

Implementation of Scheduling Algorithm for transfer of Packets in Networks

Project report submitted in partial fulfillment of the requirement for the degree of
Bachelor of Technology

in

Computer Science and Engineering Information Technology

By

DAVINDER SINGH (141281)
PALAK (141277)

Under the supervision of

Dr. Amit Kumar Singh

to



**Department of Computer Science & Engineering and Information Technology
Jaypee University of Information Technology Wanknaghat, Solan-173234, Himachal
Pradesh**

Candidate's Declaration

I hereby declare that the work presented in this report entitled “**Implementation of Scheduling Algorithm for Transfer of Packets in Networks**” in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering and Information Technology** submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology Wanknaghat is an authentic record of my own work carried out over a period from August 2016 to December 2016 under the supervision of **Dr. Amit Kumar Singh** , Assistant Professor (Senior grade)

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

(Student Signature)

Davinder Singh(141281)

Palak(141277)

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Dr. Amit Kumar Singh
Assistant Professor (senior grade)
Department of Computer Science & Engineering and Information Technology

Dated:

ACKNOWLEDGEMENT

On the submission of our thesis report on “**Implementation of Scheduling Algorithm for transfer of packets in Networks**”, we would like to extend our gratitude and sincere thanks to our supervisor **Dr. Amit Kumar Singh, Assistant Professor (senior grade), Department of Computer Science & Engineering and Information Technology** for his constant motivation and support during the course of our work. We truly appreciate and value his esteemed guidance and encouragement from the beginning. We are indebted to him for having helped us shape the problem and providing insights towards the solution.

And for providing a solid background for our studies and research thereafter.

He has been a great source of inspiration to us and we thank him from the bottom of our heart. Above all, we would like to thank all our friends whose direct and indirect support helped us.

Davinder Singh (141281), Palak (141277)

CONTENTS

1. INTRODUCTION.....	1
1.1 About scheduling algorithm	
1.2 Benefits of using Priority Queuing and Round Robin	
1.3 Problem Statement	
1.4 Objectives	
1.5 Methodology	
2. LITERATURE SURVEY.....	12
3. SYSTEM DEVELOPMENT.....	15
3.1 Design Overview	
3.2 Proposed Algorithm Approach	
3.2.1 Proposed Algorithm	
3.3 Module Description	
3.3.1 Module Description for UDP	
3.3.2 Class Diagram of UDP	
3.3.3 Module Description for TCP	
3.3.4 Class Diagram for TCP	
3.3.5 Activity Diagram of the Process	
4. PERFORMANCE ANALYSIS.....	34
4.1 Experiment Test cases for UDP	
4.2 Experiment Results for UDP	
4.3 Experiment Test Cases for TCP	
4.4 Experiment Results for TCP	
5. CONCLUSION.....	46
5.1 Conclusion	
5.2 Future Scope	

REFERENCES

LIST OF ABBREVIATIONS

FCFS	First Come First Serve
SJF	Shortest Job First
PS	Priority Scheduling
RR	Round Robin
MDRR	Mean-Difference Round Robin Algorithm
SRBRR	shortest remaining burst in round robin
SRRQT	Selective-Round-Robin Quantum of Time
SMDRR	Sub-contrary Mean Dynamic Round Robin
BJF	Best Job First
PBDRR	Priority Based Dynamic Round Robin
FP	factor of precedence
PRRDTQ	Precedence based Round Robin with Dynamic Time Quantum
ITS	Intelligent Time Slice
MLFQ	multilevel feedback queue
VM	Virtual Machine
API	Application Program Interface
TCP	Transmission Control Protocol
UDP	User datagram Protocol
JDBC	Java Database Connectivity
URL	Uniform resource Locator
ODBC	Object Database Connectivity

LIST OF FIGURES

Fig. 1	Preemptive Scheduling
Fig. 2	Non-Preemptive Scheduling
Fig. 3	Multilevel Feedback Queue Process
Fig. 4	Major Java Features
Fig. 5	Java Compilation Process
Fig. 6	Java Compiler Parts
Fig. 7	Java compilation Parts
Fig. 8	A Java Platform
Fig. 9	Java program compilation
Fig. 10	OSI Layer
Fig. 11	UDP Model Diagram
Fig. 12	Jacobson's Algorithm
Fig. 13	UDP Class Diagram
Fig. 14	TCP Model Diagram
Fig. 15	TCP Class Diagram
Fig. 16	Activity Diagram
Fig. 17	UDP Packet Transfer
Fig. 18	UDP Proposed Algorithm
Fig. 19	Proposed Time
Fig. 20	Calculating time for Proposed algorithm
Fig. 21	UDP Packet Transfer
Fig. 22	UDP Proposed Algorithm
Fig. 23	Proposed Time
Fig. 24	Calculating time for Proposed algorithm

