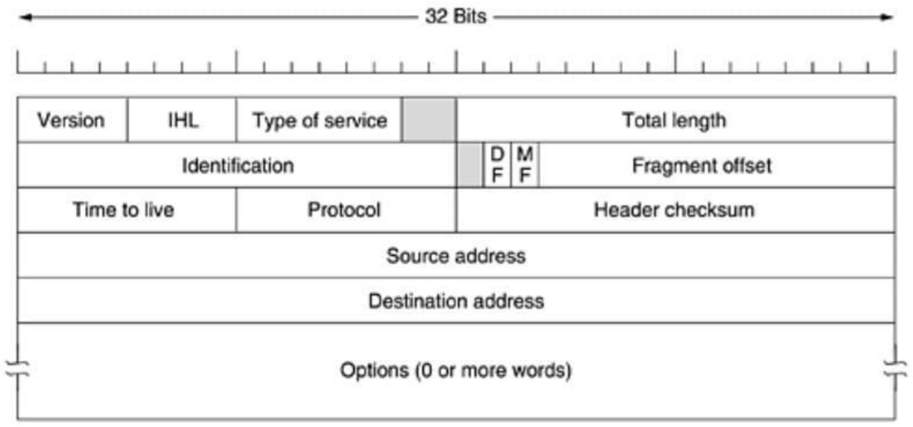
Unit 3.Internetworking

IPv4/IP Header

Key Fields in the IP Header:

Figure 5-53. The IPv4 (Internet Protocol) header.

329



1. **Version (4 bits):**
    a. This field identifies the version of the IP protocol being used.
    b. **IPv4** has a value of 4, and **IPv6** has a value of 6.
2. **IHL (Internet Header Length) (4 bits):**

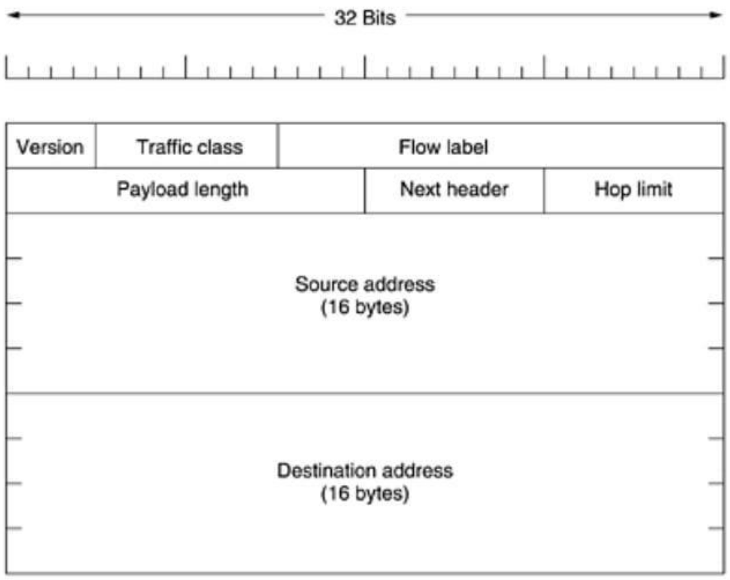
   a. Specifies the length of the IP header in 32-bit words.
   b. The minimum value of IHL is 5 (indicating a 20-byte header), but it can be larger if options are present in the header.

3. **Total Length (16 bits):**
   a. Specifies the entire length of the IP datagram, including both the header and the data (payload).
   b. The maximum value is 65,535 bytes.
4. **Identification (16 bits):**
   a. Used to uniquely identify a datagram. This field helps in fragmenting and reassembling packets when the datagram is too large to be sent in a single packet and needs to be broken into smaller pieces.
   b. Each fragment of the original datagram will carry the same identification number.
5. **Flags (3 bits):**
   a. Control fragmentation behavior of the packet.
      i. **DF (Don't Fragment)**: If set, the packet should not be fragmented. If fragmentation is required, the packet is discarded.
      ii. **MF (More Fragments)**: If set, it indicates that more fragments of the datagram follow.
      iii. The third flag is reserved and should be set to 0.
6. **Fragment Offset (13 bits):**
   a. Indicates the position or offset of a fragment in the original datagram.
   b. It is used during reassembly of fragmented packets.
7. **Time to Live (TTL) (8 bits):**
   a. Prevents packets from circulating endlessly in the network due to routing loops.
   b. Each router that processes the packet decrements the TTL value by 1.
   c. When TTL reaches 0, the packet is discarded. This field ensures that a packet does not travel forever in case of routing issues.
8. **Protocol (8 bits):**
   a. Specifies the protocol used at the transport layer (e.g., **TCP** (6), **UDP** (17), **ICMP** (1)).
   b. This helps the receiving system identify the appropriate protocol handler for the data in the payload.
9. **Header Checksum (16 bits):**
   a. This field is used to detect errors in the header of the IP datagram.
   b. It is computed based on the content of the IP header and is validated by the receiving system. If the checksum does not match, the packet is discarded.

