Internet Congestion Control

Dr. Miled M. Tezeghdanti

November 19, 2011

Introduction

- Congestion Control
  - Reduce the traffic in order to meet the network requirements
    - Senders will reduce the traffic if they notice a congestion in the network
  - Network Layer function in the OSI Model
  - Transport Layer function in the Internet
    - Performed by TCP
    - TCP friendly applications
- Flow Control
  - Reduce the traffic in order to meet the reciever requirements
    - Recievers will inform senders by their available resources
  - Layer 2, 3 , and 4 function in the OSI Model
  - Transport Layer function in the Internet
    - Performed by TCP

Flow Control

- TCP provides a means for the receiver to govern the amount of data sent by the sender.
- This is achieved by returning a "window" with every ACK indicating a range of acceptable sequence numbers beyond the last segment successfully received.
- The window indicates an allowed number of bytes that the sender may transmit before receiving further permission.
- Window:
  - specifies the number of data bytes beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept.
  - 16 bits
  - Maximum window size: 65,535 bytes

Flow Control - Window Scale Option

- Window Scale Option
  - KIND: 3
  - Length: 3
  - Scale (1 byte): power of 2 shift to the left
  - The value 'shift.cnt' may be zero (offering to scale, while applying a scale factor of 1 (2^0) to the receive window).
  - The value 'shift.cnt' may be n (offering to scale, while applying a scale factor of 2^n to the receive window).
  - The maximum value 'shift.cnt' may be 14 (offering to scale, while applying a scale factor of 2^{14} to the receive window).
  - both sides must send Window Scale options in their SYN segments to enable window scaling in either direction.
  - RFC 1323
  - +———+———+———+
  - + Kind +Length +shift.cnt+
  - +———+———+———+

Definitions

- SEGMENT
  - A segment is ANY TCP/IP data or acknowledgment packet (or both).
- RECEIVER MAXIMUM SEGMENT SIZE (RMSS)
  - The RMSS is the size of the largest segment the receiver is willing to accept.
  - This is the value specified in the MSS option sent by the receiver during connection startup.
  - If the MSS option is not used, it is 536 bytes.
  - The size does not include the TCP/IP headers and options.
- SENDER MAXIMUM SEGMENT SIZE (SMSS)
  - The SMSS is the size of the largest segment that the sender can transmit.
  - This value can be based on the maximum transmission unit of the network, the path MTU discovery algorithm, RMSS.
  - The size does not include the TCP/IP headers and options.
- FULL-SIZED SEGMENT
  - A segment that contains the maximum number of data bytes permitted (i.e., a segment containing SMSS bytes of data).

Definitions

- RECEIVER WINDOW (rwnd)
  - The most recently advertised receiver window.
- CONGESTION WINDOW (cwnd)
  - A TCP state variable that limits the amount of data a TCP can send.
  - At any given time, a TCP MUST NOT send data with a sequence number higher than the sum of the highest acknowledged sequence number and the minimum of cwnd and rwnd.
- INITIAL WINDOW (IW)
  - The initial window is the size of the sender's congestion window after the three-way handshake is completed.
- LOSS WINDOW (LW)
  - The loss window is the size of the congestion window after a TCP sender detects loss using its retransmission timer.
- RESTART WINDOW (RW)
  - The restart window is the size of the congestion window after a TCP restarts transmission after an idle period
- FLIGHT SIZE
  - The amount of data that has been sent but not yet cumulatively acknowledged.

Definitions

- DUPLICATE ACKNOWLEDGMENT
  - An acknowledgment is considered a "duplicate" when
    1. the receiver of the ACK has outstanding data
    2. the incoming acknowledgment carries no data
    3. the SYN and FIN bits are both off
    4. the acknowledgment number is equal to the greatest acknowledgment received on the given connection
    5. the advertised window in the incoming acknowledgment equals the advertised window in the last incoming acknowledgment

Congetion Detection

- How does the source determine whether or not the network is congested?
- Implicit Feedback
- Long delays
  - It takes more time for ACKs to come back
  - Large queueing delays
- Retransmission timer expiration
  - No ACK is received before the timer expiration
  - Packet loss is rarely due to transmission error (on wired lines)
  - Lost packet implies congestion

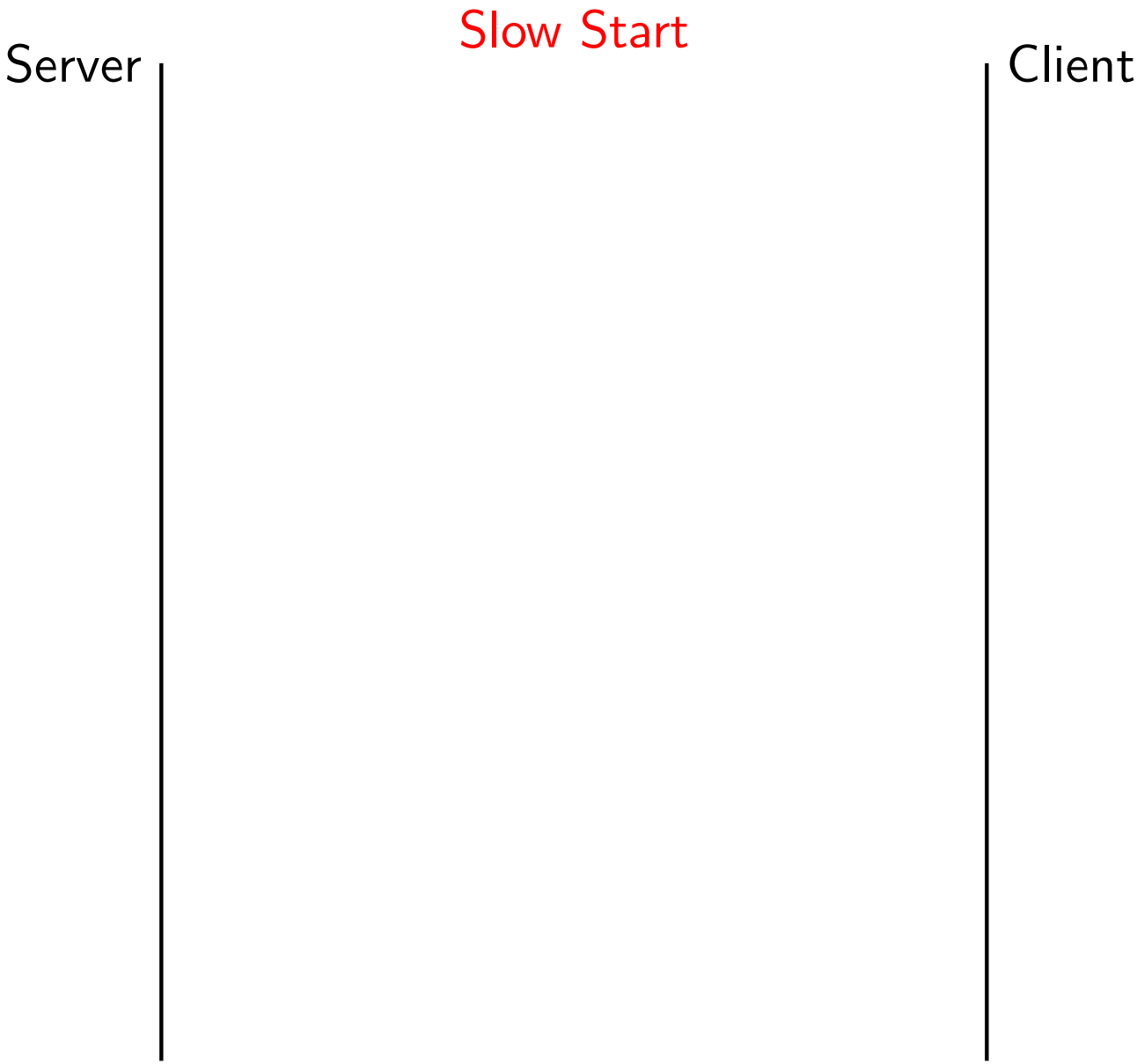
IW, RW, LW, ssthresh

- IW
  - If SMSS $>$ 2190 bytes:
    - IW = 2 * SMSS bytes and MUST NOT be more than 2 segments
  - If (SMSS $>$ 1095 bytes) and (SMSS ¡= 2190 bytes):
    - IW = 3 * SMSS bytes and MUST NOT be more than 3 segments
  - if SMSS $\leq$ 1095 bytes:
    - IW = 4 * SMSS bytes and MUST NOT be more than 4 segments
- RW = min(IW,cwnd)
- LW equals 1 full-sized segment (regardless of the value of IW)
- ssthresh
  - At startup, ssthresh = rwind
  - After a loss detection, ssthresh = max (FlightSize / 2, 2*SMSS)

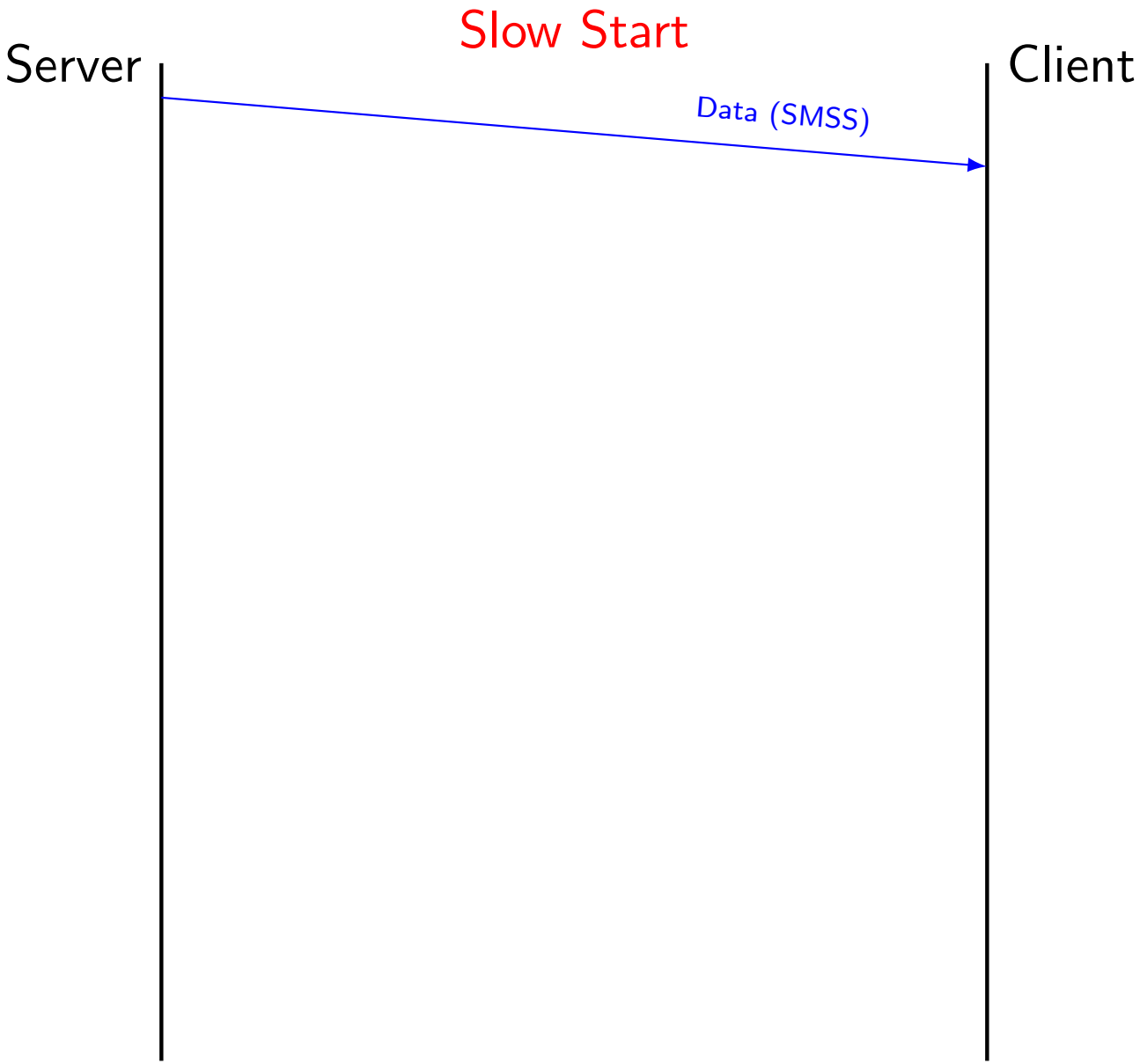
Slow Start

- How should TCP start sending data?
- Use Slow Start to increase window rapidly from a cold start
  - Determine the available capacity
  - Add a new state variable for each connection cwind (CongestionWindow)
  - Begin with cwind = 1 segment (SMSS bytes) (In fact: IW, LW, RW).
  - SMSS is negociated at the connection setup
  - Sender Maximum Segment Size: default value: 536 bytes
  - Double cwind each RTT
  - Increment cwind by 1 segment for each recieved ACK
  - This is exponential increase to probe for available bandwidth

