

GUJARAT TECHNOLOGICAL UNIVERSITY

Master in Computer Application (Integrated MCA)

Year III – (Semester-V) (W.E.F. June 2015)

Subject Name: Fundamentals of Networking (Indicative List)

Subject Code: 4450603

General Instructions for Faculty Members/Lab Instructors:

- All the programs mentioned in this list are to be performed in GNU C /Linux/Unix environment
- These programs are intended to simulate various aspects like flow control, error control and various ARQs of the LLC sub-layer protocols
- Two separate programs have to be created. The sender and the receiver.
- The sender and receiver communicate with each other using the Linux IPC mechanism Named Pipes/FIFO. The no. of pipes to be created depends upon the application.
- Bitwise operators should be used wherever applicable. Eg. To corrupt bit/s at particular position/s in error detection/correction programs.
- Various Systems Calls like read(), write(), open(), delay(), etc, may be used.
- Students should be exposed to good programming practices/best practices for the given development environment.
- Students should be made aware about the different design alternatives for a given program and the implications of adapting a particular design approach on the program efficiency/performance.
- Students should be encouraged to develop generalized solutions as far as possible.
- This list, as mentioned, is indicative. Programs other than those in the list, based on similar concepts and content of the syllabus, might be asked in the practical exam.

FRAMING TECHNIQUES:

Byte Stuffing

1. Implement a Program in GNU C which demonstrates byte-stuffing framing technique, where sender reads data from a text file, encapsulates the read data in a frame, applies byte stuffing to the frame and sends it to receiver. Assume appropriate character for flag byte and stuff byte. Sender should display the frame data before stuffing and after stuffing for each frame transmitted. Receiver should display the received frame data before de-stuffing and after de-stuffing for each frame. Receiver should keep on storing

the de-stuffed frame data in an external file. At the end, the data file contents of the sender and the receiver should match. For sender/receiver communication use IPC mechanism FIFO/Named Pipes in Linux/Unix environment. Use Bit-Wise operators in C wherever applicable. Consider a simplistic frame structure consisting of only flag bytes and frame data. All other fields like FCS, etc. are to be neglected. Use appropriate data-types for various variables/frame structure. Take appropriate frame size and file sizes. Test the program for arbitrary sequences of flag byte/stuff byte/other data combinations in the sender side data file.

Bit Stuffing

2. Implement a Program in GNU C which demonstrates bit-stuffing framing technique, where sender reads data from a text file, encapsulates the read data in a frame, applies bit stuffing to the frame and sends it to receiver. Consider the character 01111110 (binary) for flag byte. Sender should display the frame data before bit stuffing and after bit stuffing for each frame transmitted. Receiver should display the received frame data before de-stuffing and after de-stuffing for each frame. Receiver should keep on storing the de-stuffed frame data in an external file. At the end, the data file contents of the sender and the receiver files should match. For sender/receiver communication use IPC mechanism FIFO/Named Pipes in Linux/Unix environment. Use Bit-Wise operators in C wherever applicable. Consider a simplistic frame structure consisting of only flag bytes and frame data and neglect all other fields like FCS, etc. Use appropriate data-types for various variables/frame structure. Take appropriate frame size and file sizes. Test the program for arbitrary bit sequences in the sender side data file.

Character Count

3. Implement a Program in GNU C which demonstrates character count based framing technique, where sender reads data from a text file, encapsulates the read data in a frame, applies character count in the designated character count field in the frame and transmits it to receiver. Sender should display the frame data along with the character count before transmission to the receiver for each frame. Receiver should display the received frame data along with received character count for each frame. Receiver should keep on storing the frame data in an external file. At the end, the data file contents of the sender and the receiver should match. For sender/receiver communication use IPC mechanism FIFO/Named Pipes in Linux/Unix environment. Use Bit-Wise operators in C wherever applicable. Consider a simplistic frame structure consisting of only frame data and character count field. All other fields like flag bytes, FCS, etc. are to be neglected. Use appropriate data-types for various variables/frame structure. Take appropriate frame size and file sizes. Test the program for normal scenario as well as error scenario where the character count field is corrupted. Demonstrate the effect of corruption of character count field during transmission assuming there is no error detection mechanism.

ERROR DETECTION/CORRECTION TECHNIQUES

(In this program category, students may be asked to simulate different noise scenarios like single bit corruption, multi-bit corruption, etc. and check whether the receiver, using the given error detection/correction technique, is able to detect/correct the error.)

