



Education

TCP/IP Optimization for Wide Area Storage Networks

Dr. Joseph L White
Juniper Networks

- The material contained in this tutorial is copyrighted by the SNIA.

- Member companies and individuals may use this material in presentations and literature under the following conditions:
 - ◆ Any slide or slides used must be reproduced without modification
 - ◆ The SNIA must be acknowledged as source of any material used in the body of any document containing material from these presentations.

- This presentation is a project of the SNIA Education Committee.

TCP/IP Optimization for Wide Area Storage Networks

With all of the interest in using IP storage for servers, business continuance, and disaster recovery, the various IP storage protocols such as iSCSI, iFCP, and FCIP have gained increasing attention from storage networking professionals and administrators. TCP/IP provides the transport for these upper level protocols. Knowledge of TCP/IP and how it applies to block storage and bulk data transfers is critical for optimal deployment of IP storage solutions.

This session provides an overview of TCP/IP as it applies to block storage protocols and bulk data transfers. The session will begin with an overview of TCP/IP and continue with storage networking considerations when using high bandwidth, long latency, congested, or impaired networks. This session will primarily focus on modifications and enhancements used to mitigate these negative effects on TCP/IP performance.

➤ TCP Overview

- ◆ TCP Connections
- ◆ TCP Flow Control
- ◆ Long Fat Networks

➤ TCP Modifications for Block Storage

- ◆ SACK
- ◆ Network Reordering
- ◆ Congestion (packet drop) Recovery Modifications
- ◆ Compression

➤ Upper Layer Protocol Optimizations

- Demanding Requirements in the Data Center
 - ◆ High Throughput
 - ◆ Low Latency
 - ◆ Wide Scalability
 - ◆ Robustness
 - ◆ High Availability

- TCP based Storage Networking protocols must meet the data center requirements across WAN

- Technology pushed by mission critical applications
 - ◆ Data Center deployments have matured into multi-Terabit networks
 - ◆ WAN bandwidth availability is increasing
 - ◆ WAN and distributed applications are more prevalent
 - ◆ Data backup and recovery
 - ◆ Grid computing

TCP Based Storage Networking Protocols

➤ Block Storage Protocols

- ◆ FCIP
- ◆ iFCP
- ◆ iSCSI

➤ Wide Area File Systems (WAFS)

- ◆ NAS (CIFS, NFS)
- ◆ HTTP

➤ Optimizations possible in all layers

- ◆ Modify TCP/IP transport layer itself
- ◆ Optimize session upper level protocol

➤ TCP → TCP WAN acceleration uses many of the same techniques



➤ Connection Oriented Protocol

- ◆ Full Duplex - Byte Stream
- ◆ Port Numbers allow multiple connections between an IP address pair
- ◆ Certain features negotiated at connection initialization

➤ Guaranteed In-Order Delivery

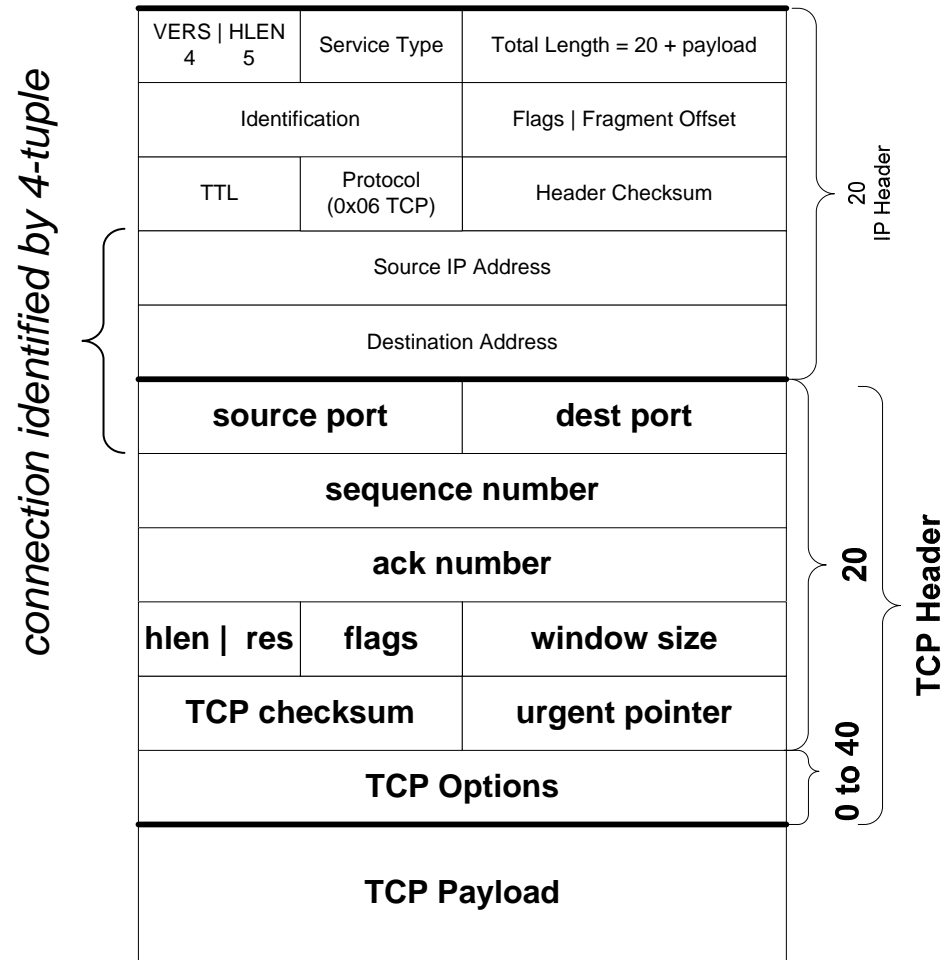
- ◆ Segments carry sequence and acknowledgement information
- ◆ Sender keeps all transmitted data until acknowledged by the receiver
- ◆ Sender maintains send timers and attempts to resend any missing bytes in the sequence

➤ Flow Control and Congestion Avoidance

- ◆ Sender maintains sending limit as a function of trips across the network and as a function of congestion events
- ◆ Receiver maintains a sliding window that is advertised back to the sender

TCP Header

- Source Port Number
- Destination Port Number
- Sequence Number
- ACK Number
- Header Length
- Flags
 - ◆ SYN
 - ◆ FIN
 - ◆ RST
 - ◆ ACK
 - ◆ PSH
 - ◆ URG
- Window Size
- Checksum
- Urgent pointer



➤ Connection Setup

- ◆ Peer to Peer
- ◆ SRC_IP, DST_IP, SRC_PORT, DST_PORT
 - > 4-Tuple is Unique identifier for the connection
 - > Assignment of port numbers: well known vs. allocated
- ◆ Triple handshake
 - > SYN : SYN-ACK : ACK
 - > TCP State Machine handles simultaneous open collisions
- ◆ Operating Parameter Negotiation
 - > TCP Options
 - > Initial Sequence Number

➤ Connection Closure

- ◆ Use FIN to indicate sender done transmitting data
 - > TCP Half Closed connection
 - > A FIN is acknowledged
- ◆ Use RST to reset the connection and force it closed
- ◆ TCP State Machine has close wait times built into it

➤ Keep-alive Timer

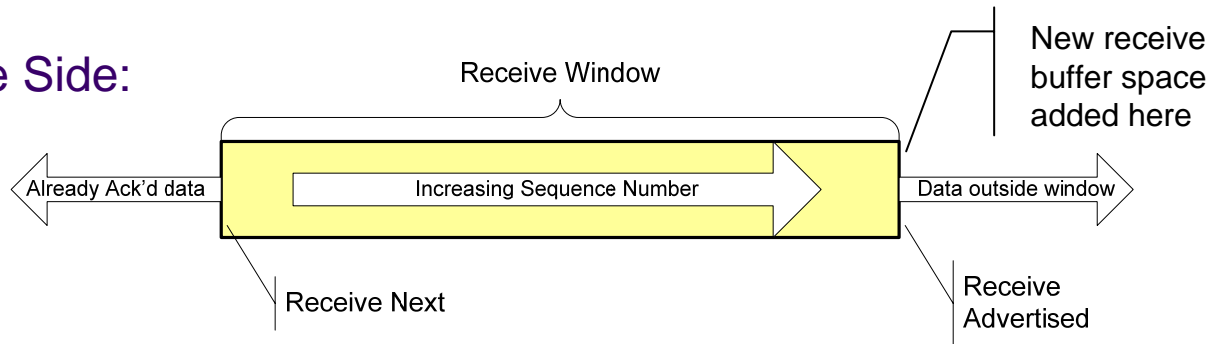
- ◆ Detects connections that have lost the other side but are otherwise idle (at 2 hour intervals)