LIST OF TABLES

Table 1.	FCFS Process Table
Table 2.	FCFS Gantt chart
Table 3.	FCFS Gantt chart
Table 4.	SJF Process Table
Table 5.	SJF Gantt chart
Table 6.	PS Process Chart
Table 7.	PS Gantt Chart
Table 8.	RR Process Chart
Table 9.	RR Gantt Chart
Table 10.	Priority Scheduling Algorithm Test case
Table 11.	Proposed Scheduling Algorithm Test Case
Table 12.	Round Robin Scheduling Algorithm Test case
Table 13.	Proposed Scheduling Algorithm Test Case
Table 14.	Priority Scheduling Algorithm Test case
Table 15.	Proposed Scheduling Algorithm Test Case
Table 16.	Round Robin Scheduling Algorithm Test case
Table 18.	Proposed Scheduling Algorithm Test Case

ABSTRACT

Scheduling algorithm helps easing decision making in an efficient manner. Scheduling becomes necessary when the request for computing ability increases. Task scheduling and load balancing are two central and perplexing areas in the field of computer engineering. Distributing processes to the processor in such a manner such that total execution time is reduced. Improving the load on processors by way of balancing the load of processes is termed as load balancing. An algorithm is used using round robin and priority scheduling to help transferring packets in efficient manner. It provides a fair chance to each class to be executed successfully and without starvation. Scheduling is required in vast number of applications like reservations, process efficiency, medical appointments etc. All such applications use scheduling to perform operations efficiently. In this report, we describe the five types of scheduling algorithms along with their merit and limitations. First Come First Serve, Shortest job first, Priority Scheduling, Round robin, Multilevel Feedback Queue are the scheduling algorithms included. We tried to focus on the two algorithms i.e. Priority Scheduling and Round Robin as they are fairest of all the other algorithms. We implemented this algorithm in UDP and TCP architectures to transfer the packets and compare our proposed algorithm with all the other basic algorithms:

Chapter 1 introduces the basic scheduling algorithms explaining their execution criterions with suitable examples by plotting their Gantt charts and finding turnaround and waiting time for each.

Chapter 2 provides the literature survey showing important factors that play major role in selecting an efficient algorithm through the research papers published during recent years. In chapter 3, we present the system design overview in which we explained the choice of programming platform with its features and some general related terms. We also explained our proposed algorithm with steps and implemented the same on network architectures (TCP & UDP) analyzing the various class diagrams.

Chapter 4 presented the experiment results and analysis.

Concluding remarks and future directions are in Chapter 5.

CHAPTER 1

Scheduling Algorithms: An Introduction

1.1 Introduction & Scheduling Algorithms:

For the better understanding of the difficult set of rules and procedures which are used to run the order in which tasks are executed by the processor we need scheduling algorithms. Many CPU scheduling algorithms have been established for the modern multiprogramming operating system [1].

1.1.1 Issues in scheduling algorithms:

Different issues as described below,

- Processor can migrate from one class to another.
- Distribution in shortest possible time.
- Utilizing all the resources.
- Circulation of processes without any progress.
- Requirement of fair scheduling.
- Non-uniform and non-pre-emptive nature of distributed system.

1.1.2 Arrangement based on pre-emption[2]:

Preemptive Scheduling: When a higher priority job arrives in a system it can interrupt the system's current flow of execution. This type of scheduling is present in all kind of systems. Figure 1 shows the preemptive scheduling:

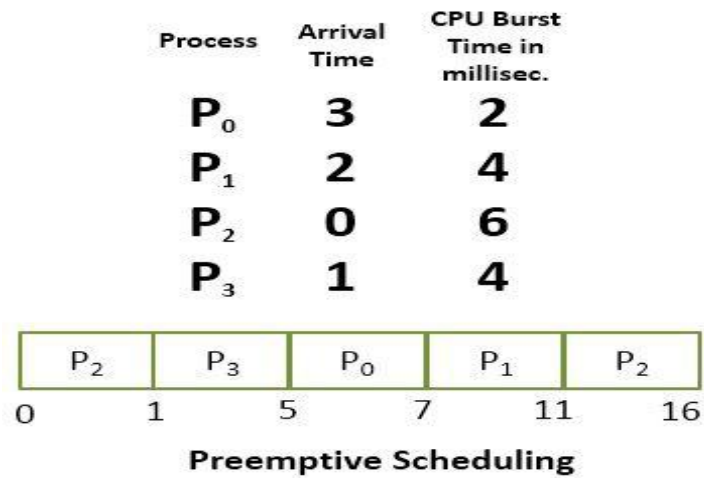


Figure 1

Non-Preemptive Scheduling: When a higher priority job arrives in a system it cannot interrupt the system's current flow of execution. FCFS is a non-pre-emptive type of scheduling. Figure 2 shows the non-preemptive scheduling:

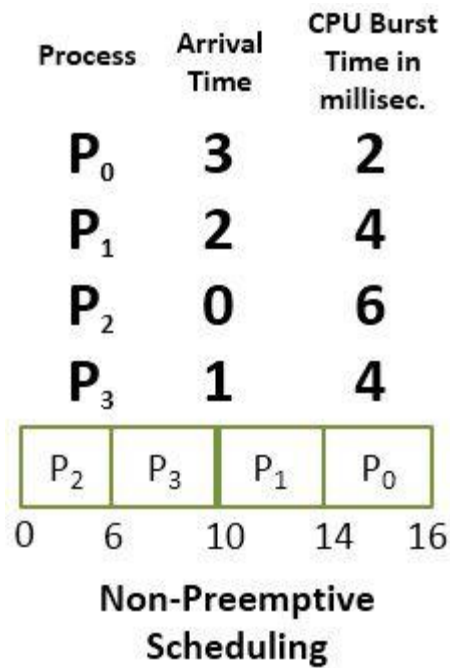


Figure 2

1.1.3 Basic Scheduling Algorithms[2]::

Some basic scheduling algorithms are discussed below:

- I. First Come First Serve Algorithm.
- II. Shortest Job First Algorithm.
- III. Priority based Algorithm.
- IV. Round Robin Algorithm.
- V. Multilevel Feedback Queue Algorithm.

