RAMAIAH UNIVERSITY OF APPLIED SCIENCES

ASSIGNMENT

Course Code	CSC303A
Course Name	Computer Networks
Programme	B. Tech.
Department	Computer Science and Engineering
Faculty	Faculty of Engineering & Technology

Name of the Student	Subhendu Maji
Reg. No	18ETCS002121
Semester/Year	5TH semester / 2018 batch
Course Leader/s	Mr. Nithin Rao R

Declaration Sheet				
Student Name	Subhendu Maji			
Reg. No	18ETCS002121			
Programme	B. Tech.		Semester/Year	5th sem / 2018 batch
Course Code	CSC303A			
Course Title	Computer Networks			
Course Date		to		
Course Leader	Mr. Nithin Rao R			

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature of the Student		Date	
Submission date stamp (by Examination & Assessment Section)			

Signature of the Course Leader and date	Signature of the Reviewer and date

Contents

Assignment - 1					
Register No.	**18ETCS002121**	Name of Student		**SUBHENDU MAJI**	
Sections		**Marking Scheme**	**Max Marks**	**First Examiner Marks**	**Second Examiner Marks**
Q.1	1.1	Disadvantages of the protocol	02		
	1.2	Modifications to overcome the disadvantages	03		
		Max Marks	**05**		
Q.2	2.1	Program to compute checksum at the transmitter	10		
	2.2	Program to check for error free data transmission at the receiver	10		
		Max Marks	**20**		
		Total Assignment Marks	**25**		

Course Marks Tabulation				
Component- 1(B) Assignment	**First Examiner**	**Remarks**	**Second Examiner**	**Remarks**
Q 1				
Q 2				
Marks (Max 25)				

Signature of First Examiner

Signature of Second Examiner

Question No. 1

Solution to Question No. 1:

Stop and Wait Protocol is a data-link layer protocol which is used for transmitting the data over the noiseless channels. It provides unidirectional data transmission which means that either sending or receiving of data will take place at a time. It provides flow-control mechanism but does not provide any error control mechanism.
The idea behind the usage of this frame is that when the sender sends the frame then he waits for the acknowledgment before sending the next frame.

The primitives of stop and wait protocol are:

Sender side

Rule 1: Sender sends one data packet at a time.

Rule 2: Sender sends the next packet only when it receives the acknowledgment of the previous packet.

Therefore, the idea of stop and wait protocol in the sender's side is very simple, i.e., send one packet at a time, and do not send another packet before receiving the acknowledgment.

Receiver side

Rule 1: Receive and then consume the data packet.

Rule 2: When the data packet is consumed, receiver sends the acknowledgment to the sender.

Therefore, the idea of stop and wait protocol in the receiver's side is also very simple, i.e., consume the packet, and once the packet is consumed, the acknowledgment is sent. This is known as a flow control mechanism.