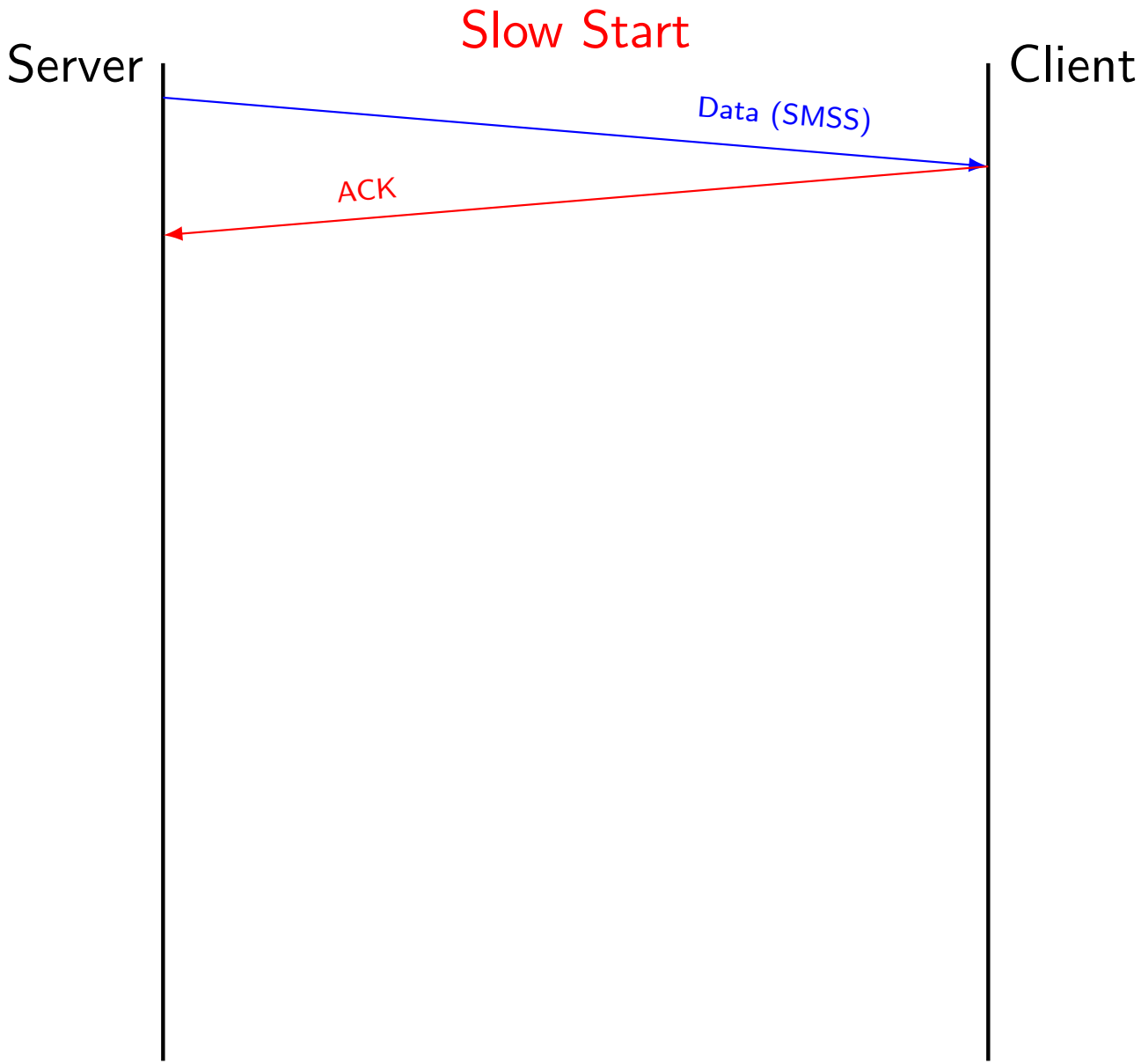
Slow Start



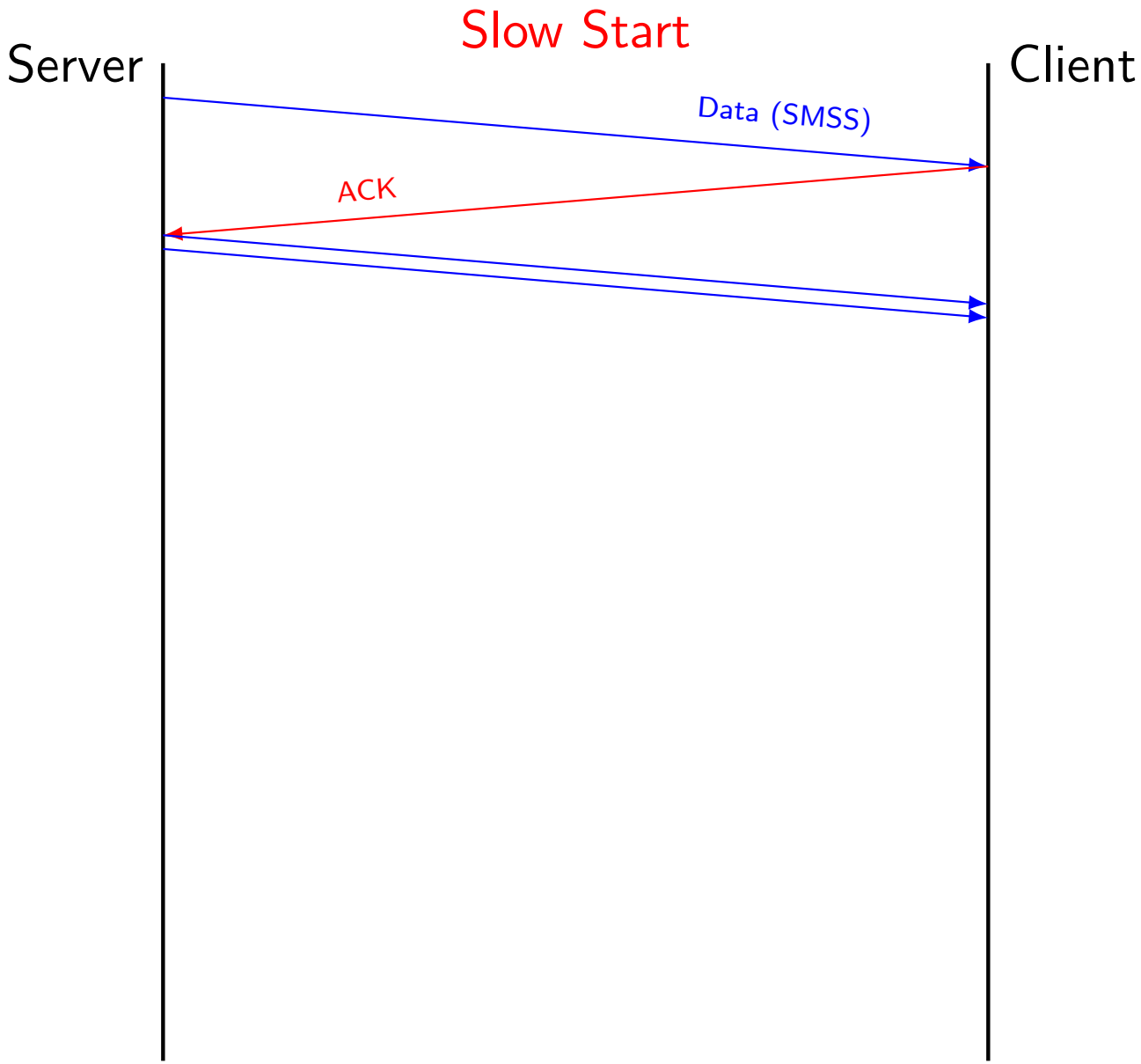
Slow Start



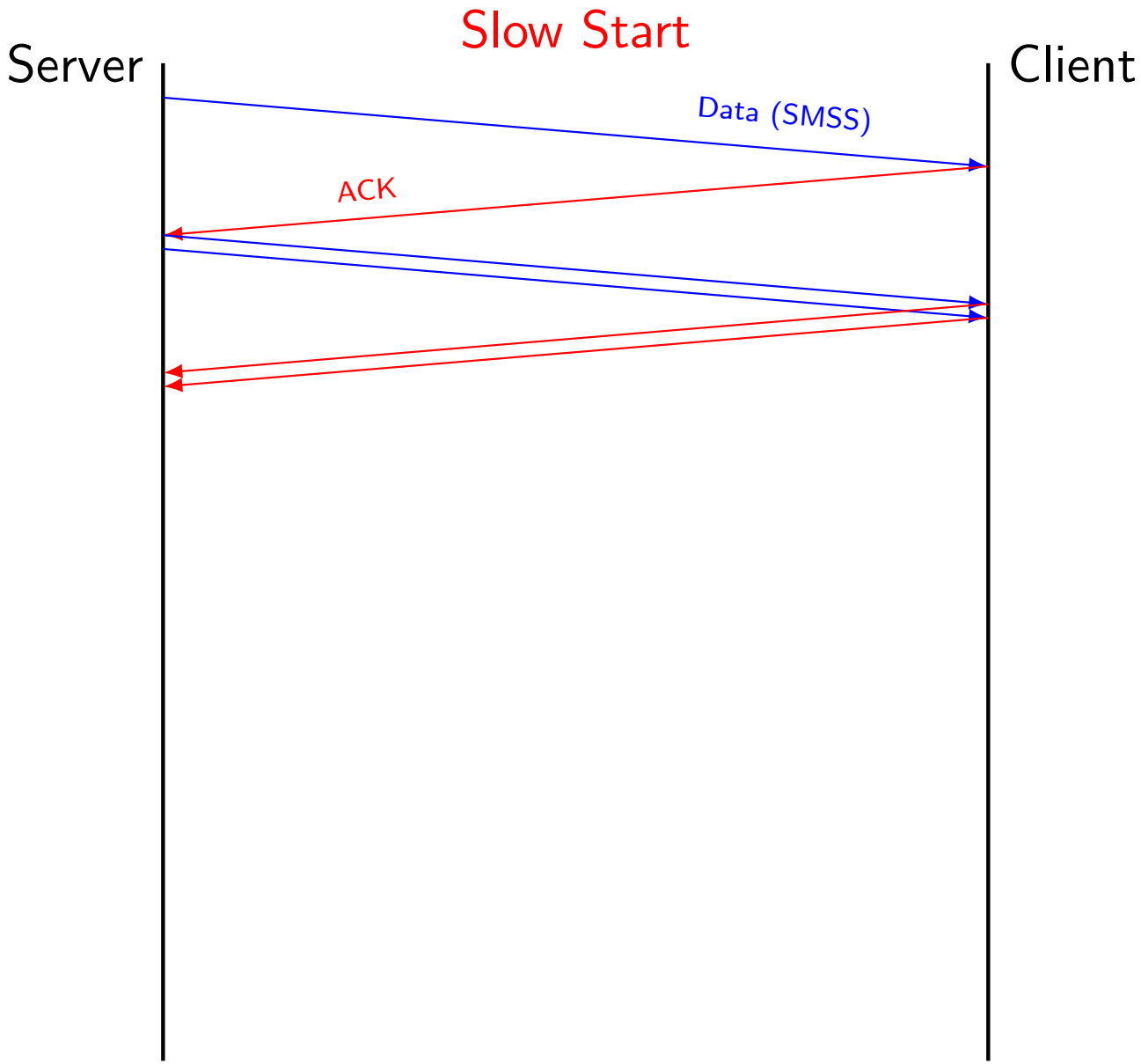
Slow Start



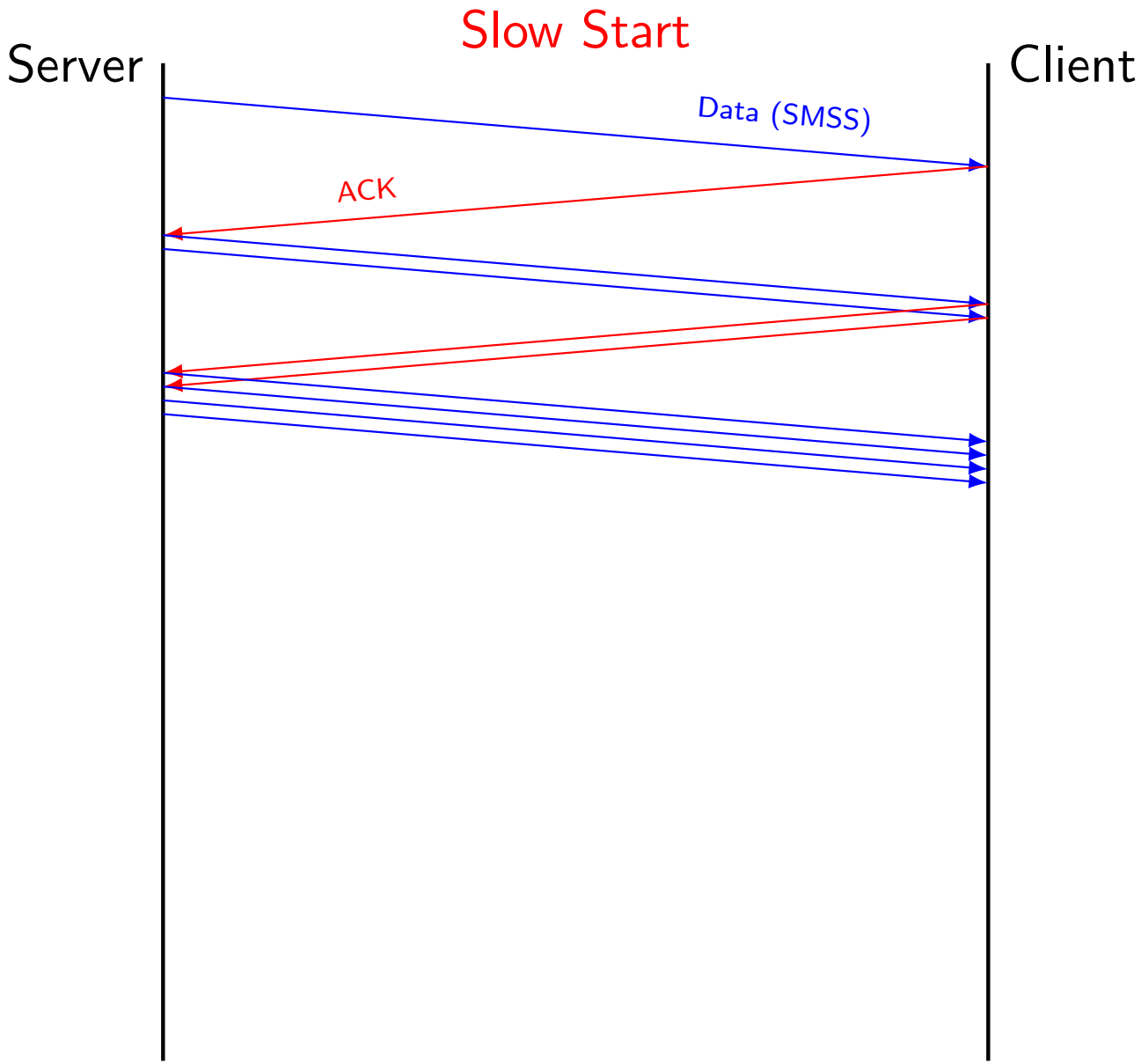
Slow Start



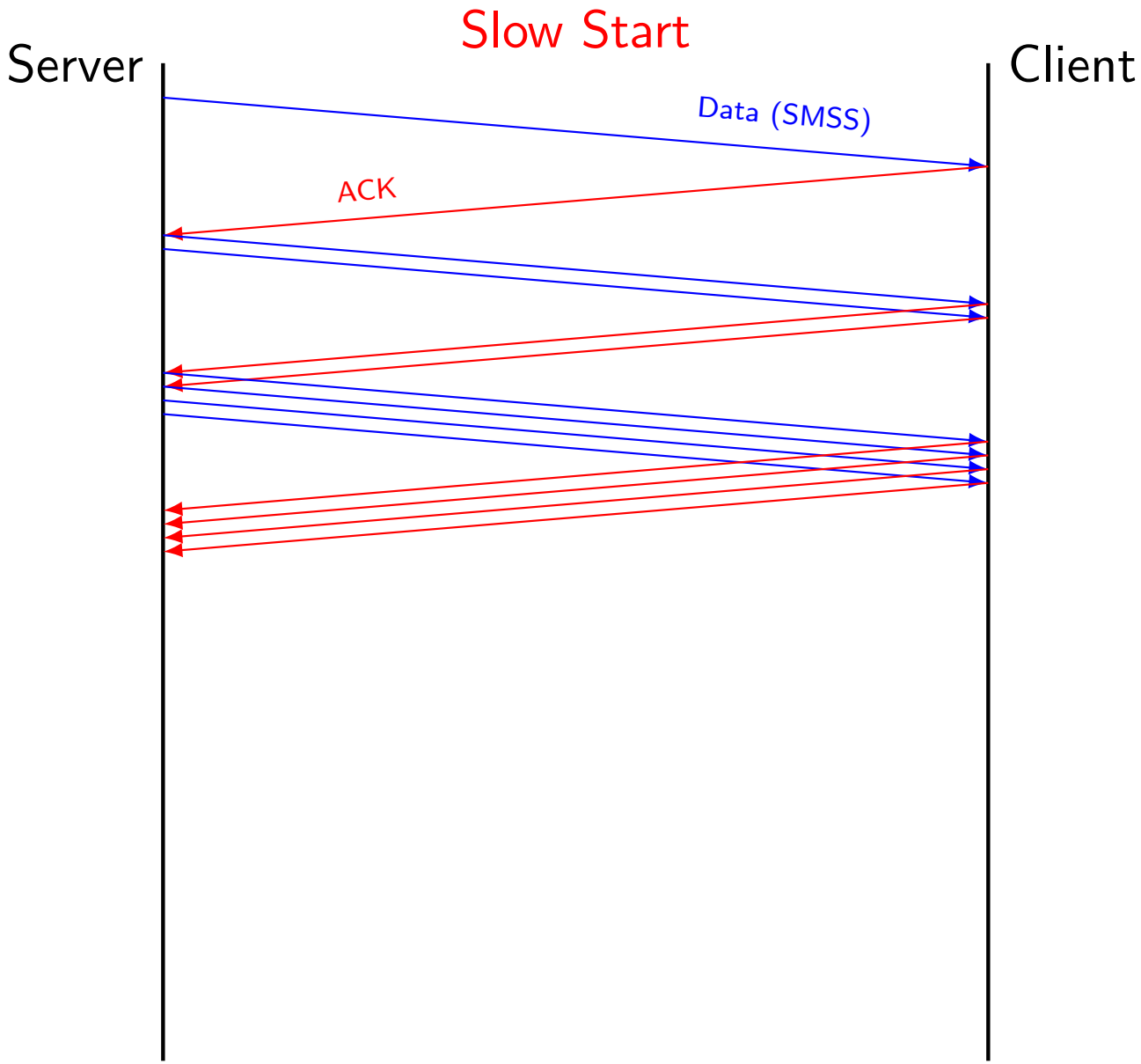
Slow Start



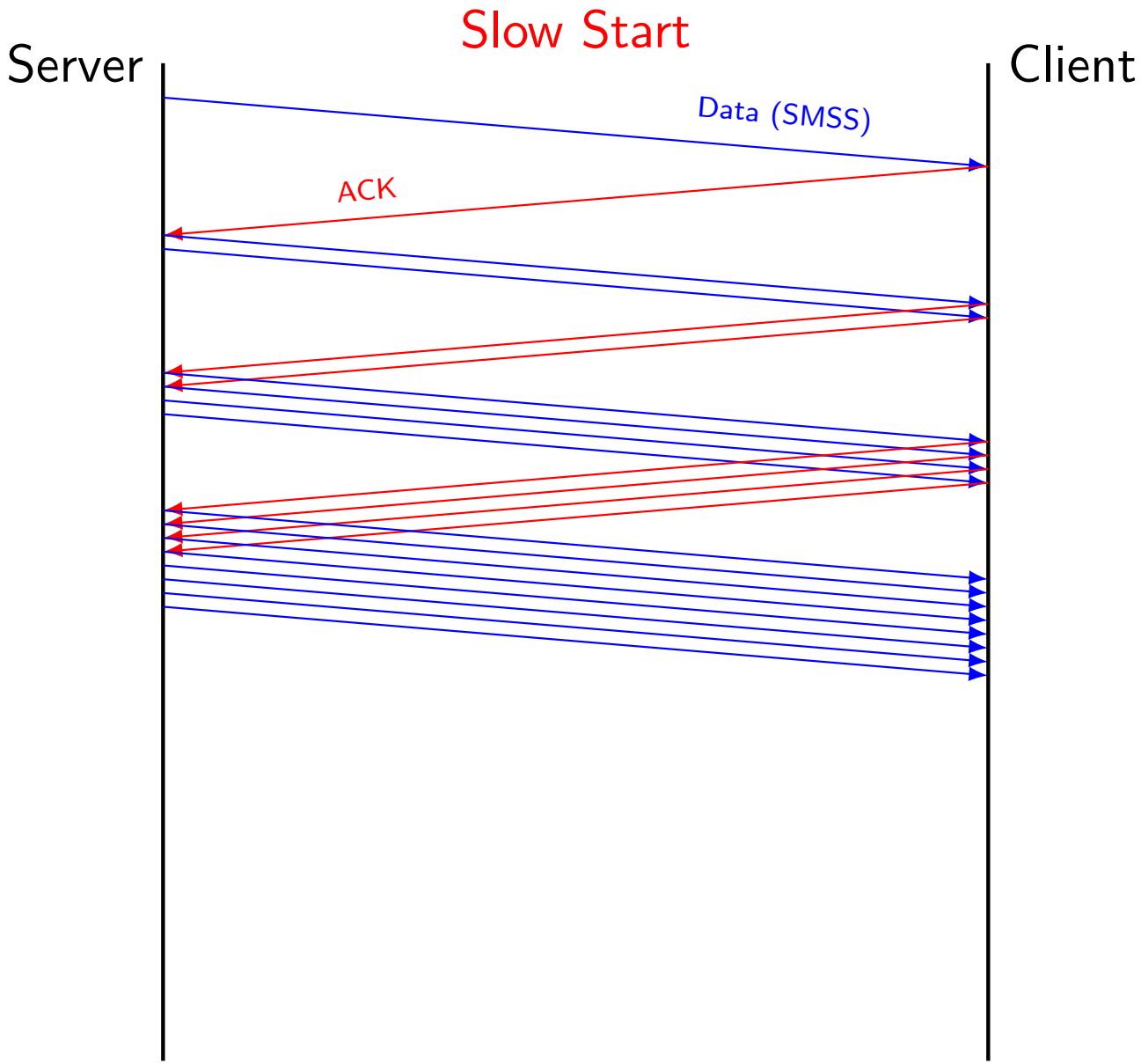
Slow Start