Single Bit Parity (VRC) method

1. Implement a Program in GNU C which demonstrates the Single Bit Even Parity Method (VRC). The sender reads a single ASCII character from K/B, applies the parity bit logic as applicable, encapsulates the character in a frame and transmits the frame to the receiver. Sender should display the ASCII character on screen before and after applying the parity bit logic. Receiver should apply the necessary parity logic and subsequently decide whether there is error or not. If, as per the receiver, there is an error, it should display an appropriate error message on screen else it should display the original character (as per the receiver). Create Bit corruption routines on the sender side in the program and accordingly demonstrate those bit error scenarios for which this method works and also those bit error scenarios for which this method fails. For sender/receiver communication use IPC mechanism FIFO/Named Pipes in Linux/Unix environment. Use Bit-Wise operators in C wherever applicable. Consider a simplistic frame structure consisting of only frame data (of single character). All other fields of frame are to be neglected. Use appropriate data-types for various variables/frame structure. Take appropriate frame size.

Block Parity (LRC) Method

2. Implement a Program in GNU C which demonstrates the Block Parity (LRC) Method (for Even Parity). The sender reads an ASCII character string from K/B, applies the block parity logic as applicable, encapsulates the character string in a frame and transmits the frame to the receiver. Sender should display the ASCII character string on screen before and after applying the block parity logic. Receiver should apply the necessary block parity logic and subsequently decide whether there is error or not. If, as per the receiver, there is an error, it should display an appropriate error message on screen else it should display the original character string (as per the receiver). Create Bit corruption routines on the sender side in the program and accordingly demonstrate

those bit error scenarios for which this method works and also those bit error scenarios for which this method fails. For sender/receiver communication use IPC mechanism FIFO/Named Pipes in Linux/Unix environment. Use Bit-Wise operators in C wherever applicable. Assume a simplistic frame structure consisting of only frame data (ASCII character string). All other fields of frame are to be neglected. Use appropriate data-types for various variables/frame structure. Take appropriate frame size.

Validity of CRC Divisor

3. Implement a Program in GNU C which determines whether a given Divisor is valid for CRC. The Divisor is to be entered in binary format from K/B. The sender encapsulates this data in a frame and transmits it to the receiver. The receiver applies the validation logic for CRC divisor and accordingly displays appropriate validation message on screen. For sender/receiver communication use IPC mechanism FIFO/Named Pipes in Linux/Unix environment. Use Bit-Wise operators in C wherever applicable. Consider a simplistic frame structure which encapsulates only the divisor information. All other fields of frame are to be neglected. Use appropriate data-types for various variables/frame structure. Take appropriate frame size.

CRC 16

4. Implement a Program in GNU C which demonstrates the CRC 16 checksum Method. Hard-Code the corresponding Divisor value ($G(x)$) for CRC 16 for both the sender and receiver. The sender reads an ASCII character string from K/B, computes the corresponding CRC 16 checksum, encapsulates the data and the CRC checksum in the frame and transmits the frame to the receiver. Sender should display the CRC 16 checksum value on screen before transmission. Receiver should apply the necessary CRC 16 logic and subsequently decide whether there is error or not. If, as per the receiver, there is an error, it should display an appropriate error message on screen else it should display the original character string. Create Bit corruption routines on the sender side in the program and accordingly Test the program for arbitrary single bit errors as well as burst errors (both even and odd bit errors) and record your observations. For sender/receiver communication use IPC mechanism FIFO/Named Pipes in Linux/Unix environment. Use Bit-Wise operators in C wherever applicable. Consider a simplistic frame structure consisting of only frame data (ASCII character string) and checksum field. All other fields of frame are to be neglected. Use appropriate data-types for various variables/frame structure. Take appropriate frame size.
5. Repeat above program no. 4 for CRC-32.
6. Repeat above program no. 4 for any valid CRC Divisor of your own choice.

