Comprehensive Guide to Transport Layer Protocols

TCP and UDP

Computer Networking: A Top-Down Approach

December 19, 2025

Contents

1 Transport Layer Overview

1.1 Role of the Transport Layer

The transport layer provides **logical communication** between application processes running on different hosts. While the network layer handles host-to-host communication, the transport layer extends this to process-to-process communication.

Key Transport Layer Actions:

- **Sender:** Breaks application messages into **segments**, adds transport header, passes to network layer
- **Receiver:** Reassembles segments into messages, passes to application layer
- Provides either **reliable** or **best-effort** delivery depending on protocol

1.2 Two Principal Internet Transport Protocols

1.2.1 TCP (Transmission Control Protocol)

- **Reliable**, in-order byte stream delivery
- **Connection-oriented** with explicit setup/teardown
- **Flow control** to prevent receiver overflow
- **Congestion control** to prevent network overload
- Examples: Web browsers (HTTP/HTTPS), email, file transfers

1.2.2 UDP (User Datagram Protocol)

- **Unreliable**, unordered datagram delivery
- **Connectionless** - no handshaking
- **No frills** extension of "best-effort" IP
- Examples: DNS, streaming media, VoIP, online games

1.3 Transport Services NOT Provided

Neither TCP nor UDP guarantees:

- **Delay bounds**
- **Minimum bandwidth guarantees**
- **Security** (though TLS/SSL can be added at application layer)

2 Multiplexing and Demultiplexing

2.1 Basic Concepts

- **Multiplexing at sender:** Handling data from multiple sockets, adding transport headers
- **Demultiplexing at receiver:** Using header information to deliver received segments to correct socket

2.2 How It Works

Key identifiers in each IP datagram:

- Source IP address
- Destination IP address
- Source port number
- Destination port number

Port numbers: 16-bit integers (0-65535)

- Well-known ports: 0-1023 (HTTP:80, HTTPS:443, DNS:53)
- Registered ports: 1024-49151
- Dynamic/private ports: 49152-65535

2.3 Connectionless (UDP) Demultiplexing

UDP socket identified by **destination port number only**

- All UDP segments with same destination port number go to same socket
- Different source IP/port combinations all directed to same socket
- Example: DNS server on port 53 receives all DNS queries

2.4 Connection-Oriented (TCP) Demultiplexing

TCP socket identified by **4-tuple**:

1. Source IP address
2. Source port number
3. Destination IP address
4. Destination port number

Each connection between client and server creates a **unique socket**. Server can support many simultaneous TCP sockets, each associated with different connecting client.

3 Connectionless Transport: UDP

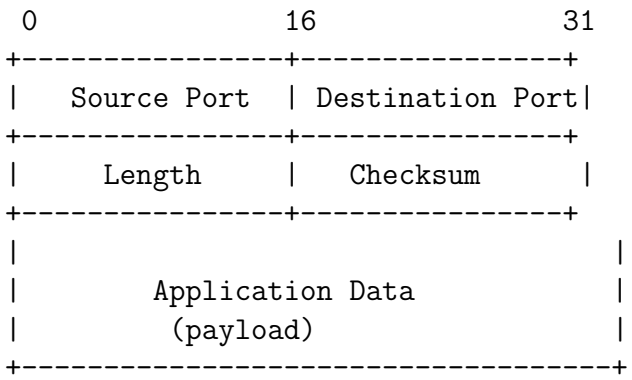
3.1 UDP Characteristics

- **"Bare bones"** Internet transport protocol
- **Best effort service:** Segments may be lost, delivered out of order
- **Connectionless:** No handshaking, each segment handled independently

3.2 Why Use UDP?

1. **No connection establishment delay** (no RTT incurred for setup)
2. **Simple:** No connection state at sender/receiver
3. **Small header size** (8 bytes vs TCP's 20+ bytes)
4. **No congestion control**
   - Can blast data as fast as desired
   - Can function when network service is compromised

3.3 UDP Segment Structure



- **Length:** Total bytes of UDP segment (header + data)
- **Checksum:** Optional error detection