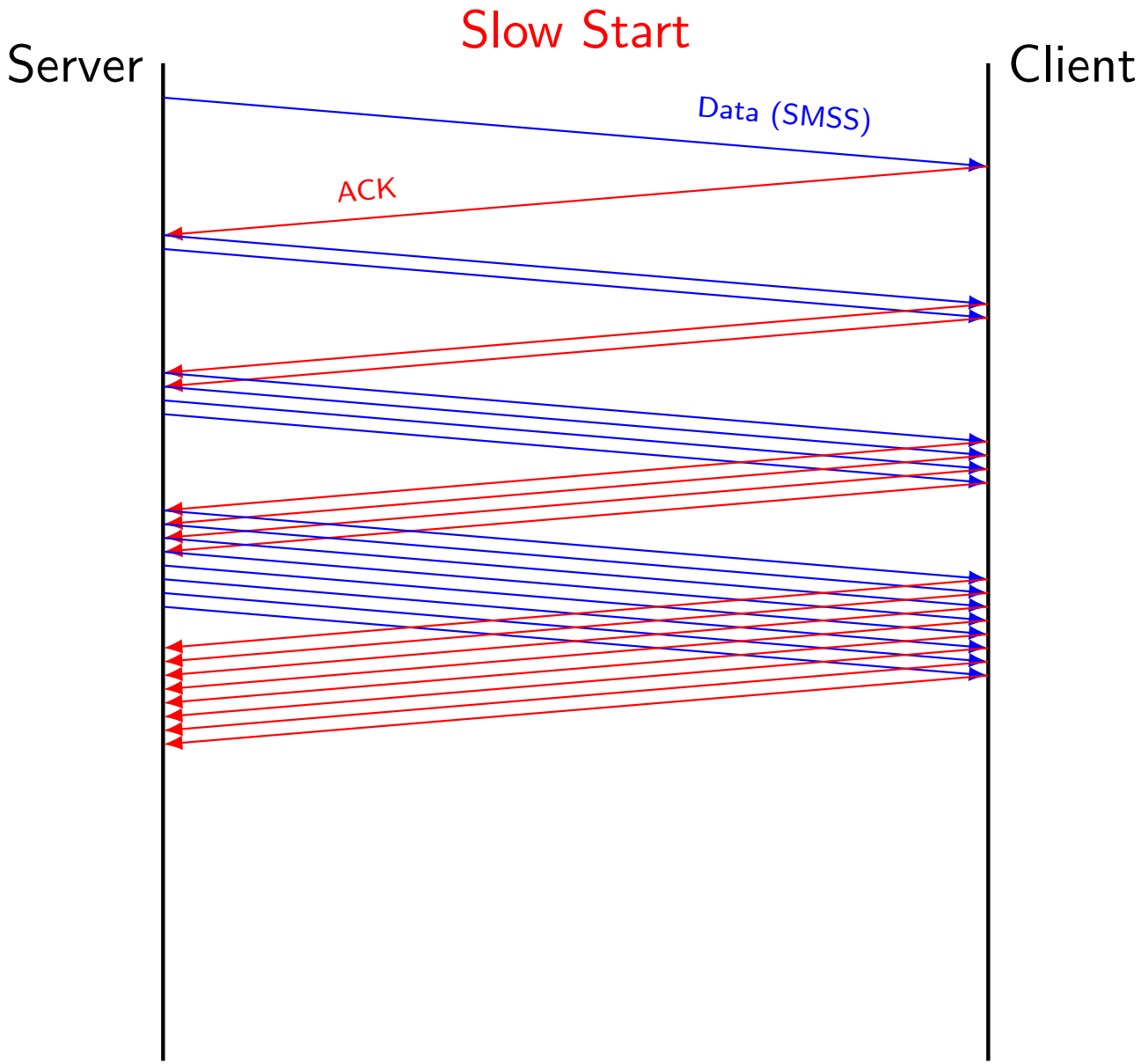
Slow Start



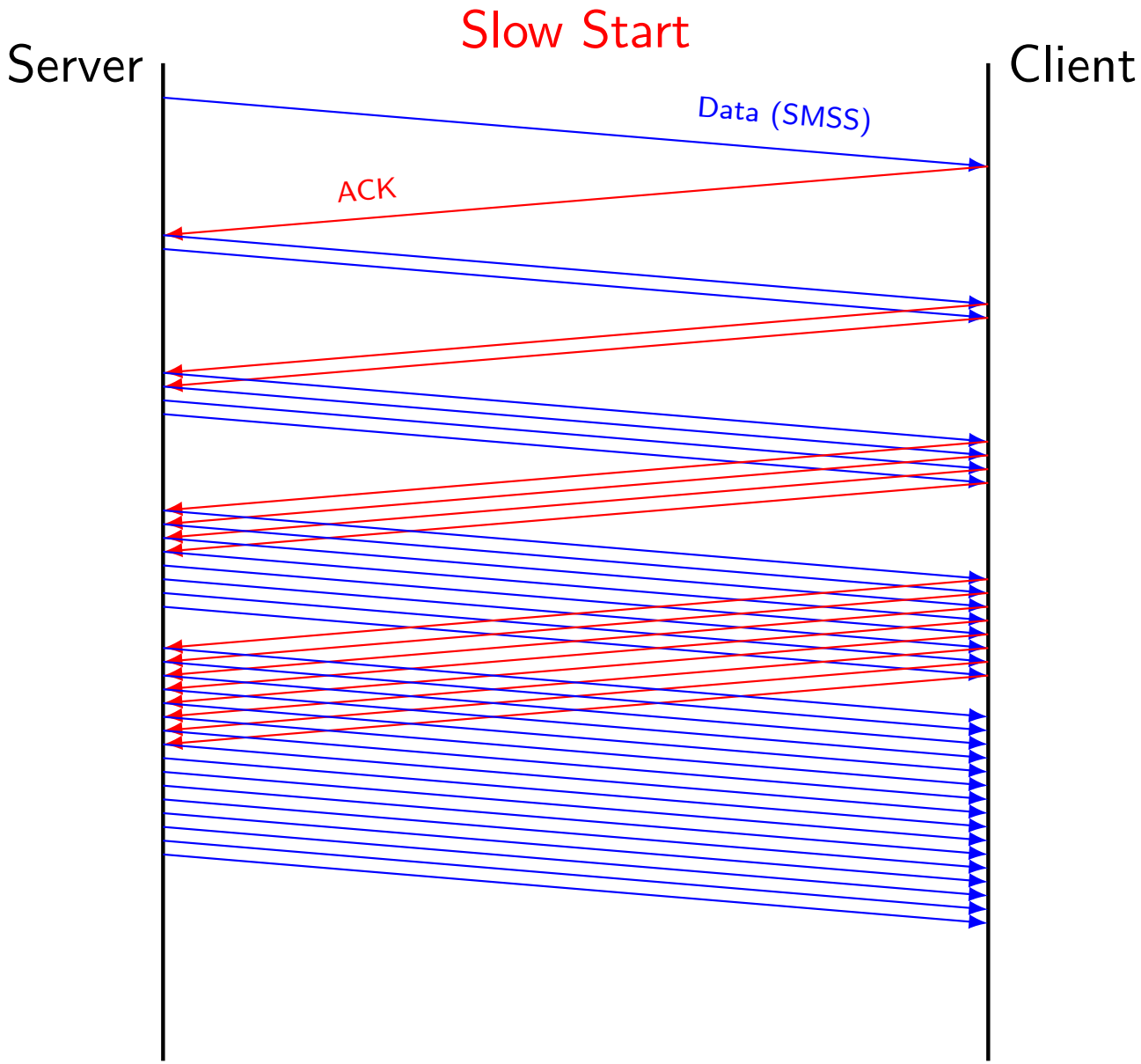
Slow Start



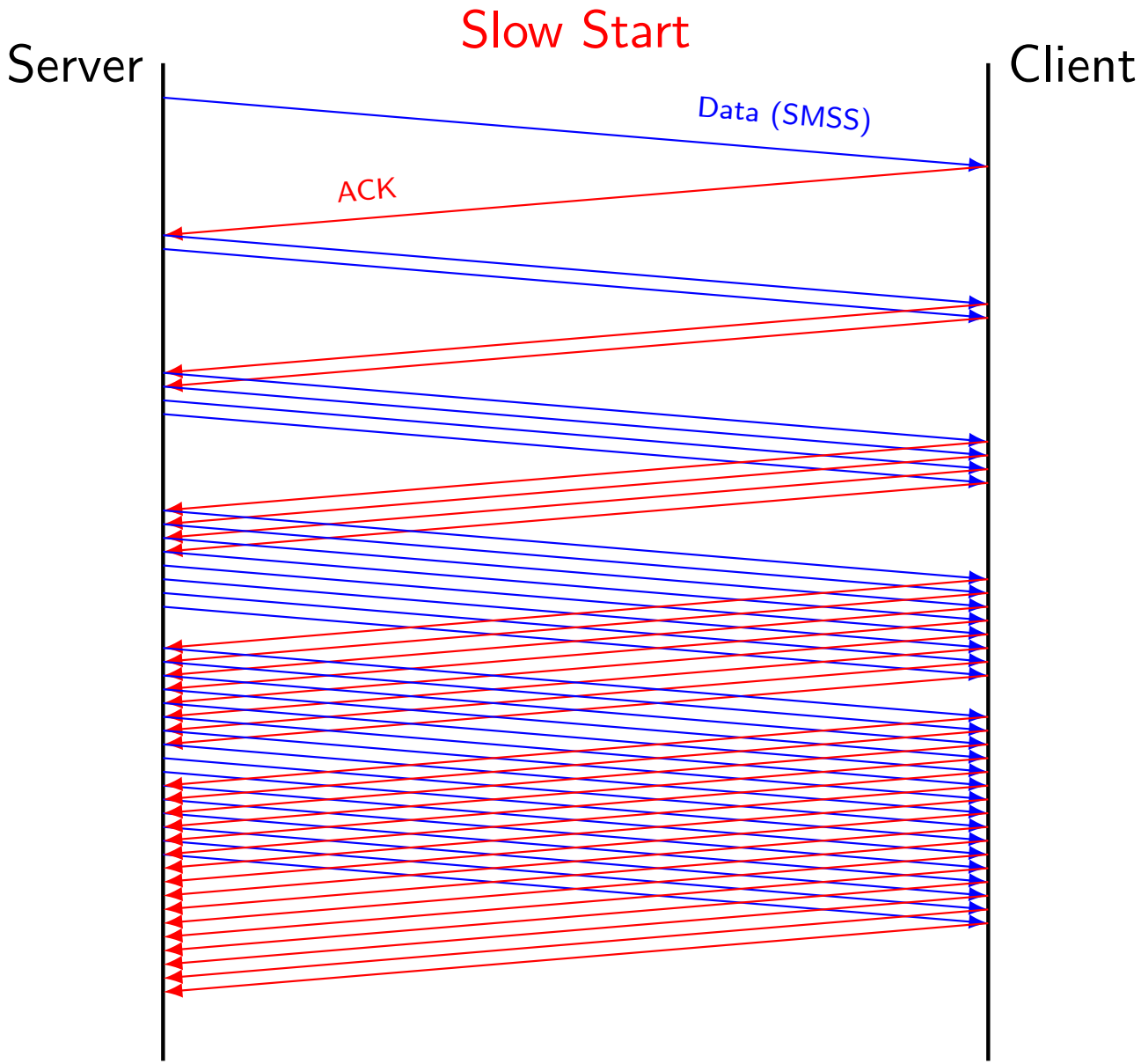
Slow Start



Slow Start



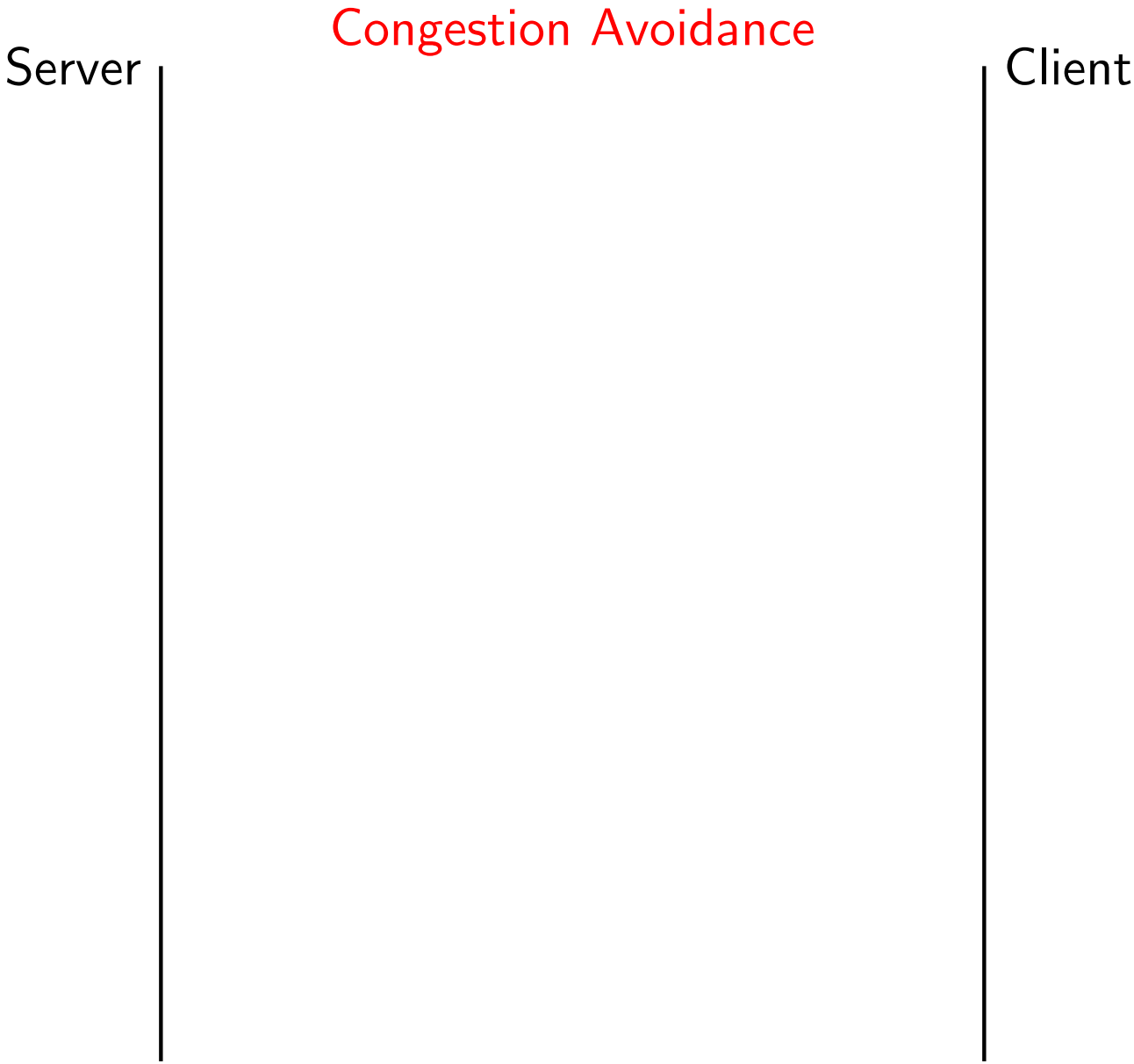
Slow Start



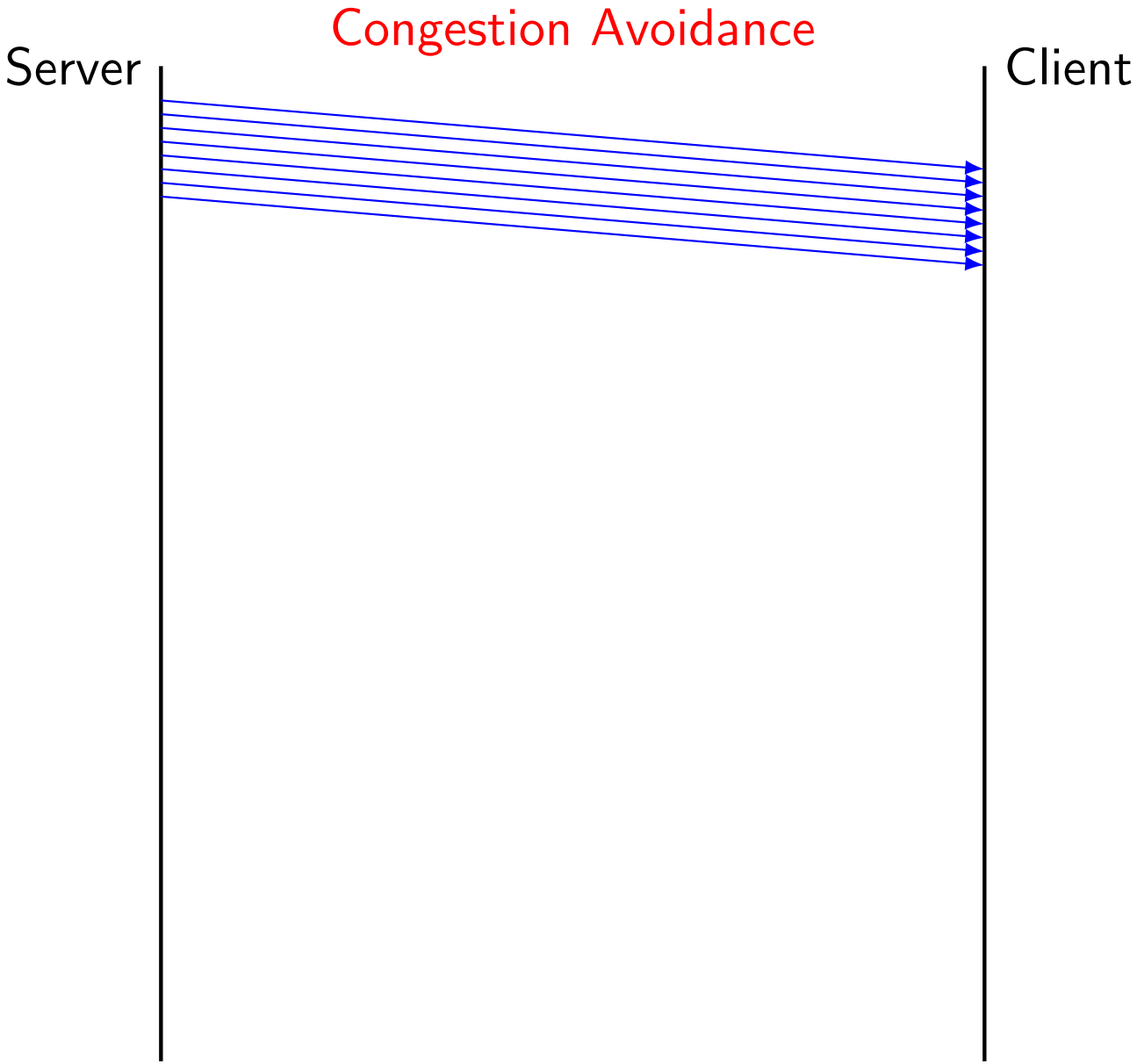
Congestion Avoidance

- Additive Increase/ Multiplicative Decrease (AIMD)
- Increment cwind by one segment (SMSS bytes) per RTT (linear increase)
- Divide cwind by two whenever a timeout occurs (multiplicative decrease)
- cwind always $\geq$ 1 SMSS
- In practice: increment a little for each ACK
  - Increment = 1/cwind
  - cwind += Increment
  - SMSS = max segment size = size of a single segment