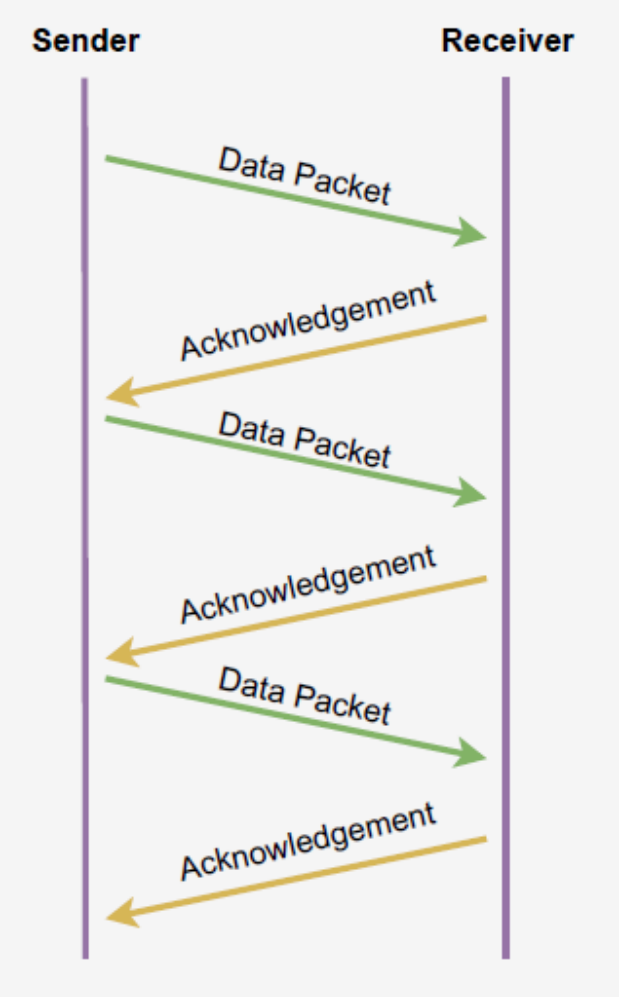


Figure 1 stop-and-wait protocol

If there is a sender and receiver, then sender sends the packet and that packet is known as a data packet. The sender will not send the second packet without receiving the acknowledgment of the first packet. The receiver sends the acknowledgment for the data packet that it has received. Once the acknowledgment is received, the sender sends the next packet. This process continues until all the packet are not sent. The main advantage of this protocol is its simplicity but it has some disadvantages also. For example, if there are 1000 data packets to be sent, then all the 1000 packets cannot be sent at a time as in Stop and Wait protocol, one packet is sent at a time.

1.1 Disadvantages of the protocol

1. Lost Data

   Suppose the sender sends the data and the data is lost. The receiver is waiting for the data for a long time. Since the data is not received by the receiver, so it does not send any acknowledgment. Since the sender does not receive any acknowledgment so it will not send the next packet. This problem occurs due to the lost data.

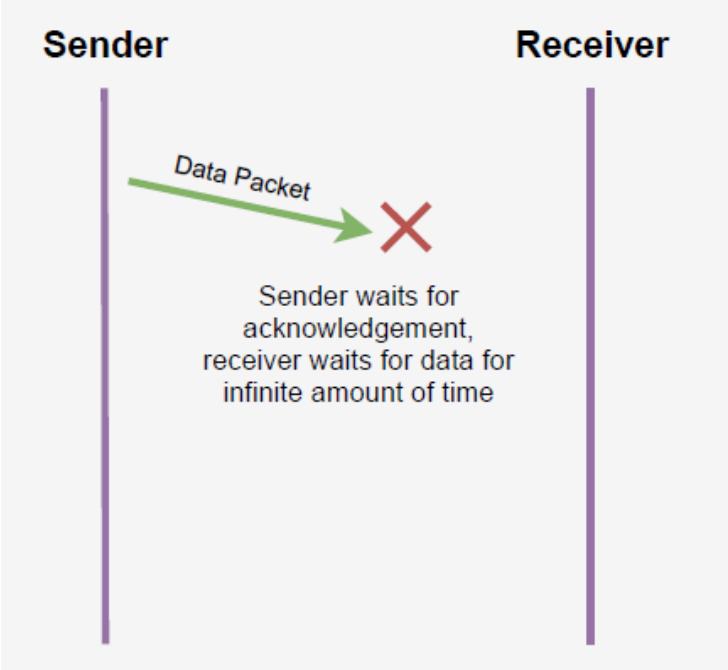


Figure 2 Data Lost

   In this case, two problems occur:

   - Sender waits for an infinite amount of time for an acknowledgment.
   - Receiver waits for an infinite amount of time for a data.

2. Lost Acknowledgment

   Suppose the sender sends the data and it has also been received by the receiver. On receiving the packet, the receiver sends the acknowledgment. In this case, the acknowledgment is lost in a network, so there is no chance for the sender to receive the acknowledgment. There is also no chance for the sender to send the next packet as in stop and wait protocol, the next packet cannot be sent until the acknowledgment of the previous packet is received.

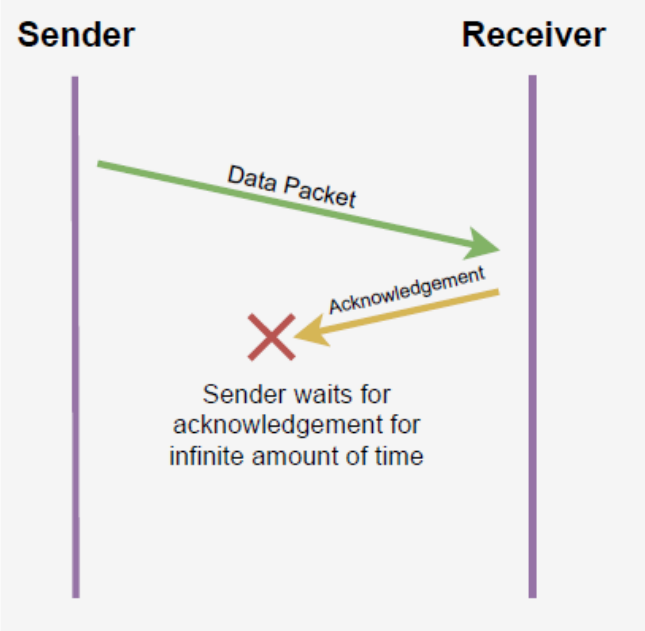


Figure 3 Acknowledgement Lost

In this case, one problem occurs:

- Sender waits for an infinite amount of time for an acknowledgment.