3.4 UDP Checksum

Goal: Detect errors (flipped bits) in transmitted segment

Sender:

1. Treat segment content as sequence of 16-bit integers
2. Compute one's complement sum
3. Put complement of sum into checksum field

Receiver:

1. Compute checksum of received segment
2. Compare with checksum field value
3. Discard if mismatch detected

Limitation: Weak protection - some error patterns undetectable

3.5 UDP Applications

3.5.1 Traditional uses

- **DNS** (Domain Name System): Simple query/response
- **SNMP** (Simple Network Management Protocol)
- **Streaming multimedia:** Loss tolerant, rate sensitive
- **Real-time applications:** Voice/Video over IP, online games
- **DHCP** (Dynamic Host Configuration Protocol)

3.5.2 Modern uses

- **HTTP/3 (QUIC):** Built on UDP with reliability added at application layer
- **RTP (Real-time Transport Protocol):** For audio/video streaming
- **Multicast applications:** Single sender to multiple receivers

3.6 Building on UDP

When reliability needed over UDP:

- Add reliability at application layer
- Add congestion control at application layer
- **Example:** QUIC provides TCP-like reliability on UDP

4 Principles of Reliable Data Transfer

4.1 The Reliable Data Transfer Problem

Goal: Create reliable communication over unreliable channel

Challenges:

- Bit errors (corruption)
- Packet loss
- Packet reordering
- Duplicate packets
- Unknown state of receiver

4.2 Reliable Data Transfer Service Model

- **rdt_send():** Called by application to send data
- **rdt_rcv():** Called when packet arrives on network layer
- **deliver_data():** Called by RDT to deliver data to application
- **udt_send():** Called by RDT to send packet via unreliable channel

4.3 Evolution of Reliable Data Transfer Protocols

4.3.1 rdt1.0: Perfectly Reliable Channel

- Assumes underlying channel is perfectly reliable
- No bit errors, no loss
- Simple finite state machine with no feedback

4.3.2 rdt2.0: Channel with Bit Errors

- Adds **checksums** for error detection
- **Acknowledgements (ACKs):** Receiver tells sender packet received OK
- **Negative Acknowledgements (NAKs):** Receiver tells sender packet had errors
- **Stop-and-wait:** Sender sends one packet, waits for response

Problem: What if ACK/NAK gets corrupted? Solution: Add **sequence numbers**

4.3.3 rdt2.1: Handling Corrupted ACKs/NAKs

- Adds 1-bit sequence number (0 or 1)
- Receiver detects duplicate packets using sequence number
- Handles corrupted ACKs by retransmission

4.3.4 rdt2.2: NAK-Free Protocol

- Uses only ACKs (no NAKs)
- ACK includes sequence number of packet being acknowledged
- Duplicate ACK triggers retransmission (like TCP)

4.3.5 rdt3.0: Channels with Errors and Loss

- Adds **timer** for timeout/retransmission
- Handles: Lost packets, premature timeouts, delayed ACKs, duplicate packets

Performance issue: Stop-and-wait is inefficient

$$\text{Utilization} = \frac{L/R}{RTT + L/R} \tag{1}$$

For high-speed links with long RTT, utilization is very low.

4.4 Pipelined Protocols

Solution to stop-and-wait inefficiency: Allow multiple "in-flight" packets
Key changes required:

1. **Range of sequence numbers** must increase
2. **Buffering** at sender and/or receiver
3. **Error recovery mechanisms** for lost/corrupted packets

4.5 Go-Back-N (GBN) Protocol

4.5.1 Sender Side

- **Window** of up to N consecutive unACKed packets
- **Cumulative ACK:** ACK(n) acknowledges all packets up to sequence number n
- **Timer** for oldest in-flight packet
- **Timeout:** Retransmit packet n and all higher sequence number packets

4.5.2 Receiver Side

- **ACK-only:** Always send ACK for correctly received packet with highest in-order sequence number
- **Discard out-of-order packets** (or optionally buffer them)
- **Re-ACK** packet with highest in-order sequence number