1's Complement based Checksum

7. Implement a Program in GNU C which demonstrates the 1's complement based checksum Method. The sender reads an ASCII character string from K/B, computes the corresponding 1's complement checksum, encapsulates the data and the one's complement checksum in the frame and transmits the frame to the receiver. Sender should display the 1's complement checksum value on screen before transmission. Receiver should apply the necessary one's complement based checksum logic and subsequently decide whether there is error or not. If, as per the receiver, there is an error, it should display an appropriate error message on screen else it should display the original character string. Create Bit corruption routines on the sender side in the program and accordingly test the program for arbitrary single bit errors as well as burst errors (both even and odd bit errors) and record your observations. For sender/receiver communication use IPC mechanism FIFO/Named Pipes in Linux/Unix environment. Use Bit-Wise operators in C wherever applicable. Consider a simplistic frame structure consisting of only frame data (ASCII character string) and checksum field. All other fields of frame are to be neglected. Use appropriate data-types for various variables/frame structure. Take appropriate frame size.

Hamming Code (for single bit error correction)

8. Implement a Program in GNU C which demonstrates the Hamming Code method. The sender reads an ASCII character from K/B, computes the corresponding Hamming Code and encapsulates the Hamming codeword into a frame and transmits the frame to the receiver. Sender should display the bit pattern of the codeword on screen before transmission. Receiver should apply the necessary Hamming code logic and identify whether there is an error or not. In case of no error, it should display the original character on screen. In case of error, it should display appropriate error message and also the bit position of the corrupted bit. Receiver should correct the error and then display the corrected character on screen. Create single Bit corruption routines on the sender side in the program and accordingly test the program for arbitrary single bit error positions and record your observations. For sender/receiver communication use IPC mechanism FIFO/Named Pipes in Linux/Unix environment. Use Bit-Wise operators in C wherever applicable. Consider a simplistic frame structure consisting of only the Hamming Code Word for the given ASCII character. All other fields of frame are to be neglected. Use appropriate data-types for various variables/frame structure. Take appropriate frame size.

Hamming Code for Burst Error Correction of known size

9. Modify the above program no. 8 for dealing with burst errors of known size.

DATA LINK LAYER PROTOCOLS

1. Protocol for an Ideal Channel and Ideal Nodes

Implement a Program in GNU C in which the sender reads data from a given file, encapsulates the data in a frame and transmits the frame to the receiver. This goes on till all the data is read from the file. The receiver goes on receiving the frames, de-capsulate the data from frame and store the data in a separate file. At the end, the original data file at the sender's end and the contents of the file at the receiver's end should tally. For sender/receiver communication, use IPC mechanism FIFO/Named Pipes in Linux/Unix environment. Use Bit-Wise operators in C wherever applicable. Consider a simplistic frame structure consisting of only frame data (ASCII character string). All other fields of frame are to be neglected. Use appropriate data-types for various variables/frame structure. Take appropriate frame size and file sizes. Assume an ideal situation where the channel is noiseless and receiver node has infinite system resources.

2. One Way Stop and Wait Protocol for Noiseless channel and Slow Receiver

Implement a Program in GNU C to demonstrate the working of the stop and wait protocol for a noiseless channel and a slow receiver. The sender reads data from a given file, encapsulates the data in a frame and transmits the frame to the receiver. The receiver receives the frame, de-capsulate the data from frame and stores the data in a separate file. Then the receiver transmits a dummy acknowledgement frame to the sender. Dummy in the sense, the contents of the acknowledgement frame is not important. It could be garbage. On receipt of the dummy acknowledgement frame from the receiver, the sender transmits the next frame. This goes on till all the data is transmitted by the sender from the sender side file. At the end, the original data file at the sender's end and the contents of the file at the receiver's end should tally. For sender/receiver communication, use IPC mechanism FIFO/Named Pipes in Linux/Unix environment. Use Bit-Wise operators in C wherever applicable. Consider a simplistic frame structure consisting of only frame data (ASCII character string). All other fields of frame are to be neglected. Use appropriate data-types for various variables/frame

structure. Take appropriate frame size and file sizes. Assume that the channel is noiseless and the receiver to be having slower processing speed then the sender.

3. One Way Stop and Wait Protocol for Noisy channel and Slow Receiver

Implement a Program in GNU C to demonstrate the working of the stop and wait protocol for a noisy channel and a slow receiver. The sender reads data from a given file, encapsulates the data in a frame and transmits the frame to the receiver. The sender starts a timer immediately upon transmitting the frame to the receiver. The receiver receives the frame and checks the frame for errors. If the frame is error free, the receiver sends a positive acknowledgement back to the sender. If the frame is erroneous, the receiver doesn't send positive acknowledgement back to the sender. In case of error free frame, the receiver de-capsulate the data from frame and stores the data in a separate file. On receipt of the positive acknowledgement frame from the receiver, the sender transmits the next frame.