3. Delayed Data or Acknowledgment

   Suppose the sender sends the data and it has also been received by the receiver. The receiver then sends the acknowledgment but the acknowledgment is received after the timeout period on the sender's side. As the acknowledgment is received late, so acknowledgment can be wrongly considered as the acknowledgment of some other data packet.

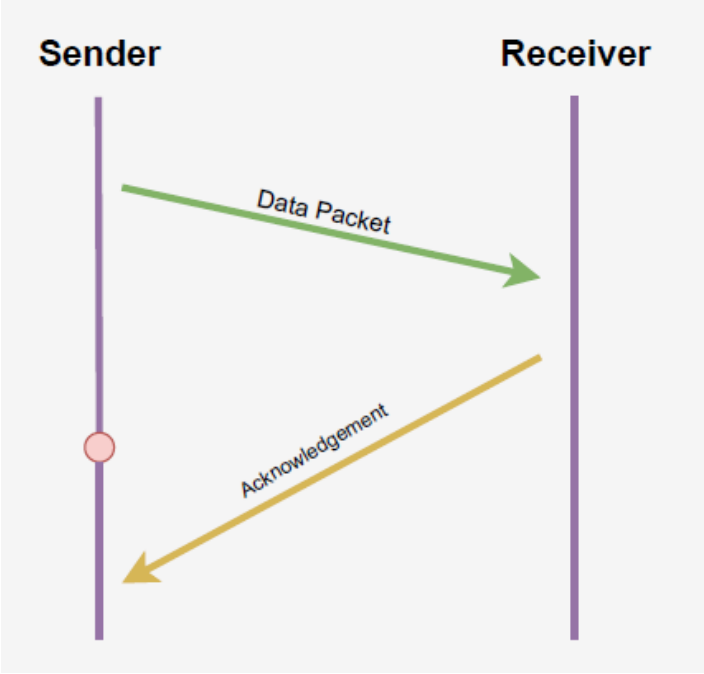


Figure 4 delayed data or acknowledgment

1.2 Modifications to overcome the disadvantages

Above 3 problems are resolved by Stop and Wait ARQ (Automatic Repeat Request) that does both error control and flow control.

1. Time Out

The problems due to lost data can be solved by using a time-out timer, after transmitting the data packet to the receiver through the communication channel, the sender starts the time out timer. Now if the data packet's feedback is received by the sender before the timer expires, then the sender stops the timer and transmits the next data packet.

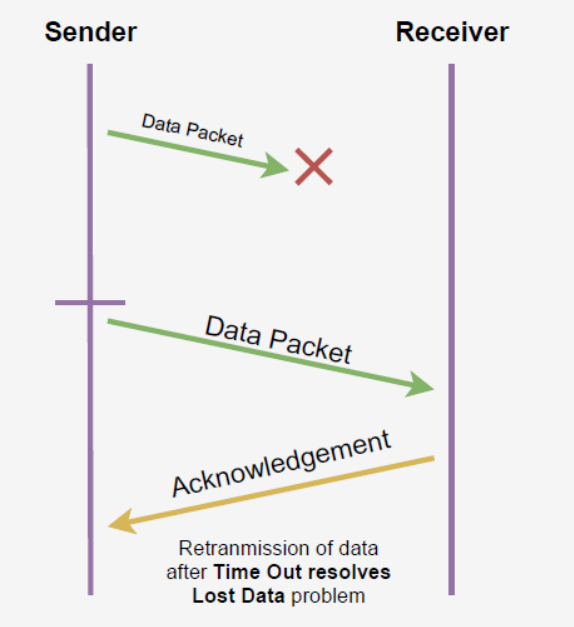


Figure 5 Time out to solve data lost problem

2. **Sequence Number** (Data)

The problem of lost acknowledgement is solved by putting sequence numbers on data packets. Assume that the feedback given by the receiver is lost, then the sender resends the same data packet after its timer expires. This prevents the occurrence of deadlock. The sequence number on the data packets helps the receiver to identify the duplicate data packet, then the receiver discards the duplicate data packet & resends the same acknowledgement.