4.6 Selective Repeat (SR) Protocol

4.6.1 Key Differences from GBN

- **Receiver buffers out-of-order packets**
- **Individual ACKs** for each correctly received packet
- **Timer for each unACKed packet**
- **Selective retransmission:** Only retransmit lost packets

4.6.2 The Sequence Number Space Dilemma

Problem: With window size = N and sequence number space size = M:

- Need $M > 2N$ to avoid ambiguity between old and new packets
- Otherwise, receiver can't distinguish between retransmission of old packet and new packet with same sequence number

Solution: Ensure sequence number space is at least twice the window size

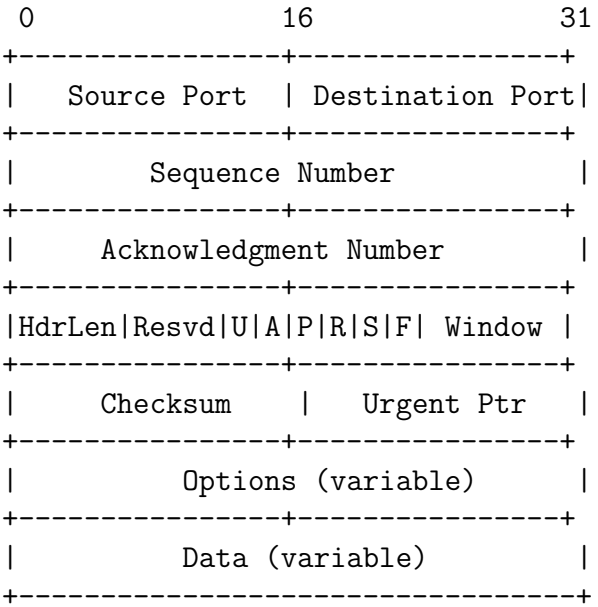
5 Connection-Oriented Transport: TCP

5.1 TCP Overview

5.1.1 Key Characteristics

- **Point-to-point:** One sender, one receiver
- **Reliable, in-order byte stream:** No message boundaries
- **Full duplex:** Bi-directional data flow
- **Pipelined:** Congestion and flow control set window size
- **Connection-oriented:** Handshaking initializes state
- **Flow controlled:** Sender won't overwhelm receiver

5.1.2 TCP Segment Structure



Key header fields:

- **Sequence number:** Byte stream "number" of first byte in segment

- **Acknowledgment number:** Sequence number of next byte expected (cumulative ACK)
- **Header length:** Size of TCP header in 32-bit words (usually 5, 20 bytes)
- **Flags:** URG, ACK, PSH, RST, SYN, FIN
- **Window:** Number of bytes receiver willing to accept (flow control)
- **Checksum:** Internet checksum of segment plus pseudo-header

5.2 TCP Sequence Numbers and ACKs

5.2.1 Byte Stream Orientation

- **Sequence numbers:** Index of first data byte in segment
- **ACKs:** Cumulative - ACK(n) means all bytes up to n-1 received

5.3 TCP Reliable Data Transfer

5.3.1 TCP Sender Events

1. **Data from application:** Create segment with sequence number, start timer if not running
2. **Timeout:** Retransmit segment that caused timeout, restart timer
3. **ACK received:** Update what is known to be ACKed, start timer if still unACKed segments

5.3.2 TCP Receiver ACK Generation

TCP Receiver Actions

- Arrival of in-order segment with expected seq # → Delayed ACK (wait up to 500ms)
- Arrival of in-order segment, one delayed ACK pending → Immediately send cumulative ACK
- Arrival of out-of-order segment → Immediately send duplicate ACK
- Arrival of segment that fills gap → Immediately send ACK

5.4 TCP Fast Retransmit

Algorithm:

1. When sender receives **3 duplicate ACKs** for same data
2. **Resend oldest unACKed segment** immediately

3. Don't wait for timeout

Rationale: 3 duplicate ACKs indicate later segments arrived, so network is still delivering packets.

