CSN-341 (Computer Networks)

Assignment-4 by Group 16
September 24, 2024

Anvit Gupta
22114009
anvit_g@cs.iitr.ac.in

Ayush Ranjan
22114018
ayush_r@cs.iitr.ac.in

Indranil Das
22114037
indranil_d@cs.iitr.ac.in

Sarvasva Gupta
22114086
sarvasva_g@cs.iitr.ac.in

Souvik Karmakar
22114096
souvik_k@cs.iitr.ac.in

Vineet Kumar
22114107
vineet_k@cs.iitr.ac.in

1. Why is flow control important in the transport layer, and how does TCP implement flow control through mechanisms like sliding windows? Discuss how improper flow control can lead to issues like buffer overflow and network congestion, and explain the potential impact on overall network performance.

Ans:
Flow control is crucial in the transport layer to manage the data flow between the sender and receiver, **ensuring that the sender does not overwhelm the receiver** with more data than it can process at a given time. This is especially important when the sender is faster than the receiver, which can result in issues like **buffer overflow** and **network congestion**.

TCP Flow Control using Sliding Windows

TCP (Transmission Control Protocol) implements flow control through a **sliding window mechanism**. This is a dynamic window that the receiver uses to inform the sender of the amount of data it can accept. The "window size" represents the available buffer space at the receiver, and the sender is allowed to transmit data only within this window. As the receiver processes and acknowledges data, the window slides, allowing more data to be sent.

1. **Sender Window:** The sender is only allowed to send data up to the size of the window (advertised by the receiver). This prevents sending too much data (i.e too fast).
2. **Receiver Window:** The receiver advertises its available buffer size to the sender, which adjusts its sending rate accordingly.

If the receiver's buffer is filled up, it will shrink the advertised window size, causing the sender to slow down or temporarily stop until it can resume sending data.

Problems with Improper Flow Control

- **Buffer Overflow:** If the sender transmits data faster than the receiver can process it, the receiver's buffer may fill up, causing packets to be dropped. This leads to retransmissions, increasing network traffic and inefficiency.
- **Network Congestion:** Excessive retransmissions due to dropped packets or improper flow control can cause congestion in the network. Congestion leads to higher latency, packet loss, and reduced throughput, affecting overall network performance. TCP's flow control mechanisms, like **Congestion Avoidance** and **Slow Start**, work to reduce these issues.

Impact on Network Performance

- **Decreased Throughput:** Buffer overflow can result in dropped packets, leading to retransmissions and reduced data throughput.

- **Increased Latency:** Congestion can increase round-trip times (RTTs), delaying the delivery of data.
- **Wasted Bandwidth:** Retransmissions due to lost or dropped packets waste bandwidth and increase network congestion.

Conclusion

Proper flow control ensures an efficient data transfer rate that matches the receiver's processing capabilities, preventing congestion and buffer overflows. TCP's sliding window mechanism plays a key role in maintaining balanced network performance.

2. Why are port numbers significant in the transport layer, and how do they facilitate multiplexing and demultiplexing of data streams? Provide examples of specific port numbers associated with well-known services, such as HTTP and DNS, and discuss the potential issues that can arise due to port number conflicts or misuse.

Ans:
Port numbers are vital in the transport layer of the TCP/IP model because they allow multiple services or applications to share a single network connection simultaneously. Port numbers help in the processes of **multiplexing** and **demultiplexing** data streams, which are crucial for handling multiple connections on the same machine.

Significance of Port Numbers

- **Multiplexing:** When a host sends data from different applications over a network, port numbers help combine these multiple streams of data into a single communication channel.
- **Demultiplexing:** Upon receiving the data, the transport layer uses port numbers to determine which application should receive which data stream, effectively splitting the combined data back into its original applications.

For instance, if you're browsing a web page and simultaneously downloading a file, the transport layer ensures the data for each task is routed to the correct application, even though both tasks use the same network connection.

Port Number Structure

- **Source Port:** Identifies the application sending the data.
- **Destination Port:** Identifies the service or application on the receiving end that the data is meant for.

Common Well-Known Port Numbers

Certain services are associated with standardized port numbers:

- **HTTP (Port 80):** Used for regular web traffic.
- **HTTPS (Port 443):** Used for secure web traffic.
- **DNS (Port 53):** Used for domain name resolution.
- **SMTP (Port 25):** Used for email transmission.

These ports allow clients to easily connect to standardized services across the internet.

Issues Related to Port Number Conflicts or Misuse

- **Port Number Conflicts:** Two different applications attempting to use the same port number on the same machine can cause conflicts. For example, if both a web server and another service attempt to use port 80, one of the services will fail to bind to the port, causing service disruption.
- **Port Misuse and Security Risks:** Unauthorized services running on commonly used ports can expose a system to attacks. For example, a malicious service running on port 443 (usually used for secure communication) could trick users into thinking they are interacting with a legitimate website.

