TCP Connection-Oriented Transport

Study-Ready Notes

Compiled by Andrew Photinakis

October 17, 2025

Contents

1 TCP Overview and Key Characteristics **2**

2 TCP Segment Structure **2**
2.1 Segment Format . . . 2
2.2 Key Field Descriptions . . . 2

3 TCP Sequence Numbers and Acknowledgments **3**
3.1 Sequence Number Space . . . 3
3.2 Acknowledgment Mechanism . . . 3

4 TCP Round Trip Time and Timeout **3**
4.1 RTT Estimation Challenge . . . 3
4.2 Exponential Weighted Moving Average (EWMA) . . . 3
4.3 Timeout Interval Calculation . . . 4

5 TCP Sender and Receiver Operations **4**
5.1 TCP Sender Events . . . 4
5.2 TCP Sender Algorithm (Simplified) . . . 5

6 TCP Retransmission Scenarios **5**
6.1 Normal Operation . . . 5
6.2 Lost ACK Scenario . . . 5
6.3 Premature Timeout . . . 5
6.4 TCP Fast Retransmit . . . 5

7 TCP Flow Control **6**
7.1 Flow Control Problem . . . 6
7.2 Receive Window Mechanism . . . 6

8 TCP Connection Management **7**
8.1 Connection Establishment Need 7
8.2 Two-Way Handshake Problems 7
8.3 TCP Three-Way Handshake 7
8.4 Human Analogy: Rock Climbing 7
8.5 Connection Termination 7

9 Study Aids and Exam Preparation **8**
9.1 Key Concepts to Master 8
9.2 Practice Questions 8

10 Summary **9**

11 References **9**

1 TCP Overview and Key Characteristics

- **Point-to-point**: One sender, one receiver
- **Reliable, in-order byte stream**: No "message boundaries"
- **Full duplex data**: Bi-directional data flow in same connection
- **MSS**: Maximum Segment Size
- **Cumulative ACKs**: Acknowledgment mechanism
- **Pipelining**: TCP congestion and flow control set window size
- **Connection-oriented**: Handshaking initializes sender/receiver state
- **Flow controlled**: Sender will not overwhelm receiver

[Summary: TCP provides reliable, connection-oriented, full-duplex communication with flow control and congestion control mechanisms to ensure efficient and orderly data transfer between two endpoints.]

2 TCP Segment Structure

2.1 Segment Format

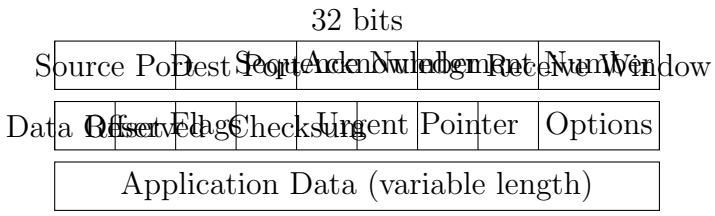

Figure 1: TCP segment structure showing key fields and their organization

2.2 Key Field Descriptions

- **Sequence Number**: Byte stream number of first byte in segment's data
- **Acknowledgment Number**: Sequence number of next expected byte (when ACK flag set)
- **Receive Window**: Number of bytes receiver is willing to accept (flow control)
- **Flags**: Control bits (SYN, ACK, FIN, RST, PSH, URG)
- **Checksum**: Internet checksum for error detection

[Mnemonic: "SP DS AN RW" - Source Port, Destination Port, Ack Number, Receive Window - key TCP header fields.]