5.5 TCP Flow Control

5.5.1 The Flow Control Problem

- Application process may read data slowly from socket buffers
- Network layer may deliver data faster than application consumes it
- Risk: Receive buffer overflow, dropped packets

5.5.2 TCP Flow Control Mechanism

- **Receiver advertises free buffer space** in rwnd field of TCP header
- **Sender limits in-flight data** to rwnd
- **Receiver:** rwnd = RcvBuffer − [LastByteRcvd − LastByteRead]
- **Sender:** LastByteSent − LastByteAcked ≤ rwnd

5.6 TCP Connection Management

5.6.1 The TCP Three-Way Handshake

```
Client                          Server
1. SYNbit=1, Seq=x         -->
2.                         <--  SYNbit=1, Seq=y,
                                 ACKbit=1, ACKnum=x+1
3. ACKbit=1, ACKnum=y+1    -->
```

5.6.2 TCP Connection Teardown

Four-way handshake:

```
Client                      Server
1. FINbit=1, Seq=x       -->
2.                      <--   ACKbit=1, ACKnum=x+1
3.                      <--   FINbit=1, Seq=y
4. ACKbit=1, ACKnum=y+1 -->
```

TIME_WAIT State: Wait 2*MSL (Maximum Segment Lifetime, typically 2 minutes) to ensure reliable termination and allow old duplicates to expire.

5.7 TCP Round Trip Time Estimation

5.7.1 Estimating RTT

Exponential Weighted Moving Average (EWMA):

$$\text{EstimatedRTT} = (1 - \alpha) \times \text{EstimatedRTT} + \alpha \times \text{SampleRTT} \tag{2}$$

where α typically $= 0.125$

5.7.2 RTT Deviation Estimation

$$\text{DevRTT} = (1 - \beta) \times \text{DevRTT} + \beta \times |\text{SampleRTT} - \text{EstimatedRTT}| \tag{3}$$

where β typically $= 0.25$

5.7.3 Timeout Interval Calculation

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

6 Principles of Congestion Control

6.1 What is Congestion?

Informally: "Too many sources sending too much data too fast for network to handle"
Manifestations:

- Long delays (queuing in router buffers)
- Packet loss (buffer overflow)
- Decreased throughput

Difference from flow control:

- **Flow control:** Receiver vs sender (end-to-end)
- **Congestion control:** Network vs senders (network-wide)

6.2 Approaches to Congestion Control

6.2.1 End-to-End Congestion Control

- No explicit network feedback
- Congestion inferred from observed loss/delay
- TCP's approach: Loss $\rightarrow$ congestion $\rightarrow$ reduce sending rate

6.2.2 Network-Assisted Congestion Control

- Routers provide direct feedback to senders
- Examples: ECN (Explicit Congestion Notification), ATM, DECbit

7 TCP Congestion Control

7.1 TCP Congestion Control Overview

Key variable: cwnd (congestion window)

- Limits amount of unACKed ("in-flight") data
- LastByteSent − LastByteAcked ≤ min(cwnd, rwnd)
- **Sending rate** ≈ cwnd/RTT bytes/sec

7.2 TCP Slow Start

7.2.1 Algorithm

- **Initial cwnd = 1 MSS** (Maximum Segment Size)
- **Increase cwnd by 1 MSS** for each ACK received
- **Effect:** Doubles cwnd every RTT (exponential growth)

7.2.2 Slow Start Threshold (ssthresh)

- Switch point from slow start to congestion avoidance
- Initial value: Typically 65,535 bytes
- After loss: Set to cwnd/2

7.3 Congestion Avoidance

7.3.1 Algorithm

- **Increase cwnd by 1 MSS per RTT**
- **Implementation:** Increase cwnd by MSS×(MSS/cwnd) per ACK
- **Additive increase:** Linear growth vs slow start's exponential

7.4 Fast Retransmit and Fast Recovery

7.4.1 Fast Recovery (TCP Reno)

After fast retransmit:

1. ssthresh = cwnd/2
2. cwnd = ssthresh + 3 × MSS (account for 3 duplicate ACKs)
3. Enter congestion avoidance (not slow start)