➤ MTU Discovery

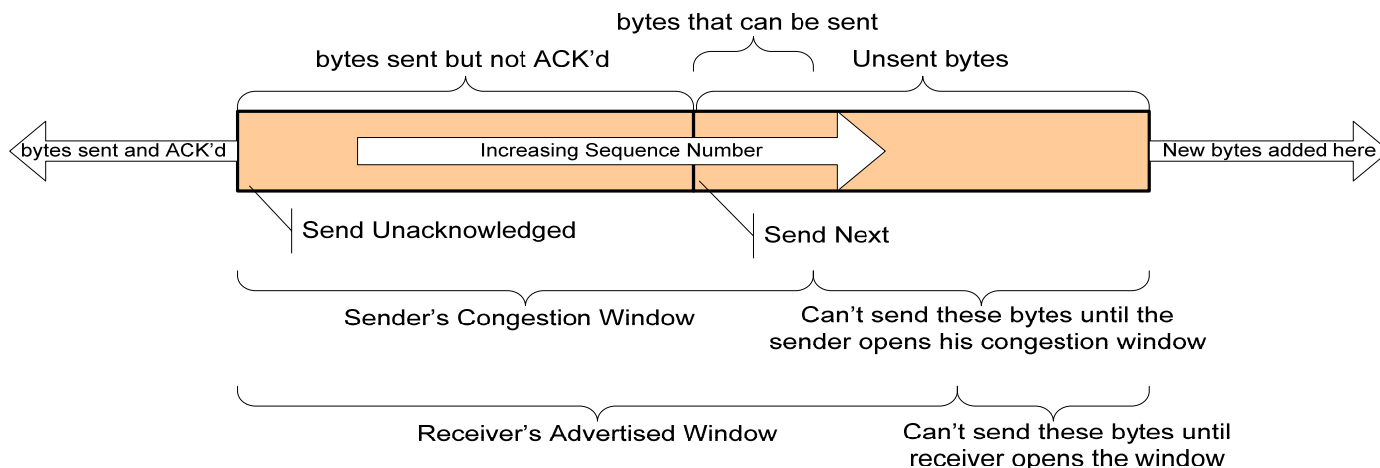
- ◆ Packets probe the network path to determine maximum packet size

TCP Sliding Window and Flow Control

Receive Side:



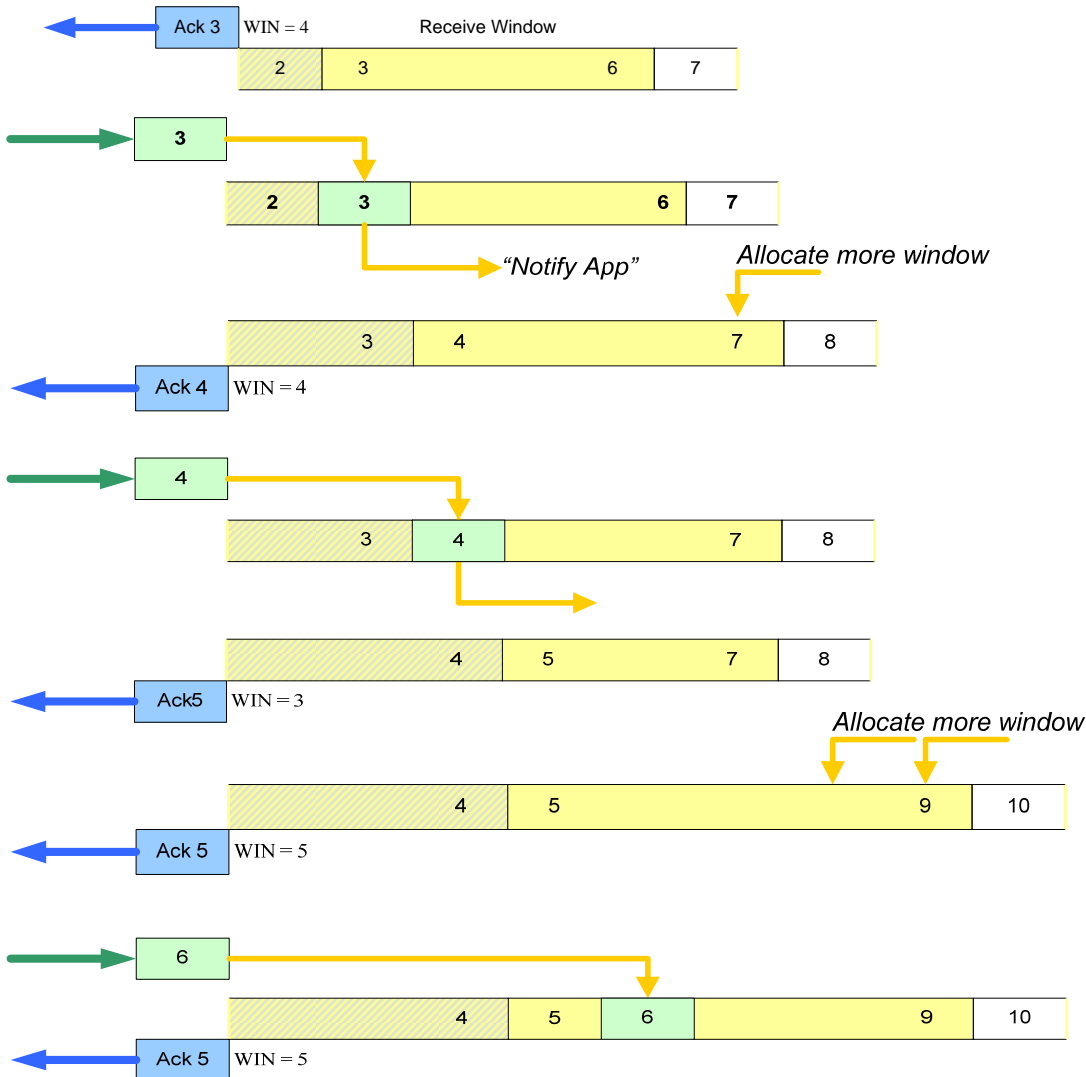
Transmit (Sender) Side:



- Sliding window protocol
 - › Receiver advertises a window to the sender
 - › Receiver performs Out of Order reassembly
 - › Duplicate segments or out of window segments detected and discarded
 - ◆ Avoiding silly window syndrome
 - › Smallest advertised size is minimum of one segment or $\frac{1}{2}$ actual receive buffer, a zero sized window advertised otherwise

- Persist timer maintained by sender
 - › Generates a window probe on timeout
 - › Used to recover lost window updates
 - › If advertised window is zero, send 1 byte to get the other side to send back an ACK with a window update.
 - › Similar back-off scheme to the retransmission timer (5-60 seconds)

TCP Sliding Receive Window Illustration



Everything previous ACK'd
And ready to receive...

TCP Segment Received

Receive Buffer allocated
and ACK sent

TCP Segment Received

Receive Buffer NOT allocated
But ACK still sent

Receive Buffer added
causes Window Update
(not considered a duplicate ACK)

Out of Order segment
Generates a Duplicate ACK

- Data Stream chopped into chunks (segments)
 - ◆ Sent as IP Datagrams
 - ◆ Segments protected by TCP checksum

- Sender maintains a Congestion Window (cwnd)
 - ◆ Uses AIMD (Additive Increase, Multiplicative Decrease) algorithm
 - ◆ Highest sequence number that can be sent is last ACK + MIN (cwnd, rwnd)

- Slow Start
 - ◆ Rate of packet injection into the network equals to rate which ACKs are received
 - ◆ Leads to exponential sender cwnd ramp until:
 - > A congestion event occurs
 - > The limit of the receiver's advertised window is reached
 - > The limit of the sender's un-acknowledged data buffering is reached
 - > The limit of the network to send data is reached (network saturation)

- Nagle Algorithm (RFC 896)
 - ◆ don't send less than one segment of data unless all sent data has been ACK'd

- ◆ ACK number is next expected sequence number
 - › This is the last sequence number received + 1
- ◆ Segments must be in order to advance ACK Number
 - › SACK allows additional information to be sent
- ◆ Duplicate ACK
 - › Sent when segment is received out of order
 - › May indicate a missing segment to the sender
- ◆ Delayed ACK
 - › Do not send an ACK right away, wait a short time to see if Additional segments arrive which can also be ACK'd
- ◆ ACK value valid in a Data Segment
 - › If no data, the segment is often called a 'Pure ACK'
- ◆ ACK/N
 - › *one possible modification*
 - › Wait for several (N) segments to arrive before sending an ACK
 - › Send anyway after a short time interval