3 TCP Sequence Numbers and Acknowledgments

3.1 Sequence Number Space

- Sequence numbers count **bytes** in the byte stream, not segments
- Each byte of data is assigned a sequence number
- Initial sequence numbers chosen during connection establishment

3.2 Acknowledgment Mechanism

- **Cumulative ACKs**: ACK(n) acknowledges all bytes up to sequence number n-1
- ACK number indicates the **next expected byte**
- Out-of-order segment handling is implementation-dependent

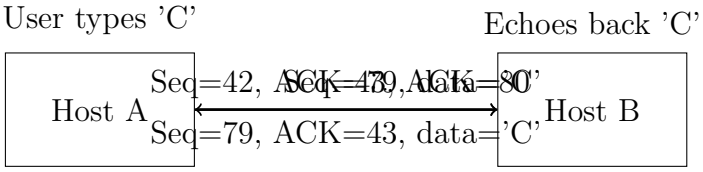

Figure 2: TCP sequence number exchange in telnet scenario

4 TCP Round Trip Time and Timeout

4.1 RTT Estimation Challenge

- Timeout must be longer than RTT, but RTT varies
- **Too short**: Premature timeout, unnecessary retransmissions
- **Too long**: Slow reaction to segment loss

4.2 Exponential Weighted Moving Average (EWMA)

$$\text{EstimatedRTT} = (1 - \alpha) \times \text{EstimatedRTT} + \alpha \times \text{SampleRTT}$$

Where:

- SampleRTT: Measured time from segment transmission to ACK receipt
- α: Smoothing factor (typically 0.125)
- Influence of past samples decreases exponentially

4.3 Timeout Interval Calculation

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 \times \text{DevRTT}$$

$$\text{DevRTT} = (1 - \beta) \times \text{DevRTT} + \beta \times |\text{SampleRTT} - \text{EstimatedRTT}|$$

Where β is typically 0.25.

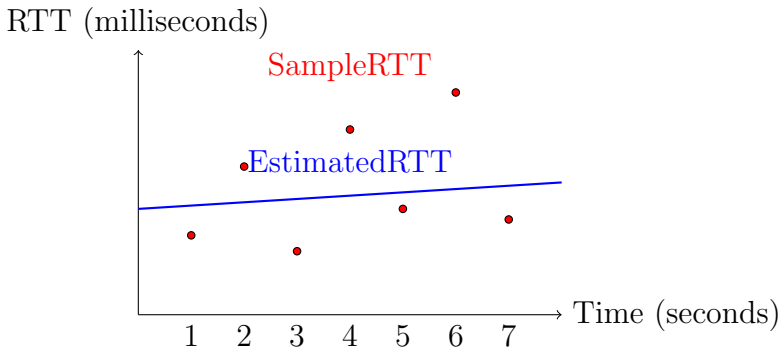

Figure 3: RTT measurement and estimation showing SampleRTT variability and smoothed EstimatedRTT

[Summary: TCP uses exponential weighted moving averages to estimate RTT and its deviation, setting timeout intervals that balance quick loss detection with avoiding premature timeouts.]

5 TCP Sender and Receiver Operations

5.1 TCP Sender Events

- **Data received from application**:
  - Create segment with sequence number
  - Sequence number is byte-stream number of first data byte
  - Start timer if not already running
- **Timeout**:
  - Retransmit segment that caused timeout
  - Restart timer
- **ACK received**:
  - If ACK acknowledges previously unACKed segments
  - Update what is known to be ACKed
  - Start timer if there are still unACKed segments

5.2 TCP Sender Algorithm (Simplified)

```
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum

while True:
    wait for event

    if ACK received with ACK field value y:
        if y > SendBase:
            SendBase = y
            if there are not-yet-acked segments:
                start timer
            else:
                stop timer

    if data received from application:
        create segment with seq \# = NextSeqNum
        pass segment to IP
        NextSeqNum = NextSeqNum + length(data)
        if timer not running:
            start timer

    if timeout:
        retransmit not-yet-acked segment with smallest seq \#
        start timer
```

6 TCP Retransmission Scenarios

6.1 Normal Operation

- Segments transmitted and acknowledged in order
- Cumulative ACKs cover all bytes up to acknowledged sequence number

6.2 Lost ACK Scenario

- ACK is lost in transit
- Subsequent cumulative ACK covers for earlier lost ACK
- No retransmission needed if later ACK arrives

6.3 Premature Timeout

- Timer expires before ACK arrives (ACK is delayed)

- Sender retransmits segment
- Receiver receives duplicate, uses sequence numbers to detect and discard