Rationale: Duplicate ACKs indicate network still delivering packets

7.4.2 TCP Tahoe vs TCP Reno

- **TCP Tahoe:** On any loss, cwnd = 1, enter slow start
- **TCP Reno:** On triple duplicate ACK, use fast recovery; on timeout, cwnd = 1

7.5 TCP Throughput Analysis

7.5.1 Average TCP Throughput

Simplified model: Window size varies between W/2 and W

- Average window size = 0.75W
- Average throughput = 0.75W / RTT bytes/sec

7.5.2 Throughput vs Loss Rate

Mathis Equation:

$$\text{Average TCP throughput} \approx \frac{1.22 \times \text{MSS}}{\text{RTT} \times \sqrt{L}} \tag{5}$$

where L = packet loss probability

7.6 TCP Fairness

Fairness Goal: If K TCP sessions share bottleneck link of bandwidth R, each should get average rate of R/K.

Under idealized conditions: TCP converges to fair allocation through AIMD (Additive Increase, Multiplicative Decrease).

In practice:

- Different RTTs: Connections with smaller RTT get higher throughput
- Parallel connections: Applications can open multiple connections
- UDP traffic: Doesn't respond to congestion
- Non-TCP protocols: May not implement AIMD

7.7 Explicit Congestion Notification (ECN)

7.7.1 How ECN Works

1. End hosts indicate ECN capability during TCP handshake
2. Routers mark packets when experiencing congestion
3. Receivers echo congestion indication in ACKs
4. Senders reduce rate as if packet loss occurred

Benefits:

- Early congestion notification before packet loss
- Reduced packet loss and retransmissions
- Improved performance for loss-sensitive applications

8 Evolution of Transport-Layer Functionality

8.1 BBR (Bottleneck Bandwidth and RTT)

8.1.1 Philosophy

- Model-based rather than loss-based
- Directly measures: (1) Bottleneck bandwidth, (2) Round-trip propagation delay
- Goal: Keep bottleneck link full but not overfilled

8.1.2 BBR States

1. **Startup:** Exponential growth (like slow start)
2. **Drain:** Drain queue built during startup
3. **ProbeBW:** Cycle between higher/lower rates to measure bandwidth
4. **ProbeRTT:** Occasionally reduce rate to measure minimum RTT

8.2 QUIC (Quick UDP Internet Connections)

8.2.1 Why QUIC?

Problems with TCP+TLS for HTTP:

1. Head-of-line blocking: Loss of one packet delays entire stream
2. Slow handshake: 1-3 RTTs for TCP+TLS setup
3. Ossification: Middleboxes interfere with TCP evolution

8.2.2 QUIC Design

- Transport on top of UDP: Bypasses middlebox interference
- Integrated security: TLS 1.3 built-in
- Stream multiplexing: Multiple independent streams
- Connection migration: Survive IP address changes

8.2.3 QUIC Features

1. 0-RTT connection establishment: Resume previous connections
2. Forward error correction: Optional redundancy
3. Improved congestion control: Can evolve faster than TCP
4. Per-stream flow control: Independent of connection flow control

8.3 Other Transport Protocols

8.3.1 RTP (Real-time Transport Protocol)

- For real-time applications: VoIP, video conferencing
- Runs over UDP
- Features: Sequence numbers, timestamps, payload type identification, SSRC

8.3.2 SCTP (Stream Control Transmission Protocol)

- Message-oriented (like UDP) but reliable (like TCP)
- Multi-homing: Multiple network paths
- Multi-streaming: Multiple independent streams within connection

9 Summary and Practical Applications

9.1 TCP vs UDP: Decision Guide

Use TCP when:

- Reliability is critical (file transfer, web pages, email)
- In-order delivery needed (most applications)
- Flow control needed (mismatched sender/receiver speeds)
- Congestion control needed (fair sharing of network)

Use UDP when:

- Low latency is critical (games, voice, real-time video)
- Loss tolerance is high (live streaming)
- Simple query/response (DNS, DHCP)
- Multicast/broadcast (video distribution)
- Building custom protocol (QUIC, RTP)