Port numbers are key to organizing and routing network traffic properly, but conflicts or misuse can cause communication failures or expose networks to vulnerabilities.

Conclusion

Port numbers are essential for managing data flow and ensuring efficient use of network resources. By providing unique identifiers for services, they facilitate the simultaneous use of multiple applications over the same connection, while also introducing potential risks if not managed properly.

3. Compare and contrast the Selective-Repeat and Go-Back-N protocols in the context of reliability and efficiency. How does piggybacking in bidirectional communication improve efficiency, and what are the practical considerations when implementing piggybacking in real-world applications?

Ans:

Comparison of Selective-Repeat and Go-Back-N Protocols

1. **Reliability**

   - **Selective-Repeat (SR):** In SR, only the specific packets that are lost or corrupted are retransmitted. The receiver individually acknowledges each correctly received packet, even if it is out of order. This ensures a higher level of reliability, as only the erroneous packets are retransmitted, reducing unnecessary retransmissions.
   - **Go-Back-N (GBN):** In GBN, if a packet is lost or corrupted, all subsequent packets are retransmitted, even if they were received correctly. The receiver can only accept packets in order, so out-of-order packets are discarded. Although it doesn't affect reliability compared to SR but at a cost of more retransmissions, which causes inefficiency.

2. **Efficiency**

   - **Selective-Repeat:** SR is more efficient than GBN when the network has a high error rate or when many packets are being transmitted. Since SR retransmits only the erroneous packets, bandwidth is conserved, making it more efficient in scenarios with frequent errors.
   - **Go-Back-N:** GBN can be less efficient because it requires retransmission of all packets following a lost packet, even if they were received correctly. This leads to wasted bandwidth when the network conditions cause frequent packet loss. It may be better if there was a burst of packet drops.

3. **Window Size**

   - **Selective-Repeat:** SR requires a larger buffer size at the receiver to store out-of-order packets until missing packets are received, as the receiver accepts out-of-order packets. The window size is typically half of the sequence number space, i.e., $W \leq 2^{m-1}$ where m is the number of bits in the sequence number field.
   - **Go-Back-N:** GBN does not allow out-of-order packets and only requires buffering for a single packet. Its window size can be as large as the sequence number space, i.e., $W \leq 2^m$, where m is the number of bits in the sequence number field.

Piggybacking in Bidirectional Communication

Piggybacking is a technique where acknowledgments (ACKs) are embedded into outgoing data packets instead of sending standalone ACK packets. This improves efficiency by reducing the number of packets on the network, as ACKs and data are combined into one packet.

- **Efficiency Gains:** Instead of sending two separate packets (one for data and one for the acknowledgment), piggybacking combines them, reducing the overhead and saving bandwidth. It works best in bidirectional communication, where both sides are sending data regularly

Practical Considerations for Implementing Piggybacking

1. **Timing:** Piggybacking can introduce delays in acknowledgments if the sender has no data to send, as ACKs are held back until a data packet is available. In real-time applications like voice or video conferencing, delayed ACKs can negatively impact performance.
2. **Network Conditions:** In networks with high latency, waiting for data to piggyback on may degrade performance, as immediate acknowledgment may be preferred for time-sensitive protocols.
3. **Asymmetry in Traffic:** If one side of the communication sends significantly more data than the other, piggybacking may not be fully utilized, leading to situations where standalone ACKs still need to be sent.

4. In scenarios where low latency is critical, how would you evaluate the trade-offs between using TCP and UDP? Consider the impact on reliability, error correction, and the specific needs of applications:
a) online gaming
b) video conferencing
c) stock trading.

Ans:
In scenarios where low latency is critical, the trade-offs between using **TCP** and **UDP** revolve around balancing reliability, error correction, and speed. Here's an evaluation of the two protocols across different applications:

1. Online Gaming

Trade-offs:

- **UDP** is preferred for online gaming due to its low-latency nature, as games prioritize speed over perfect data accuracy. UDP doesn't wait for acknowledgments or retransmit lost packets, reducing lag. For real-time gameplay, some packet loss is tolerable as the game state is updated frequently.
- **TCP** would introduce delays as it implements flow control, error correction, and retransmission, which could cause lag spikes. This is detrimental for fast-paced games like first-person shooters or fighting games.

Example:

Multiplayer games (e.g., Fortnite or Call of Duty) often use **UDP** to send quick updates about player positions and actions without worrying about lost data since a new update will soon follow.

Impact:

Speed and responsiveness are critical in gaming, so **UDP** is typically favored. However, in some cases, TCP might be used for initial handshakes or non-time-sensitive actions like logging into a game server.

2. Video Conferencing

Trade-offs:

- **UDP** is often used in video conferencing protocols like RTP (Real-time Transport Protocol), which ensures that packets arrive quickly even if some are lost. Video and voice streams need minimal delay, and dropped frames are often less noticeable than pauses.
- **TCP** would introduce delays as it ensures every packet is received in the correct order and retransmits lost packets. This can result in noticeable latency and jitter, which would degrade the experience in live video conversations.

