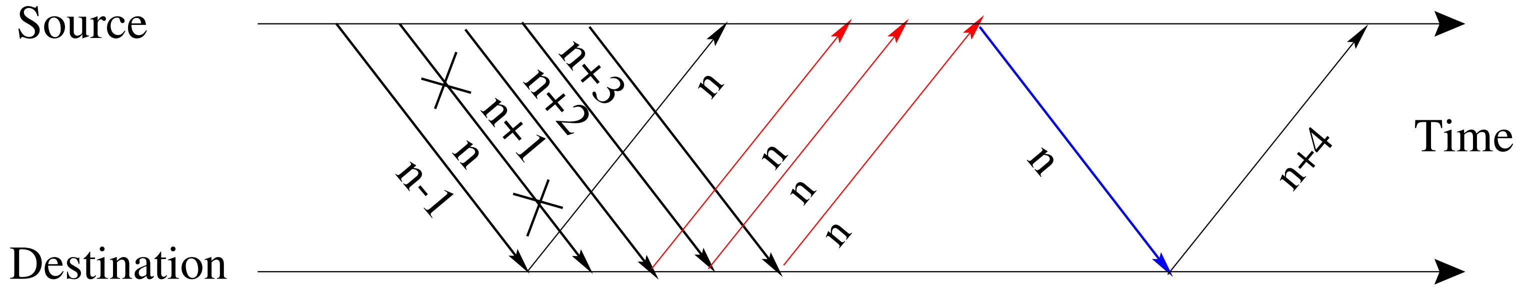
Fast Retransmit algorithm

– At the destination: Send a duplicate ACK for every out-of-order data segment.

– At the source: Consider the receipt of 3 duplicate ACKs as a loss signal.

– The retransmission timer is always used as a final means to detect losses.

– **TCP-Tahoe**: Add Fast Retransmit but keep Slow Start after loss detection.



Fast Recovery algorithm

– Called after Fast Retransmit to recover from losses without slow starting.

– While recovering, it tries to not drain the pipe by estimating the output rate.

– According to CA, the pipe must not contain more than $ssthresh$ packets.

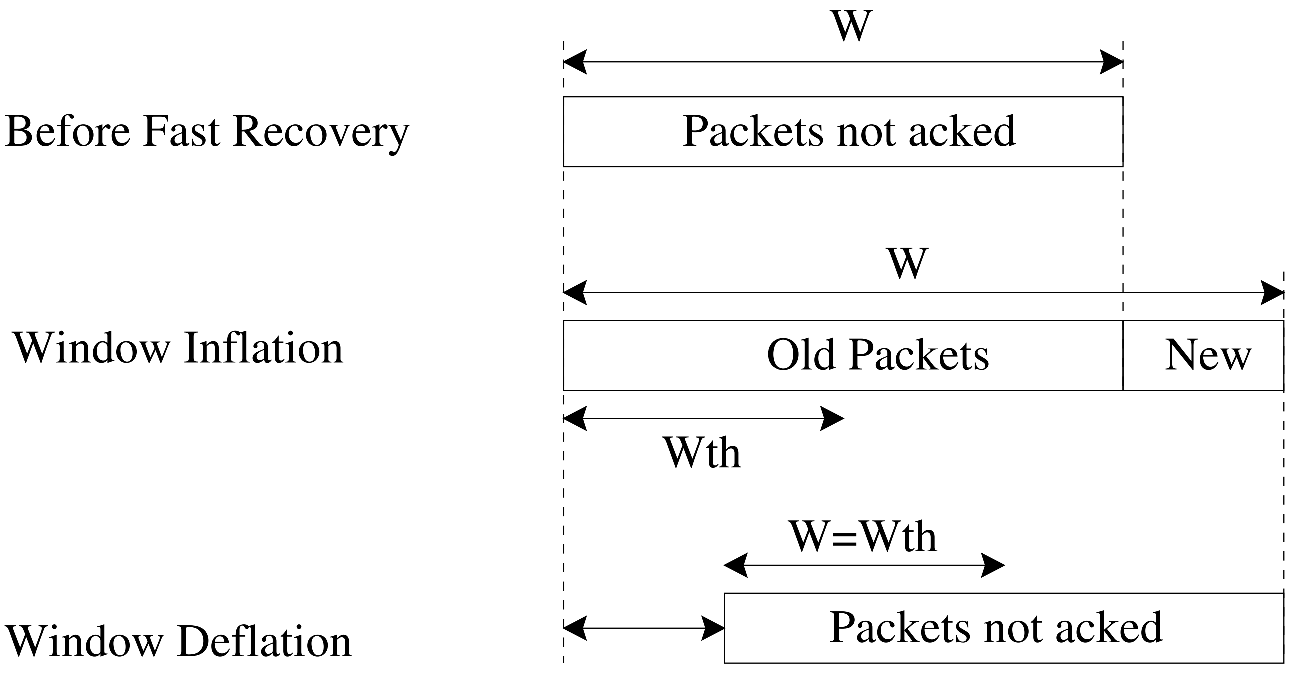
– In addition to retransmissions, new data is sent when the pipe size estimate fells below $ssthresh$.

– In case of failure (e.g. stop of the ACK clock), timeout occurs followed by a Slow Start.

The different versions of TCP differ in their Fast Recovery phase ...

TCP-Reno

Main idea: Consider a duplicate ACK as a signal that one packet has left the network.



TCP-Reno

- Upon loss detection by Fast Retransmit:
  - Set W_{th} to $W/2$.
  - Set W to $W_{th} + 3$ (to account for the 3 duplicate ACKs).
  - Retransmit the lost packet.
- When a duplicate ACK is received:
  - $W = W + 1$ (window inflation).
  - Send a new packet if the window allows.
- When a new ACK is received:
  - Reduce W to W_{th} (window deflation).
  - End Fast Recovery.