In case of erroneous frame reaching the receiver, the sender times out and transmits the same frame again to the receiver. This goes on till all the data is transmitted by the sender from the sender side file. At the end, the original data file at the sender's end and the contents of the file at the receiver's end should tally. For sender/receiver communication, use IPC mechanism FIFO/Named Pipes in Linux/Unix environment. Use Bit-Wise operators in C wherever applicable. Consider a simplistic frame structure consisting of only data frame sequence no., acknowledgement frame sequence no. and frame data (ASCII character string). All other fields of frame are to be neglected. Use appropriate data-types for various variables/frame structure. Take appropriate frame size and file sizes. Assume that the channel is noisy and the receiver to be having slower processing speed then the sender. Set appropriate timeout value for timer. Simulate an error scenario whereby 25% of the data frames transmitted by the sender are corrupted. The error scenario should be created by simply generating random number based logic at receiver and accordingly treating the arrived frame to be corrupted or not.

Logic for deciding whether arrived acknowledgement is corrupted or not should also be developed on similar lines on sender side. FCS (Frame Check Sequence) of any kind is NOT expected for this program. Also assume that data frames sent by the sender never get lost and the acknowledgements sent by the receiver are never lost or corrupted. At the end of the program, inspect the data file contents at the receiver side to ensure that there is no data omission or duplication even in wake of errors.

- i. Modify the above program no. 3 in which instead of 25%, 33% data frames get corrupted.
- ii. Modify the above program no. 3 to simulate a scenario in which 25% of data frames get lost. However, assume that data frames never get corrupted. All other assumptions remain as they are.

- iii. Modify the above program no. 3 to simulate a scenario where 33% of acknowledgement frames get lost. However, assume that data frames never get lost or corrupted and acknowledgement frames never get corrupted.
- iv. Modify the above program no. 3 to simulate a scenario where 25% of acknowledgement frames get corrupted. However, assume that data frames never get lost or corrupted and acknowledgement frames never get lost.
- v. Modify the above program no. 3 to simulate delayed acknowledgements. Then demonstrate that by appropriate increase in time-out value, how the delayed acknowledgements are correspondingly reduced.
- vi. Modify the above program no. 3 in which the receiver sends a negative acknowledgement to the sender in wake of a corrupted frame instead of passively waiting for a time-out. Demonstrate the increase in system throughput after implementing this scheme.

4. One Way Go Back n Protocol for Noisy channel and Slow Receiver

Implement a Program in GNU C to demonstrate the working of the Go Back n Protocol for a noisy channel and a slow receiver. The sender reads data from a given file, encapsulates the data in a frame and transmits the frame to the receiver. The sender sends multiple frames to the receiver as per go back n protocol. The sender starts a timer immediately upon transmitting the frame to the receiver for each frame. The receiver receives each frame and if it is error free, transmits a positive acknowledgement frame back to the sender. The receiver de-capsulate the frame and stored the transmitted data into an external file. In case of erroneous frame, the receiver doesn't respond and waits for the time out. Assume that the channel is Noisy and the receiver is slow. Decide an appropriate value for the no. of frames to be pipelined at a time (Batch Size). Demonstrate that when a given frame is corrupted, all frames prior to the corrupted frame which were successfully accepted without error by receiver are also retransmitted. At the end, the original data file at the sender's end and the contents of the file at the receiver's end should tally. For sender/receiver communication, use IPC mechanism FIFO/Named Pipes in Linux/Unix environment. Use Bit-Wise operators in C wherever applicable. Consider a simplistic frame structure consisting of only data frame sequence no., acknowledgement frame sequence no. and frame data (ASCII character string). All other fields of frame are to be neglected. FCS (Frame Check Sequence) of any kind is NOT expected for this program. Use appropriate data-types for various variables/frame structure. Take appropriate frame size and file sizes. Assume that the channel is noisy and the receiver to be having slower processing speed then the sender. Set appropriate timeout value for timer. Simulate an error scenario whereby 25% of the data frames transmitted by the sender are corrupted. The error scenario should be created by simply generating random number based logic at receiver and accordingly treating the arrived frame to be corrupted or not. Logic for deciding whether arrived acknowledgement is corrupted or not should also be developed on similar lines on sender side. Also assume that data frames sent by the sender never get lost and the acknowledgements sent by the receiver are never lost or corrupted. The acknowledgements are cumulatively treated by the sender. At the end of the program,

inspect the data file contents at the receiver side to ensure that there is no data omission or duplication even in wake of errors.