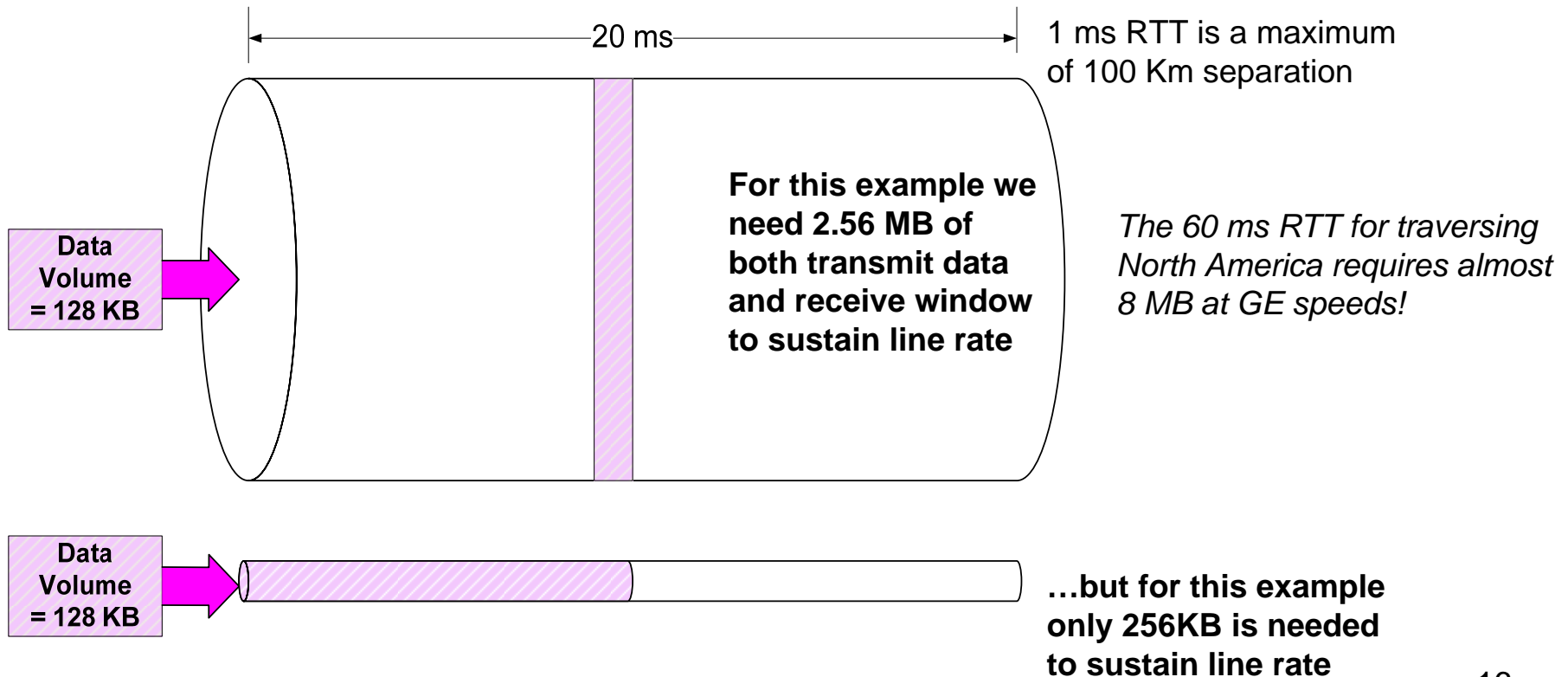
- TCP can only move one window's worth of data per RTT
 - ◆ On the receive side the buffering must be allocated to the connection
 - ◆ On the transmit side the data must be held until ACK'd
 - ◆ Data segment must travel to the receiver
 - ◆ ACK for that segment must travel back

- TCP Basic Window Limits
 - ◆ TCP header contains 16-bit advertised window size in units of bytes
 - ◆ Allows a 64KB-1 maximum window size

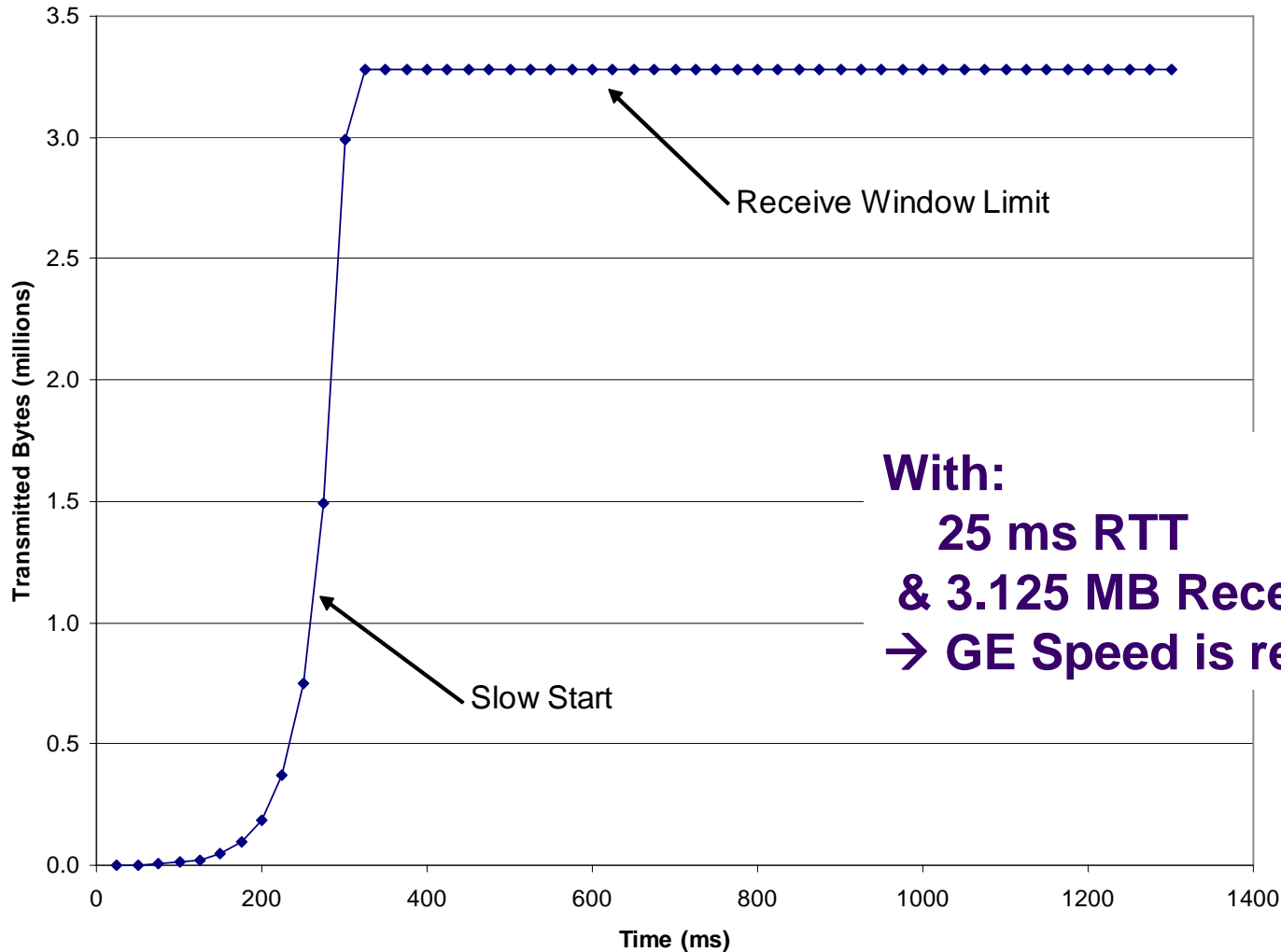
- Scaled Window TCP Option
 - ◆ During connection setup a scale factor is negotiated in each direction.
 - ◆ The scale factor is 1 byte in size with a maximum allowed value of 14.
 - ◆ Each 'count' in the advertised window field is worth $2^{\text{scaleFactor}}$ bytes.
 - ◆ The resolution for window advertisement is reduced but in practice this is not important since the scale factor does not have to be very large to reach a large maximum window size.
 - ◆ For example, with a scale factor of 10, the window field carries KB of available window with a maximum allowed window of 64 MB.

Long Fat Networks (LFN)

- LFNs are characterized by a large bandwidth-delay product
 - The bandwidth-delay product is the amount of data that the network can hold between sender and receiver (= transmission rate x RTT latency)
 - LFNs therefore place large buffering demands TCP/IP WAN connections
 - At 1Gb/s = 128 KB buffering and data needed per 1 ms of RTT



Receive Window Plateau



With:
25 ms RTT
& 3.125 MB Receive window
→ GE Speed is reached

➤ Timestamp Option

- ◆ enables PAWS
- ◆ allows RTT calculation on each ACK instead of once per window
- ◆ produces better smoothed averages
- ◆ Timer rate guidelines: $1 \text{ ms} \leq \text{period} \leq 1 \text{ second}$