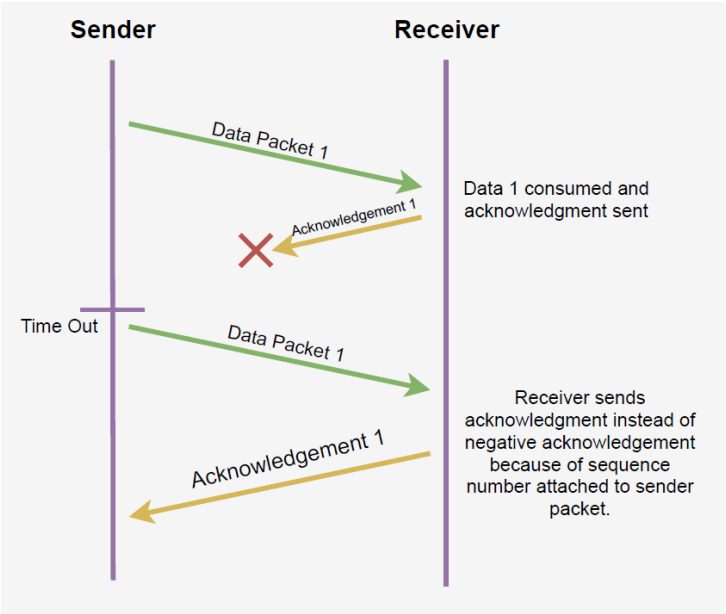


Figure 6 sequence number with data to solve acknowledgement lost problem

3. **Delayed Acknowledgement**

The problem of delayed feedback is solved by putting sequence numbers on the acknowledgement. Assume that the feedback given by the receiver is lost, then the sender resends the same data packet after its timer expires. The sequence number on the data packets helps the receiver to identify the duplicate data packet, then the receiver discards the duplicate data packet & resends the same feedback. Meanwhile if the delayed feedback also reaches the sender, the sender discards the duplicate feedback based on the feedback sequence number.

This is resolved by introducing sequence number for acknowledgement also.

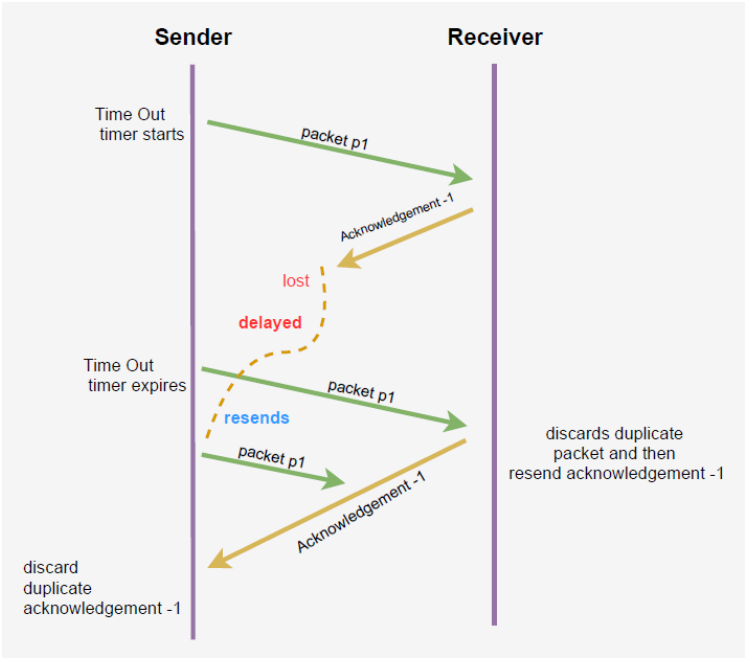


Figure 7 sequence number with acknowledgement to solve delayed data/acknowledgemnt problem

Question No. 2

Solution to Question No. 2:

2.1 Program to compute checksum at the transmitter

Algorithm to compute checksum at the transmitter

```
STEP 1: Start
STEP 2: Input data
STEP 3: divide data into n blocks with k elements each and store
each block in an array
STEP 4: do binary addition of all the blocks and store in sum array
STEP 5: if the length of the sum is more than k, again do the
binary addition of the carry and the sum
STEP 6: complement the sum to get the checksum
STEP 7: Print checksum
STEP 8: Stop
```