TCP-Reno

- Advantage: A perfect recovery when one packet is lost from a window.
- Problems:
  * Unable to recover faster than one loss per RTT.
  * Every loss in a window causes a reduction of W_{th} (the estimate) by 2.
  * Unable to recover from more than two losses from the same window.
    The ACK clock stops and timeout and Slow Start are required.
    Tahoe has been proven to give better performance in case of bursty losses.
  * Sensitivity to the loss of ACKs : Overestimation of the number of packets in the network.
    Result: Burstiness, network drain and possible failure of the recovery.

TCP-New-Reno

– Objective: A solution to the problem of multiple losses from the same window.

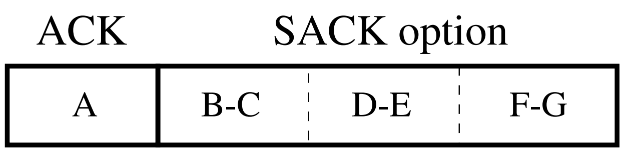
– When Fast Recovery is called, it records in $recover$ the highest sequence number transmitted.
The receipt of an ACK less than $recover$ (called **partial** ACK) is considered as a loss indication.
This loss is retransmitted directly without waiting for 3 duplicate ACKs and without reducing W_{th}. Fast Recovery ends when $recover$ is acknowledged.

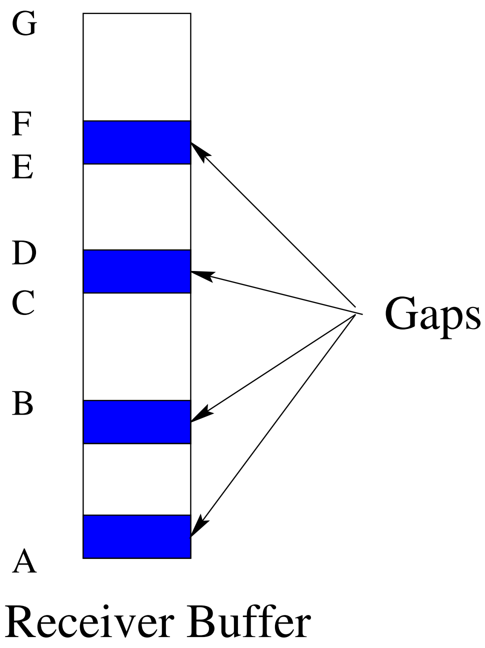
– Advantage: Able to recover from many losses if retransmissions are not lost.

– Problem: As Reno, it cannot recover from more than one loss per RTT.

Selective ACK

SACK supplies the source with the packets in the receiver buffer.
It reports the three most recently received blocs of contiguous data.





Selective ACK

- With this information, the sender can:
  * Recover faster than one loss per RTT.
  * Avoid the retransmission of packets already received.
  * Estimate more correctly the number of packets in the pipe (to keep it as much as possible equal to W_{th}).

- It must be combined with a retransmission policy at the sender.

Many algorithms have been proposed to exploit this information ...

Some Algorithms

– The difference is in the estimation of the number of packets in the pipe ($pipe$). A retransmission or a new data is sent if $pipe$ is less than W_{th}.

– TCP-SACK:

  * $pipe- = 1$ when an duplicate ACK is received.
  * $pipe- = 2$ when a partial ACK is received.
  * $pipe+ = 1$ when a packet is transmitted.
  * Drawbacks:
    An underestimation of $pipe$ if some ACKs are lost.
    The result is an underutilization of network resources and burstiness.

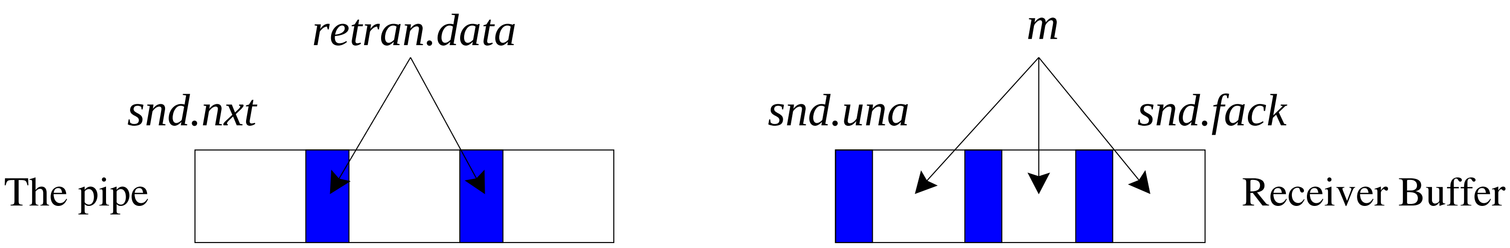
Some Algorithms

– **Forward ACK**: $pipe = snd.nxt - snd.fack + retran.data$

– **Total ACK**:

∗ ACK=CACK + number of packets in the receiver buffer (m).

∗ $pipe = snd.nxt - (snd.una + m)$



Some Algorithms

- Forward ACK:
  Solves the problem of sensitivity of TCP-SACK to the loss of ACKs.
  But results in a underestimation of $pipe$ (overload on the network).

- Total ACK:
  The same value estimated by TCP-SACK but robust against ACK loss.

- $snd.nxt$: The highest sequence number transmitted.
  $snd.fack$: The highest sequence number received.
  $retran.data$: The number of retransmissions.
  $snd.una$: The last acknowledged sequence number.