9.2 Key Takeaways

1. TCP provides reliability at the cost of overhead and latency
2. UDP provides simplicity and low latency but requires applications to handle reliability
3. Congestion control is essential for network stability and fairness
4. Modern protocols like QUIC are rethinking transport layer design
5. The right choice depends on application requirements and network conditions

9.3 Future Directions

9.3.1 QUIC and HTTP/3 Adoption

- Increasing deployment across Internet
- Potential to replace TCP+TLS for web traffic
- Challenges: UDP blocking, middlebox interference

9.3.2 Multipath Transport

- MPTCP for smartphones, multi-homed hosts
- Multipath QUIC in development
- Benefits: Resilience, bandwidth aggregation

9.3.3 Integration with New Network Technologies

- 5G networks: Low latency, network slicing
- Satellite Internet: High delay, intermittent connectivity
- IoT networks: Constrained devices, unusual traffic patterns

10 Advanced Topics

10.1 Multipath TCP (MPTCP)

10.1.1 Concept

Use multiple network paths simultaneously for:

- Increased throughput
- Resilience to path failure
- Applications: Smartphones (WiFi + cellular), data centers

10.1.2 Design Challenges

- Congestion control across paths: Avoid shifting congestion
- Reordering: Different paths have different delays
- Middlebox compatibility: Many middleboxes break MPTCP

10.2 TCP in Data Centers

10.2.1 Special Characteristics

- Very low latency requirements (microseconds)
- High bandwidth (10-100 Gbps)
- Short flows (mice) and long flows (elephants) mixed
- Bursty traffic patterns

10.2.2 Data Center TCP Variants

- **DCTCP:** Uses ECN for fine-grained congestion feedback
- **TIMELY:** Uses RTT measurements for congestion detection
- **pHost:** Per-host congestion control to avoid incast

10.3 Performance Enhancing Proxies (PEPs)

10.3.1 Problem

- TCP performs poorly on high-loss, high-delay links (satellite, cellular)
- TCP confuses wireless loss with congestion loss

10.3.2 PEP Solutions

- Split TCP connections: Break end-to-end TCP into segments
- Local retransmissions: Recover from wireless losses quickly
- Compression: Reduce data volume
- **Controversial:** Breaks end-to-end semantics, interferes with encryption

10.4 Transport Layer Security Integration

10.4.1 TLS over TCP

- Traditional approach: TLS runs on top of TCP
- Problem: Two layered handshakes (TCP + TLS)
- Solution: TLS False Start, TLS 1.3 1-RTT/0-RTT handshake

10.4.2 QUIC Approach

- Integrated transport and security
- Always encrypted (even metadata)
- Prevents ossification: Middleboxes can't observe/inspect protocol

11 Debugging and Troubleshooting

11.1 Common TCP Issues

1. **Connection refused:** Server not listening on port
2. **Connection timeout:** Firewall blocking, network unreachable
3. **Slow performance:**
   - High RTT: Geographical distance, congestion
   - Packet loss: Network issues, wireless problems
   - Small window: Receiver flow control limiting
4. **Retransmissions:** Network loss, congestion

11.2 Tools

- **Wireshark:** Packet capture and analysis
- **tcpdump:** Command-line packet capture
- **netstat:** Connection statistics
- **ss:** Modern socket statistics (replaces netstat)
- **iperf:** Network performance testing

12 Best Practices

12.1 For Application Developers

1. Choose the right transport protocol based on needs
2. Implement proper error handling for UDP applications
3. Use standard ports for well-known services
4. Consider connection pooling for TCP applications
5. Implement timeouts for all network operations

12.2 For Network Administrators

1. Don't block ICMP: Needed for PMTUD (Path MTU Discovery)
2. Configure appropriate buffers: Too small causes loss, too large increases delay
3. Enable ECN on routers and hosts
4. Monitor retransmission rates for network health
5. Consider QoS for latency-sensitive applications

13 Conclusion

The transport layer provides critical services for end-to-end communication in computer networks. TCP and UDP, while designed decades ago, continue to evolve to meet the demands of modern applications and networks. Understanding their principles, trade-offs, and evolution is essential for network engineers, application developers, and anyone working with networked systems.

