Department of Computer Science and Engineering
College of Engineering Guindy
Anna University, Chennai- 25

CS6111- Computer Networks

20- Sep- 2024

Preparatory Content

Year/ Sem/Batch: III/V/P

Course Instructor: Ms.M.S.Karthika Devi
Asst. Prof (Sr. Gr.)

Flow Control

Flow control enables a receiver to regulate the flow of data from a sender so that the receiver is not overwhelmed.

Flow control is a technique for ensuring that the sending entity does not overwhelm the receiving entity with data preventing buffer overflow

- **Transmission time — Time taken for a station to emit all bits of a frame into medium • Proportional to the length of the frame**

- **Propagation time — Time for a bit to traverse the link between source and destination.**

Consider a situation in which the sender transmits frames faster than the receiver can accept them.
If the sender keeps pushing out frames at high rate, at some point the receiver will be completely flooded and will start losing some frames. This problem may be solved by introducing **flow control.**
Most flow control protocols contain a feedback mechanism to inform the sender when it should transmit the next frame.

Mechanisms for Flow Control:

- **Stop and Wait Protocol:** This is the simplest file control protocol in which the sender transmits a frame and then waits for an acknowledgement, either positive or negative, from the receiver before proceeding.

  If a positive acknowledgement is received, the sender transmits the next packet; else it retransmits the same frame.

However, this protocol has one major flaw in it. If a packet or an acknowledgement is completely destroyed in transit due to a noise burst, a deadlock will occur because the sender cannot proceed until it receives an acknowledgement.

This problem may be solved using timers on the sender's side. When the frame is transmitted, the timer is set. If there is no response from the receiver within a

certain time interval, the timer goes off and the frame may be retransmitted.

This process is handled by one more method that is known as **STOP AND WAIT ARQ (Automatic repeat Request).**

Any time an error is detected in an exchange, specified frames are retransmitted. This process is called automatic repeat request (ARQ).

Stop-and-wait ARQ

Stop-and-wait ARQ is a method used in communication to send information between two connected devices. It ensures that information is not lost and received in the correct order. It is the ARQ method. A stop-and-wait ARQ sender sends one frame at a time;

After sending each frame, the sender doesn't send any other frames until it receives an acknowledgement (ACK) signal. After receiving a good frame, the receiver sends an ACK. If the ACK does not reach the sender before a certain time, known as the timeout, the sender sends the same frame again.

In this case the sender resends the same packet. Eventually the receiver gets two copies of the same frame, and sends an ACK for each one. The sender, waiting for a single ACK, receives two ACKs, which may cause problems if it assumes that the second ACK is for the next frame in the sequence.

To avoid these problems, the most common solution is to define a **1 bit sequence number** in the header of the frame.

This sequence number alternates **(from 0 to 1)** in subsequent frames. When the receiver sends an ACK, it includes the sequence number of the next packet it expects.

This way, the receiver can detect duplicated frames by checking if the frame sequence numbers alternate. If two subsequent frames have the same sequence number, they are duplicates, and the second frame is discarded. Similarly, if two subsequent ACKs reference the same sequence number, they are acknowledging the same frame.

Stop-and-wait ARQ is inefficient compared to other ARQs, because the time between packets, if the ACK and the data are received successfully, is twice the transit time (assuming the turnaround time can be zero). The throughput on the

channel is a fraction of what it could be. To solve this problem, one can send more than one packet at a time with a larger sequence number and use one ACK for a set. This is what is done in GO-BACK-N ARQ and the Selective Repeat.

Sliding Window

Sliding window is a technique for controlling transmitted data packets between two network computers where reliable and sequential delivery of data packets is required.

In the sliding window technique, each data packet (for most data link layers) and byte (in TCP) includes a unique consecutive sequence number, which is used by the receiving computer to place data in the correct order. The objective of the sliding window technique is to use the sequence numbers to avoid duplicate data and to request missing data.

Sliding window is also known as windowing.
The sliding window technique places varying limits on the number of data packets that are sent before waiting for an acknowledgment signal back from the receiving computer. The number of data packets is called the **window size**.
In sliding window protocols the sender's data link layer maintains a **'sending window'** which consists of a set of sequence numbers corresponding to the frames it is permitted to send.

Similarly, the receiver maintains a **'receiving window'** corresponding to the set of frames it is permitted to accept.

The window size is dependent on the retransmission policy and it may differ in values for the receiver's and the sender's window.