I. First Come First Served Scheduling

In this algorithm the jobs are processed in the order of in which they appear in the ready queue. All the jobs are put into one ready queue. The PCB is linked onto the tail of the queue. In this type of algorithm the waiting time is very long. Below are the processes from P1 to P3 with their arrival time equal to 0:

Table 1 shows the process execution.

Table 1: Process execution in FCFS

Process	Burst time	Waiting Time	Turnaround time
P1	48	0	48
P2	6	48	54
P3	6	54	60
Average	-	34	54

Below given Gantt chart can be obtained in case of FCFS if order is above mentioned.

Table 2:Gantt chart for FCFS

P1	P2	P3
0	48	54
60		

If the order of the processed is P3, P3,P1:

Table 3: Gant chart

P2	P3	P1
0	6	12
60		

This algorithm cannot utilize parallel resources and waiting time is long=6 ms.

II. Shortest Job First Scheduling:

This algorithm takes into account the burst of next process to be executed. CPU is assigned to the process that has smallest burst time. If the burst of two processes is same then FCFS criteria can be used. Below example depicts the algorithm briefly: Table 4 shows the process execution.

Table 4: Process execution in SJF

Process	Burst time	Waiting Time	Turnaround time
P1	12	6	18
P2	16	32	48
P3	14	18	32
P4	6	0	6
Average	-	14	26

ms

Following gantt chart can be obtained when SJF scheduling is observed with process order as:

Table 5:Gantt chart for SJF

P4	P4	P3	P2	
0	6	18	32	48

For given processes the SJF generates minimum average waiting time. The only complexity with this algorithm is that it's hard to find out the burst of next process.

If short term scheduling is required then this algorithm can be implemented. This algorithm is either preemptive or non preemptive.

III. Priority Scheduling:

Priority is given to every process according to any criteria i.e. it can be numbers like 0 to 20 or alphabets A to B. Process with highest priority will get the CPU. FCFS criteria can be followed if two processes with similar priority are encountered. Below chart depicts the algorithm adequately: Processes are from P1 to P5 and lowest number represent highest numbers. Table 6 shows the process execution.

Table 6: Process execution in PS

Process	Burst time	Priority	Waiting Time	Turnaround time
P1	20	6	12	32
P2	2	2	0	2
P3	4	8	32	36
P4	2	10	36	38
P5	10	4	2	12
Average	-		16.4	24

Following Gantt chart can be obtained using the priority scheduling:

Table 7:Gantt chart for RR

P2	P5	P1	P3	P4
0	2	12	32	36

38

This algorithm can either be preemptive or nonpreemptive. Process with highest priority will be pre-empted other it will be placed on the head of the ready queue. This algorithm suffers from indefinite blocking i.e. starvation.

IV. Round Robin Scheduling:

This is one of the fairest algorithm of all the other algorithms and yet very simple. In this algorithm time quantum is defined which is basically a time interval for which the process will have CPU allocated to it. A circular queue of all the incoming processes is maintained and the scheduler goes round the queue to pick the process and execute it for a time slice. Process will give up the CPU voluntarily if its time is completed. If the process still needs more time it will be attached to tail of the circular queue. This algorithm has maximum average waiting time. Below chart explain the process adequately. Time quantum is 4ms. Table 8 shows the process execution.

Table 8: Process execution in RR

Process	Burst time	Waiting Time	Turnaround time
P1	48	12	60
P2	6	8	14
P3	6	14	20
Average	-	11.33	31.33

Following Gantt chart can be obtained in RR algorithm:

Table 9: Gantt chart for RR

P1	P2	P3	P1	P1	P1	P1	P1
0	4	7	10	14	18	22	26
30							

No process gets the time quantum more than 1 defined quantum. It is pre-emptive.

V. Multilevel Feedback Queue Scheduling:

In this algorithm once a process arrives in the system it assigned to a permanent queue. No change of queue is allowed after that. This algorithm has the less overhead of all other basic algorithms. Due to that it is highly inflexible also. This algorithm does have a mechanism of changing a process from one queue to another. It can be performed as:

- I. Process will be moved to a less priority queue if it is consuming more CPU time.
- II. Process will be moved to a high priority queue if it can be executed fast.