As networks continue to evolve with 5G, IoT, and new applications, transport protocols will continue to adapt, but the fundamental principles of reliable data transfer, flow control, and congestion control will remain essential.

14 Appendix: Key Formulas

14.1 RTT Estimation

$$\text{EstimatedRTT} = (1 - \alpha) \times \text{EstimatedRTT} + \alpha \times \text{SampleRTT} \tag{6}$$

$$\text{DevRTT} = (1 - \beta) \times \text{DevRTT} + \beta \times |\text{SampleRTT} - \text{EstimatedRTT}| \tag{7}$$

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 \times \text{DevRTT} \tag{8}$$

14.2 TCP Throughput

$$\text{Average throughput} \approx \frac{1.22 \times \text{MSS}}{\text{RTT} \times \sqrt{L}} \tag{9}$$

14.3 Stop-and-Wait Utilization

$$U_{\text{sender}} = \frac{L/R}{\text{RTT} + L/R} \tag{10}$$

14.4 Flow Control

$$\text{rwnd} = \text{RcvBuffer} - [\text{LastByteRcvd} - \text{LastByteRead}] \tag{11}$$

$$\text{LastByteSent} - \text{LastByteAcked} \leq \min(\text{cwnd}, \text{rwnd}) \tag{12}$$

14.5 Congestion Control

$$\text{Sending rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec} \tag{13}$$

15 Appendix: TCP State Diagram

TCP Connection States

Connection Establishment:

- CLOSED → LISTEN → SYN_RCVD → ESTABLISHED
- CLOSED → SYN_SENT → ESTABLISHED

Data Transfer:

- ESTABLISHED (bidirectional data flow)

Connection Termination:

- ESTABLISHED → FIN_WAIT_1 → FIN_WAIT_2 → TIME_WAIT → CLOSED
- ESTABLISHED → CLOSE_WAIT → LAST_ACK → CLOSED

16 Appendix: Protocol Comparison Table

Feature	TCP	UDP	QUIC	SCTP
Connection-oriented	Yes	No	Yes	Yes
Reliable	Yes	No	Yes	Yes
Ordered delivery	Yes	No	Yes	Yes
Flow control	Yes	No	Yes	Yes
Congestion control	Yes	No	Yes	Yes
Stream multiplexing	No	No	Yes	Yes
Message boundaries	No	Yes	Yes	Yes
Built-in security	No	No	Yes	No
Header size (bytes)	20+	8	Variable	12+
Multipath support	Extension	No	Extension	Yes

Table 1: Comparison of Transport Layer Protocols

Service	Port	Protocol
HTTP	80	TCP
HTTPS	443	TCP
FTP (Data)	20	TCP
FTP (Control)	21	TCP
SSH	22	TCP
Telnet	23	TCP
SMTP	25	TCP
DNS	53	UDP/TCP
DHCP (Server)	67	UDP
DHCP (Client)	68	UDP
TFTP	69	UDP
POP3	110	TCP
IMAP	143	TCP
SNMP	161	UDP
HTTPS (Alt)	8080	TCP

Table 2: Common Well-Known Port Numbers

17 Appendix: Common Port Numbers

18 References

1. Kurose, J. F., & Ross, K. W. (2021). *Computer Networking: A Top-Down Approach* (8th ed.). Pearson.

2. RFC 793: Transmission Control Protocol. Internet Engineering Task Force (IETF).

3. RFC 768: User Datagram Protocol. Internet Engineering Task Force (IETF).

4. RFC 9000: QUIC: A UDP-Based Multiplexed and Secure Transport. Internet Engineering Task Force (IETF).

5. RFC 5681: TCP Congestion Control. Internet Engineering Task Force (IETF).

6. RFC 3168: The Addition of Explicit Congestion Notification (ECN) to IP. Internet Engineering Task Force (IETF).

7. Cardwell, N., et al. (2016). BBR: Congestion-Based Congestion Control. *ACM Queue*, 14(5).

8. Mathis, M., et al. (1997). The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3).