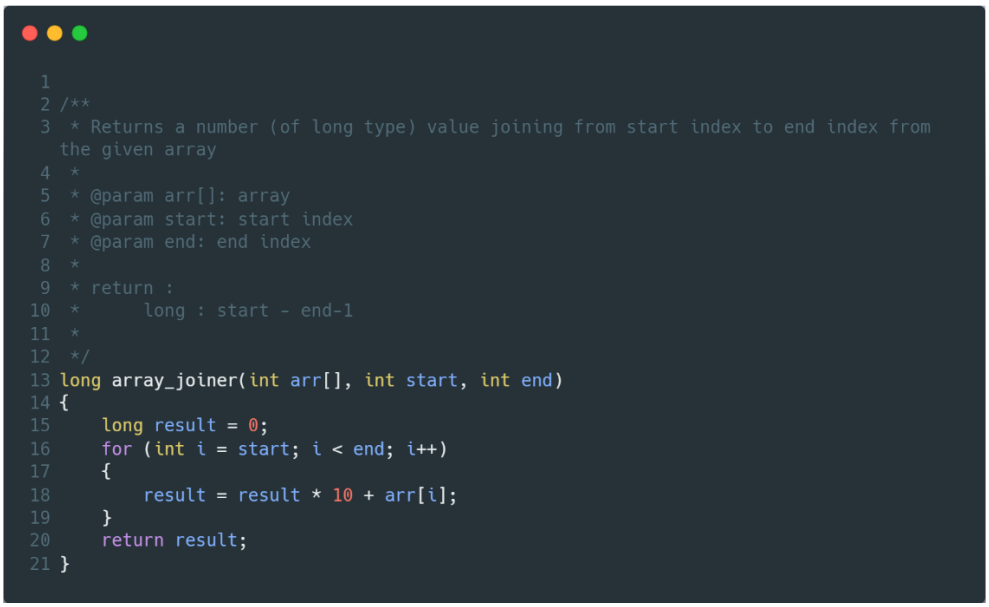
Program to compute checksum at the transmitter

```c
#include <stdio.h>

#define BLOCK_SIZE 8
#define SIZE 4

int sum[10][20];
int last_index = 0, last_known_len = 0;
int checksum[BLOCK_SIZE];

// function prototypes
void reverse_array(int[], int, int);
long array_joiner(int[], int, int);
long generate_checksum(int);
int binary_adder(int, int, int);

int main()
{
    int data[BLOCK_SIZE * SIZE];

    // getting 32 bit long data
    printf("Enter Data: ");
    for (size_t i = 0; i < (BLOCK_SIZE * SIZE); i++)
    {
        scanf("%d", &data[i]);
    }

    // printing each block
    for (size_t i = 0; i < SIZE; i++)
    {
        printf("Block %d : ", i + 1);
        for (size_t j = BLOCK_SIZE * i; j < BLOCK_SIZE * (i + 1); j++)
        {
            printf("%d", data[j]);
        }
        printf("\n");
    }
    // divding 32 bit data into block size and storing in array
    long blocks[SIZE];
    for (size_t i = 0; i < SIZE; i++)
    {
        blocks[i] = array_joiner(data, BLOCK_SIZE * i, BLOCK_SIZE * (i + 1));
    }

    // adding first two blocks and storing in 0th postion of sum array
    binary_adder(blocks[0], blocks[1], 0);

    // adding remaining block and storing one after another
    for (int i = 2; i < SIZE; i++)
    {
        binary_adder(array_joiner(sum[i - 2], 0, last_known_len + 1), blocks[i], i -
1);
    }

    // depending on the block size adding carry bits with the remaining bits
    binary_adder(array_joiner(sum[last_index - 1], (last_known_len - BLOCK_SIZE + 1),
last_known_len + 1), array_joiner(sum[last_index - 1], 0, (last_known_len -
BLOCK_SIZE + 1)), last_index);

    // printing result => last index = index of last inserted value of the sum array
    generate_checksum(last_index);
    printf("Checksum : ");
    for (size_t i = 0; i < BLOCK_SIZE; i++)
    {
        printf("%d ", checksum[i]);
    }

    int transmit_data[BLOCK_SIZE * (SIZE + 1)];

    // creating transmit data which is concatination of checksum and original data
    for (size_t i = 0; i < (BLOCK_SIZE * (SIZE + 1)); i++)
    {
        if (i < BLOCK_SIZE)
        {
            transmit_data[i] = checksum[i];
        }
        else
        {
            transmit_data[i] = data[(i - BLOCK_SIZE)];
        }
    }

    printf("\nData to be transmitted : ");

    //printing data to be transmitted
    for (size_t i = 0; i < (BLOCK_SIZE * (SIZE + 1)); i++)
    {
        printf("%d ", transmit_data[i]);
    }

    return 0;
}
```

Figure 8 Driver Program

```c
/**
 * Reverses the array from start to end index
 *
 * @param arr[]: array which needs to be reversed
 * @param start: start index
 * @param end: end index
 *
 * return
 *      -
 */
void reverse_array(int arr[], int start, int end)
{
    int temp;
    while (start < end)
    {
        temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
}
```