Problems due to TCP header

Limitation on the window size:

– $rwnd$ is coded in the TCP header on 16 bits. This gives a maximum limit on the window of 64 KBytes.

– For a given RTT (in seconds), this limits the throughput to $524288/RTT$ bps (934 Kbps for a satellite link of 0.56 s RTT).

– Solution: A Window Scale Option that carries a number (on 14 bits) to be multiplied with the original window field. This let $rwnd$ grow up till 2^{30} Bytes.

Problems due to TCP header

Limitation on the sequence space:

– TCP assumes that the maximum life of an IP packet is 2 minutes (MSL).
  The same sequence number is not used within twice this period.
  Sequence number on 32-bit $\Rightarrow$ throughput limited to 143 Mbps.

– Solution:
  Time-stamps option with a PAWS (Protect Against Wrapped Sequence Numbers) algorithm at the receiver.
  A received packet is discarded if it is sent before the last in-sequence one.

Losses during Slow Start

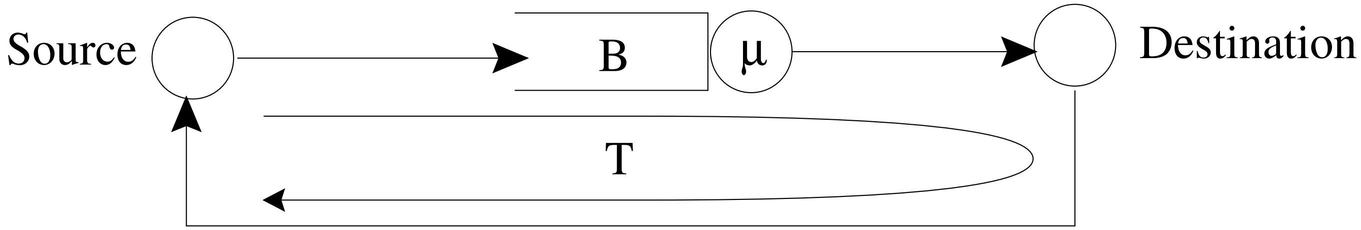
– Many losses appear during Slow Start if at the beginning of the connection the network capacity has been overestimated.

– Result: Failure of even the most intelligent Fast Recovery algorithm.

– Solution: Estimate the capacity via additional mechanisms.

– Proposition: Use the ACK clock at the beginning of Slow Start to estimate the BDP and then set $ssthresh$ to this value.

Buffering requirements



- Congestion avoidance
  * TCP window varies between $(B + \mu T)/2$ and $(B + \mu T)$.
  * TCP throughput equals μ when $W > \mu T = BDP$.
  * Thus, a bandwidth utilization $\simeq$ 100% can be obtained if $B > \mu T$.

Buffering requirements

– Slow Start (SS)

∗ Due to Slow Start burstiness, a small B may overflow before reaching W_{th} even if W_{th} is correctly set.

∗ Result: Early buffer overflow, underestimation of the network capacity and throughput deterioration.

∗ In case of Tahoe, multiple consecutive Slow Start phases have been discovered.

∗ For a given increase rate during Slow Start, a minimum buffer size is required.

∗ But if the buffer is small ...

Solutions to losses during SS

- Reduce the Slow Start threshold. This may improve the performance in some cases (not very small buffers).
- Space the packets during Slow Start. Sending them at approximately μ solves always the problem.
- Decrease continuously the window increase rate during Slow Start. Solves the problem while preserving the ACK clock.

Effect of long RTT

A long RTT is typical of satellite links (250 ms one-way for GEO):

- More time to recover from losses $\Longrightarrow$ Solution: SACK.
- Affects the three-way handshake procedure required to set up a TCP connection.
  Poor performance especially for short data transfers (e.g. HTTP transactions).
  Solution: Transaction TCP (T/TCP).
- Slows down the window increase.
  This affects mainly the Slow Start phase where the window is small.
  Long time to reach a high rate at the beginning of the connection or after an idle period. The result is a poor performance for short transfers.
  Problem exacerbated with the Delay ACK mechanism.

TCP level Solutions to accelerate W increase

Use of a **Large Window** W_I at the beginning of the connection.

– Slow Start duration is reduced from

$$RTT \log_2(W_{th}) \quad \text{to} \quad RTT(\log_2(W_{th}) - \log_2(W_I))$$

– The window proposed (in MSS) $W_I = \min(4MSS, \max(2MSS, 4380))$.

– Drawback: A large W_I increases the amount of losses.

TCP level Solutions to accelerate W increase

- Objective: Overcome the impact of Delay ACK on the duration of Slow Start.
- Propositions:
  * Delay ACKs only in congestion avoidance and use the standard algorithm.
  * Consider the number of segments acknowledged (Unlimited Byte Counting).
  * Limited Byte Counting : Don't increase W by more than two segments.
- Comparison:
  * (1) gives the best performance but requires a cooperation from the sender.
  * UBC results in the fastest increase but it is too aggressive and too bursty.
  * LBC limits the size of bursts when ACKs are lost.

TCP level Solutions to accelerate W increase

- Objective: Avoid the Slow Start phase by spreading $ssthresh$ packets over the estimated RTT.
- Result: Important gain in performance especially for short transfers.
- Problem: Possible unfairness w.r.t. to standard TCP if $ssthresh$ or RTT are wrongly estimated.
- Proposed implementations:
  * Start with normal Slow Start, estimate $ssthresh$, then spread packets.
  * Fast Start: Spread directly packets using previous values of $ssthresh$ but mark them with low priority in order to not disturb other connections.

An application level solution (XFTP)

Open simultaneously N TCP connections at the application level:

– N times faster window increase (an initial window of N instead of 1).

– Distribution of losses among the N connections (helps the Fast Recovery phase of each connection).

