

CS 3CN3 and SE 4C03 Winter 2009

05 Transport Protocols

William M. Farmer

Department of Computing and Software
McMaster University

3 February 2009



Interprocess Communication

Problem: How can a process on one host access a service provided by a process on another host?

- ▶ Processes are constantly coming into and going out of existence.
- ▶ Processes are often replaced (e.g., when the host is rebooted).
- ▶ Processes may provide multiple services.
- ▶ Processes usually do not broadcast the services they provide.

Solution: Interprocess communication is performed between **protocol ports** instead of the processes themselves.

Protocol Ports

A **protocol port** is a nonnegative integer used as an abstract delivery point.

- ▶ Ports are assigned to processes by the operating system.
- ▶ There are 2^{16} ports ranging from 0 to 65535.
- ▶ There are two sets of ports, one for UDP and one for TCP.
- ▶ Each active port is assigned a queue to hold incoming packets.

Reserved and Ephemeral Ports

- The **reserved ports** (0 to 1023) are assigned statically.
 - ▶ Many of them are assigned to a standard server process (e.g., TCP 23 is assigned to the telnet server).
 - ▶ On Unix systems, they are reserved for processes running as root.
- The **ephemeral ports** (1024 to 65535) are assigned dynamically.
 - ▶ They are usually assigned to client processes.
 - ▶ When the process assigned to an ephemeral port terminates, the port is put back in the pool of available ports.

User Datagram Protocol (UDP)

The **User Datagram Protocol (UDP)** defines a mechanism for sending **UDP datagrams** from one application process to another.

- Provides **unreliable connectionless communication**.
- Protocol ports are used to distinguish between processes.
- UDP datagrams are encapsulated in IP datagrams.
- The IP protocol type for UDP is 17.
- UDP datagrams are completely independent from each other at the transport layer.

Format of a UDP Datagram

- Header
 - ▶ UDP source port (16 bits).
 - ▶ UDP destination port (16 bits).
 - ▶ UDP message length (16 bits).
 - ▶ UDP checksum (16 bits) holds checksum of the UDP datagram plus some of the header information of the encapsulating IP datagram including the source and destination addresses.
- Data area

Stream Delivery

Problem: Many applications require reliable stream delivery.

- Need to handle large volumes of data.
- Need to handle lost, out-of-order, and duplicated data.

Reliable Stream Delivery Service

1. **Stream orientation**: Data is sent as a stream of bits (or bytes).
2. **Virtual circuit connection**: An illusion of a communication circuit is created.
3. **Buffered transfer**: The stream of bits is divided into a stream of packets.
4. **Unstructured stream**: The stream delivery service treats the stream as if it were completely unstructured.
5. **Full duplex connection**: A connection consists of two independent streams flowing in different directions.
6. **Reliability**: Reliability is achieved through **positive acknowledgment with retransmission**.

Positive Acknowledgment with Retransmission

- The receiver of a packet sends an **acknowledgment** message back to the sender.
- Unacknowledged packets are retransmitted.
- Each packet is assigned a sequence number so that lost or out-of-order packets can be detected.

The Transmission Control Protocol (TCP)

- Provides a **reliable stream delivery service**.
- Can be used with the IP datagram delivery service and many other packet delivery services as well.
- The IP protocol type for TCP is 6.
- Like UDP, uses protocol ports for addressing processes.
- Views the data stream as a stream of octets.
- Divides the octet stream into **segments** each composed of a **header** and a **data area**.

TCP Segment Header Fields

- Source port.
- Destination port.
- Sequence number.
- Acknowledgment number.
- Header length.
- Code bits.
- Window advertisement.
- Checksum.
- Urgent pointer.
- Options.

TCP Segment Header Notes

- The sequence number is the position in the octet stream of the first octet held in the data area.
- The code bits (URG, ACK, PSH, RST, SYN, and FIN) are used to identify the type of the segment.
- The **maximum segment size (MSS)** is transferred in the options field.
- The checksum field holds the checksum of the TCP segment plus some of the header information of the encapsulating IP datagram including the source and destination addresses.

TCP Connections

- A **TCP connection** is identified by the endpoints of the connection.
 - ▶ An **endpoint** is identified as a host-port pair (where the host is identified by an IP address).
 - ▶ Multiple connections to the same endpoint are possible.
- To open a TCP connection requires cooperation by both endpoints.
 - ▶ One endpoint performs a **passive open** by requesting a port at which to listen.
 - ▶ The other endpoint requests an **active open** to establish a connection.

Establishing a TCP Connection

A three-way handshake establishes a TCP connection:

1. The sender transmits a segment with the SYN bit but not the ACK bit set.
 - ▶ The segment contains the sender's initial sequence number x .
2. The receiver acknowledges the SYN segment by transmitting a segment with both the SYN and ACK bits set.
 - ▶ The segment contains the receiver's initial sequence number y .
 - ▶ The segment's acknowledgment value is $x + 1$.
3. The sender acknowledges the SYN/ACK segment by transmitting a segment with the ACK bit but not the SYN bit set.
 - ▶ The segment's acknowledgment value is $y + 1$.