Figure 9 Function to reverse an array



```c

/**
 * Returns a number (of long type) value joining from start index to end index from the given array
 *
 * @param arr[]: array
 * @param start: start index
 * @param end: end index
 *
 * return :
 *      long : start - end-1
 *
 */
long array_joiner(int arr[], int start, int end)
{
    long result = 0;
    for (int i = start; i < end; i++)
    {
        result = result * 10 + arr[i];
    }
    return result;
}
```

Figure 10 function to join array and return a number

```c
/**
 * calculates complement of the given index in sum array and returns a number (of
long type) joining it
 *
 * @param n: index of sum array whose complement needs to be calculated
 *
 * return:
 *      - long : complemented value of sum[n]
 *
 */
long generate_checksum(int n)
{

    for (size_t i = 0; i < BLOCK_SIZE; i++)
    { // if length of last sum value is less than BLOCK_SIZE
        // means, there are (BLOCK_SIZE - len(last_known_index)) 0 before the value
        // hence the complement will be 1
        if (i < (BLOCK_SIZE - last_known_len - 1))
        {
            checksum[i] = 1;
        }
        else
        { // doing complement of each value
            if (sum[last_index - 1][i - (BLOCK_SIZE - last_known_len - 1)] == 0)
            {
                checksum[i] = 1;
            }
            else
            {
                checksum[i] = 0;
            }
        }
    }
    return array_joiner(checksum, 0, BLOCK_SIZE);
}

```

Figure 11 Function to generate checksum

```c
/**
 * Generates and stores the value of two binary number in sum array at given index
 *
 * @param binary1: 1st binary number
 * @param binary2: 2nd binary number
 * @param n: index of sum array to store the result
 *
 * return:
 *      - int : 0
 */
int binary_adder(int binary1, int binary2, int n)
{
    last_index++;
    int result[20];
    int i = 0, remainder = 0;

    // calculating binary addition and and storing  in sum[n]
    while (binary1 != 0 || binary2 != 0)
    {
        sum[n][i++] = (binary1 % 10 + binary2 % 10 + remainder) % 2;
        remainder = (binary1 % 10 + binary2 % 10 + remainder) / 2;
        binary1 = binary1 / 10;
        binary2 = binary2 / 10;
    }
    if (remainder != 0)
        sum[n][i++] = remainder;

    --i;
    // reversing the array as the value stored is reversed
    reverse_array(sum[n], 0, i);

    // updating length of last added value
    last_known_len = i;

    return 0;
}
```

Figure 12 Function to add two binary numbers

Output

```
# make file

checksum_transmit:
	@gcc checksum_transmit.c
	@./a.out

checksum_receive:
	@gcc checksum_receive.c
	@./a.out

clean:
	@rm *.out

.PHONY: clean
```

Figure 13 makefile

Let's say we want to send a 32-bit long data: 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 0 1 1
It can be divided into $k = 4$ blocks.

Block 1	1	1	0	0	1	1	0	0
Block 2	1	1	1	1	0	0	0	0
Block 3	1	0	1	0	1	0	1	0
Block 4	1	1	0	0	0	0	1	1

Adding all the block we get

Carry bits	**1**	**0 1**	**0**	**1**	**1**	**1**	**1**	**1**		
Block 1			1	1	0	0	1	1	0	0
Block 2			1	1	1	1	0	0	0	0
Block 3			1	0	1	0	1	0	1	0
Block 4		**+**	1	1	0	0	0	0	1	1
Sum	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**1**

Adding carry bits with the sum,

Sum			0	0	1	0	1	0	0	1
Carry								+	1	1
Final Sum			**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**
checksum			**1**	**1**	**0**	**1**	**0**	**0**	**1**	**1**

Data to be transmitted becomes:

1 1 0 1 0 0 1 1 – 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 0 1 1

```
 subhendu@LAPTOP-AL8CTHTV  /mnt/d/RUAS-sem-05/CN/assignment  make checksum_transmit
Enter Data: 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 0 1 1
Block 1 : 11001100
Block 2 : 11110000
Block 3 : 10101010
Block 4 : 11000011
Checksum : 1 1 0 1 0 0 1 1
Data to be transmitted : 1 1 0 1 0 0 1 1 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 0 1 1 %
 subhendu@LAPTOP-AL8CTHTV  /mnt/d/RUAS-sem-05/CN/assignment
```

Figure 14 execution of checksum_transmit

2.2 Program to check for error free data transmission at the receiver

Algorithm to check for error free transmission at the receiver

STEP 1: Start

STEP 2: Input received data

STEP 3: divide data into n blocks with k elements each and store each block in an array

STEP 4: First k elements or the first block is the checksum

STEP 4: do binary addition of all the blocks and the checksum and store in sum array

STEP 5: if the length of the sum is more than k, again do the binary addition of the carry and the sum

STEP 6: if the final sum contains all 1's, then the received data is error free, else the received data is **not** error free.