– Smaller window reduction in case of losses (from 1/2 to $(N - \frac{1}{2})/N$ in case of 1 loss).

– Drawback: A large N increases the burstiness and then the losses.
An adaptive algorithm has been proposed to control N as a function of RTT.
Can be viewed as an unsocial solution that substitutes a user by N users.

Other Solutions

– Application level:

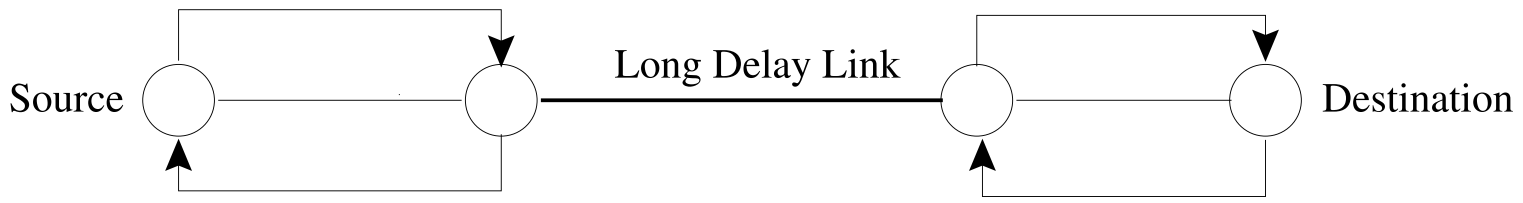
  * Persistent TCP: Combines short transfers into a single one (HTTP 1.1).
  * Caching: Better supported by satellites due to their broadcast nature.

– Lower level solution: Path MTU (Maximum Transfer Unit) Discovery:

  * Find the largest segment size that can cross the path without fragmentation.
  * A large MSS results in a faster growth of the window in terms of bytes.
  * Drawback: An additional delay especially on satellite links.
    But, support of common MTU values reduces this problem.
    And, MTU values can be cached to be used later.

Network level solution

TCP Spoofing (or Split TCP):



Network level solution

– TCP Spoofing:

  * Terminate the TCP connection at the entry of the long delay link (virtual destination).
  * Transmit packets on this link using a well tuned protocol (e.g. STP).
  * If the destination is not located on the output of the link (in general, it is the case), establish another TCP connection to the destination and send the packets (virtual source).
  * The virtual source is responsible for error and congestion control on the right-hand side of the long delay link.

TCP Spoofing : Advantages

- On the two sides of the long delay link: Faster window increase due to a shorter RTT per connection.
- On the long delay link: Design of a link-specific transport protocol able to use efficiently (good utilization and fairness) the available bandwidth and to avoid the long Slow Start phase of TCP.
- Possible use of an another TCP better suited to the right-hand side of the link (e.g. case of a wireless network).
- Better reaction to congestion on the two sides of the link (due to a shorter feedback delay).

TCP Spoofing : Drawbacks

– Break the end-to-end semantic of the Internet (data is acknowledged before reaching the receiver).

– Vulnerable to path changes and router crashes.

– Don't function when IP packets are encrypted.

– Require symmetric paths (a solution is to use IP-tunneling between the destination and the output router).

Impact of RTT on retransmission timer precision

– Standard TCP updates the RTT estimate (and the variance) once per RTT.

– A long RTT impairs the accuracy of the retransmission timer. Long time is required to track any change in the end-to-end delay.

– Solution: The Time-stamps option.

  * Stamp every data packet.
  * Echo the stamp in the ACK.
  * Measure the RTT and update the estimate for every received ACK.

Impact of RTT fluctuations

- Mobile networks, Satellite constellations, Path changes.
- Invalidate the current estimate of the RTT.
- Results: Spurious retransmission, unnecessary window reduction, Slow Start and a Go-Back-N retransmission (all the packets in the window are retransmitted in an aggressive manner).
- Solutions:
  * Make TCP conservative by setting a large minimum value for the retransmission timer (one second currently).
  * Use of the Time-stamps option (the Eiffel algorithm).

Impact of RTT on TCP Fairness

– An AIMD flow control is fair if throughputs of the different connections increase at the same rate.

– Problem of TCP: Throughput increase rate of a connection is inversely proportional to the RTT.
Result: TCP favors connections with small RTT.

– Some mathematical results:

  * In a synchronized environment (e.g. case of a drop-tail buffer), the average throughput of a connection is proportional to $1/T^\alpha$ with $1 < \alpha < 2$.
  * In a non-synchronized environment (e.g. RED buffer), the average throughput is proportional to $1/T$.

Network level solutions

– Implementation of mechanisms in routers to enforce fairness.

– Required mechanism: An intelligent packet-drop policy that distributes drops between flows exceeding their fair share.

– Propositions:

  * General drop policies (e.g. Random Early Detection, Random Drop):
    · Apply the same algorithm (e.g. random drop) to all incoming packets without accounting for the buffer occupancy of the corresponding connection.
    · Improve the performance but shown to be not enough especially in presence of non-responsive flows.

Network level solutions

* Per-connection drop policies (e.g. Flow RED, Longuest Queue Drop, Virtual Queuing):
  - Main idea: Guarantee a minimum number of places per-connection to protect low rate connections from aggressive ones.
  - Result: In the absence of any information on the rates of the different connections, fair buffer sharing is the most important mechanism for an isolation of flows and a fair sharing of bandwidth.

TCP level solutions

– Idea : Change the window increase algorithm at the source during Congestion Avoidance to make it more aggressive for long delay connections.

– Constant Rate window increase algorithm:

  * The window is increased by $W = W + (c * RTT * RTT)/W$.
  * Result: All the throughputs increase at a constant rate c.
  * Problems: How to determine c and burstiness for long RTT.