Figure 3 shows the Multilevel queue scheduling:

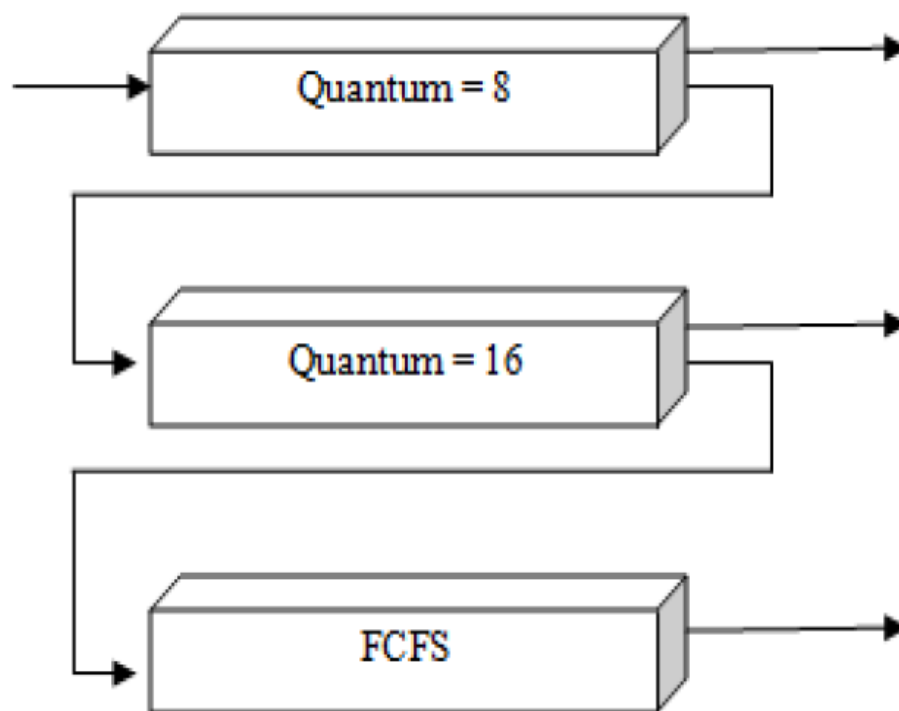


Figure 3:Multilevel Feedback Queuing

1.1.4 QUALITATIVE PARAMETERS [2]

Following are the parameters used to judge the efficiency of above written algorithms.

- I. **CPU Utilization/Efficiency:** Utilization of the CPU to its full potential.
- II. **Throughput:** maximizing the throughput is the motive.
- III. **Turnaround time:** Minimize the time between a process is submitted and processed.
- IV. **Waiting time:** The time spent in the ready queue should be minimized.
- V. **Response Time:** Minimising response to a process.
- VI. **Fairness:** Every process should get fair share of the process.

1.2 Benefit of Using Priority Scheduling Round Robin[]:

After analysis of above mentioned algorithms it's made quite clear that the algorithm is said to be efficient if it executes job in a fair manner and result in less avg. waiting and turnaround time.

Now we have established in coming literature survey that time quantum and priority serves as a bottleneck for efficiency of various scheduling algorithm. How two or more algorithm works together based on dynamic properties is another concept. What should be the limit of time quantum for optimality of RR algorithm is perplexed question to encounter. Another question of dynamicity comes which is not easy task to take under consideration. But the solution should satisfy performance goals and should be accepted by operating system.

Researches shows that although the so much efforts has been put into taking response time and other related factors to be first criteria but vast amount of work has been done to select right time quantum and dynamic properties. For these reasons RR and PS scheduler are the first algorithm to choose from.

Also we must add that processes are different types like I/O bound and CPU bound etc. so efficiency of an algorithm is highly subjective and an efficient algorithm is represented by goal it is meant to achieve.

1.3 PROBLEM STATEMENT:

- I. To find out an efficient scheduling algorithm that will provide fair packet transfer with better turnaround time and waiting time etc.
- II. To implement the found algorithm using java platform.

1.4 OBJECTIVES

- I. Implementation of efficient Scheduling Algorithm.
- II. To reduce turnaround time and waiting time as compared to existing algorithms.
- III. Notify the user of packet loss

1.5 METHODOLOGY

- I. To understand what are scheduling algorithms and its types.
- II. To find improvements in the algorithms.
- III. Implementing the proposed Algorithm on Java Platform.
- IV. To find the results in terms of turnaround time and waiting time.

CHAPTER 2

LITERATURE SURVEY

Three important factors play crucial role in scheduling algorithms-

- I. Arrival time of a process present in the ready queue
- II. Priority of a process present in the ready queue
- III. CPU burst time of the processes in the ready queue.

Vast number of scheduling algorithms has been developed.

We categorized the research field accordingly:-

(i)The static time used in the RR scheduling is attempted to make dynamic and

(ii) Research on priority as one of the main factor in determining the efficiency of an algorithm is also given enough thoughts in following papers. Doing this lead to development of more number of applications than that of Static time.

Kiran et al.[10] have presented an algorithm called MDRR. Their suggested algorithm analyses the mean burst time of all the processes in the ready queue. Then it discover out the variance between a process burst time and intended mean burst time. This step is repetitive for all the processes in the ready queue. It is executed for one time slice. After the expiration of time slice next process can be executed

The authors have presented [11] an algorithm ERR which apply dynamic time quantum in RR scheduling. In the planned approach, time quantum is the ratio of the sum of the burst times to the no. of processes.