10. **Source IP Address (32 bits for IPv4, 128 bits for IPv6):**
    a. Identifies the sending host or network.
    b. It is used by routers and receiving devices to know where the packet originated from.
11. **Destination IP Address (32 bits for IPv4, 128 bits for IPv6):**
    a. Identifies the intended recipient of the packet.
    b. It allows routers and the receiving device to forward the packet toward its destination.
12. **Options (Variable length, optional):**
    a. This section is optional and allows the inclusion of additional functionality, such as:
        i. **Security**: Can specify security options for routing and transmission.
        ii. **Timestamp**: Used to measure delays or perform diagnostics.
        iii. **Record Route**: Allows the collection of a route history.
        iv. **Loose Source Routing**: Specifies a list of intermediate hosts that the packet must visit.
        v. **Strict Source Routing**: Specifies a strict route that the packet must follow

IPv6 Header

Figure 5-68. The IPv6 fixed header (required).



IPv6 Header Structure

- **Version**: Always set to 6 for IPv6, which helps distinguish it from IPv4 packets.
- **Traffic Class**: Used to differentiate packets with different real-time delivery requirements, especially useful for multimedia data.
- **Flow Label**: Experimental field for establishing pseudoconnections with specific properties, such as reserved bandwidth for real-time applications.
- **Payload Length**: Specifies the length of the data following the IPv6 header (40 bytes).
- **Next Header**: Indicates the type of next header, whether it's an extension header or transport layer protocol (e.g., TCP or UDP).
- **Hop Limit**: Similar to the TTL (Time To Live) in IPv4, it ensures that packets do not circulate indefinitely.
- **Source and Destination Addresses**: Both are 128-bit addresses, providing an incredibly large address space.

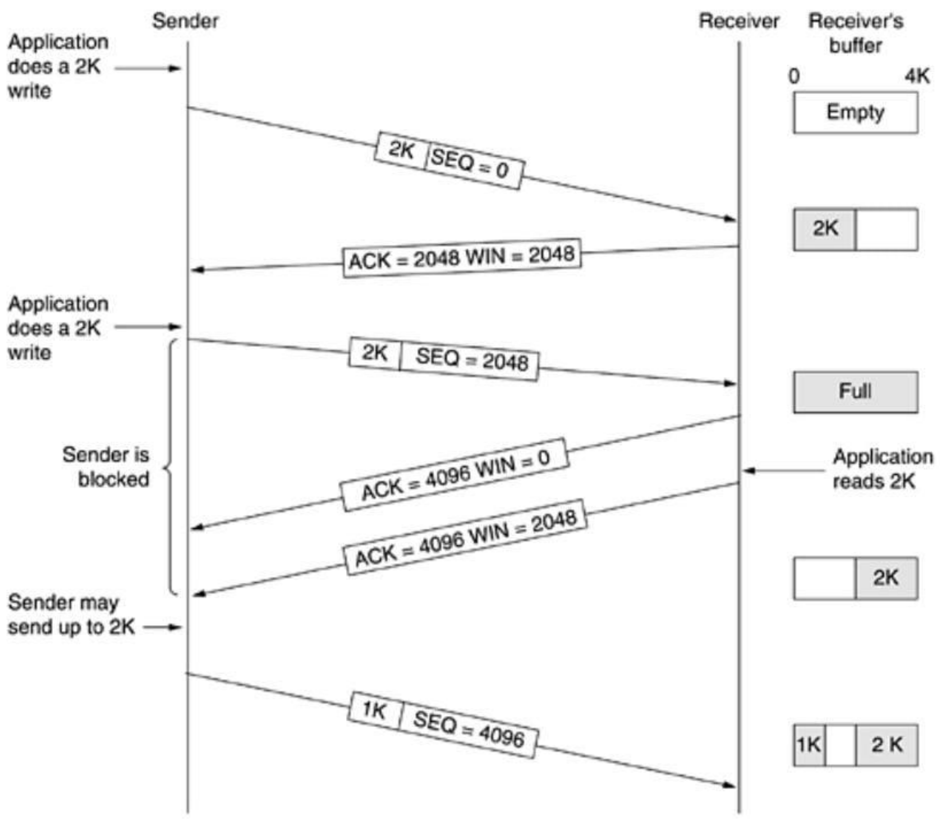
Unit 4. TCP

TCP transmission policy

.8 TCP Transmission Policy

nentioned earlier, window management in TCP is not directly tied to acknowledgements as in most data link protocols. For example, suppose the receiver has a 4096-byte buffer, as wn in Fig. 6-34. If the sender transmits a 2048-byte segment that is correctly received, the iver will acknowledge the segment. However, since it now has only 2048 bytes of buffer ce (until the application removes some data from the buffer), it will advertise a window of 8 starting at the next byte expected.