STEP 7: print the sum

STEP 8: print 'ERROR' if the sum contains 0, else print 'NO ERROR'

STEP 9: Stop

Program to compute checksum at the transmitter

```c
#include <stdio.h>

#define BLOCK_SIZE 8
#define SIZE 5

int sum[10][20];
int last_index = 0, last_known_len = 0;

// function prototypes
void reverse_array(int[], int, int);
long array_joiner(int[], int, int);
int binary_adder(int, int, int);

int main()
{
    int data[BLOCK_SIZE * SIZE];

    printf("Enter Received Data: ");
    for (size_t i = 0; i < (BLOCK_SIZE * SIZE); i++)
    {
        scanf("%d", &data[i]);
    }

    // printing each block
    for (size_t i = 0; i < SIZE - 1; i++)
    {
        printf("Block %d : ", i + 1);
        for (size_t j = BLOCK_SIZE * (i + 1); j < BLOCK_SIZE * (i + 2); j++)
        {
            printf("%d", data[j]);
        }
        printf("\n");
    }

    //printing checksum
    printf("Checksum : ");
    for (size_t i = 0; i < BLOCK_SIZE; i++)
    {
        printf("%d ", data[i]);
    }

    // divding 32 bit data into block size and storing in array
    long blocks[SIZE];
    for (size_t i = 0; i < SIZE; i++)
    {
        blocks[i] = array_joiner(data, BLOCK_SIZE * i, BLOCK_SIZE * (i + 1));
    }

    // adding first two blocks and storing in 0th postion of sum array
    binary_adder(blocks[0], blocks[1], 0);

    // adding remaining block and storing one after another
    for (int i = 2; i < SIZE; i++)
    {
        binary_adder(array_joiner(sum[i - 2], 0, last_known_len + 1), blocks[i], i -
1);
    }

    // depending on the block size adding carry bits with the remaining bits
    binary_adder(array_joiner(sum[last_index - 1], (last_known_len - BLOCK_SIZE + 1),
last_known_len + 1), array_joiner(sum[last_index - 1], 0, (last_known_len -
BLOCK_SIZE + 1)), last_index);

    int flag = 1;

    printf("\nSum of checksum and all blocks : ");
    for (size_t i = 0; i < BLOCK_SIZE; i++)
    {
        printf("%d ", sum[last_index - 1][i]);

        // checking if any value is other than 1
        // which means theres error in data
        if (sum[last_index - 1][i] == 0)
        {
            flag = 0;
        }
    }

    // if flag=1 , there's no error
    if (flag)
    {
        printf("\n NO ERROR in Data");
    }
    else
    {
        printf("\n ERROR in Data");
    }

    return 0;
}
```

Figure 15 driver program

```c
/**
 * Reverses the array from start to end index
 *
 * @param arr[]: array which needs to be reversed
 * @param start: start index
 * @param end: end index
 *
 * return
 *      -
 */
void reverse_array(int arr[], int start, int end)
{
    int temp;
    while (start < end)
    {
        temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
}
/**
 * Returns a number (of long type) value joining from start index to end index from the given array
 *
 * @param arr[]: array
 * @param start: start index
 * @param end: end index
 *
 * return :
 *      long : start - end-1
 *
 */
long array_joiner(int arr[], int start, int end)
{
    long result = 0;
    for (int i = start; i < end; i++)
    {
        result = result * 10 + arr[i];
    }
    return result;
}

/**
 * Generates and stores the value of two binary number in sum array at given index
 *
 * @param binary1: 1st binary number
 * @param binary2: 2nd binary number
 * @param n: index of sum array to store the result
 *
 * return:
 *      - int : 0
 */
int binary_adder(int binary1, int binary2, int n)
{
    last_index++;
    int result[20];
    int i = 0, remainder = 0;

    // calculating binary addition and and storing  in sum[n]
    while (binary1 != 0 || binary2 != 0)
    {
        sum[n][i++] = (binary1 % 10 + binary2 % 10 + remainder) % 2;
        remainder = (binary1 % 10 + binary2 % 10 + remainder) / 2;
        binary1 = binary1 / 10;
        binary2 = binary2 / 10;
    }
    if (remainder != 0)
        sum[n][i++] = remainder;

    --i;
    // reversing the array as the value stored is reversed
    reverse_array(sum[n], 0, i);

    // updating length of last added value
    last_known_len = i;

    return 0;
}