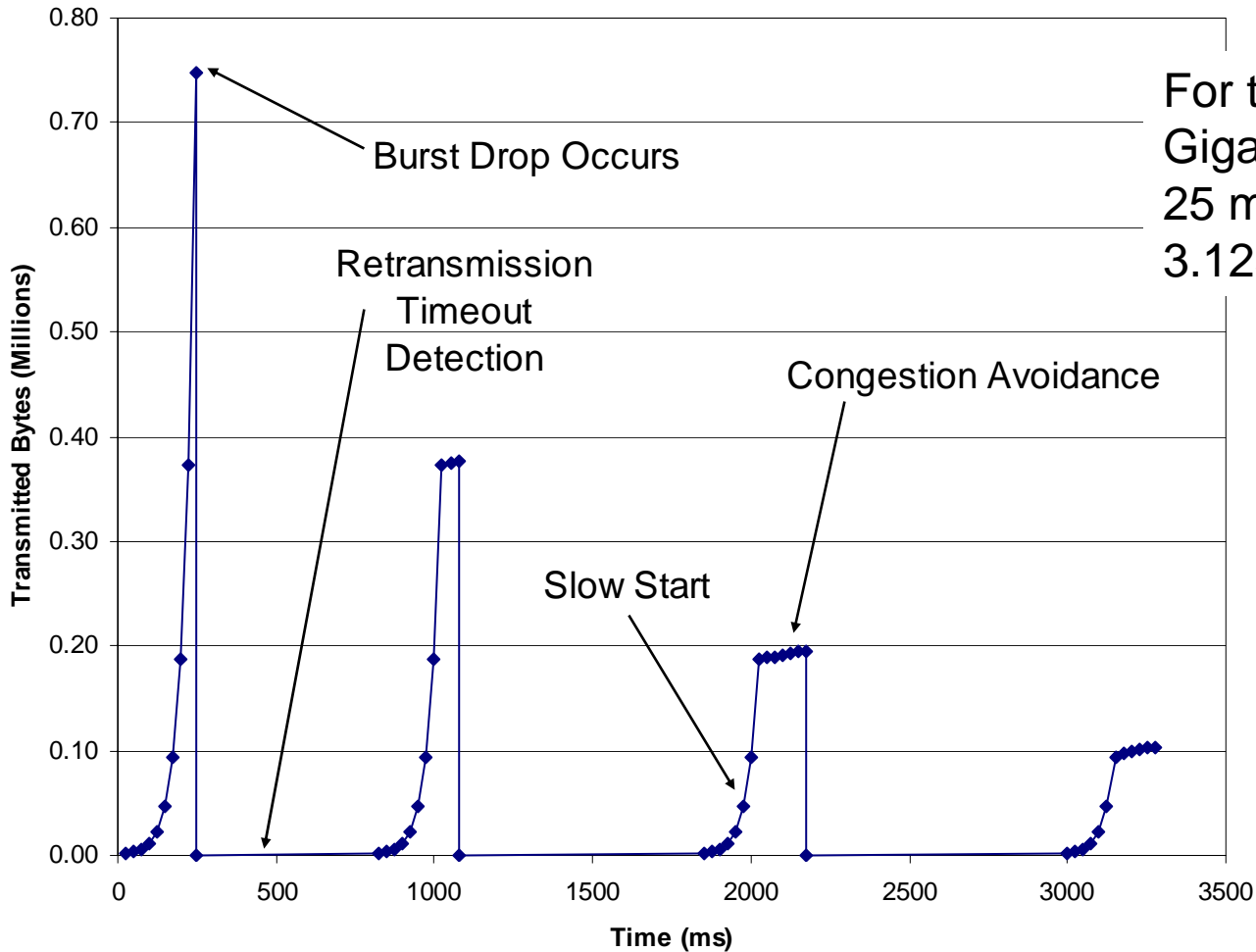
➤ PAWS: protection against wrapped sequences

- ◆ In very fast networks where data can be held, protects against old sequence numbers accidentally appearing as though they are in the receiver's valid window
- ◆ uses timestamp as 32-bit extension to the sequence number
- ◆ requires that the timestamp increment at least once per window

- Packet (segment) loss can occur for several reasons
 - ◆ Congestion
 - ◆ Ethernet/IP proactive flow control schemes (RED)
 - ◆ Faulty equipment
 - ◆ Uncorrectable bit errors

- When packet loss does occur it can be extremely detrimental to the throughput of the TCP connection
 - ◆ Extent of disruption determined by the pattern of drops
 - ◆ There are TCP features and modifications which mitigate effects of packet loss

TCP Retransmission Timeouts



For this example:
Gigabit Ethernet Speed
25 ms RTT
3.125 MB Receive window

➤ Retransmission timer

- ◆ times oldest sent, unacknowledged data
- ◆ cancelled and restarted on non-duplicate ACK
- ◆ used to detect packet loss

➤ Requires RTT estimation for connection

- ◆ smoothed estimator updated once per round trip
- ◆ timestamp option allows RTT estimate for each segment
- ◆ RTT estimate used to set the timeout expiration value
- ◆ Karn's algorithm
 - cancel RTT estimation during retransmission timeout
- ◆ Can use better resolution timer (1ms, 10 ms) than normal TCP implementation's 500ms

➤ Multiple Retransmissions and connection closure

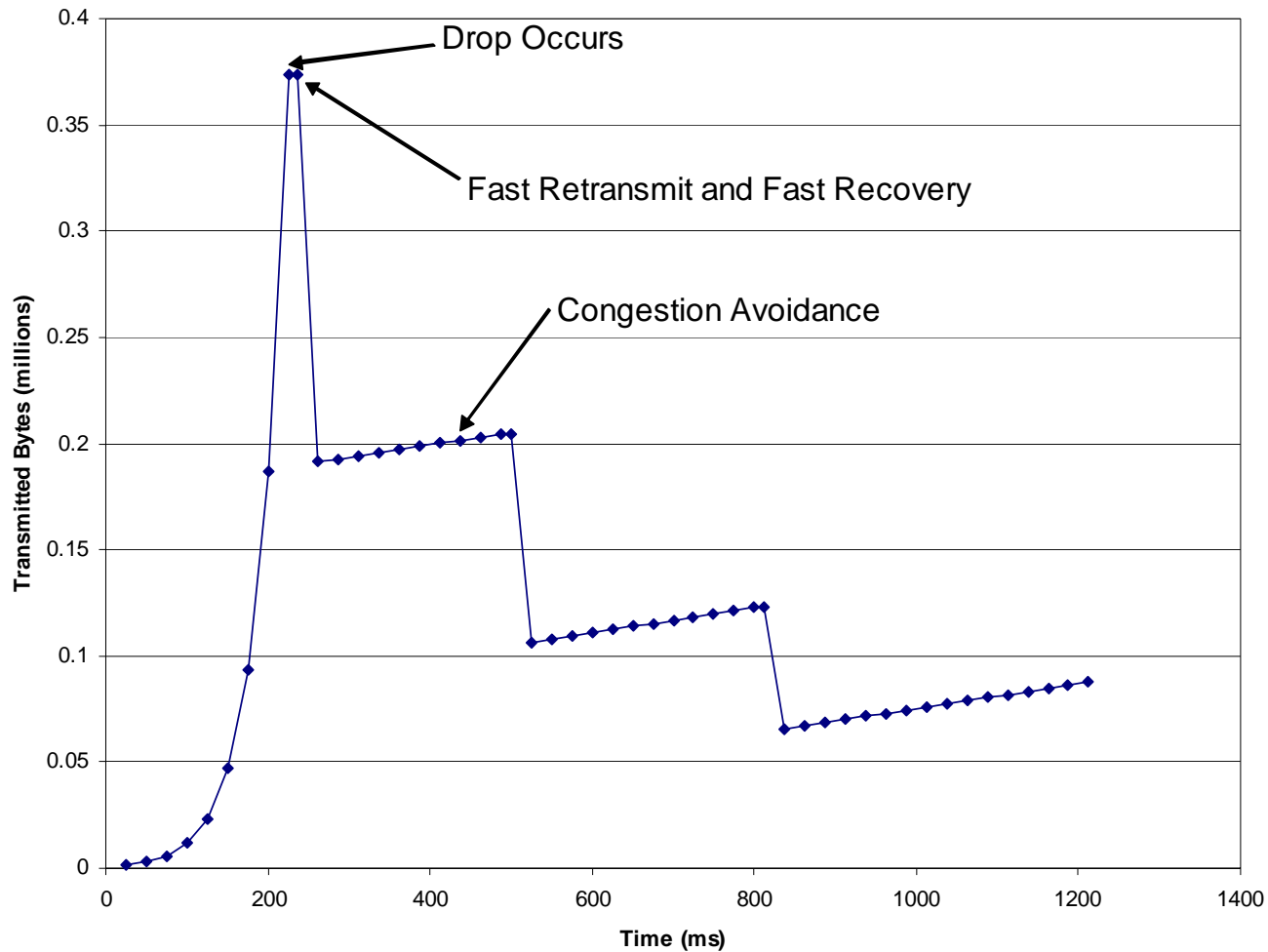
- ◆ After some number of retransmission attempts (usually 13) the TCP connection is closed
- ◆ Usually the timeout value increases with successive timeouts
- ◆ Many different patterns used in the various TCP stacks
 - Some close in a few seconds, Some in several minutes

- Congestion event
 - ◆ .. is any dropped or reordered frames resulting in
 - > fast recovery
 - > Retransmission timeout
 - often referred to as going back to slow start

- Once in congestion avoidance the sender will ramp linearly above the most recent fast recovery cwnd threshold (slow start threshold)
 - ◆ Increment the sender's cwnd by $1/\text{cwnd}$ per ACK instead of by 1 segment per ACK
 - ◆ segment size/8 often used instead of $1/\text{cwnd}$ as approximation
 - ◆ below this threshold still exponential growth of the sender's cwnd

- Dropped frames detected by looking for duplicate ACKs
- On 3 duplicate ACKs received by the sending TCP:
 - ◆ **Fast Retransmit**
 - > Send the oldest unacknowledged segment again
 - > Does not stop data transmission for new data
 - ◆ **Fast Recovery**
 - > The cwnd is reduced to $\frac{1}{2}$ its value + 3 segments
 - > $\frac{1}{2}$ the cwnd value is marked as the slow start threshold
 - > Does not stop segment sending
 - > Once new data acknowledged
 - Pull the sender's cwnd back to the slow start threshold
 - > If new data not acknowledged then a retransmission timeout will eventually occur
- *Networks which reorder extensively can cause TCP to react even though no packets are lost!*