Figure 6-34. Window management in TCP.



TCP Transmission Policy

1. **Window Management in TCP**:
    a. **Buffer Size and Window**: TCP uses a sliding window mechanism for flow control. The receiver advertises its available buffer space to the sender using the "window size."
    b. **Advertised Window**: If the receiver has a 4096-byte buffer and the sender transmits 2048 bytes, the receiver will acknowledge the segment and advertise a 2048-byte window. If the receiver's buffer becomes full, it may

advertise a window size of 0, instructing the sender to stop sending data until space becomes available.

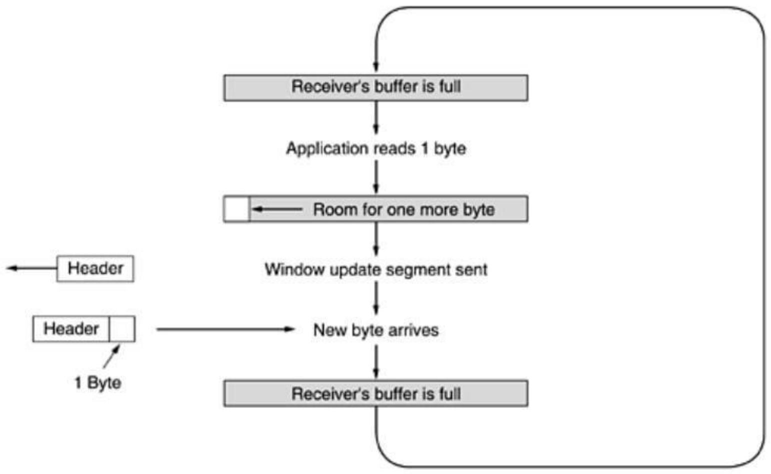
   c. **Exceptions for Zero Window**: If the window size is 0, the sender can still send urgent data or a 1-byte segment to reinitiate the window update.

2. **Delayed Acknowledgements and Window Updates**:
   a. **Efficiency**: TCP does not require immediate acknowledgements or window updates. To improve performance, many TCP implementations delay acknowledgements and window updates, allowing the sender to bundle multiple acknowledgements into a single packet.
   b. **Example**: In an interactive session like Telnet, delaying acknowledgements for 500 ms can reduce bandwidth usage and the number of packets sent.
3. **Nagle's Algorithm**:
   a. **Purpose**: Nagle's algorithm minimizes the number of small packets by buffering data at the sender until the first byte is acknowledged. Once the acknowledgement is received, the sender sends all buffered data in one segment.
   b. **When Disabled**: Nagle's algorithm is often disabled for applications like X Window System, where delays in sending small packets could negatively affect real-time user interaction, such as mouse movements.
4. **Silly Window Syndrome**:
   a. **Problem**: This occurs when the receiver sends window updates for very small amounts of data, leading to inefficient use of bandwidth.
   b. **Solution**: The receiver should avoid sending window updates for small amounts of data (less than the maximum segment size). It waits until it has enough space to send a larger update, reducing the number of small, inefficient transmissions.
5. **Handling Out-of-Order Segments**:
   a. **Buffering**: The receiver may receive out-of-order segments, which are buffered until the missing segment is received. Acknowledgements are sent only for the last correctly received byte.
   b. **Retransmission**: If a segment is lost (detected via timeout), the sender retransmits it, and the receiver acknowledges all the data once it is received.
6. **Congestion Control in TCP**:
   a. **Congestion Window and Slow Start**: TCP uses congestion control to manage network load. It starts by sending a small amount of data, increasing the transmission rate exponentially (slow start) until a timeout occurs or the receiver's window is reached.

b. **Threshold and Linear Growth**: After a timeout, TCP reduces the congestion window by half and grows it linearly for each successful transmission. This mechanism helps avoid network congestion by slowing down data flow when congestion is detected.

Silly window Syndrome

Figure 6-35. Silly window syndrome.