6.4 TCP Fast Retransmit

- **Triple duplicate ACKs**: Receiving 3 ACKs for same data
- Indicates segments received after a missing segment
- Lost segment is likely, so retransmit without waiting for timeout
- More responsive than timeout-based retransmission

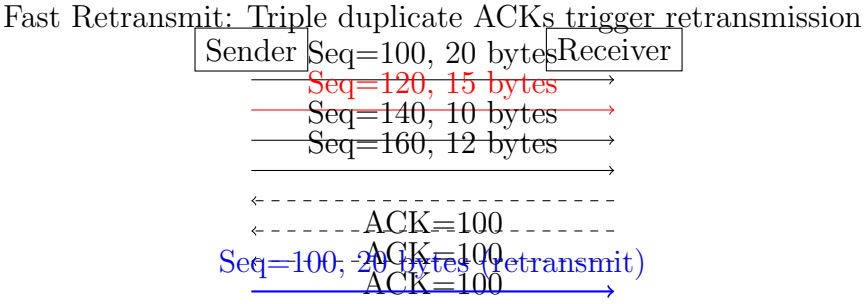

Figure 4: TCP fast retransmit mechanism triggered by triple duplicate ACKs

[Concept Map: TCP Reliability → Sequence numbers (byte counting) + Cumulative ACKs (efficiency) + Timers (loss detection) + Fast retransmit (quick recovery) = Complete reliability over unreliable networks.]

7 TCP Flow Control

7.1 Flow Control Problem

- Network layer may deliver data faster than application removes it from socket buffers
- Without control, receiver buffers would overflow
- Need mechanism to prevent sender from overwhelming receiver

7.2 Receive Window Mechanism

- TCP receiver advertises free buffer space in **rwnd** field
- **RevBuffer** size set via socket options (typically 4096 bytes)
- Many operating systems auto-adjust buffer size
- Sender limits unACKed ("in-flight") data to received rwnd value

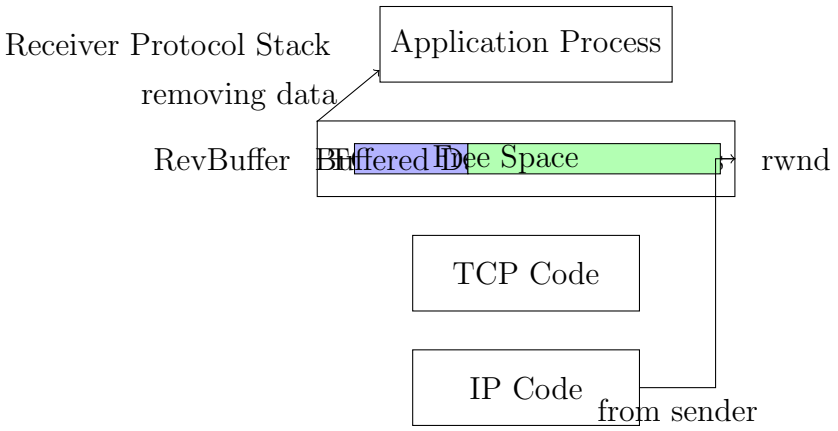

Figure 5: TCP flow control: receiver advertises free buffer space via rwnd field

8 TCP Connection Management

8.1 Connection Establishment Need

- Agree to establish connection (mutual agreement)
- Agree on connection parameters (initial sequence numbers)
- Initialize connection state at both ends

8.2 Two-Way Handshake Problems

- **Half-open connections**: Client terminates, server doesn't know
- **Duplicate data acceptance**: Delayed retransmissions accepted as new data
- **Message reordering**: Cannot handle network reordering properly