Fast Retransmit, Fast Recovery

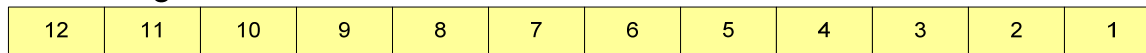


Network Reordering

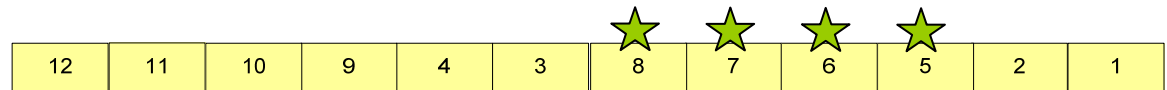
- Networks which dynamically load balance cause excessive false congestion events due to extensive reordering and TCP normally only uses a value of 3 for the duplicate ACK threshold

Direction of segment travel 

Sent Segments

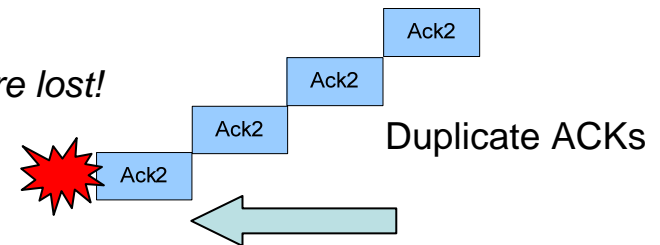


Received Segments



*Causes Fast Retransmit and Fast Recovery
by the sender even though no segments were lost!*

- Can be helped by ignoring more duplicate ACKs before Fast Retransmit/Recovery
- Have to be careful not to miss a retransmit that should have gone out
- Several ways to implement this



- ◆ Generally called SACK

- ◆ SACK Blocks
 - ◆ TCP option carries 1 to 4 Left Edge (LE) – Right Edge (RE) Pairs
 - ◆ For each block
 - > All the bytes from the LE to the RE -1 have been received
 - > The same SACK block may received repeatedly

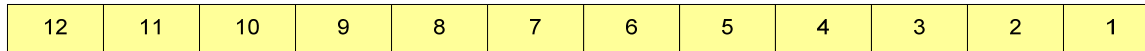
- ◆ Receiver
 - ◆ Maintains list of most recently received contiguous blocks
 - ◆ Whenever Out Of Order received,
 - > send dup ACK as normal
 - > Plus 3-4 most recent SACK blocks

- ◆ Sender
 - ◆ Maintains map of acknowledged contiguous blocks
 - ◆ Retransmit un-ACK'd data blocks to try and fill in multi-segment holes

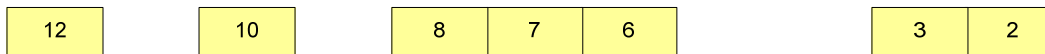
SACK Illustration

Direction of segment travel 

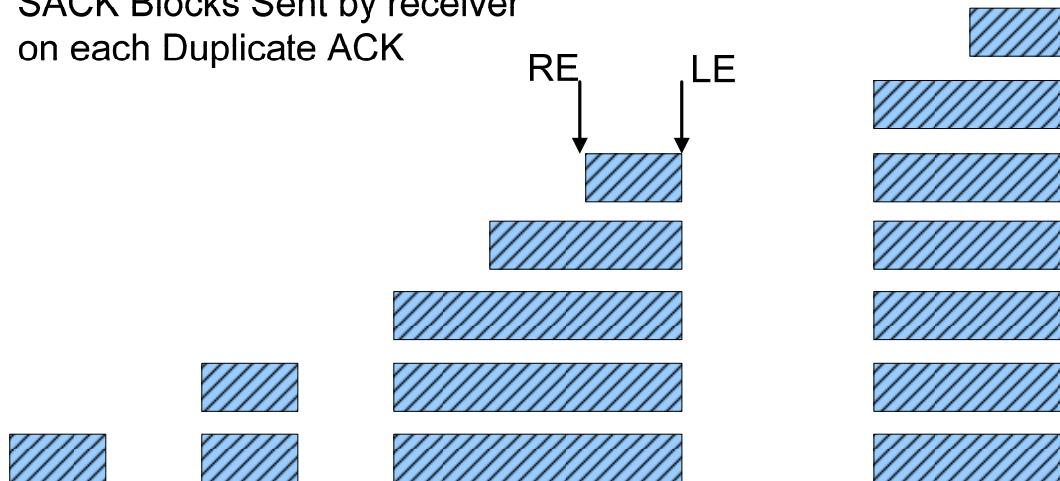
Sent Segments



Received Segments



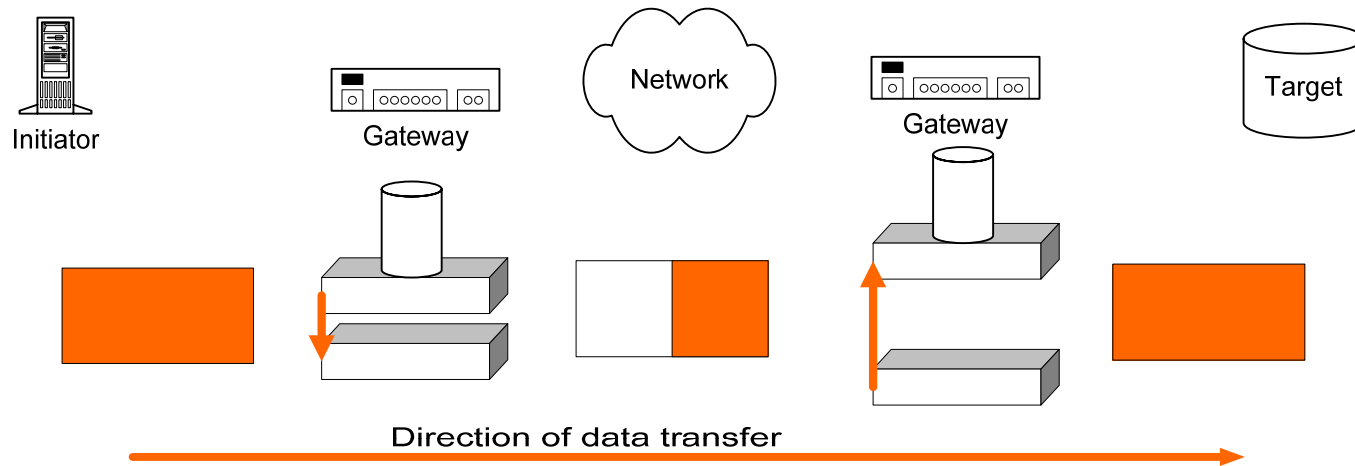
SACK Blocks Sent by receiver
on each Duplicate ACK



 Direction of duplicate ACK travel

- Want to make full utilization of available bandwidth to provided the maximum possible sustained throughput
 - ◆ Scaled receive windows
 - ◆ Implement Selective Acknowledgement (SACK)
 - ◆ Detect retransmission timeouts faster
 - › and with less back-off for successive timeouts
 - ◆ Reduce the amount of data transferred
 - › Compression (Also used by WAFS & TCP WAN Accelerators)
 - ◆ Modify Congestion Controls
 - ◆ Bandwidth Management, Output Rate Limiting, Traffic Shaping
 - › Attempt to avoid congestion in the first place by matching the speed the network can sustain
 - ◆ Aggregate or shotgun multiple TCP sessions together

Compression



➤ Compression Ratio

- ◆ The size of the incoming data divided by the outgoing data
- ◆ Determined by the data pattern and algorithm
- ◆ History buffers help the compression ratio since they retain more data for potential matches

➤ Compression Rate

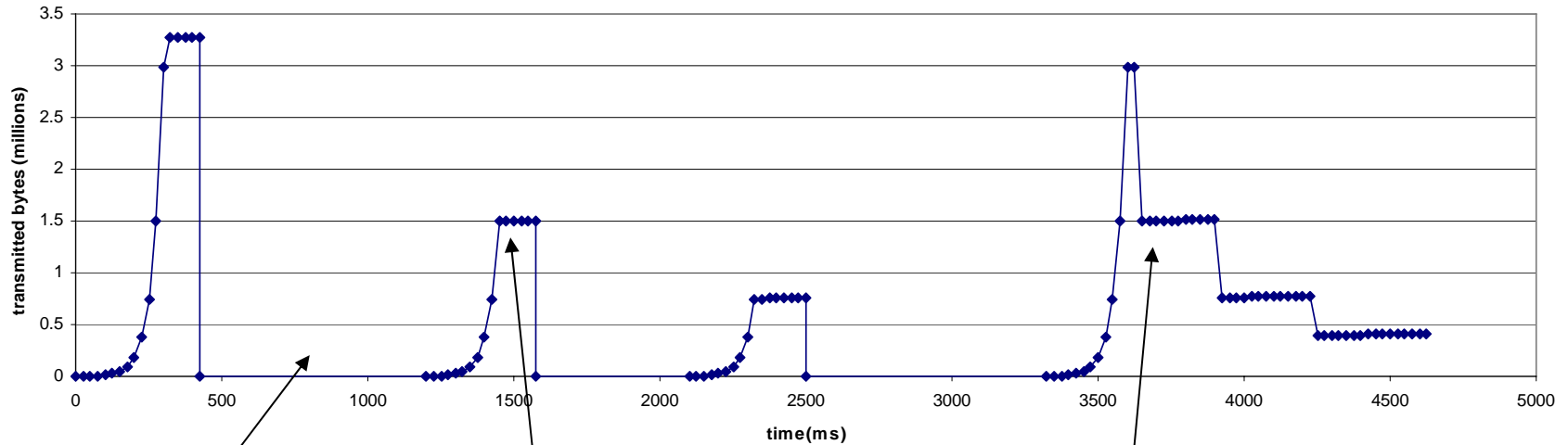
- ◆ Speed of incoming data processing
- ◆ Different algorithms need different processing power