```

Figure 16 function to reverse an array, join an array and add two binary numbers

Output

```
# make file

checksum_transmit:
	@gcc checksum_transmit.c
	@./a.out

checksum_receive:
	@gcc checksum_receive.c
	@./a.out

clean:
	@rm *.out

.PHONY: clean
```

Figure 16 make file

Let's say we received data:

1 1 0 1 0 0 1 1 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 0 1 1

Where,

Block 1	1	1	0	0	1	1	0	0
Block 2	1	1	1	1	0	0	0	0
Block 3	1	0	1	0	1	0	1	0
Block 4	1	1	0	0	0	0	1	1
checksum	1	1	0	1	0	0	1	1

Adding all the blocks and checksum we get

Block 1			1	1	0	0	1	1	0	0
Block 2			1	1	1	1	0	0	0	0
Block 3			1	0	1	0	1	0	1	0
Block 4			1	1	0	0	0	0	1	1
checksum		**+**	1	1	0	1	0	0	1	1
Sum			**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

Hence, the data is error free.

```
 subhendu@LAPTOP-AL8CTHTV  /mnt/d/RUAS-sem-05/CN/assignment  make checksum_receive
Enter Received Data: 1 1 0 1 0 0 1 1 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 0 1 1
Block 1 : 11001100
Block 2 : 11110000
Block 3 : 10101010
Block 4 : 11000011
Checksum : 1 1 0 1 0 0 1 1
Sum of checksum and all blocks : 1 1 1 1 1 1 1 1
 NO ERROR in Data%
 subhendu@LAPTOP-AL8CTHTV  /mnt/d/RUAS-sem-05/CN/assignment
```

Figure 17 execution of checksum_receive with correct received data

Running the program with last bit changed,

```
 subhendu@LAPTOP-AL8CTHTV  /mnt/d/RUAS-sem-05/CN/assignment  make checksum_receive
Enter Received Data: 1 1 0 1 0 0 1 1 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 0 1 0
Block 1 : 11001100
Block 2 : 11110000
Block 3 : 10101010
Block 4 : 11000010
Checksum : 1 1 0 1 0 0 1 1
Sum of checksum and all blocks : 1 1 1 1 1 1 1 0
 ERROR in Data%
 subhendu@LAPTOP-AL8CTHTV  /mnt/d/RUAS-sem-05/CN/assignment
```

Figure 18 execution of checksum_receive with incorrect received data

Advantage of checksum

The checksum detects all the errors involving an odd number of bits as well as the error involving an even number of bits.

Disadvantage of checksum

The main problem is that the error goes undetected if one or more bits of a subunit is damaged and the corresponding bit or bits of a subunit are damaged and the corresponding bit or bits of opposite value in second subunit are also damaged. This is because the sum of those columns remains unchanged.

Example:

Let's say we have data:

1 1 0 1 0 0 1 1 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 0 1 1

Where,

Block 1	1	1	0	0	1	1	0	0
Block 2	1	1	1	1	0	0	0	0
Block 3	1	0	1	0	1	0	1	0
Block 4	1	1	0	0	0	0	1	1
checksum	1	1	0	1	0	0	1	1

But the data received is

1 1 0 1 0 0 1 1 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 0 1 0 1 0 1 0 1 **1** 1 1 0 0 0 0 1 **0**

```
 subhendu@LAPTOP-AL8CTHTV  /mnt/d/RUAS-sem-05/CN/assignment  make checksum_receive
Enter Received Data: 1 1 0 1 0 0 1 1 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 0 1 0 1 0 1 0 1 1 1 1 0 0 0 0 1 0
Block 1 : 11001100
Block 2 : 11110000
Block 3 : 10101011
Block 4 : 11000010
Checksum : 1 1 0 1 0 0 1 1
Sum of checksum and all blocks : 1 1 1 1 1 1 1 1
 NO ERROR in Data%
 subhendu@LAPTOP-AL8CTHTV  /mnt/d/RUAS-sem-05/CN/assignment
```

Although data is corrupted, the error is undetected.

1. https://www.javatpoint.com/stop-and-wait-protocol
2. https://www.geeksforgeeks.org/stop-and-wait-arq/
3. https://www.tutorialspoint.com/error-detecting-codes-checksums