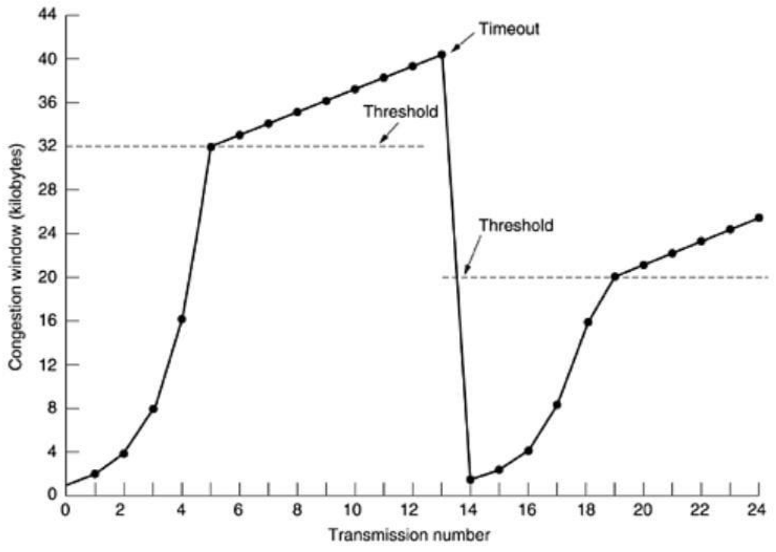
Silly Window Syndrome (SWS)

1. **Definition**: Silly Window Syndrome (SWS) is a phenomenon in TCP where the receiver advertises a very small window size to the sender, resulting in inefficient data transmission. This leads to multiple small packets being sent, instead of fewer, larger packets, wasting bandwidth and causing unnecessary overhead.
2. **Causes**:
    a. **Small Available Buffer Space**: The receiver's buffer is not large enough to accommodate a full segment, so it advertises a small window size to the sender.
    b. **Premature Window Updates**: The receiver sends window size updates for small amounts of data, causing the sender to transmit small packets frequently, instead of waiting for a larger window to send more data.
    c. **Incorrect TCP Window Management**: Sometimes, TCP implementations send updates for very small amounts of free buffer space, which leads to frequent, inefficient data transmissions.
3. **Impact**:
    a. **Inefficient Bandwidth Usage**: The sender may be forced to send several small segments instead of larger ones, leading to inefficient use of the available bandwidth.
    b. **Increased Overhead**: The overhead associated with sending many small packets, including header information, is higher than sending fewer, larger packets.
    c. **Network Congestion**: The sender's network might get congested with small packets, resulting in wasted network resources.
4. **Solution**:
    a. **Receiver-Side Buffering**: The receiver should wait until its buffer is sufficiently full before sending a window update, thus avoiding advertising tiny windows. This minimizes the frequency of small packet transmissions.
    b. **Delayed Window Updates**: The receiver should delay window size updates until it has accumulated enough free buffer space, ensuring that the sender can transmit larger segments.
    c. **TCP Implementation Changes**: Some implementations disable SWS by implementing more sophisticated buffering techniques that avoid sending updates for small windows.
    d. **TCP Flow Control Adjustments**: TCP stack implementations may include algorithms that monitor the window size and prevent sending small windows unless absolutely necessary.

5. **Example**: For instance, in a scenario where a receiver has a buffer space of 100 bytes and it advertises a window size of 20 bytes, the sender might send many small packets of 20 bytes, leading to excessive overhead. Instead, the receiver could wait until the buffer space is larger and advertise a window size of 80 or 100 bytes, allowing the sender to send larger chunks of data at once.

TCP Congestion control

Figure 6-37. An example of the Internet congestion algorithm.



TCP Congestion Control: Simplified Explanation

When the network is overloaded (congested), packets can get delayed or lost. TCP tries to prevent congestion by controlling how much data is sent at once.

1. **Congestion Detection:**
    a. Initially, packet loss due to transmission errors was a common cause of timeouts. However, today, timeouts are mostly caused by congestion (network overload). TCP assumes that lost packets are due to congestion.
2. **Two Types of Windows:**
    a. **Receiver Window:** Specifies how much data the receiver can handle.
    b. **Congestion Window:** Specifies how much data the sender believes the network can handle without causing congestion. The sender uses the smaller of these two windows to decide how much data to send.
3. **Slow Start Algorithm (to prevent congestion):**
    a. When a TCP connection is established, the sender starts with a small congestion window (the size of one maximum segment).
    b. It sends one segment and waits for acknowledgment. If acknowledged, the sender increases the congestion window, and the next segment sends more data.
    c. This process continues, and the congestion window grows exponentially, doubling each time (exponential growth). This is called **slow start**, though it's fast in terms of growing the window.
4. **Threshold and Linear Growth:**
    a. A **threshold** (initially 64 KB) is set to limit how much the congestion window can grow exponentially.
    b. Once the congestion window reaches the threshold, the growth switches to **linear growth** (adding one segment at a time), which is less aggressive and helps avoid congestion.
5. **Timeout and Backoff:**
    a. If a timeout occurs (likely due to congestion), the congestion window is reset to one segment, and the threshold is halved. The **slow start** process begins again from this smaller threshold, gradually increasing the congestion window.