- A higher compression ratio generally requires more processing power for the same rate
- Many algorithms
- Can also be used for data at rest
- Encrypted data incompressible

TCP/IP Congestion Control Modifications

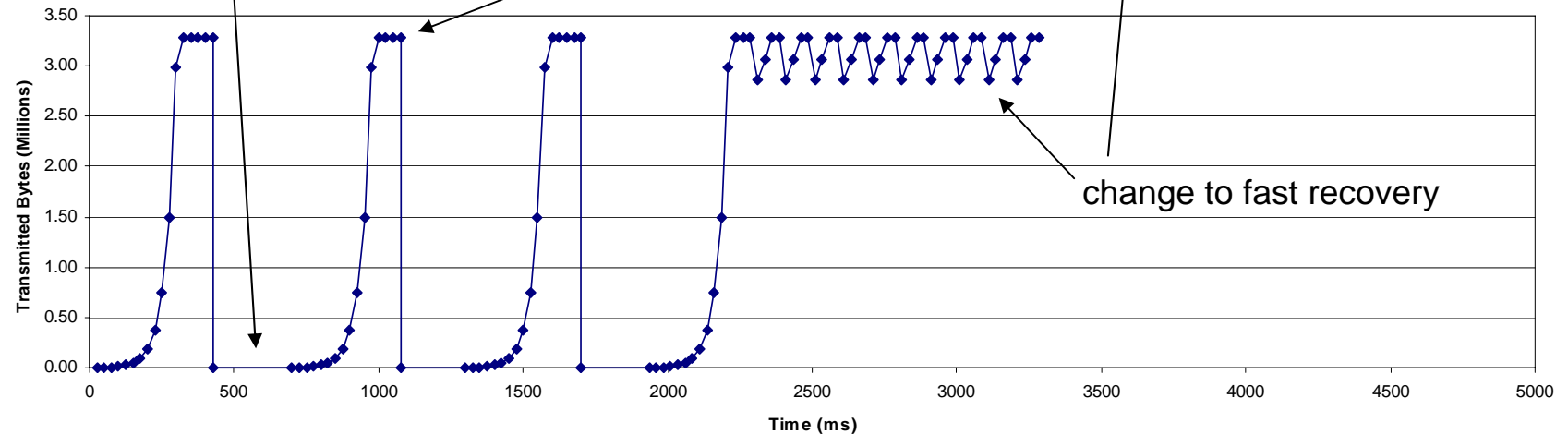
- Don't want to break TCP's fundamental congestion behavior
- Modify Congestion Controls
 - ◆ Different ramp up or starting value for slow start (aka QuickStart)
 - ◆ Less reduction during fast recovery
 - ◆ Ignore or reduce the effects of congestion avoidance
 - ◆ Eifel Algorithm
- Modify Fast retransmit and Fast Recovery detection scheme
 - ◆ Ignore a larger fixed number of duplicate ACKs but backstop with a short timer
 - ◆ RFC 4653 – TCP-NCR (non-congestion robustness)
 - Retransmission detection is based upon cwnd of data leaving network instead of a fixed limit

TCP Congestion Control Modifications



change to retransmission timeout

change to congestion avoidance



change to fast recovery

Some High Speed TCP/IP Projects

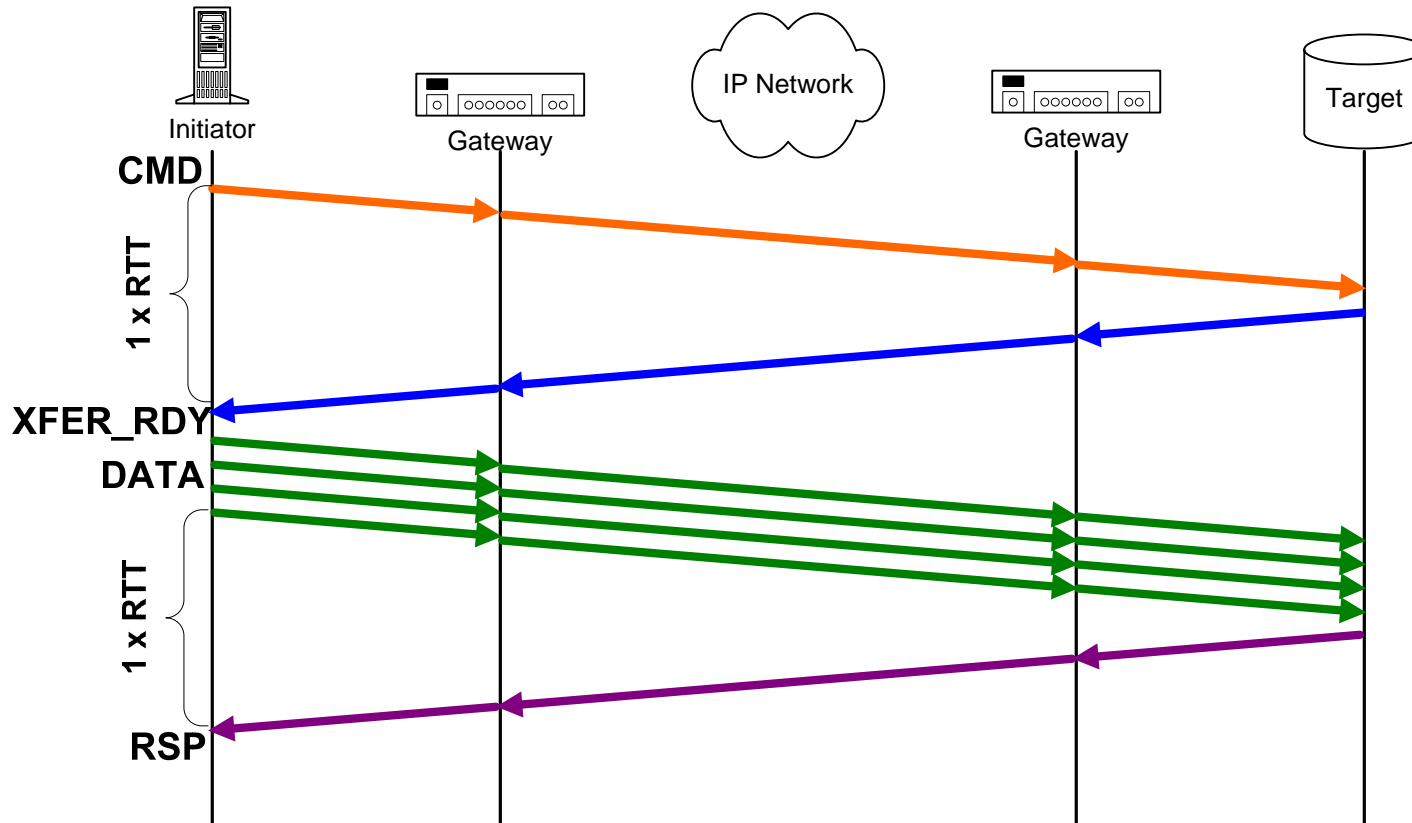
- Scalable TCP (STCP)
- High Speed TCP (HS-TCP)
- FAST TCP
- H-TCP
- Industry Proprietary Implementations
 - ◆ WAN Acceleration Companies
 - ◆ Distance Extension for SAN
 - ◆ Distance Extension for TCP