Kishore et al.[8] gave a scheduling algorithm based on Median based time quantum. All jobs are first ordered in increasing order of their burst time and then RR scheduling is implemented.

Bisht et al. [13] used dynamic time quantum in RR scheduling. But this time it only needs fractional value of time quantum given to the processes for their execution.

In[8], highest burst time and median is used to calculate time quantum and then it is used for execution of processes.

Hagery[5] proposed a method called SRRQT. He considered all the RR scheduling basics. It provides one of the best results used by via dynamic quantum time.

Bhoi et al.[15] presented a paper which provided harmonic mean to calculate time quantum.

Al-Husainy[3] proposed BJF algorithm that takes into consideration the fact that scheduling algorithm are mostly analyse via three properties priority ,arrival time ,CPU burst time. He calculate a factor f that takes into account the fact that the job with highest priority, less CPU burst and came first will be executed first. This algorithm is actually far better than FCFS, SJF, RR, PS etc.

Yaashuwanth and Ramesh[17] presented this paper gave the idea that small size processor can be used which will take the load of other processes to be executed. A new design was proposed in which all the defects of the RR scheduling algorithm were removed. Considered time slice was different for each Process and independent of each task.

Saxena and Agarwal[12] presented this paper shows concept of RR scheduling is elongated by calculating fp i.e. precedence factor and IST i.e. intelligent time slice that measure order and time given to each process for its execution. RR scheduling becomes a lot better.

Varma et al.[14] calculated mean average as time quantum and used bp for precedence to improve RR scheduling further.

Behera and Swain[16] presented a PRRDTQ algorithm. This algorithm gives precedence to the processor with shorter burst time and then applies RR on it. Turnaround time and response time becomes much better. Algorithm performs better than MRR [16] and PBDRR [29] w.r.t. CPU performance.

Jamal and Zubair[4] tried to improve performance of round robin by considering the factors like the waiting time and turnaround time along with number of context switches.

CHAPTER 3

SYSTEM DEVELOPMENT

3.1 Design Overview:

Design involves basic UDP and TCP architecture with the estimation of delay and bandwidth measurements. It was designed by the way of a client uploading the file to the server. Any project is worked out by designing appropriate diagrams. Likewise the design was realized with the help of various diagrams. Two diagrams explain the model of the project for UDP and TCP both. Class diagrams and activity diagrams were also drawn for better understanding of the project. Once the design is over it is we require to decide which software is suitable for the application. The platform used for the project has been explained via describing its major features and common terms used.

Java Technology[19]

Java technology is both a programming language and a platform.

The Java Programming Language

The Java programming language is a high-level language that can be characterized by all of the following buzzwords: Figure 3 shows the Major features of java:

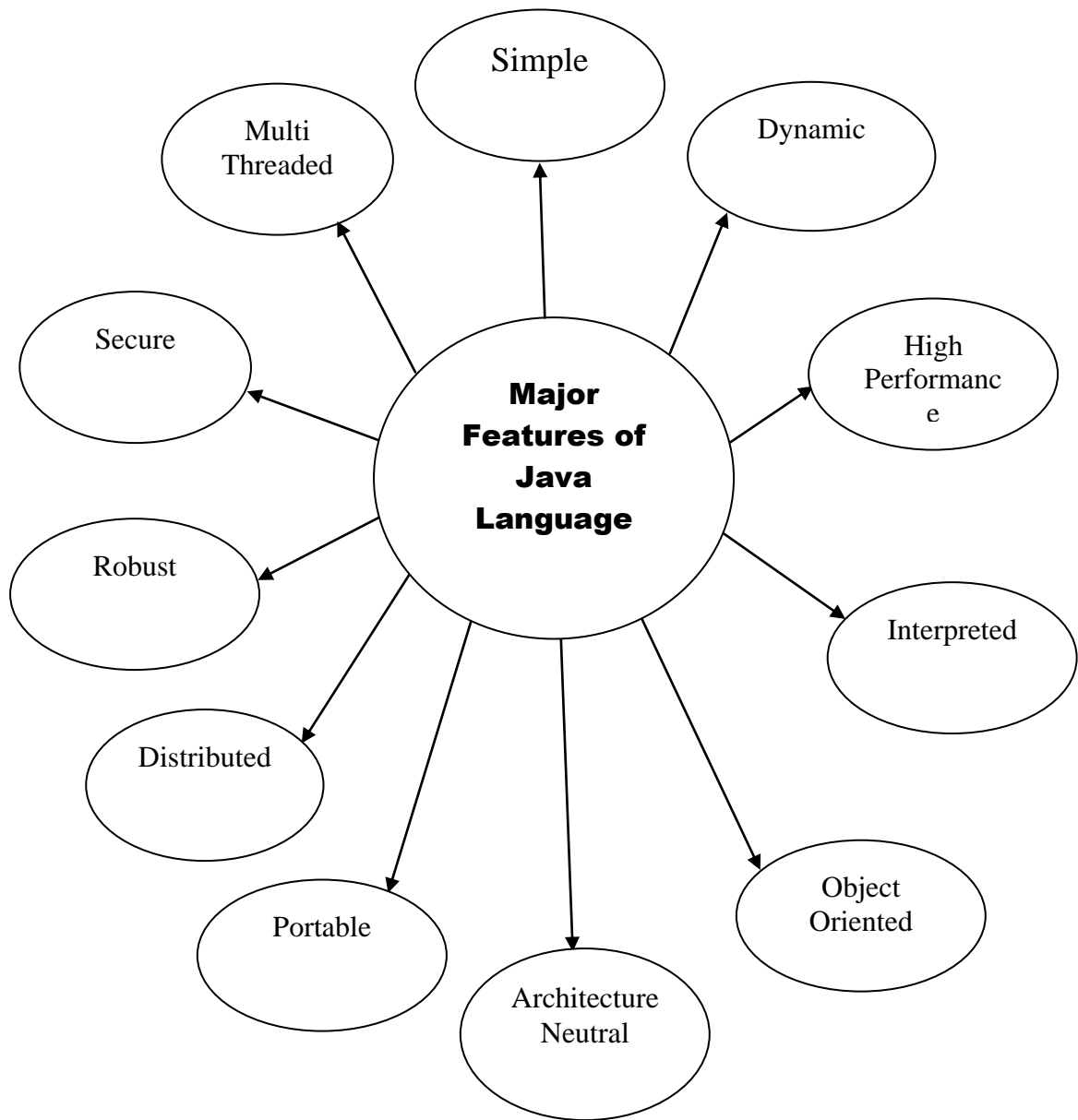


Figure 4: Major Features of Java Language

Guideline on the PC. Aggregation happens just once; understanding happens each time the program is executed. The accompanying figure delineates how this functions.

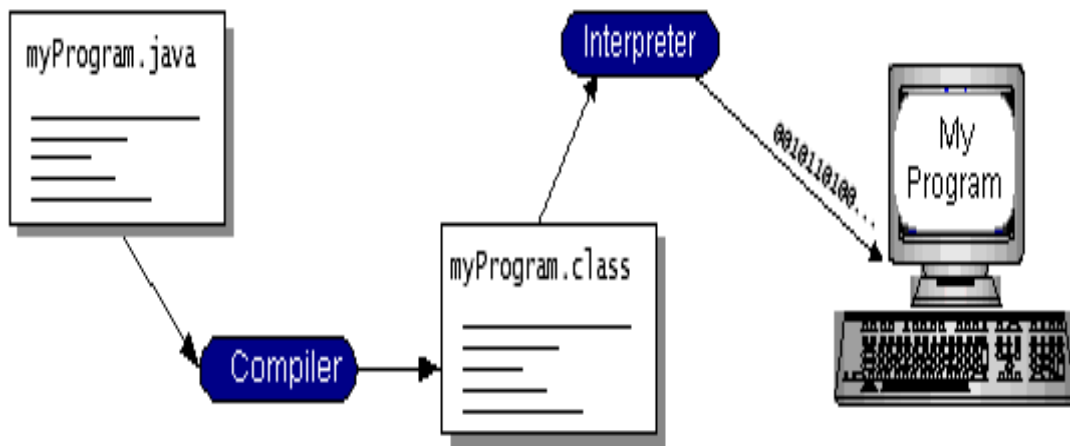


Figure 5: Java Compilation

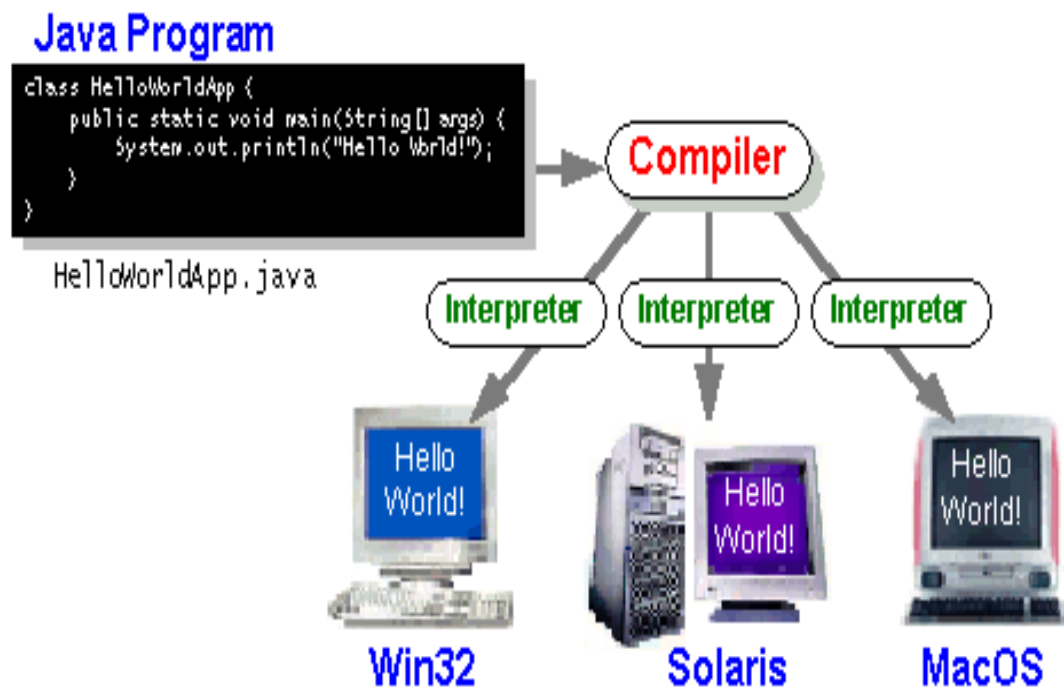


Figure 6: Java Compiler Parts

The Java Platform[20]:

A *platform* is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms. The Java platform has two components:

- The *Java Virtual Machine* (Java VM)
- The *Java Application Programming Interface* (Java API)

You've just been acquainted with the Java VM. It's the base for the Java stage and is ported onto different equipment based stages.

The accompanying figure delineates a program that is running on the Java stage. As the figure appears, the Java API and the virtual machine protect the program from the equipment. Figure 7 shows Java Compiler Parts:

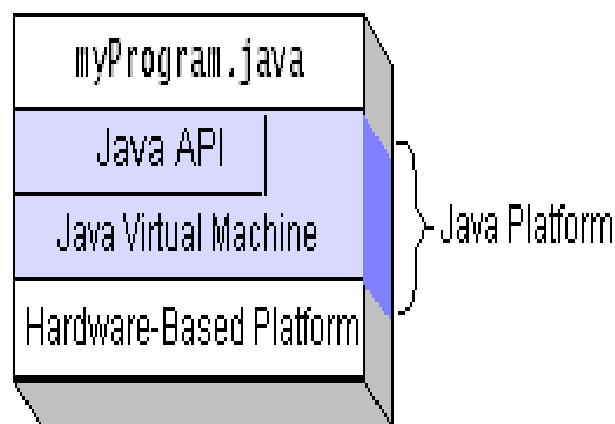


Figure 7: Java Compiler Parts

Local code will be code that after you assemble it, the ordered code keeps running on a particular equipment stage. As a stage free condition, the Java stage can be a bit slower than local code.

In any case, shrewd compilers, very much tuned mediators, and in the nick of time byte code compilers can convey execution near that of local code without undermining compactness.

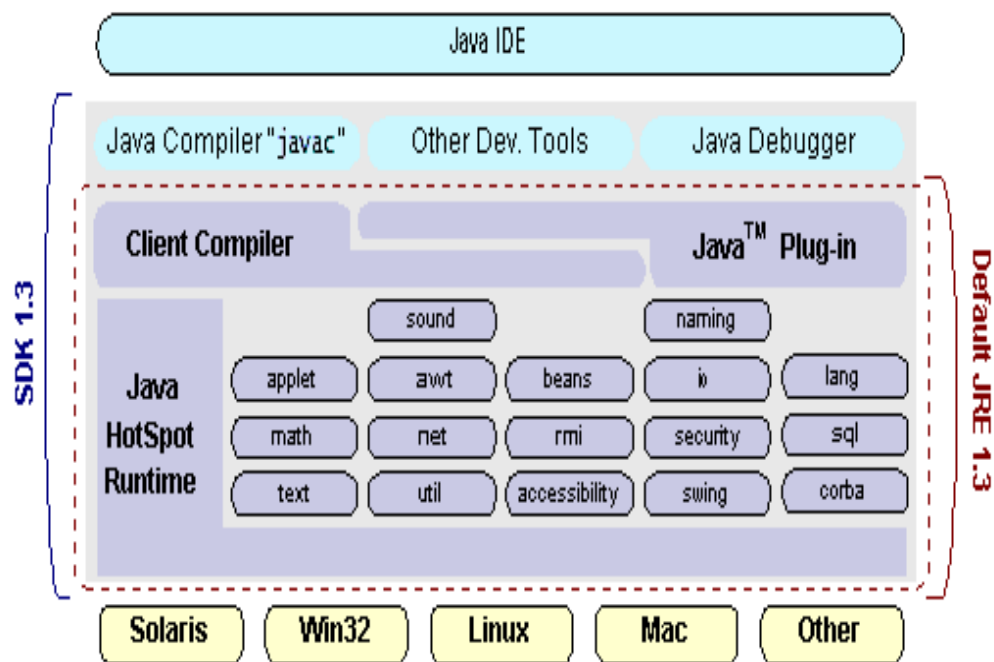


Figure 8: A Java Platform

Why Java:

- Get started quickly
- Write less code
- Write better code
- Develop programs more quickly
- Avoid platform dependencies with 100% Pure Java
- Write once, run anywhere
- Distribute software more easily

