DAYANANDA SAGAR COLLEGE OF ARTS SCIENCE AND COMMERCE

Shavige Malleshwara Hills, 1st Stage, Kumaraswamy Layout, Bengaluru, Karnataka.

Department of Computer Application-MCA (BU)

A project report on

APPLICATION OF TOC ON COMPUTER NETWORK IN DETAIL DESIGN AND IMPLEMENT A FINITE STATE MACHINE (FSM) FOR SIMPLE NETWORK PROTOCOL

Submitted To:

Mrs. Akshatha
Assistant Professor
DSCASC

Submitted By:

Chandana N (P03CJ24S126024)
Keshava Murthy N J (P03CJ24S126046)
Ashwini T (P03CJ24S126014)
Bhargav B (P03CJ24S126023)

INDEX

INTRODUCTION

This project explores the application of **Theory of Computation (TOC)** in the field of **Computer Networks**. In the rapidly evolving landscape of computer networks, the need for efficient and reliable communication protocols has never been more critical. As devices become increasingly interconnected, the underlying mechanisms that facilitate this communication must be robust, adaptable, and easy to manage. One of the foundational concepts in designing such systems is the **Finite State Machine (FSM),** a mathematical model that provides a structured way to represent the behavior of systems in response to various inputs.

This report focuses on the design and simulation of a simple network protocol using an **FSM** approach. The protocol will encompass essential operations such as establishing a connection, transferring data, and terminating the connection. By simulating these processes, we can gain insights into the dynamics of network communication and the role that FSMs play in ensuring smooth and efficient interactions between devices.

FSM OVERVIEW

Definition:

A Finite State Machine (FSM) is a computational model consisting of a finite number of states, transitions between those states, and actions. It is used to represent the behavior of systems in response to external inputs.

States and Events:

In the context of a network protocol, states represent the current status of the communication process, while events trigger transitions between these states.

TOC IN COMPUTER NETWORKS

Protocol Design:

TOC principles are applied in defining the rules and structures of communication protocols. FSMs are often used to model the behavior of protocols, ensuring correctness and efficiency.

Network Security:

TOC is crucial in cryptography and security protocols, helping to design secure algorithms and verify their properties through formal methods.

Error Detection and Correction:

TOC provides the theoretical basis for algorithms used in error detection and correction, ensuring reliable data transmission over networks.

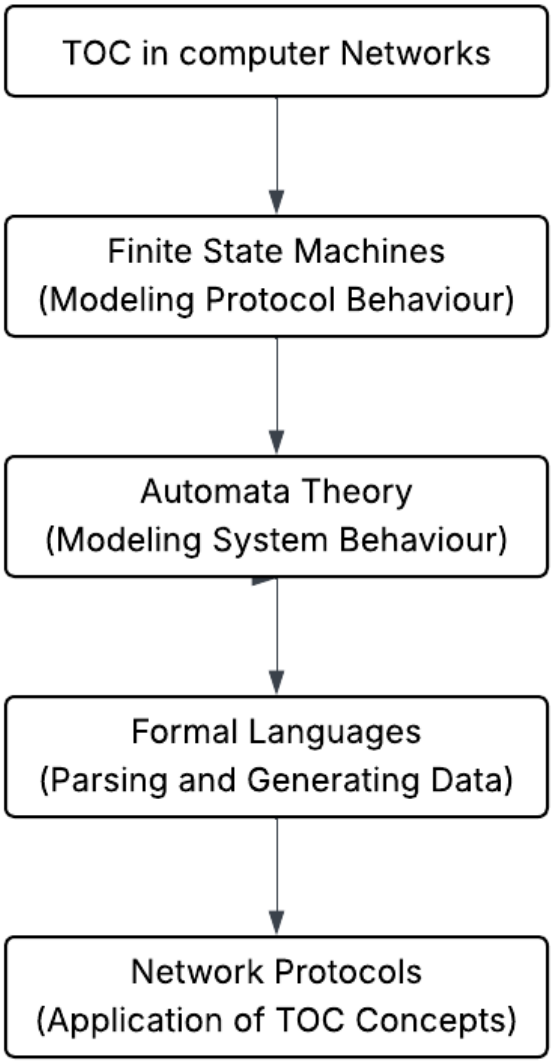
Routing Algorithm:

TOC aids in designing efficient routing algorithms by analyzing the complexity of different strategies and their performance in various network topologies.

Network Performance Analysis:

TOC allows for modeling network performance metrics, enabling the analysis of factors affecting network efficiency.

PROJECT DESCRIPTION



States of the Protocol:

Idle:

The initial state where the device is waiting for a connection.

Connecting:

The state when a connection is being established.

Transmitting Data:

The state when data is being sent.

Disconnected:

The state when the connection is terminated.

Events and Transitions:

Connect:

Initiates the connection process.

Data:

Represents the transmission of data.

Disconnect:

Terminates the connection.