- Make use of some combination of the modifications and options described above

Upper layer Protocol Optimization

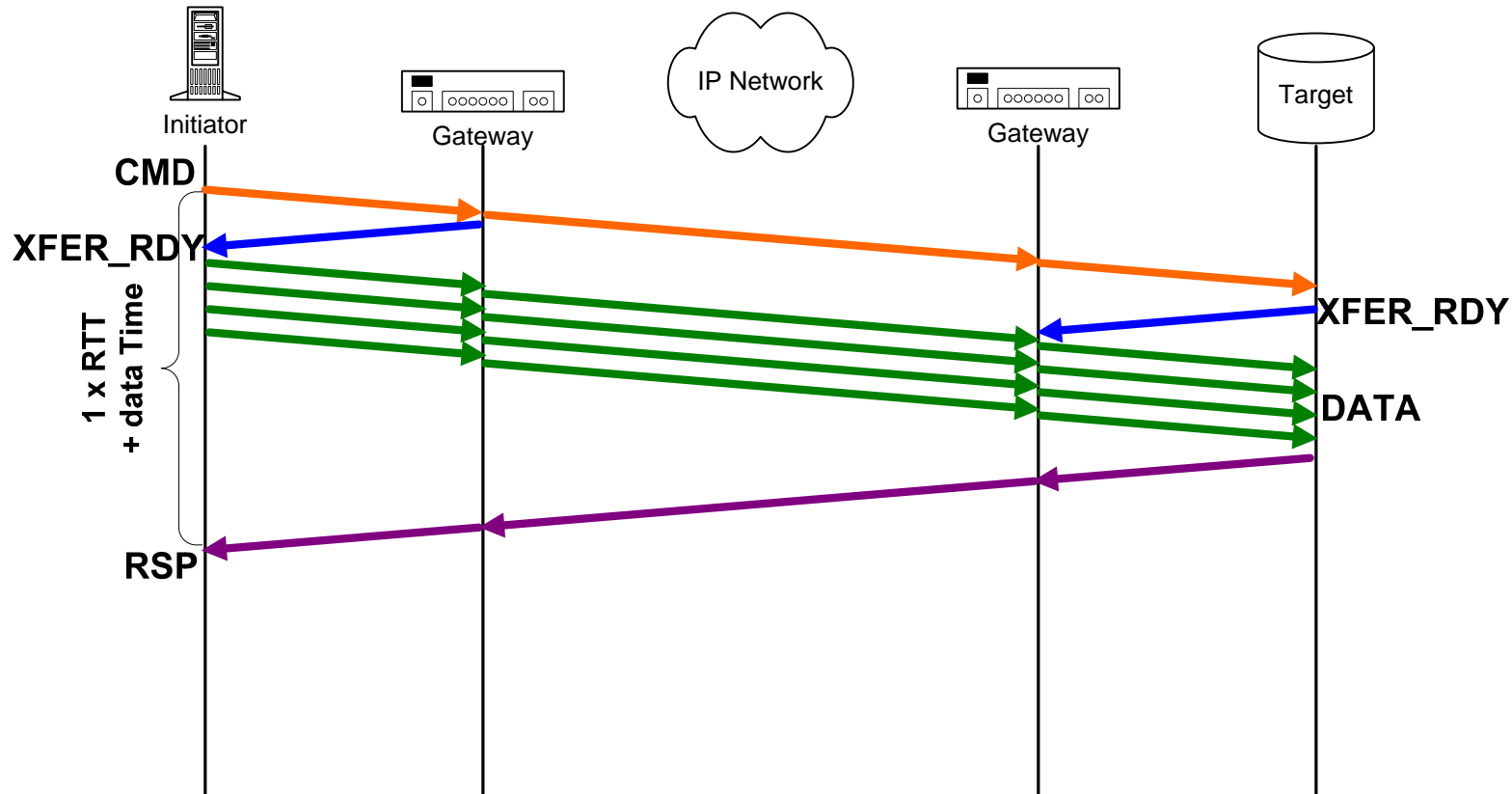
- FC Fabric Connectivity across the WAN
 - ◆ Generally requires a pair of cooperating Gateways or Accelerators
 - ◆ Protocol might be modified FCIP, modified iFCP, or proprietary
 - ◆ TCP Segmentation FC/SCSI storage specific modifications:
 - > Always push data when FCP End of Sequence encountered
 - > Can instead push each FC/FCP frame (typically 2 segments per data frame)
- SCSI Command Acceleration
 - ◆ Acceleration For Write Commands
 - > Called Fast Write or Write Acceleration
 - > iSCSI has immediate data and unsolicited data
 - ◆ Acceleration For Tape Devices
 - > Called Tape Pipelining or Tape Acceleration
 - > Write acceleration plus early response for write commands
 - > Read ahead for sequential read commands
- Other Protocols (HTTP, NFS, CIFS, etc)
 - ◆ Basic Acceleration similar to SCSI command acceleration
 - ◆ Makes heavier use of file system caching, 'byte' caching, and high ratio compression algorithms

Normal Write Command



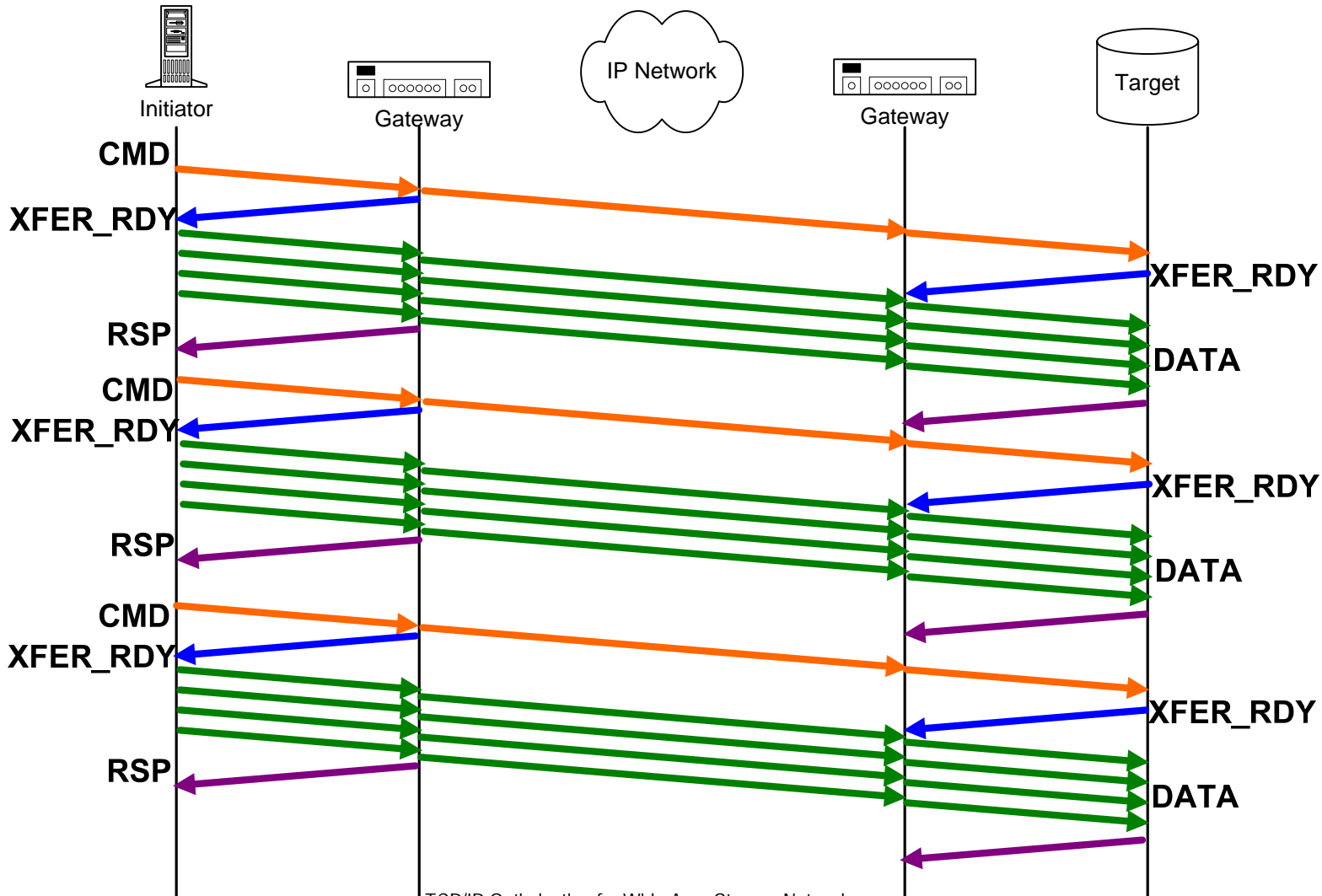
- A write command normally takes $2 \times \text{RTT}$ + the data transmission and device response time to complete