The sequence numbers within the sender's window represent the frames sent but as yet not acknowledged. Whenever a new packet arrives from the network layer, the upper edge of the window is advanced by one.

When an acknowledgement arrives from the receiver the lower edge is advanced by one.

Go-Back-N ARQ

Go-Back-N ARQ is a specific technique of the ARQ protocol, in which the sending process continues to send a number of frames specified by a **window size** even without receiving an ACK packet from the receiver. It is a special case of the general sliding window protocol with the transmit window size of N and receive window size of 1.

The receiver process keeps track of the sequence number of the next frame it expects to receive, and sends that number with every ACK it sends. The receiver will discard any frame that does not have the exact sequence number it expects (either a duplicate frame it already acknowledged, or an out-of-order frame it expects to receive later) and will resend an ACK for the last correct in-order frame.

Once the sender has sent all of the frames in its **window**, it will detect that all of the frames since the first lost frame are **outstanding,** and will go back to sequence number of the last ACK it received from the receiver process and fill its window starting with that frame and continue the process over again.

Go-Back-N ARQ is a more efficient use of a connection than Stop and Wait ARQ, since unlike waiting for an acknowledgement for each packets, the connection is still being utilized as packets are being sent.

w-1 + 1 < Sequence Number Space
i.e., w < Sequence Number Space
Maximum Window Size = Sequence Number Space - 1

In other words, during the time that would otherwise be spent waiting, more packets are being sent. However, this method also results in sending frames multiple times – if any frame was lost or damaged, or the ACK acknowledging them was lost or damaged, then that frame and all following frames in the window (even if they were received without error) will be re-sent. To avoid this, Selective Repeat ARQ can be used.

Selective Repeat ARQ

It may be used as a protocol for the delivery and acknowledgement of message units, or it may be used as a protocol for the delivery of subdivided message sub-units.

When used as the protocol for the delivery of **messages**, the sending process continues to send a number of frames specified by a **window size** even after a frame loss.

Unlike Go Back N ARQ, the receiving process will continue to accept and acknowledge frames sent after an initial error; this is the general case of the sliding window protocol with both transmit and receive window sizes greater than 1.

The receiver process keeps track of the sequence number of the earliest frame it has not received, and sends that number with every ACK it sends. If a frame from the sender does not reach the receiver, the sender continues to send subsequent frames until it has emptied its **window**.

The receiver continues to fill its receiving window with the subsequent frames, replying each time with an ACK containing the sequence number of the earliest missing frames. Once the sender has sent all the frames in its **window**, it re-sends the frame number given by the ACKs, and then continues where it left off.

Maximum Window Size = Sequence Number Space / 2

The size of the sending and receiving windows must be equal, and half the maximum sequence number.

Implementation of Flow control using Stop and Wait mechanism

Server side code:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>


#define PORT 8080

#define MAX 1024


void error(const char *msg) {

    perror(msg);

    exit(1);
```

```
}

int main() {
    int sockfd, newsockfd;
    struct sockaddr_in serv_addr, cli_addr;
    socklen_t clilen;
    char buffer[MAX];
    int frame = 0;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) error("ERROR opening socket");

    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(PORT);

    if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
        error("ERROR on binding");

    listen(sockfd, 5);
    clilen = sizeof(cli_addr);
    newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
```

```
    if (newsockfd < 0) error("ERROR on accept");

    while (1) {
        bzero(buffer, MAX);
        read(newsockfd, buffer, MAX);
        printf("Received frame: %s\n", buffer);
        frame++;
        sprintf(buffer, "ACK %d", frame);
        write(newsockfd, buffer, strlen(buffer));
    }

    close(newsockfd);
    close(sockfd);
    return 0;
}
```

Client side code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
```

```
#define MAX 1024

void error(const char *msg) {
    perror(msg);
    exit(1);
}

int main() {
    int sockfd;
    struct sockaddr_in serv_addr;
    char buffer[MAX];
    int frame = 0;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) error("ERROR opening socket");

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(PORT);

    if (connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
        error("ERROR connecting");
```

```
    while (1) {
        frame++;
        sprintf(buffer, "Frame %d", frame);
        write(sockfd, buffer, strlen(buffer));
        bzero(buffer, MAX);
        read(sockfd, buffer, MAX);
        printf("Received: %s\n", buffer);
        sleep(1); // Simulate delay
    }

    close(sockfd);
    return 0;
}
```