IMPLEMENTATION USING C CODE

```
#include <stdio.h>
#include <string.h>
typedef enum {
    IDLE,
    CONNECTING,
    TRANSMITTING_DATA,
    DISCONNECTED
} State;

typedef struct {
    State state;
} NetworkProtocolFSM;

void initFSM(NetworkProtocolFSM *fsm) {
    fsm->state = IDLE;  // Initial state
}

void connect(NetworkProtocolFSM *fsm) {
    if (fsm->state == IDLE) {
        printf("Connecting...\n");
        fsm->state = CONNECTING;
    } else {
        printf("Cannot connect. Current state: %d\n", fsm->state);
    }
}

void transmit_data(NetworkProtocolFSM *fsm) {
    if (fsm->state == CONNECTING) {
        printf("Connection established. Transmitting data...\n");
        fsm->state = TRANSMITTING_DATA;
```

```
    } else if (fsm->state == TRANSMITTING_DATA) {
        printf("Already transmitting data.\n");
    } else {
        printf("Cannot transmit data. Current state: %d\n", fsm->state);
    }
}

void disconnect(NetworkProtocolFSM *fsm) {
    if (fsm->state == TRANSMITTING_DATA || fsm->state == CONNECTING) {
        printf("Disconnecting...\n");
        fsm->state = DISCONNECTED;
    } else {
        printf("Already disconnected or idle. Current state: %d\n", fsm->state);
    }
}

void reset(NetworkProtocolFSM *fsm) {
    if (fsm->state == DISCONNECTED) {
        printf("Resetting to Idle state...\n");
        fsm->state = IDLE;
    } else {
        printf("Cannot reset. Current state: %d\n", fsm->state);
    }
}

const char* get_state(NetworkProtocolFSM *fsm) {
    switch (fsm->state) {
        case IDLE: return "Idle";
        case CONNECTING: return "Connecting";
        case TRANSMITTING_DATA: return "Transmitting Data";
        case DISCONNECTED: return "Disconnected";
```

```
        default: return "Unknown State";
    }
}

// Simulation of the network protocol
void simulate_network_protocol() {
    NetworkProtocolFSM fsm;
    initFSM(&fsm);
    printf("Initial State: %s\n", get_state(&fsm));

    // Attempt to connect
    connect(&fsm);  // Transition to Connecting
    printf("Current State: %s\n", get_state(&fsm));

    // Attempt to transmit data
    transmit_data(&fsm);  // Transition to Transmitting Data
    printf("Current State: %s\n", get_state(&fsm));

    // Attempt to disconnect
    disconnect(&fsm);  // Transition to Disconnected
    printf("Current State: %s\n", get_state(&fsm));

    // Attempt to reset
    reset(&fsm);  // Transition back to Idle
    printf("Current State: %s\n", get_state(&fsm));
}

int main() {
    simulate_network_protocol();
 return 0;
}
```