Example:

Services like **Zoom** and **Google Meet** use **UDP-based** protocols to deliver real-time audio and video, handling packet loss through error concealment techniques.

Impact:

UDP is better suited for real-time video conferencing as it reduces delays, with slight degradation in quality being preferable to lag

3. Stock Trading

Trade-offs:

- **TCP** is typically used in financial applications like stock trading, as the **reliability** of data is critical. Every transaction, trade execution, and order needs to be accurate and verified. Even though this introduces slightly more latency, the financial risks associated with incorrect or lost data outweigh the need for ultra-low latency.
- **UDP** could be used in scenarios where traders receive live market data, but for actual trade execution, **TCP** is crucial to ensure transactions are not lost.

Example:

Stock exchanges and financial firms typically rely on **TCP** for data integrity and secure trading sessions.

Impact:

For critical transactions, **TCP**'s guarantees of data integrity and correctness make it the clear choice, despite the potential for slightly increased latency.

Summary of Trade-offs

- **TCP:** Ensures reliability and correctness through error correction, retransmission, and flow control, making it suitable for applications where data accuracy is more important than speed (e.g., stock trading).
- **UDP:** Prioritizes speed over reliability, making it the better choice for applications requiring real-time performance, where slight data loss is tolerable (e.g., gaming and video conferencing).

These decisions come down to the specific needs of the application:

- **Online gaming and video conferencing:** Choose **UDP** for low latency.
- **Stock trading:** Choose **TCP** for reliability and accuracy, where data integrity is critical.

5. Discuss the importance of sequence numbers and acknowledgments in TCP. How do these mechanisms ensure reliable data transmission and prevent issues of:
a) packet duplication
b) out-of-order delivery
c) Provide an example of how TCP recovers from lost or out-of-order packets.

Ans:

Importance of Sequence Numbers and Acknowledgments in TCP

Sequence numbers and **acknowledgments (ACKs)** are key mechanisms that ensure reliable, ordered, and error-free data transmission in **TCP (Transmission Control Protocol)**.

1. **Sequence Numbers:** Each byte of data transmitted by TCP is assigned a sequence number. This allows both the sender and receiver to keep track of which bytes have been transmitted and received. The first sequence number is randomly generated during connection establishment (TCP 3-way handshake)
2. **Acknowledgments (ACKs):** TCP uses acknowledgments to confirm receipt of data. For every byte received, the receiver sends an acknowledgment to the sender indicating the next sequence number it expects, ensuring that the sender knows which data has been successfully received.

a) Packet Duplication Prevention

Sequence numbers help avoid packet duplication. Since every byte (as per TCP) of data is assigned a unique sequence number, the receiver can easily detect if the same data is transmitted twice. If a duplicated packet arrives (e.g., due to network issues), the receiver can discard it because its sequence number will indicate that it has already been received and acknowledged.

b) Out-of-Order Delivery Prevention

Although packets might arrive out of order due to network conditions, **sequence numbers** ensure that the receiver can reorder the data before passing it to the application layer. The receiver uses the sequence numbers to assemble the data in the correct order, even if packets arrive out of sequence. **TCP's acknowledgment mechanism** only acknowledges the highest in-order byte received, encouraging the sender to retransmit missing segments if out-of-order packets are detected.

c) Recovery from Lost or Out-of-Order Packets

TCP employs **retransmission** and **sliding window** mechanisms to handle lost or out-of-order packets:

- If a packet is lost (i.e., the receiver does not acknowledge it), the sender will not receive an ACK for the missing segment within the **timeout period**. When this happens, TCP retransmits the missing packet.
- TCP also uses **cumulative acknowledgments**: If packet 3 is lost, but packets 4 and 5 are received, the receiver will keep sending an ACK for packet 2 (the last successfully received in-order packet). The sender detects that packet 3 is missing and retransmits it.

Example

Imagine a TCP transmission where the sender transmits packets with sequence numbers 1 to 5. If packet 3 is lost, but packets 4 and 5 are received:

1. The receiver will send ACK with ackNo = 3 repeatedly, indicating it has only received up to byte 2 in sequence and is expecting 3rd Packet [We are saying packet for simplicity although it's bytes]
2. The sender receives the ACK thus sliding the window start to 3rd Packet and when timeout occurs the whole window from 3rd to 5th Packet is retransmitted.
3. Once packet 3 is received, the receiver sends an acknowledgment for packet 5 with ACK of ackNo = 6.

In summary, **sequence numbers** ensure that every byte of data is uniquely tracked, while **ACKs** verify that data has been received. Together, these mechanisms prevent data duplication, handle out-of-order delivery, and enable recovery from packet loss, ensuring reliable data transmission over TCP.