- i. Modify the above program no .4 to simulate a scenario where 33% of acknowledgements are corrupted. Assume that data frames never get lost or corrupted and acknowledgement frames never get lost.
- ii. Modify the above program no. 4 to simulate a scenario where 25% of acknowledgements are lost. Assume that data frames never get lost or corrupted and acknowledgement frames never get corrupted.
- iii. Modify the above program no. 4 to simulate a scenario where 33% of data frames are lost. Assume that data frames never get corrupted and acknowledgement frames never get lost or corrupted.
- iv. Implement two way (bi directional) versions of above program no. 4 and all its modifications i, ii and iii.

5. One Way Selective Repeat Protocol for Noisy channel and Slow Receiver

Implement a Program in GNU C to demonstrate the working of the Selective Repeat protocol for a noisy channel and a slow receiver. The sender reads data from a given file, encapsulates the data in a frame and transmits the frame to the receiver. The sender sends multiple frames to the receiver as per predefined batch size. The sender starts a timer immediately upon transmitting the frame to the receiver for each frame. The receiver receives each frame and if it is error free, transmits a positive acknowledgement frame back to the sender. The receiver de-capsulate the frame and stored the transmitted data into an external file. In case of erroneous frame, the receiver sends a negative acknowledgement to the sender.

On receipt of negative acknowledgement or in case of time out, the sender retransmits the corresponding frame again. Assume that the channel is Noisy and the receiver is slow. Decide an appropriate value for the no. of frames to be pipelined at a time (Batch Size). Demonstrate that when a given frame is corrupted, all frames prior to the corrupted frame which were successfully accepted without error by receiver are retained by the receiver and receiver sends negative acknowledgement to the sender only for the corrupted frame. At the end, the original data file at the sender's end and the contents of the file at the receiver's end should tally. For sender/receiver communication, use IPC mechanism FIFO/Named Pipes in Linux/Unix environment. Use Bit-Wise operators in C wherever applicable. Consider a simplistic frame structure consisting of only data frame sequence no., acknowledgement frame sequence no. and frame data (ASCII character string). All other fields of frame are to be neglected. FCS (Frame Check Sequence) of any kind is NOT expected for this program. Use appropriate data-types for various variables/frame structure. Take appropriate frame size and file sizes. Assume that the channel is noisy and the receiver to be having slower processing speed then the sender. Set appropriate timeout value for timer. Simulate an error scenario whereby 25% of the data frames transmitted by the sender

are corrupted. The error scenario should be created by simply generating random number based logic at receiver and accordingly treating the arrived frame to be corrupted or not. Logic for deciding whether arrived acknowledgement is corrupted or not should also be developed on similar lines on sender side. Also assume that data frames sent by the sender never get lost and the acknowledgements sent by the receiver are never lost or corrupted. The acknowledgements are cumulatively treated by the sender. At the end of the program, inspect the data file contents at the receiver side to ensure that there is no data omission or duplication even in wake of errors.

- i. Modify the above program no. 5 to simulate a scenario where 33% of acknowledgements are corrupted. Assume that data frames never get lost or corrupted and acknowledgement frames never get lost.
- ii. Modify the above program no. 5 to simulate a scenario where 25% of acknowledgements are lost. Assume that data frames never get lost or corrupted and acknowledgement frames never get corrupted.
- iii. Modify the above program no. 5 to simulate a scenario where 33% of data frames are lost. Assume that data frames never get corrupted and acknowledgement frames never get lost or corrupted.
- iv. Implement two way (bi directional) versions of above programs no. 5 and all its modifications i. ii and iii.. Include the feature of piggybacking in these programs.

Reference Books for Practical:

1. W. Richard Stevens, “Advanced Programming in Unix Environment”, Pearson Education Publications ,Second Edition (to study how system calls can be used)
2. Vijay Mukhi, “C Odyssey: Unix the open Boundless C”, BPB Publications, Paperback Edition (1992) (for learning how to read and write data using named pipes)

Accomplishments of the student after completing the Course:

1. Understand and use the process of protocols and other techniques using C’ under Linux environment
2. Analyze and develop protocol/algorithm to solve real problems
3. Able to implement various protocols, Framing Techniques, Error detection and correction techniques.