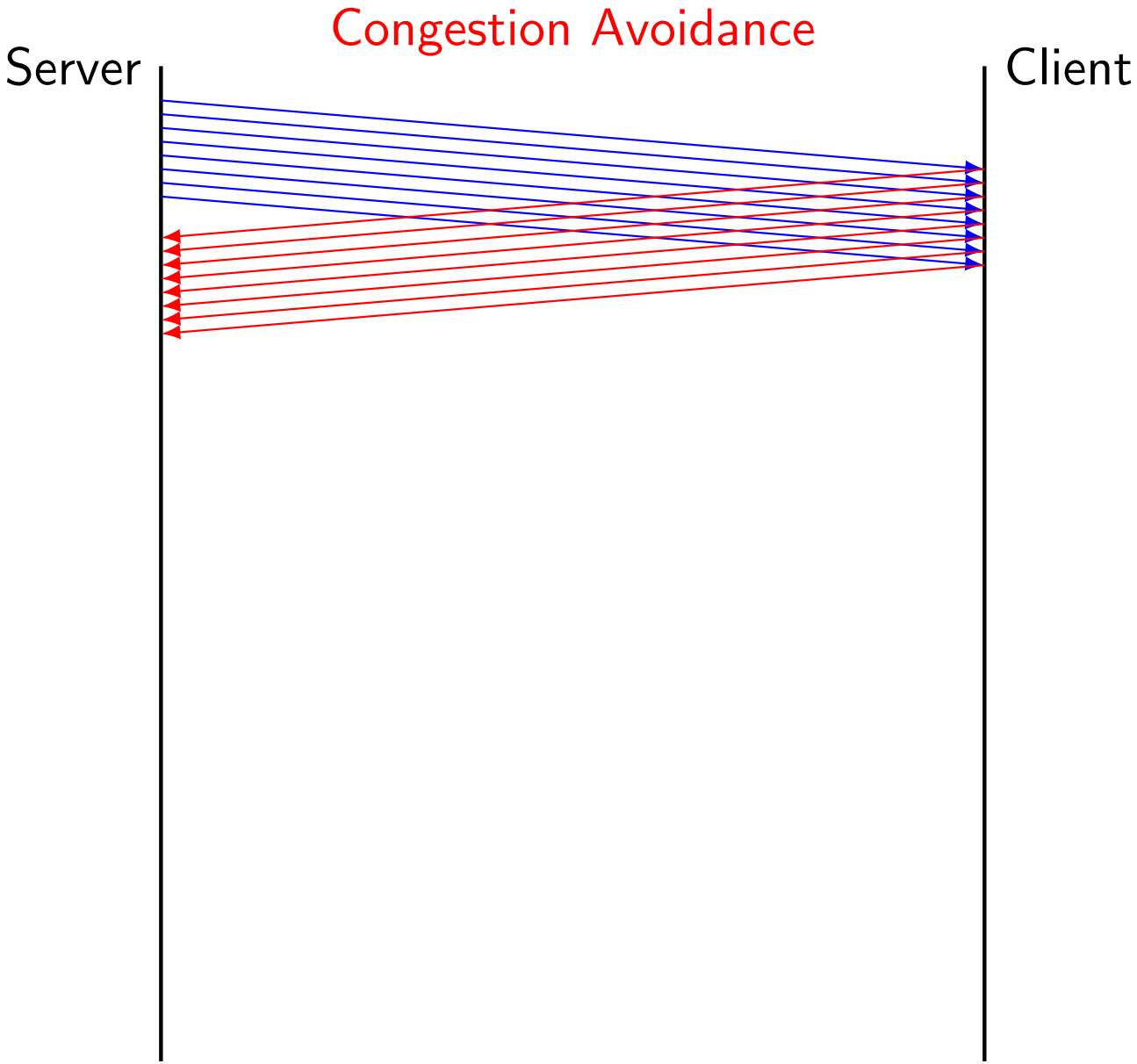
Congestion Avoidance



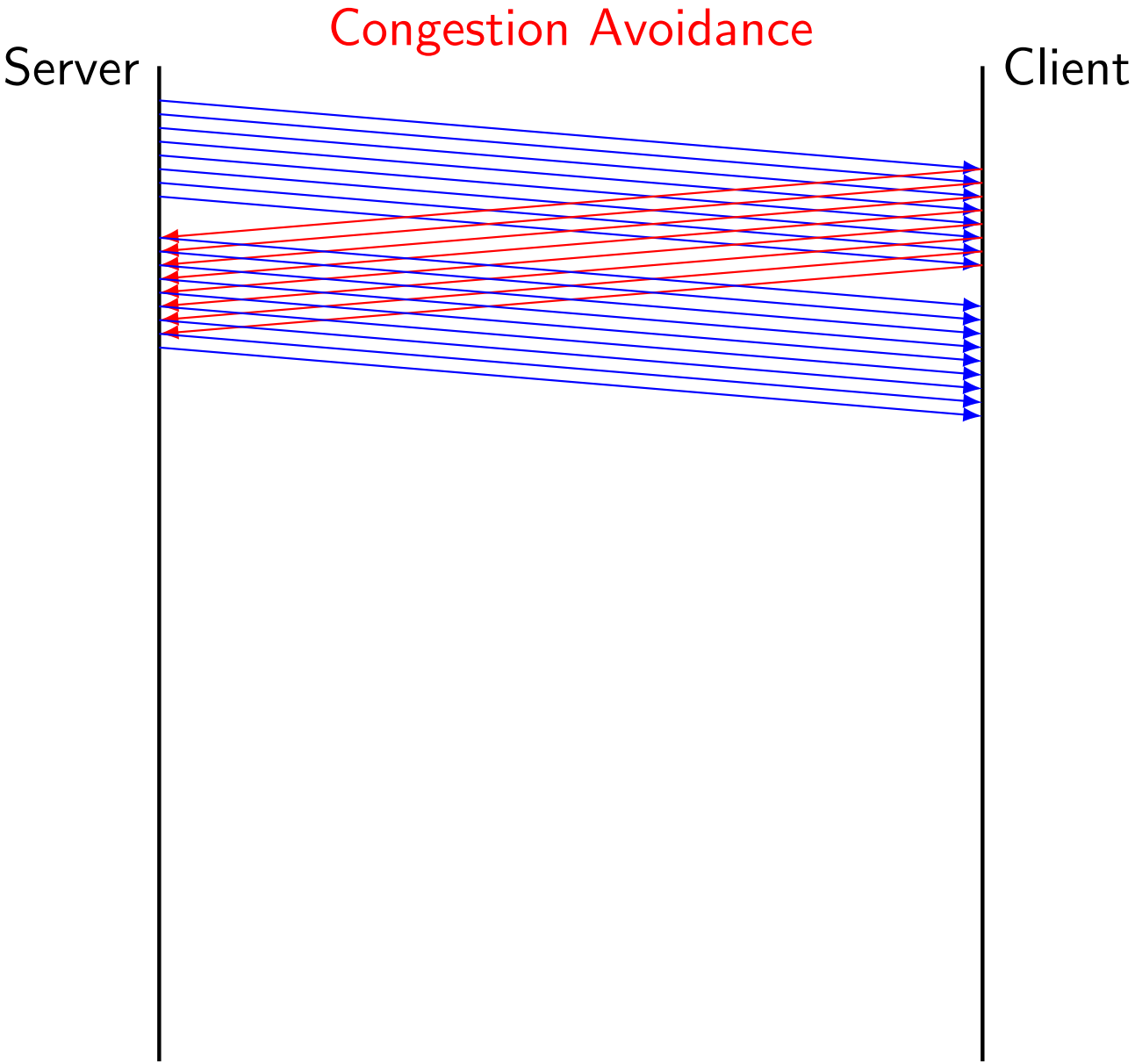
Congestion Avoidance



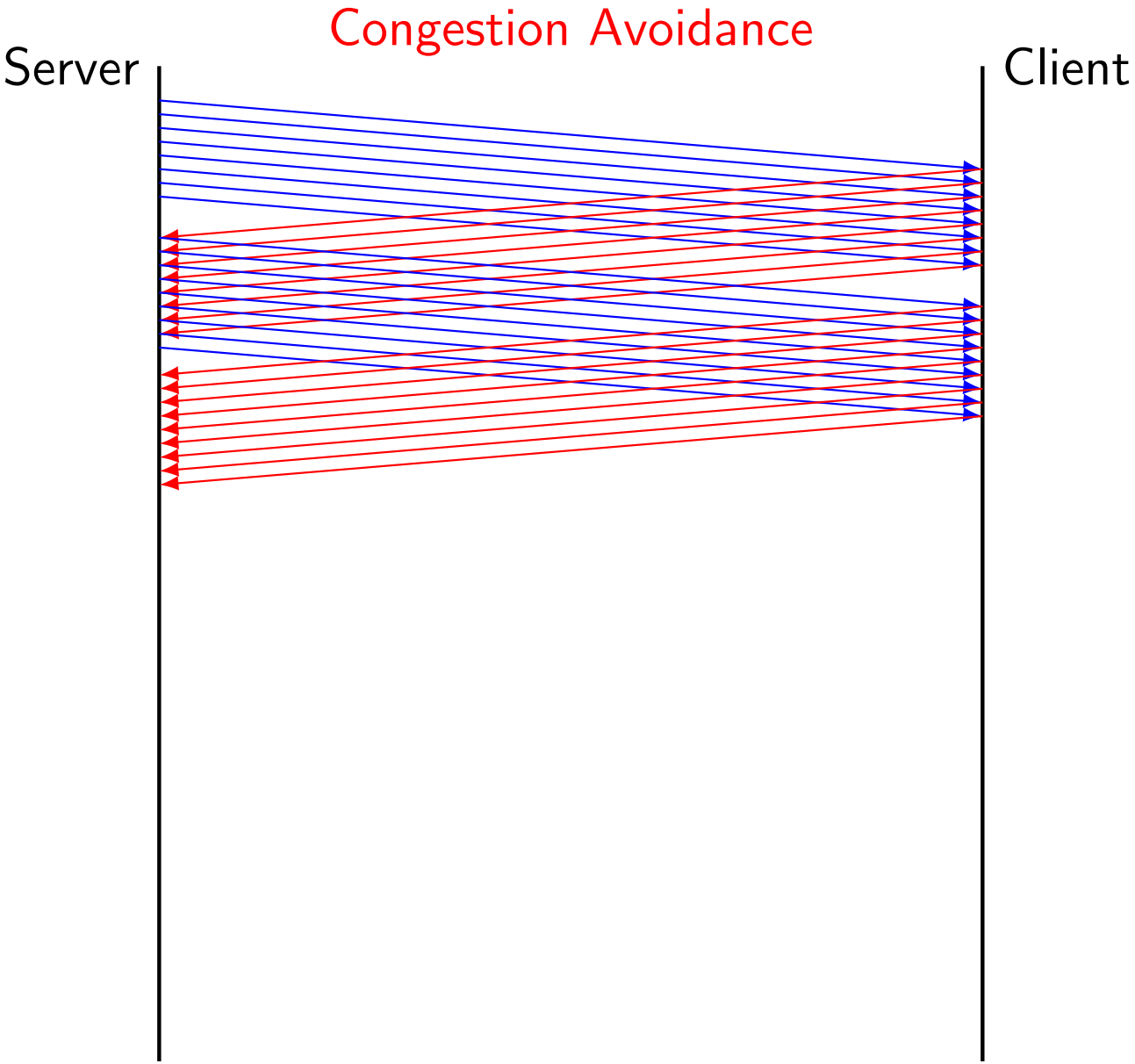
Congestion Avoidance



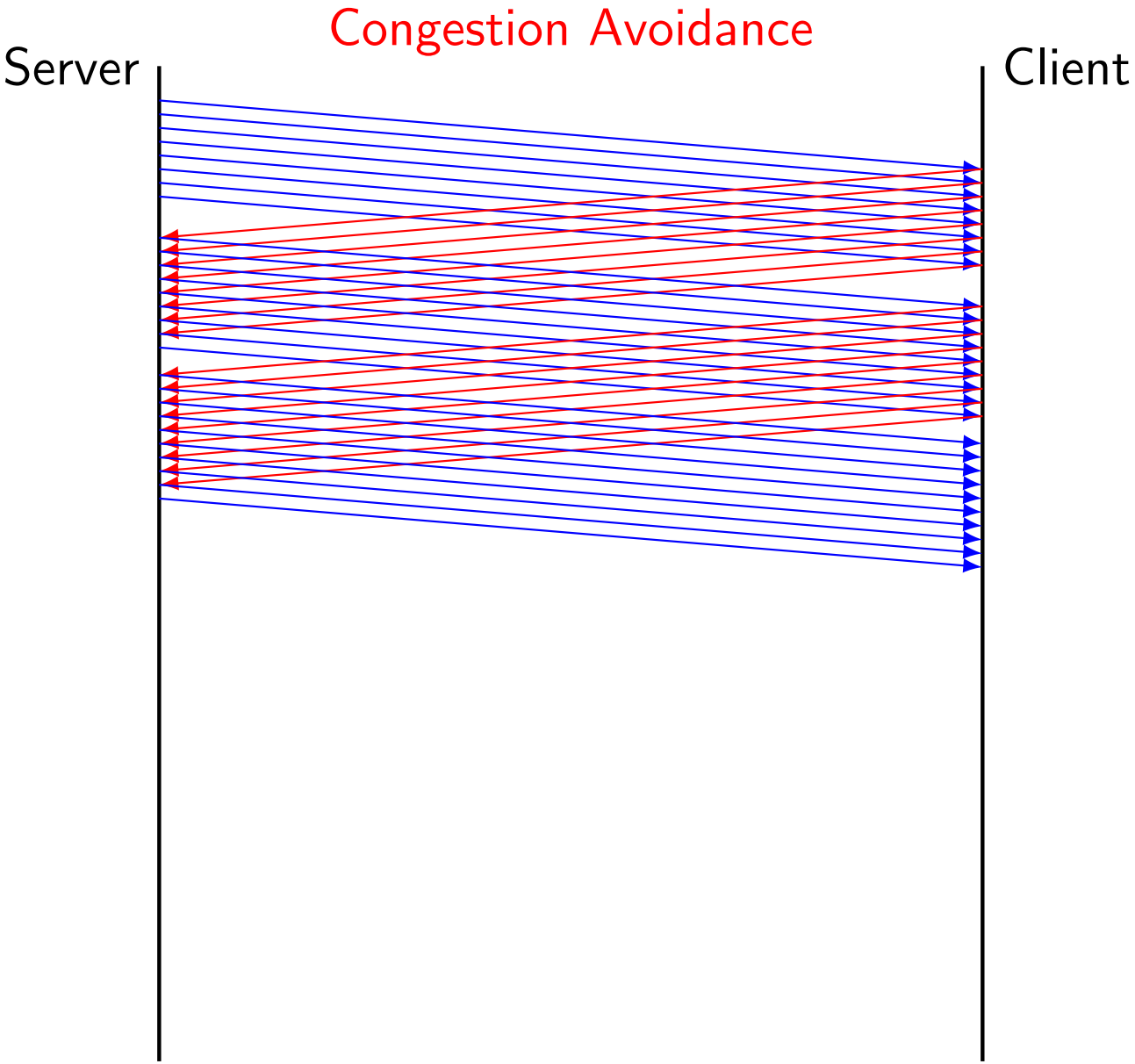
Congestion Avoidance



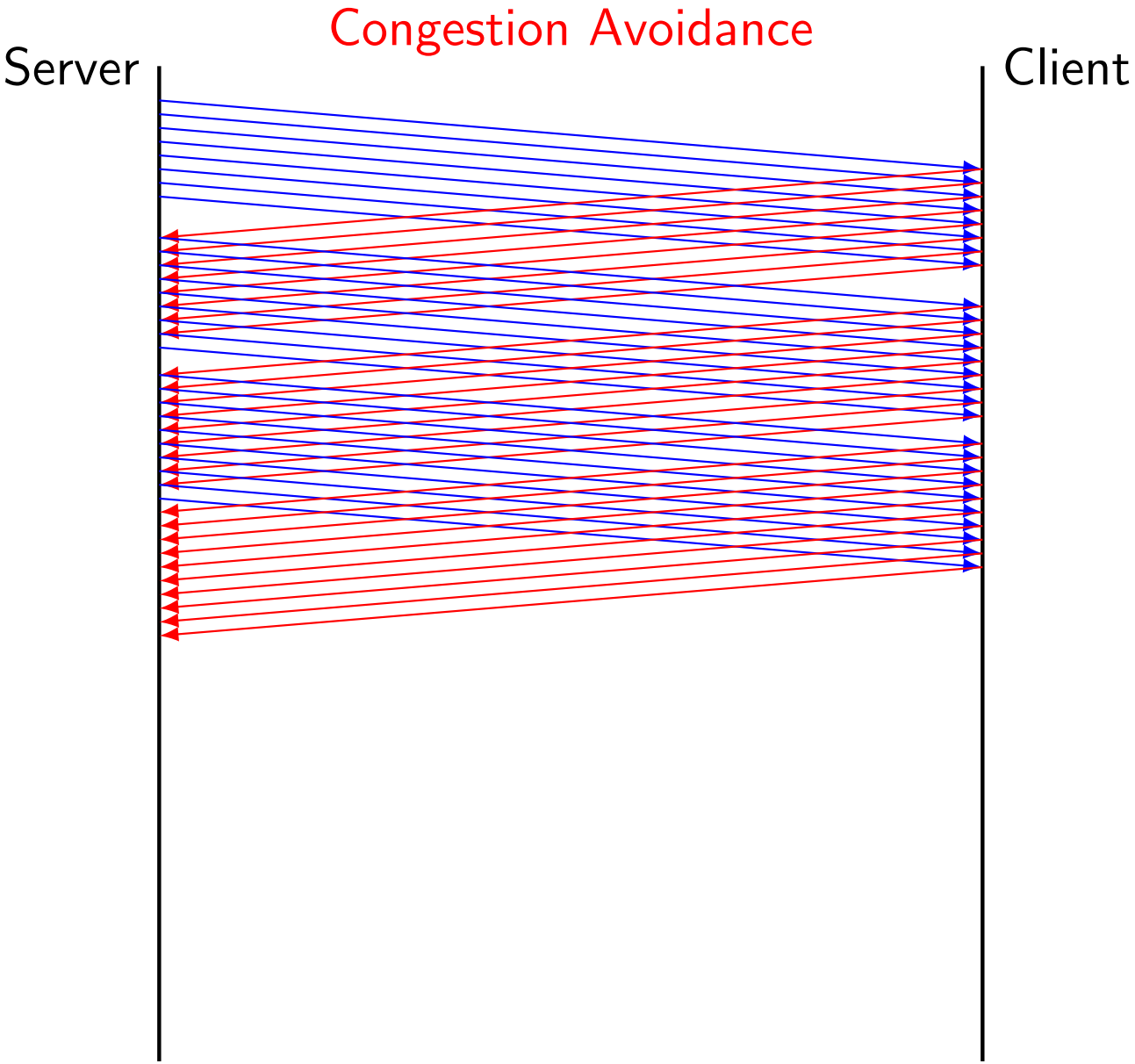