TCP level solutions

– Increase By K algorithm:

  * Increases W by K segments every RTT when RTT is long.
  * Can be implemented at the source by $W = W + K/W$.
  * Or at the receiver by acknowledging data in smaller chunks.
  * Less burstiness than CR algorithm and better performance in case of drop tail buffers.

TCP in a lossy environment

– The main idea behind TCP: Create losses in order to detect congestion (no explicit information sent by the network).

– But, on some unreliable paths (e.g. wireless links with high BER or weak link-level error recovery), losses can appear at the link-level due to many phenomena other than congestion (e.g. corruption, disconnection, path changes).

– Problem of TCP: Considers any loss as a congestion signal and reduces its throughput which results in a poor performance if non-congestion losses are frequent.

TCP in a lossy environment

– In a lossy environment where packets are lost w.p. p:

$$\text{Average throughput} \propto 1/(RTT\sqrt{p})$$

– Solutions:

  * Hide lossy links from the sender (requires no modification to existing TCP). It is equivalent to cleaning Internet links in order to keep losses in routers.
  * End-to-end solutions : Enhance TCP with additional mechanisms to reduce the impact of non-congestion losses.

How to hide lossy links?

Link-level error recovery: Local Retransmissions (ARQ):

- Efficient when the link is not very lossy (extra bandwidth consumed only upon retransmission) and the RTT is not very long.
- Drawback: TCP is not fully shielded:
  - Interaction between TCP timer and ARQ timer.
  - Duplicate ACKs (case of an out-of-order link layer).
- Solutions:
  - Limit the number of retransmissions.
  - Suppression of duplicate ACKs (a TCP-aware protocol).

How to hide lossy links?

Link-level error recovery: Forward Error Correction (FEC):

– Together with data, redundant informations are transmitted over the lossy link to enable the reconstruction of errors at the output of the link.

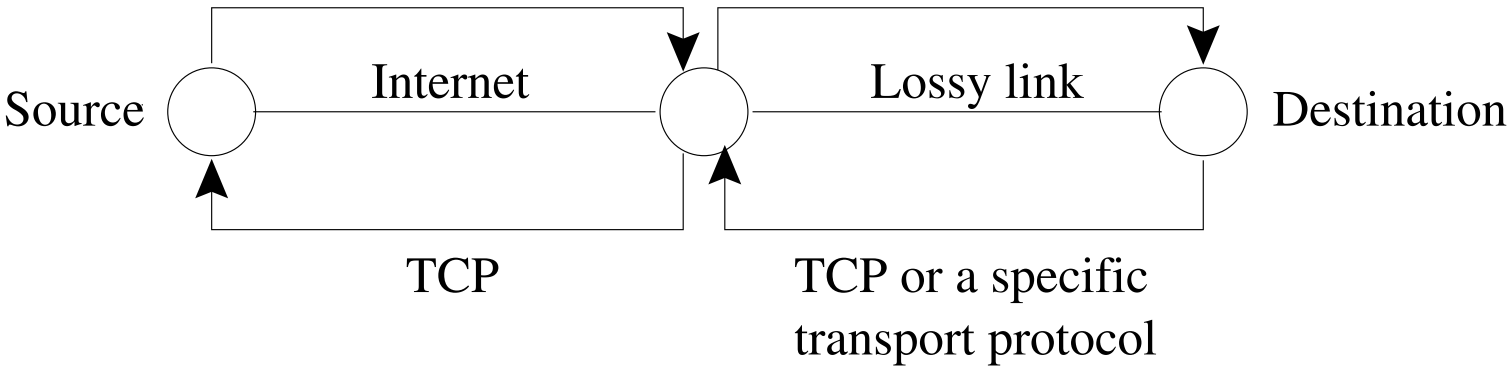
– Convenient when the RTT is long (i.e. no retransmission) and when losses are frequent. It shields completely the sender.

– Drawbacks:

 * Bandwidth consumption and coding/decoding overhead.
 * Sensitivity to error burstiness (alleviated by interleaving the data after the addition of FEC).

How to hide lossy links?

TCP-level solutions: Split TCP (e.g. Indirect TCP I-TCP):



How to hide lossy links?

TCP-level solutions: Split TCP (e.g. Indirect TCP I-TCP):

– The source connection is terminated at the input of the lossy link.

– A protocol well tuned to a lossy environment is used on the lossy link (an enhanced version of TCP (e.g. TCP-SACK) or a specific transport protocol).

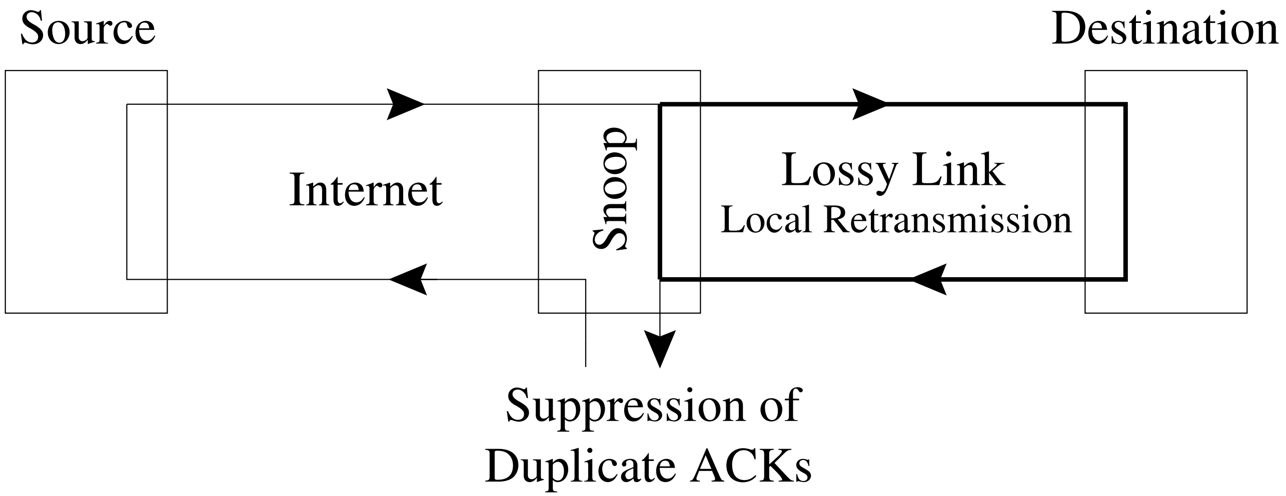
– Drawbacks:

  * Violation of the end-to-end semantics.
  * Difficult handover in case of a wireless network due to the important amount of data in Base Stations (intermediate router).

How to hide lossy links?

TCP-level solutions: Snoop protocol:

– Objective: Solve the end-to-end problem of Split TCP.



How to hide lossy links?

- Snoop protocol:
  - ∗ An agent at the input of the lossy link monitors packets in both directions.
  - ∗ It stores TCP packets to retransmit them later on behalf the source.
  - ∗ And it stops duplicate ACKs to not trigger a fault congestion signal.
  - ∗ Packets are retransmitted locally when three duplicate ACKs are received or a local Timeout expires.
- Remarks:
  - ∗ Can be considered as a link layer protocol aware of TCP packets.
  - ∗ Requires that no congestion losses exist between the Snoop agent and the destination (i.e. the lossy link forms the last hop).

End-to-end solutions

Two trends:

– Help the source to distinguish non-congestion losses from congestion ones in order to react differently.

  * Explicit indication (ELN, ICMP messages).
  * Implicit indication (loss predictors).

– Decouple congestion detection from loss detection. Congestion losses are reduced due to early congestion detection which facilitates the task of loss distinguishing at the source.

  * Explicit indication (ECN).
  * Implicit indication (Vegas).

Explicit Indication of type of losses

- Explicit indication of corruption losses:
  - The receiver sends an Explicit Loss Notification message (TCP option) to the source indicating that a packet is lost due to corruption.
  - The source retransmits the loss without reducing the window.
  - Problems:
    - Corrupted packets are discarded before reaching the TCP layer (possible corruption of the TCP header).
    - The receiver may not be the last hop to the destination.

Explicit Indication of type of losses

* Solutions:
  - Protect the TCP header with FEC or send the message by the input of the lossy link (problem of asymmetric paths).
  - Send a corruption-experienced ICMP message to the source if the lossy link is not the last hop (NASA SCPS-TP).
  - Use the inter-packet arrival time to infer the type of the loss.

– Explicit indication of packet drop:

* Use of the Source Quench ICMP message sent by a router upon drop.
* Problem: Message loss.

Implicit Indication of type of losses

– Loss predictors: Try to predict the type of the next loss from measurements of the window and RTT without any additional feedback.

– Idea: The network starts to be overloaded when an increase in the window results in a increase in RTT.

– Three predictors are used: Vegas, Normalized Throughput Gradient and Normalized Delay Gradient.

– Results: Best performance for Vegas but in general not promising since the network reaction (RTT) is usually independent of the window size.

Explicit indication for early congestion detection

Explicit Congestion Notification proposal:

- Combined with an early congestion detection buffer (e.g. RED).
- Instead of dropping a packet when the congestion is not serious, set the Congestion Experienced bit in the IP header.
- The receiver echoes the CE bit in a flag in the TCP header.
- The source reduces its window when a notification is received.
- But it continues to reduce its window when a packet is lost.

Losses due to Disconnections

– A typical problem in a mobile environment (handover, call blocking, obstacles).

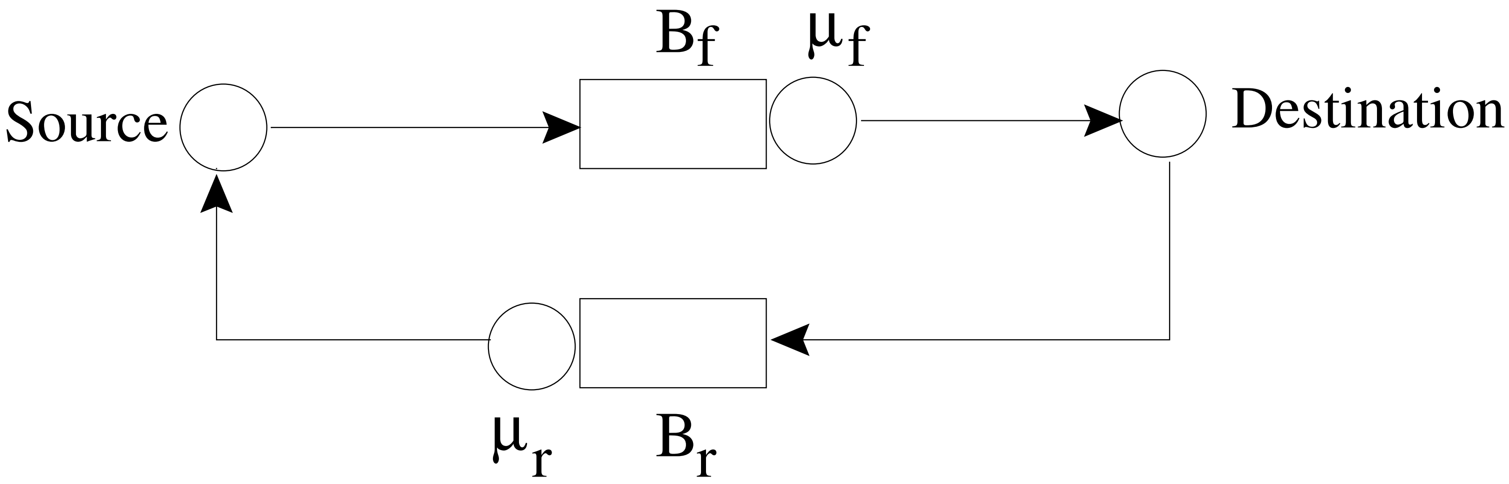
– Impact on TCP:

  * Force the source to timeout and to close its window due to a stop of the ACK clock.
  * Serial timeouts in case of frequent disconnections (or long disconnection).
    · The source backs off its retransmission timer (up to 64s) which results in a long waiting time before the flowing of data once the mobile is reconnected.
    · A reduction of the Slow Start threshold to a very small value.

Losses due to Disconnections

- Solutions:
  - Idea: Stop the congestion control at the source when disconnection appears and awake the source when it disappears.
  - Implementations:
    - M-TCP: Keep always an unacknowledged byte at the base station, close the sender window upon disconnection (by acking the stored byte with a zero receiver window), then reopen it.
    - SCPS-TP: Stop the source with a link-outage ICMP message and reopen it with a link-restored ICMP message.
    - The receiver triplicates the last ACK it has sent to avoid the long Timeout.

TCP and asymmetric paths



$$K = \frac{\mu_f}{\mu_r} > 1$$

TCP and Asymmetric paths

– Bandwidth asymmetry: The reverse path is so slow so that

$$\frac{\text{Forward rate}}{\text{Data packet size}} \times \frac{\text{ACK size}}{\text{Reverse rate}} > 1$$

Problem exacerbated in case of two-way traffic.

– Problems:

∗ Congestion on the reverse path due to a high ACK rate.
∗ Interactions between forward and reverse connections.

Reverse path congestion

- Results:
  * Increase in the end-to-end delay:
    - Less throughput for a given window.
    - Slower window growth (disastrous for satellite links).
  * Loss of ACKs (one ACK over K reach the source):
    - Slower window growth.
    - Burstiness at the sender (a $B_f > K$ is required).
    - Problems with the Fast Recovery algorithm.

Reverse path congestion

- Solutions to:
  * Congestion (Receiver side):
    - TCP/IP header compression (e.g. SLIP).
    - Decreasing the ACK frequency (ACK Congestion Control).
    - ACK filtering in B_r (a new ACK is inserted in place of an old one).
  * Burstiness (Sender side):
    - Limiting the size of bursts (spacing the bursts).
    - Reconstructing the ACK stream at the output of the slow link.

Case of multiple connections

– Interaction between multiple forward connections:

* This arises the problem of fairness in sharing the reverse slow channel between the ACKs of the different connections.
* The running connections overflows the reverse buffer B_r with their ACKs.
* A new connection finds a problem to increase its window due to the loss of its first ACKs (timeouts and slow window increase during Slow Start). It remains blocked until the dominant connections reduce their throughput.
* Solution: Intelligent management of buffer B_r that improves fairness.

Case of multiple connections

– Interaction between forward and reverse connections:

  * This arises the problem of fairness in sharing the reverse channel between data packets and ACKs.
  * Also, ACKs wait long time in B_r behind data packets (data packets can be 20 times larger than ACKs). This waiting leads to an increase in RTT of forward connections and to burstiness at the source.
  * Solution: An intelligent management of buffer B_r to guarantee fairness in bandwidth sharing and an intelligent scheduling of data packets and ACKs to reduce the waiting time for ACKs (e.g. Weighted Round Robin).

Conclusions

– Main problems of TCP to be solved:

  * Reliance of the transmission on the ACK clock.
  * Coupling between congestion and error detection.

– Beliefs:

  * Packet spacing is unavoidable for TCP operation in extreme conditions (satellite links, asymmetric paths).
  * End-to-end detection of non-congestion losses is difficult without any feedback from the network. The open question is which is better, to split the connection and to keep the source unchanged or to solve the problem on end-to-end by changing the source and adding the required signaling.

References

[ALL98] M. Allman, "On the Generation and Use of TCP Acknowledgments", *ACM Computer Communication Review*, Oct 1998.

[RFC2414] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's Initial Window", *RFC 2414*, Sep 1998.

[RFC2488] M. Allman, D. Glover, and L. Sanchez, "Enhancing TCP Over Satellite Channels using Standard Mechanisms", *RFC 2488*, Jan 1999.

[AKO96] M. Allman, H. Kruse, and S. Ostermann, "An Application-Level Solution to TCP's Satellite Inefficiencies", *First International Workshop on Satellite-based Information Services (WOSBIS)*, Nov 1996.

[ABN95] E. Altman, J. Bolot, P. Nain, D. Elouadghiri- M. Erramdani, P. Brown, and D. Collange, "Performance Modeling of TCP/IP in a Wide-Area Network", *INRIA Research Report N=3142*. (available in http://www.inria.fr:80/RRRT/RR-3142.html). A shorter version in: *34th IEEE Conference on Decision and Control*, Dec 1995.

[AD98] M. Aron and P. Druschel, "TCP: Improving Startup Dynamics by Adaptive Timers and Congestion Control", *Rice Computer Science*, Technical Report TR98-318.

[BB95] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts", *15th International Conference on Distributed Computing Systems (ICDCS)*, May 1995.