8.3 TCP Three-Way Handshake

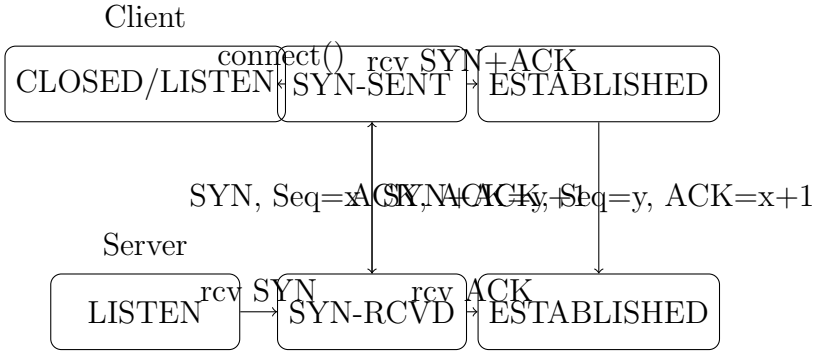

Figure 6: TCP three-way handshake: state transitions and message exchange

8.4 Human Analogy: Rock Climbing

1. **"On belay?"** (SYN): Climber asks if belayer is ready
2. **"Belay on."** (SYN-ACK): Belayer confirms readiness
3. **"Climbing."** (ACK): Climber acknowledges and begins

8.5 Connection Termination

- Client and server each close their side of connection
- Send TCP segment with FIN bit = 1
- Respond to received FIN with ACK
- FIN and ACK can be combined
- Simultaneous FIN exchanges handled gracefully

2-Way Handshake	3-Way Handshake	Advantage
Vulnerable to half-open connections	Prevents half-open connections	Complete state synchronization
Duplicate data acceptance	Sequence number validation	Prevents old duplicate confusion
Simple but unreliable	Robust and reliable	Handles network variability
Not used in practice	TCP standard	Industry standard

Table 1: Comparison of 2-way vs 3-way handshake

[Summary: TCP uses a three-way handshake for reliable connection establishment, solving problems of half-open connections and duplicate data that plague simpler two-way handshakes, with clear state synchronization.]

9 Study Aids and Exam Preparation

9.1 Key Concepts to Master

- Understand TCP segment structure and field purposes
- Be able to calculate and interpret sequence numbers and ACKs
- Know the RTT estimation formulas and timeout calculation
- Explain TCP flow control mechanism and rwnd usage
- Trace through three-way handshake and connection termination
- Compare and contrast different retransmission scenarios

9.2 Practice Questions

1. **Calculate TCP timeout** given the following: EstimatedRTT = 200ms, DevRTT = 50ms, $\alpha = 0.125$, $\beta = 0.25$. If the next SampleRTT is 180ms, what is the new TimeoutInterval?

2. **Trace the three-way handshake** between client (starting seq=1000) and server (starting seq=5000). Show the sequence and acknowledgment numbers in each segment.

3. Explain how **TCP flow control** prevents buffer overflow at the receiver. What happens if the application stops reading data from the socket?

4. Compare **timeout-based retransmission** vs **fast retransmit**. Under what conditions is each mechanism triggered, and which is more efficient?

5. A TCP sender has SendBase = 2000 and NextSeqNum = 2500. The receiver's window is 1000 bytes. How much data can the sender transmit without receiving new ACKs? What happens when rwnd becomes 0?

[Mnemonic: "SYN SYN-ACK ACK" - The three steps of TCP handshake, easy to remember for connection establishment.]

10 Summary

- TCP provides reliable, connection-oriented byte stream service
- Sequence numbers count bytes, not segments, enabling accurate tracking
- Cumulative ACKs provide efficient acknowledgment mechanism
- Adaptive timeout calculation using EWMA of RTT and its deviation
- Flow control prevents receiver buffer overflow through rwnd advertising
- Three-way handshake ensures reliable connection establishment
- Fast retransmit provides quick recovery from segment loss
- Connection management handles both establishment and termination gracefully

11 References

- Kurose, J.F., & Ross, K.W. (2020). *Computer Networking: A Top-Down Approach (8th ed.)*. Pearson.
- RFC 5681: TCP Congestion Control
- RFC 793: Transmission Control Protocol

- Course: COMPSCI 453 Computer Networks, University of Massachusetts
- Professor: Jim Kurose, College of Information and Computer Sciences
- Textbook website: http://gala.cs.umass.edu/kurose_ross
- Interactive exercises: http://gaia.cs.umass.edu/kurose_ross/interactive/