Congestion Avoidance



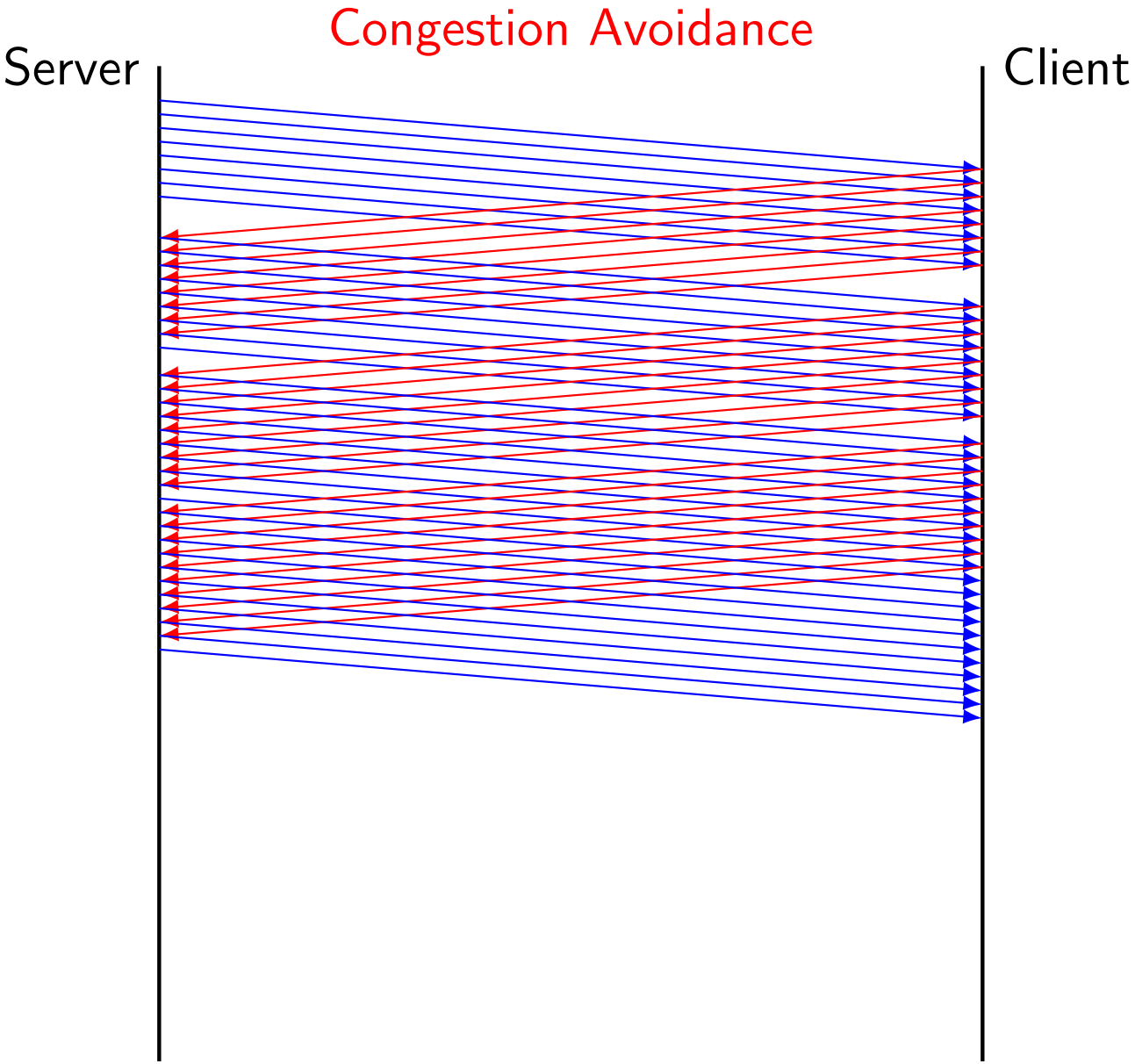
Congestion Avoidance



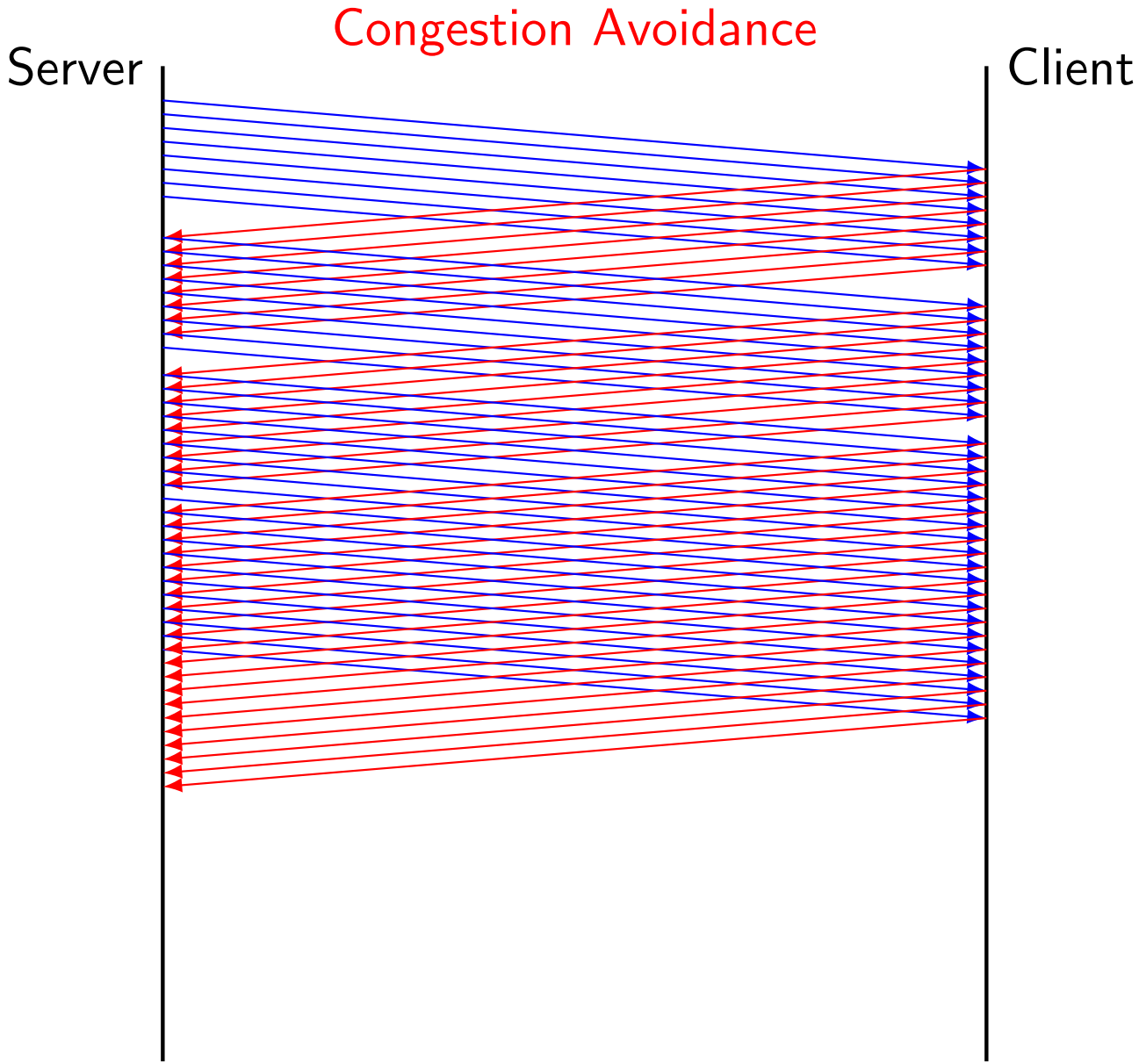
Congestion Avoidance



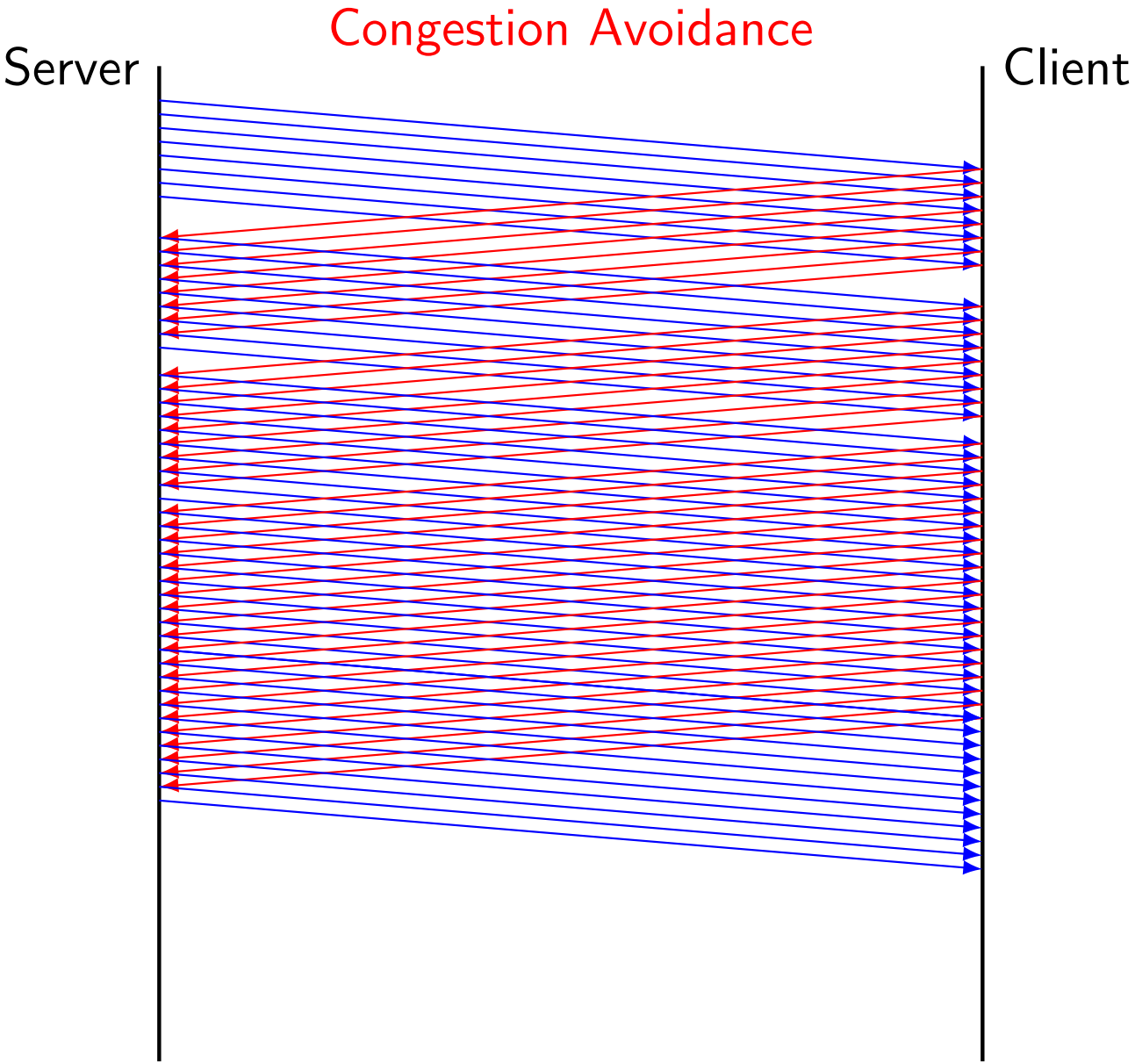
Congestion Avoidance



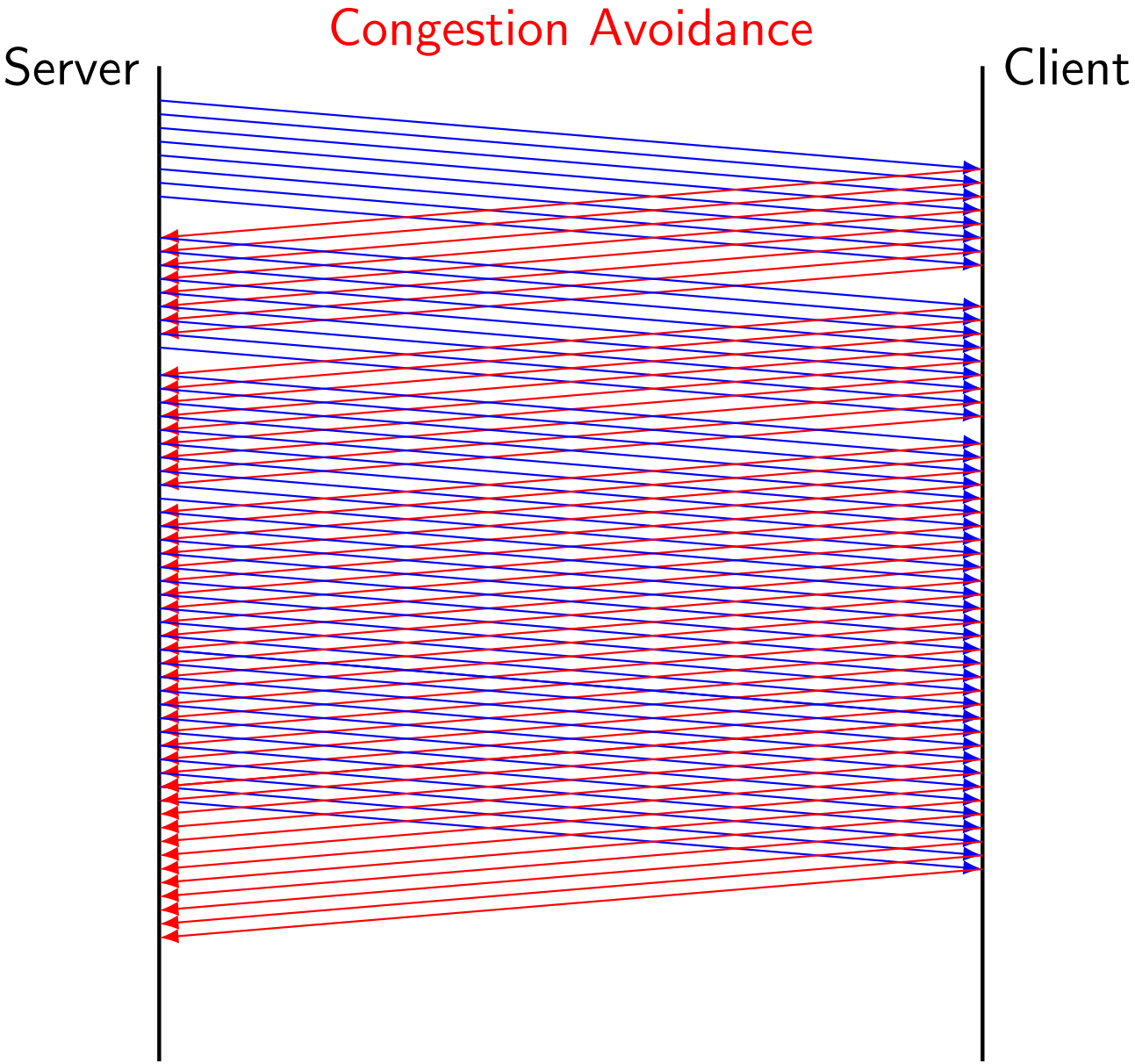
Congestion Avoidance



Congestion Avoidance