OUTPUT

programiz.com/c-programming/online-compiler/

Gmail YouTube Maps Scholarship AWS GitHub All Bookmarks

Programiz
C Online Compiler

Ads by Google
Stop seeing this ad Why this ad? ⓘ

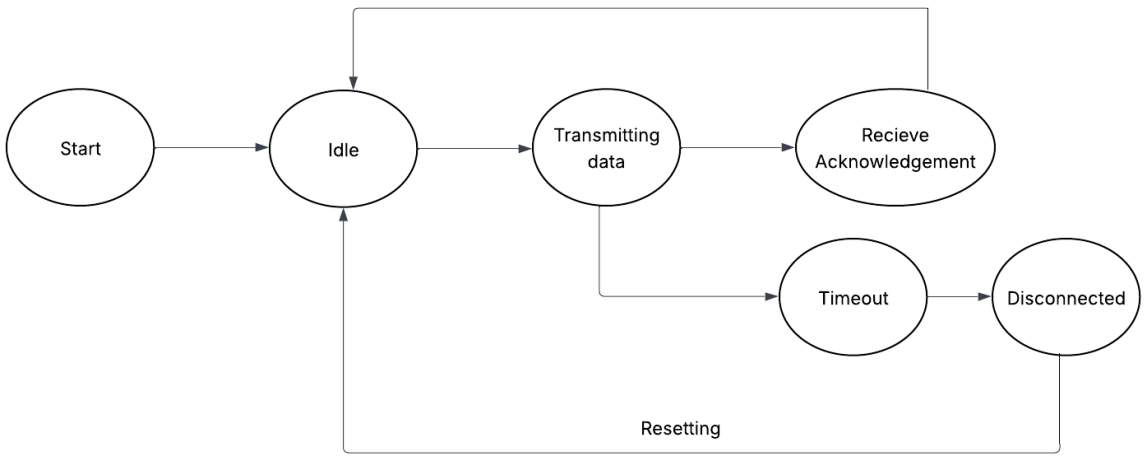
Programiz PRO ›

main.c Share Run Output Clear

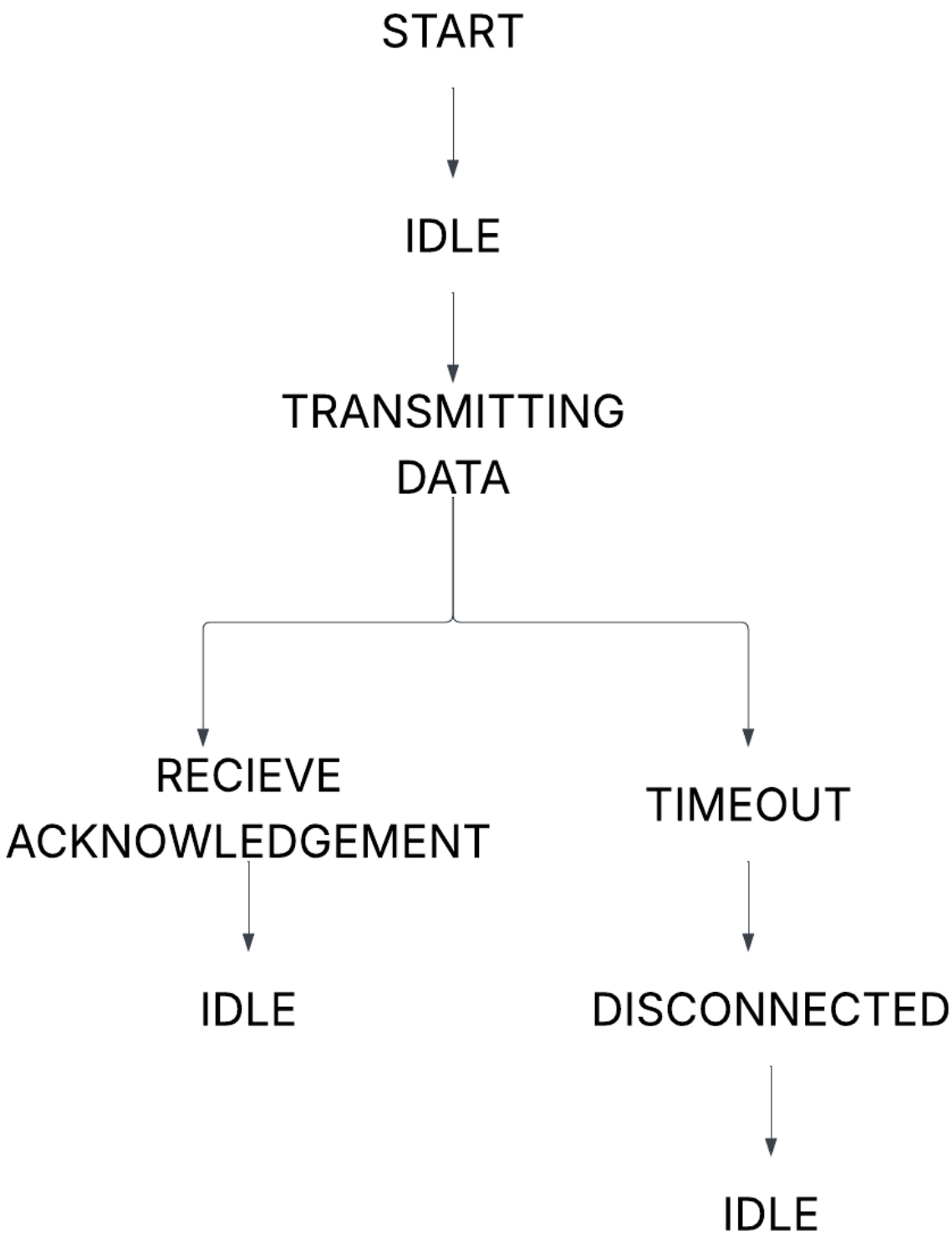
```
74      printf("Current State: %s\n", get_state(&fsm));
75
76      // Attempt to transmit data
77      transmit_data(&fsm);  // Transition to Transmitting Data
78      printf("Current State: %s\n", get_state(&fsm));
79
80      // Attempt to disconnect
81      disconnect(&fsm);  // Transition to Disconnected
82      printf("Current State: %s\n", get_state(&fsm));
83
84      // Attempt to reset
85      reset(&fsm);  // Transition back to Idle
86      printf("Current State: %s\n", get_state(&fsm));
87  }
88
89  int main() {
90      simulate_network_protocol();
91      return 0;
```

```
Initial State: Idle
Connecting...
Current State: Connecting
Connection established. Transmitting data...
Current State: Transmitting Data
Disconnecting...
Current State: Disconnected
Resetting to Idle state...
Current State: Idle


=== Code Execution Successful ===
```



PARSING DIAGRAM



CONCLUSION

In conclusion, the Theory of Computation plays a vital role in the design and implementation of computer networks, particularly through the use of Finite State Machines (FSMs). By modeling the various states and transitions that occur during network communication, FSMs help us understand and manage the complex interactions between devices.

The simulation of a simple network protocol using an FSM illustrates how these concepts come to life in practical applications. We can see how the protocol transitions through different states—connecting, transmitting data, disconnecting, and resetting—each with its own set of rules and behaviors.

The insights gained from applying the Theory of Computation to computer networks empower engineers and developers to create more efficient, robust, and secure communication protocols. As our world becomes increasingly interconnected, the importance of these foundational concepts will only continue to grow, ensuring that our networks can handle the demands of the future.