Closing a TCP Connection

- Once a TCP connection is established, the two endpoints have exactly the same status.
- A four-way handshake closes a TCP connection:
 1. One endpoint transmits a segment with the FIN bit set.
 2. The other endpoint transmits a segment that acknowledges the FIN segment (which closes one direction of the connection).
 3. Later the second endpoint transmits a segment with the FIN bit set.
 4. The first endpoint transmits a segment that acknowledges the FIN segment (which closes the other direction of the connection).

The Other Code Bits

- The URG code bit and the urgent pointer are used to send **urgent out-of-sequence** information such as an interrupt or abort message.
- Either endpoint can **instantaneously kill** a connection by transmitting a segment with the RST bit set.
- An application can force data to be transmitted immediately in a partially filled TCP segment by issuing a **push command**.
 - ▶ The data is transmitted in a segment with the PSH bit set.
 - ▶ When the segment is received, it is forwarded to the application immediately.

TCP Acknowledgments

- A **TCP acknowledgment** is the position in the octet stream of the first octet that has not yet been received.
- Advantages:
 - ▶ Acknowledgments are simple.
 - ▶ Lost acknowledgments do not necessarily lead to retransmission.
 - ▶ The sender has to only managed one piece of information: the position in the octet stream marking the first unacknowledged segment.
- Disadvantage:
 - ▶ Several segments may be retransmitted because an earlier segment was lost.
- Acknowledgments for segments going in one direction can be “piggybacked” on segments going the other direction.

Sliding Window Technique

- TCP uses the **sliding window technique** (with variable window size) to:
 - ▶ Maximize the throughput of octets.
 - ▶ Control end-to-end octet flow.
- Each acknowledgment contains a **window advertisement** that specifies how much room is available in receiver's buffer.
- The window advertisement is used by the sender to adjust its window size.

Retransmission

- When a TCP segment is transmitted, a timer is started, and if this timer expires before an acknowledgment is received, the TCP segment is retransmitted.
- The timeout value for the connection is computed by an **adaptive retransmission algorithm**.
 - ▶ The timeout value is adjusted as the performance of the connection changes.
 - ▶ The algorithm computes the **sample round trip time** as a weighted average of the round trip times for the segments sent across the connection.
 - ▶ The timeout value is then computed from the sample round trip time using the estimated variance of the round trip times.

Acknowledgment Ambiguity

- The round trip time is hard to measure when a packet is retransmitted because the acknowledgment does not say which copy of the packet was received. This is called **acknowledgment ambiguity**.
- **Karn's algorithm**: Estimate the sample round trip time using only unambiguous acknowledgments and use a backoff strategy to gradually increase the timeout value when retransmission occurs.
- The algorithm works well even under adverse situations.

Congestion Collapse (1)

- If segments are retransmitted due to congestion (which the connection endpoints cannot observe), retransmission can aggravate congestion and cause **congestion collapse**.
- **Multiplicative decrease congestion avoidance**:
 - ▶ The sender maintains a **congestion window** in addition to the **receiver's window** (given by the window advertisement).
 - ▶ The sender's window is computed as the minimum of the receiver's window and the congestion window.
 - ▶ When a segment is lost (and thus when there is possible congestion), the congestion window is halved and the timeout value is doubled.
 - ▶ The congestion window always allows at least one segment so that the connection is not completely shutdown.

Congestion Collapse (2)

- The **slow-start recovery** technique increases the congestion window size one segment at a time after a period of congestion to avoid wild oscillations in the congestion window size.
- None of these algorithms and techniques are computationally expensive.
- The TCP protocol handles congestion very well for arbitrary environments (particularly based on wired networks), but there are more efficient ways of handling congestion for specific environments.
 - ▶ **Example:** TCP over wireless networks.

Silly Window Syndrome

- Symptoms:
 - ▶ Each acknowledgment from the receiver advertises a small window.
 - ▶ Each segment sent carries a small amount of data.
- Outcome: suboptimal throughput.
- Heuristics for Avoiding the Syndrome:
 - ▶ Receive-side silly window avoidance.
 - ▶ Send-side silly window avoidance.

Receive-Side Silly Window Avoidance

- Heuristic: Do not advertise small windows but wait until the window is one half the buffer size or equal to the maximum segment size.
- Two implementations:
 1. Send acknowledgment without advertisement.
 2. Delay acknowledgment (recommended by the TCP protocol).
- Advantage: Can increase throughput.
- Disadvantage: May cause more retransmissions.

Send-Side Silly Window Avoidance

- Known as the [Nagle algorithm](#) (RFC 896).
- Heuristic: While waiting for acknowledgments, clump additional data together in one packet and wait to send it until either all acknowledgments have been received or there is enough data to fill a maximum-size packet.
- Outcome: Information is sent as fast as the network and destination can handle it.