Write Command Acceleration

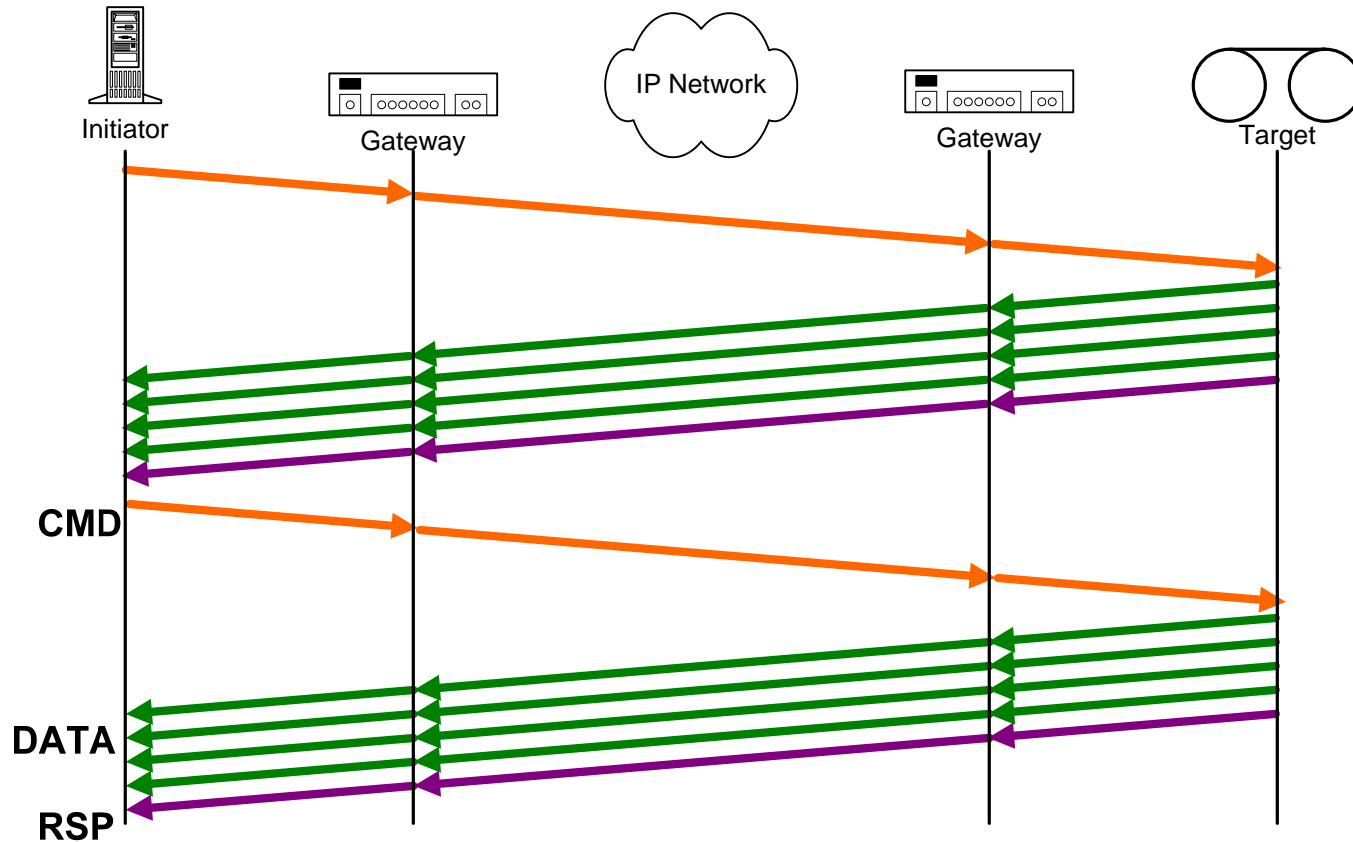


- An accelerated write command takes $1 \times \text{RTT} + \text{the data transmission time}$

Accelerated Tape Write Commands

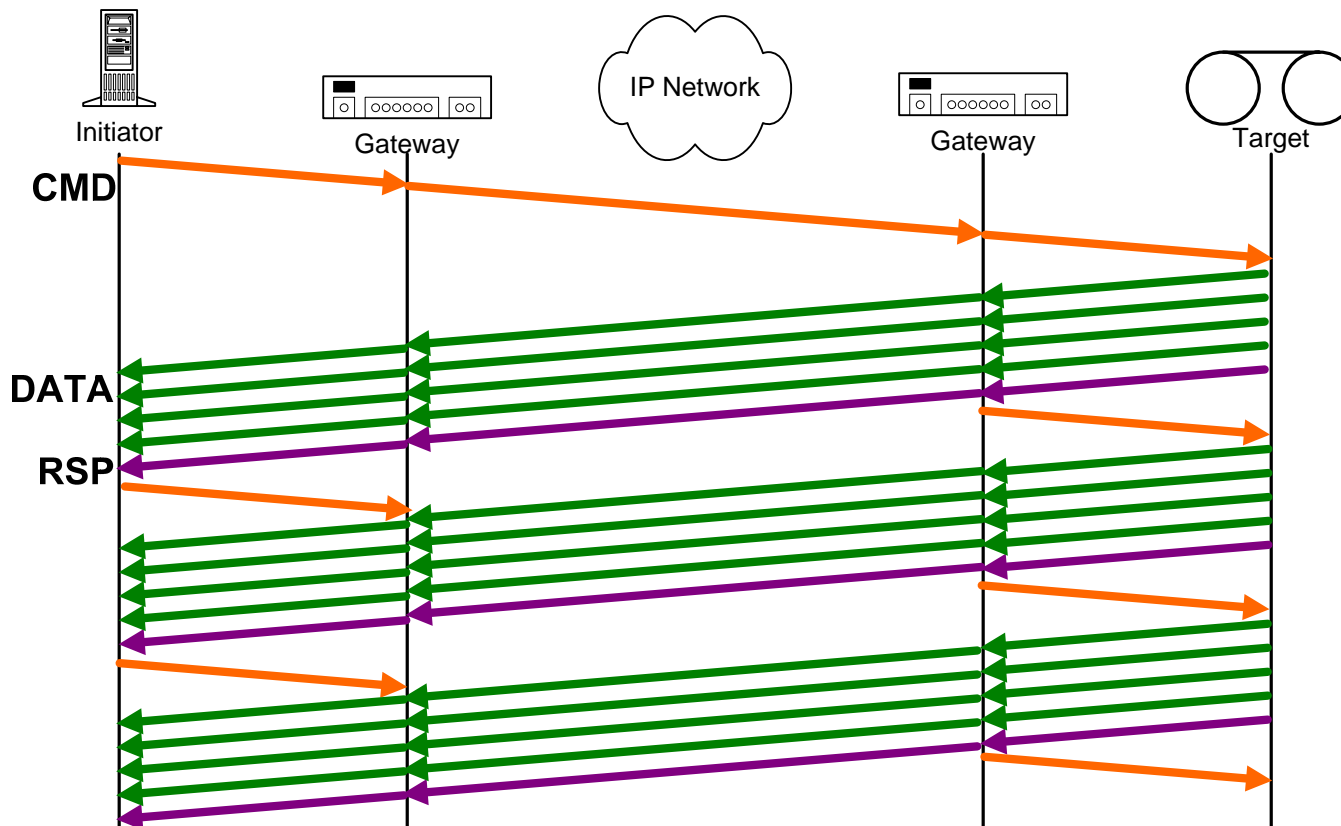


Normal Tape Read Commands



- Tape devices only allow 1 outstanding command for read as well

Accelerated Tape Read Commands



- Remote Gateway reads ahead by issuing commands itself
 - ◆ Data sent to and buffered by Local gateway until command received
 - ◆ Works well because a tape is a sequential access device

- Please send any questions or comments on this presentation to SNIA: tracknetworking@snia.org

**Many thanks to the following individuals
for their contributions to this tutorial.**

SNIA Education Committee

Joseph L White

Howard Goldstein

Appendix: Relevant Internet RFCs

- RFC 793 – Transmission Control Protocol
- RFC 896 – Congestion control in IP/TCP internetworks
- RFC 1122 – Requirements for Internet Hosts - Communication Layers
- RFC 1323 – TCP Extensions for High Performance
- RFC 2018 – TCP Selective Acknowledgment Options
- RFC 2140 – TCP Control Block Interdependence
- RFC 2581 – TCP Congestion Control
- RFC 2861 – TCP Congestion Window Validation
- RFC 2883 – An Extension to the Selective Acknowledgement (SACK) Option for TCP
- RFC 2988 – Computing TCP's Retransmission Timer
- RFC 3042 – Enhancing TCP's Loss Recovery Using Limited Transmit
- RFC 3124 – The Congestion Manager
- RFC 3155 – End-to-end Performance Implications of Links with Errors
- RFC 3168 – The Addition of Explicit Congestion Notification (ECN) to IP
- RFC 3390 – Increasing TCP's Initial Window
- RFC 3449 – TCP Performance Implications of Network Path Asymmetry
- RFC 3465 – TCP Congestion Control with Appropriate Byte Counting (ABC)
- RFC 3517 – A Conservative Selective Acknowledgment based Loss Recovery Algorithm for TCP
- RFC 3522 – The Eifel Detection Algorithm for TCP
- RFC 3649 – HighSpeed TCP for Large Congestion Windows
- RFC 3742 – Limited Slow-Start for TCP with Large Congestion Windows
- RFC 3782 – The NewReno Modification to TCP's Fast Recovery Algorithm
- RFC 4015 – The Eifel Response Algorithm for TCP
- RFC 4138 – Forward RTO-Recovery (F-RTO)
- RFC 4653 – Improving the Robustness of TCP to Non-Congestion Events
- RFC 4782 – Quick-Start for TCP and IP
- RFC 4828 – TCP Friendly Rate Control (TFRC): The Small-Packet (SP) Variant

Appendix: TCP Options

- End of option list

0

Kind	Len	Values...
------	-----	-----------

- No operation

Options are usually 4 byte aligned with leading NOPs

1

- Maximum (*Receive*) Segment Size [SYN only]

2	4	MSS
---	---	-----

- Window Scale Factor [SYN only]

NOP	3	3	shift
-----	---	---	-------

- Timestamp

NOP	NOP	8	10	Timestamp Value	Timestamp Echo Reply
-----	-----	---	----	-----------------	----------------------

- Selective ACK Permitted [SYN packet only]

NOP	NOP	4	2
-----	-----	---	---

- Selective ACK block

NOP	NOP	5	L	Left Edge	Right Edge
-----	-----	---	---	-----------	------------

...

$$L = 2 + N * 8, N \text{ is number of left-right pairs}$$