[BPSK96] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. Katz, "A comparison of Mechanisms for Improving TCP Performance over Wireless Links", *ACM Sigcomm*, Aug 1996.

[BPK97] H. Balakrishnan, V. Padmanabhan, and R. Katz, "The Effects of Asymmetry on TCP Performance", *ACM Mobicom*, Sep 1997.

[BA00] C. Barakat and E. Altman, "Performance of Short TCP Transfers", *Networking 2000 (Performance of Communications Networks*, May 2000.

[BAD00] C. Barakat, E. Altman, and W. Dabbous, "On TCP Performance in a Heterogeneous Network : A Survey", IEEE Communications Magazine, Jan 2000.

[BCDA98] C. Barakat, N. Chaher, W. Dabbous, and E. Altman, "Improving TCP/IP over Geostationary Satellite Links", *IEEE Globecom*, Dec 1999.

[BV98] S. Biaz and N. H. Vaidya, "Distinguishing Congestion Losses from Wireless Transmission Losses: A Negative Result", *Seventh International Conference on Computer Communications and Networks (IC3N)*, Oct 1998.

[BV99] S. Biaz and N. H. Vaidya, "Discriminating Congestion Losses from Wireless Losses using Inter-Arrival Times at the Receiver", *IEEE Symposium ASSET*, Mar 1999.

[RFC1644] R. Braden, "T/TCP - TCP Extensions for Transactions: Functional Specification", *RFC 1644*, Jul 1994.

[BP95] L. Brakmo and L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet", *IEEE Journal on Selected Areas in Communications*, Oct 1995.

[BS97] K. Brown and S. Singh, "M-TCP: TCP for Mobile Cellular Networks", *ACM Computer Communication Review*, Oct 1997.

[DMT96] R. Durst, G. Miller, and E. Travis, "TCP Extensions for Space Communications", *ACM Mobicom*, Nov 1996.

[FF96] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", *ACM Computer Communication Review*, Jul 1996.

[FLO91] S. Floyd, "Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic", *ACM Computer Communication Review*, Oct 1991.

[FLO95] S. Floyd, "TCP and Explicit Congestion Notification", *ACM Computer Communication Review*, Oct 1994.

[FJ93] S. Floyd and V. Jacobson, "Random Early Detection gateways for Congestion Avoidance", *IEEE/ACM Transactions on Networking*, Aug 1993.

[RFC2582] S. Floyd and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm", *RFC 2582*, Apr 1999.

[GJKGF99] R. Goyal, R. Jain, S. Kota, M. Goyal, S. Fahmy, and B. Vandalore, "Traffic Management for TCP/IP over Satellite-ATM Networks", *IEEE Communication Magazine*, Mar 1999.

[HK99] T. Henderson and R.H. Katz, "Transport Protocols for Internet-Compatible Satellite Networks", *IEEE Journal on Selected Areas in Communications*, Feb 1999.

[HSMK98] T. Henderson, E. Sahoria, S. McCanne, and R. H. Katz, "Improving Fairness of TCP Congestion Avoidance", *IEEE Globecom*, Nov 1998.

[HOE96] J. Hoe, "Improving the Start-up Behavior of a Congestion Control Scheme for TCP", *ACM Sigcomm*, Aug 1996.

[JAC88] V. Jacobson, "Congestion avoidance and control", *ACM Sigcomm*, Aug 1988.

[RFC1144] V. Jacobson, "Compressing TCP/IP Headers for Low-speed Serial Links", *RFC 1144*, Feb 1990.

[JAC92] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance", *RFC 1323*, May 1992.

[K98] A. Kumar, "Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link", *IEEE/ACM Transactions on Networking*, Aug 1998.

[LMS97] T. V. Lakshman, U. Madhow, and B. Suter, "Window-based error recovery and flow control with a slow acknowledgment channel: a study of TCP/IP performance", *IEEE Infocom*, 1997.

[LM97] T.V. Lakshman and U. Madhow, "The performance of TCP/IP for networks with high bandwidth-delay products and random loss", *IEEE/ACM Transactions on Networking*, Jun 1997.

[LIMO97] D. Lin and R. Morris, "Dynamics of Random Early Detection", *ACM SIGCOMM*, Sep. 1997.

[LK00] R. Ludwig and R. Katz, "The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions", *ACM Computer Communication Review*, Jan 2000.

[MM96] M. Mathis and J. Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control", *ACM Sigcomm*, Aug 1996.

[RFC2018] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options", *RFC 2018*, Oct 1996.

[PK98] V. Padmanabhan and R. Katz, "TCP Fast Start: A Technique for Speeding Up Web Transfers", *IEEE Globecom*, Nov 1998.

[PS97] C. Partridge and T. Shepard, "TCP Performance Over Satellite Links", *IEEE Network*, Sep 1997.

[RFC793] J. Postel, "Transmission Control Protocol", *RFC 793*, Sep 1981.

[RFC2001] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", *RFC 2001*, 1997.

[SLSC98] B. Suter, T.V. Lakshman, D. Stiliadis, and A.K. Choudhary, "Design Considerations for Supporting TCP with Per-flow Queueing", *IEEE Infocom*, Mar 1998.

[VH97] V. Visweswaraiah and J. Heidemann, "Improving Restart of Idle TCP Connections", *Technical Report 97-661*, University of Southern California, Nov 1997.

[WML98] J. Waldby, U. Madhow, and T. V. Lakshman, "Total Acknowledgments: A Robust Feedback Mechanism for End-to-End Congestion Control", *ACM Sigmetrics*, 1998.

[ZDRD97] Y. Zhang, D. DeLucia, B. Ryu, and S. Dao, "Satellite Communications in the Global Internet: Issues, Pitfalls, and Potential", *INET*, Jun 1997.