Experiment 8

4/4/24

18

LEAKY BUCKET

Aim

To implement leaky bucket algorithm

Algorithm

1. Start
2. Include necessary header files. NOF_PACKETS ← 10
3. Define a function custom_rand that takes an integer a as argument and generates a random number between 1 and a using random() function.
4. Declare necessary variables
5. Initialize packet_sz array with random packet size between 10 and 50 bytes
6. Display generated packet sizes
7. Enter o_rate and b_size via input
8. Iterate through each packets in the packet_sz array:
   i) Check if adding current packet size to the remaining packet size exceeds the bucket size. If so, reject the packet.
   ii) If packet size does not exceeds bucket size, update remaining packet size and display information about incoming packets, remaining packets and time for transmission
   iii) Simulate transmission by iterating through the time required for transmission. If there are remaining bytes to transmit, determine

the output packet size based on o-rate and transmit the packet.

- update remaining packet size after transmission and display remaining bytes to transmit.
- If there are no packets to transmit, display a message indicating no packets for transmission.

5. Stop

Result

The program is executed and output is printed.

Experiment 9

4/9/24

20

DISTANCE VECTOR

Aim

To implement distance vector algorithm.

Algorithm

1. Start
2. Define a structure node to store distance and next hop information
3. Initialize routing table rt to store the informations for each router
4. Input no. of nodes and cost matrix
5. Initialise distance and next hop information based cost matrix
6. Repeat until Convergence:
   i) iterate through each node i
   ii) From each node i, iterate through each node j
   iii) For each pair of nodes (i, j) iterate through each node n.
   iv) If distance from i to j via n is is shorter than current distance, update distance and next hop information.
7. Print routing table for each router, showing the shortest distance and next hop
8. Repeat step 6 until no further updates are made
9. Stop.

Result

The program is executed and output is printed.

Experiment 10

11/4/24

22

FILE TRANSFER PROTOCOL

Aim

To implement file transfer protocol

Algorithm

Server

1. Import necessary libraries
2. Define SIZE with value 1024
3. Define a function write_file to receive data from a socket and write it to a file.
   - → Open file recv.txt in write mode
   - → Continuously receive data from the socket until there is no more data.
   - → Write the received data to file
   - → Clear buffer after each read operation.
4. Define main function
   - → Declare necessary variables
   - → Create socket using socket() function
   - → Initialize server address with IP address & port no.
   - → Bind socket using bind() function.
   - → Listen for incoming connections using listen() function.
   - → Accept connection using accept() function.
   - → Call write_file() function to recieve data from client and write it to a file
   - → Print success message.

5. Return 0 to indicate successful execution
6. Stop

Client
1. Include necessary files
2. Define a function send-file that takes a file pointer
3. Define Constant SIZE with value 1024
4. Declare variables n and data as character arrays of size SIZE
   → use a while loop to read data from file into array
   → use send() function to send data over socket
5. Define main function
   → declare necessary variables
   → initialize IP variable with server's IP address and port with server's port number.
   → create a socket using socket() function
   → print success message if socket created.
   → Initialize server_addr structure with server's address details
   → Attempt to connect to server using connect() function.
   → open file in read mode
   → call send_file function with file pointer and socket descriptor.
6. close socket using close() function and return 0.
7. stop.

Result

The programs are executed and output is printed.

Experiment 11

18/4/24

25

UDP DATAGRAM - WIRESHARK

Aim

To observe data transfered using UDP in client server communication and to study UDP datagrams

Description

Wireshark is a free network protocol analyzer that runs on windows, Mac, and linux/unix computers. It operates in computers using ethernet, 802.11 wireless LAN and many other link-layer technology. It typically stores and display the contents of various protocol fields in captured messages. A packet sniffer recieves a copy of packets that are send/received from/by application and protocols executing in the machine. Packet sniffer has two components named packet capture library and packet analyzer.

The major components of wireshark interface are:

→ Command menus

The two noteworthy menus are file menu that allows you to save captured packet data and exit the wireshark application. The capture menu allows you to begin packet capture and stop capturing when needed.

→ Packet listing window

Displays a one line summary of each packet captured, including the packet number assigned by wireshark, the time of capturing, packet's source & destination addresses, protocol type, and protocol specific information contained in packet

→ Packet - header details window

provides details about packet selected in packet-listing window.

→ Packet - Contents window

displays entire contents of the captured frame in both ASCII and hexadecimal format.

Result

Observed data transfer using UDP in client server communication by using wireshark.