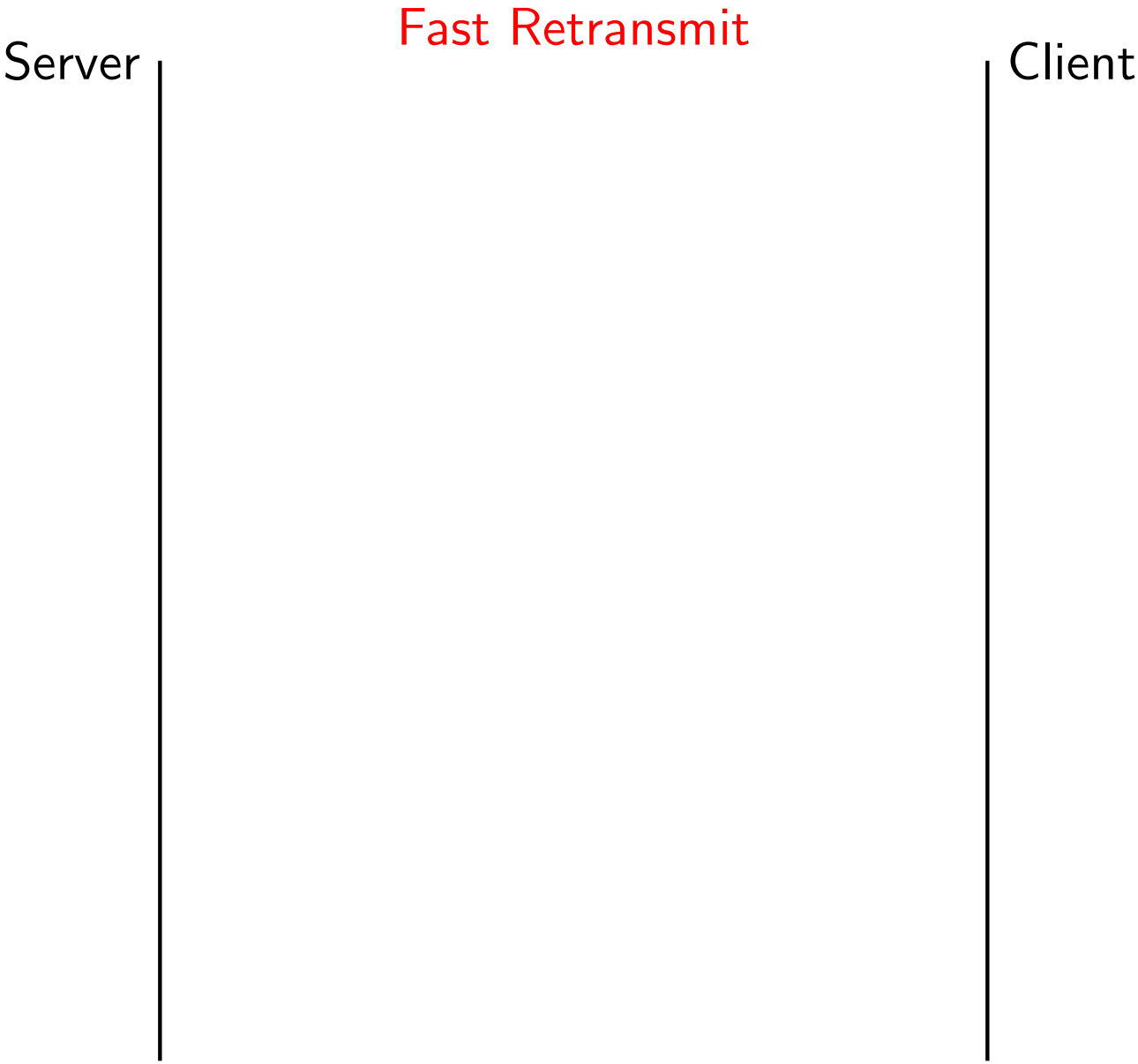
Congestion Avoidance



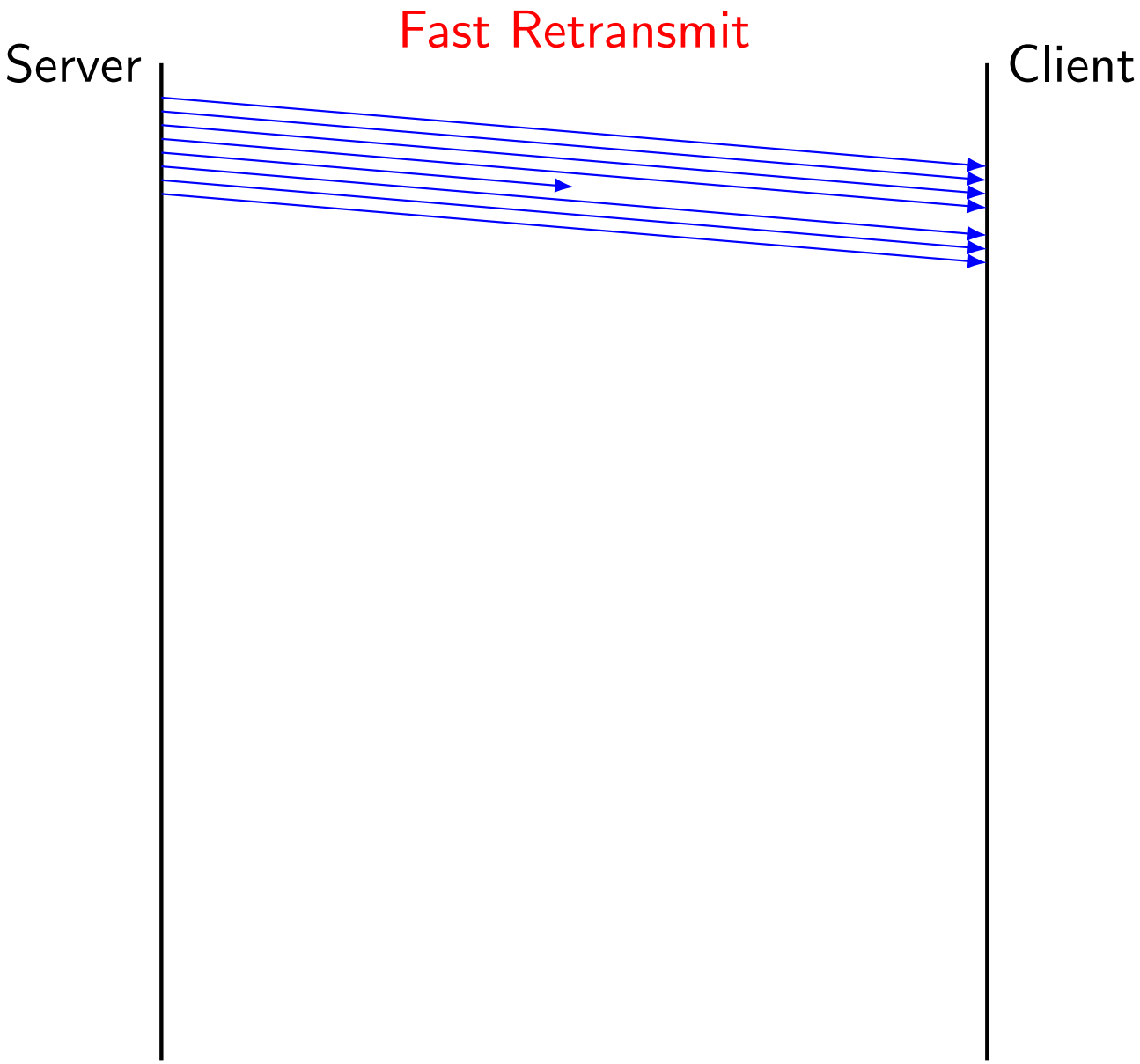
Fast Retransmit

- Receiving small number of duplicate ACKs (3) signals segment loss
- Lost segment can be retransmitted before the retransmission timer to expiration
- This improves channel utilization

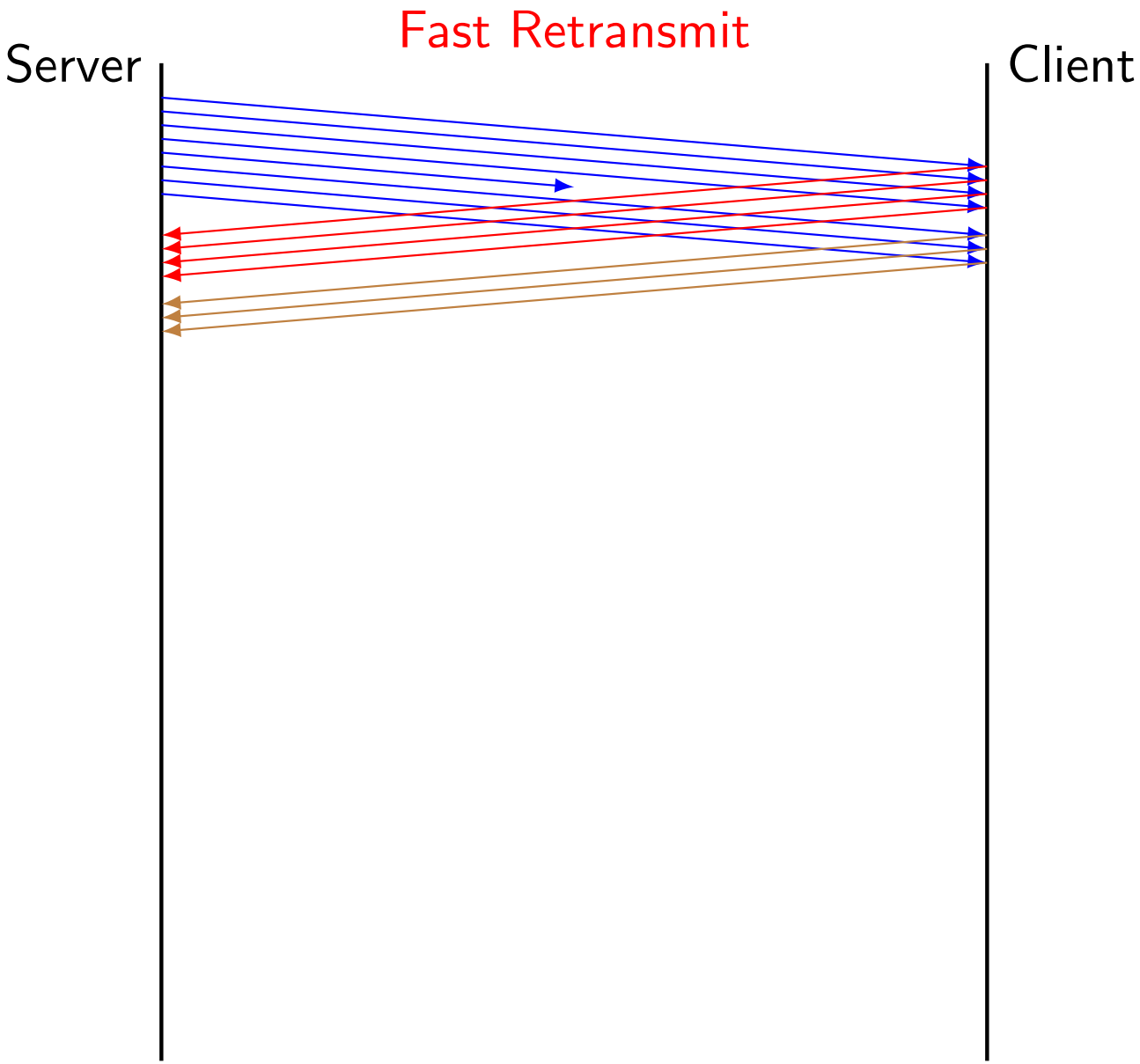
Fast Retransmit



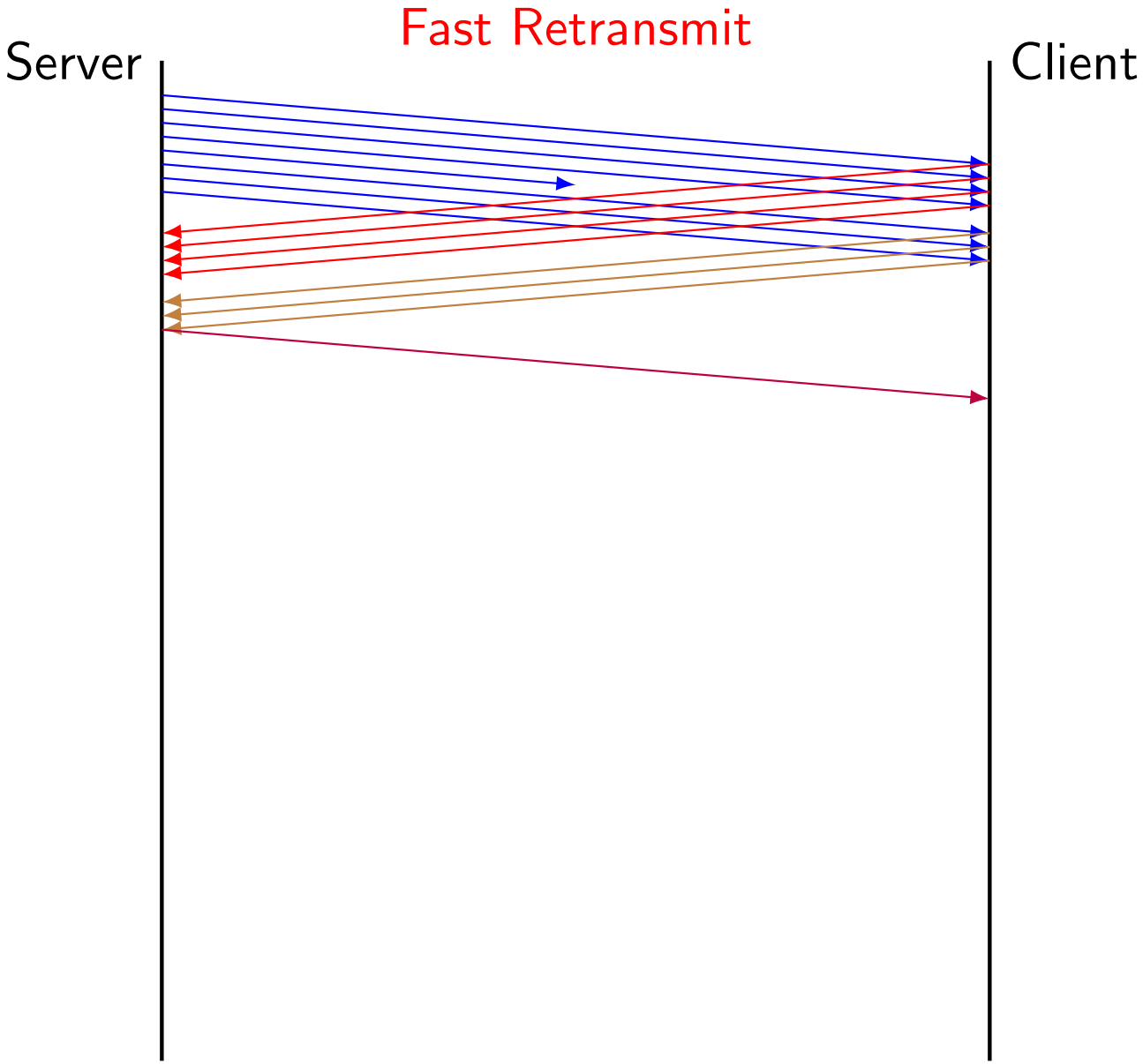
Fast Retransmit



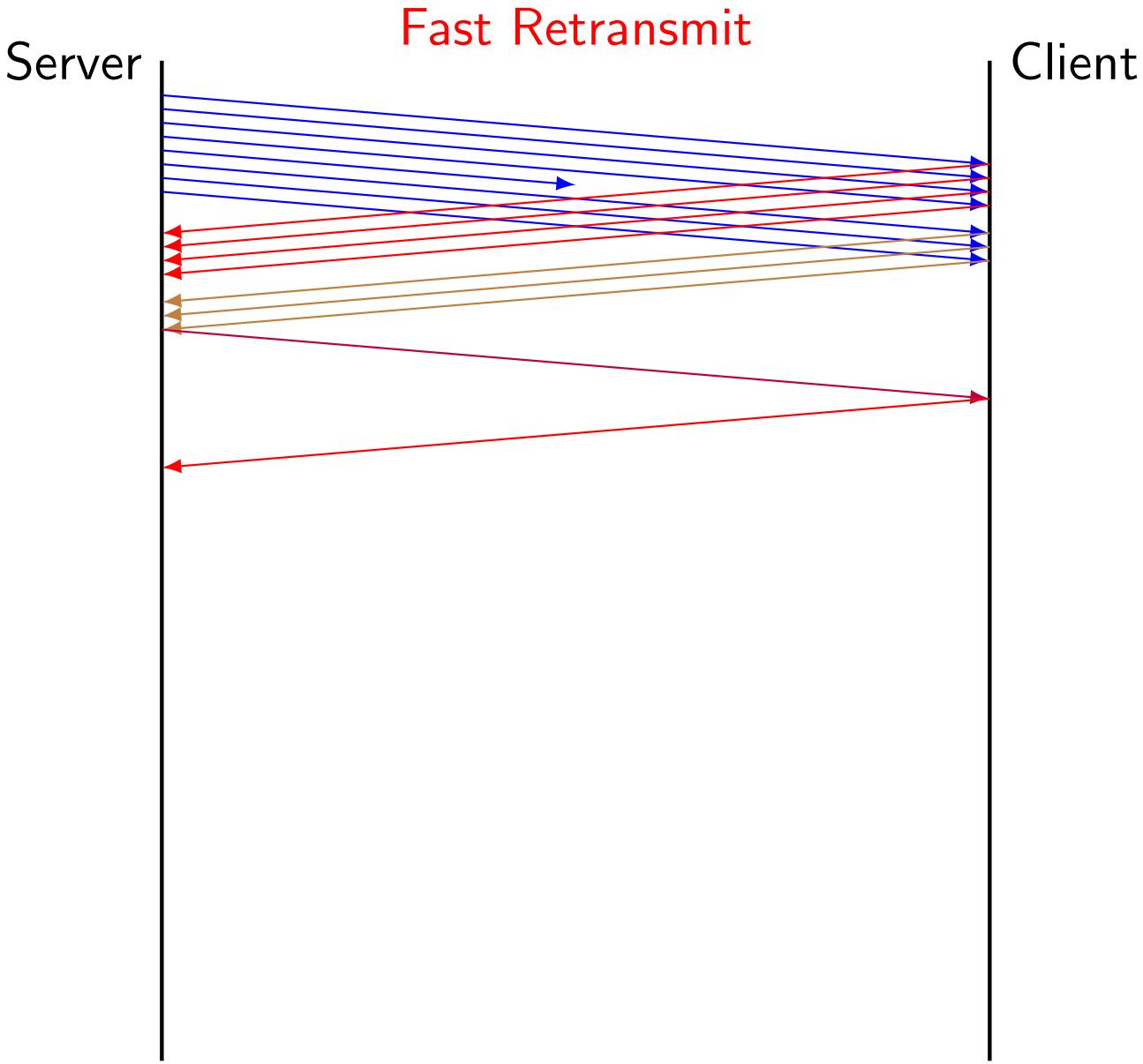
Fast Retransmit



Fast Retransmit



Fast Retransmit



Fast Recovery

- After the Fast Retransmit algorithm sends what appears to be the missing segment, the "Fast Recovery" algorithm governs the transmission of new data until a non-duplicate ACK arrives
- Each duplicate ACK indicates some packet has left pipe
- Skip the Slow Start phase
- Start at ssthresh and do additive increase after Fast Retransmit

TCP Tahoe

- TCP Tahoe (distributed with 4.3BSD Unix - 1988)
- Original implementation of Van Jacobson's mechanisms
- Includes:
  - Slow Start
  - Congestion Avoidance
  - Fast Retransmit

TCP Reno

- TCP Reno (distributed with 4.3BSD Unix - 1990)
- TCP Reno skips Slow Start (Fast Recovery) after three duplicate ACKs
- Includes:
  - Slow Start
  - Congestion Avoidance
  - Fast Retransmit
  - Fast Recovery
- With Fast Recovery, Slow Start only occurs:
  - At cold start
  - After a coarse-grain timeout
  - This is the difference between TCP Tahoe and TCP Reno

TCP New Reno

- Reno with modified Fast Recovery
- New-Reno continues with Fast Recovery if a partial ACK is received
- cwind increased by 1 for each duplicate ACK until ACK for last packet sent before loss detection is received

TCP SACKS

- Reno + selective ACKs
- Added the SACK option within ACKs
- Allows receiver to specify the range of packets that were received out of order

TCP Vegas

- Idea: source watches for some sign that router's queue is building up and congestion will happen too
- RTT grows
- Algorithm
  - Let BaseRTT be the minimum of all measured RTTs (commonly the RTT of the first packet)
  - Compute ExpectedRate = cwind/BaseRTT
  - Source calculates sending rate (ActualRate) once per RTT
  - Source compares ActualRate with ExpectedRate
    - Diff = ExpectedRate - ActualRate
    - if Diff $< \alpha$ increase cwind linearly
    - if Diff $> \beta$ decrease cwind linearly
    - if $\alpha \leq$ Diff $\leq \beta$ leave cwind unchanged
    - In practice $\alpha = 2$ segments(2*SMSS) and $\beta = 4$ segments (4*SMSS)
- Modified Slow Start Mechanism
  - Vegas allows exponentuial growth every other RTT
  - In between, the cwind stays fixed so a valid comparaison of the expected and actual rates can be made
  - when Diff $< \gamma$, Vegas changes from Slow Start mode to linear