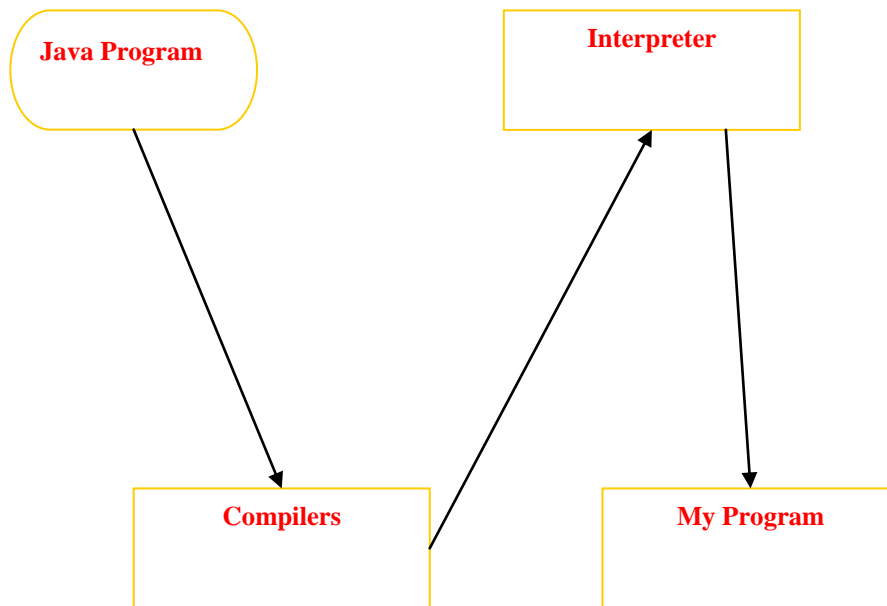


Figure 9: Java Program Compilation

TCP/IP stack[21]

The TCP/IP stack is shorter than the OSI one. Figure 10 shows OSI layer:

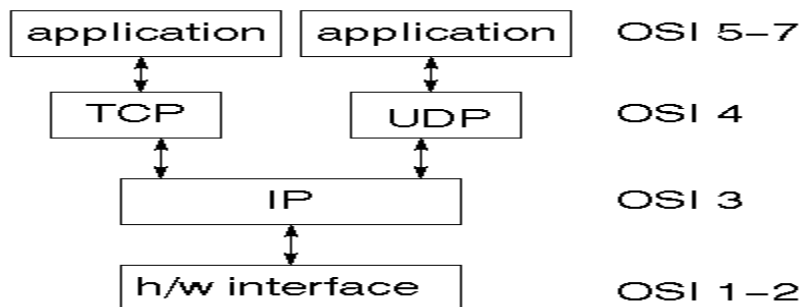


Figure 10: OSI layer

UDP:

UDP is likewise connectionless and temperamental. What it adds to IP is a checksum for the substance of the datagram and port numbers. These are utilized to give a customer/server display - see later.

TCP:

TCP supplies rationale to give a dependable association situated convention above IP. It gives a virtual circuit that two procedures can use to impart.

3.2 Proposed Algorithm Approach:

- Combination of two techniques i.e. Priority and Round robin Algorithm.
- After doing the literature survey three points are major for a process: Burst time, Arrival time, Priority.
- New factor can be calculated by considering all above factors [3].
- In a real time environment, factor f is the more deciding factor than the arrival time and burst time.
- $F=(\text{priority}*\text{priority_ratio})+(\text{burst_time}*\text{burst_time_ratio})+(\text{arrival_time}*\text{arrival_time_ratio})$
- Then CPU burst time can be considered more important factor than the arrival time.
- Therefore, in this equation more weight is given to priority (80%).
- CPU bursts should be 80 % shorter than the time quantum [1] (rule of thumb).
- To make target at least 70%, 0.7 weight is given to the Burst time.
- $F=(\text{priority}*0.8)+(\text{burst time}*0.7)+(\text{Arrival time}*0.2)$
- Based on the priority and arrival time of packet, factor f is given to each packet.
- As performance of RR algorithm is directly proportional to time quantum.
- Small value generates too many context switches otherwise FCFS.
- To impart dynamicity nature Harmonic mean[4] is taken into account as:

$$TQ=n/(1/bt_1+1/bt_2+1/bt_3+1/bt_4+\dots\dots\dots+1/bt_n)$$

n=no. of packets

Bt_i=CPU bt of ith packet.

3.2.1 Proposed Algorithm:

Input: CPU BT, AT, and P_i of each process.

- I. Start the algorithm with all packets ready.
- II. Give every packet a number BTT such that packet with low burst time has high value of BTT.
- III. Give every packet a number ATT such that packet with low burst time has high value of ATT.
- IV. Calculate the factor f ;

$$F=(P_i*0.8)+(BTT_i)+(ATT_i *0.2)$$

P_i = Priority of packets

BTT_i = Burst time of packets

ATT_i = Arrival time of packets

- V. Packet with highest f will be executed first.
- VI. Calculate the time quantum TQ according to harmonic mean;

$$TQ=n/(1/bt_1+1/bt_2+1/bt_3+1/bt_4+.....+1/bt_n)$$

- VII. Assign the TQ to packet($i=1$ to n)
- VIII. Execute the packet in Round Robin fashion until ready queue is empty.
- IX. Stop.

3.3 Module Description: Implemented on Java Platform using Eclipse IDE Version: Oxygen.2 Release (4.7.2)

3.3.1 Module Description for UDP: Figure 11 shows UDP Model Diagram

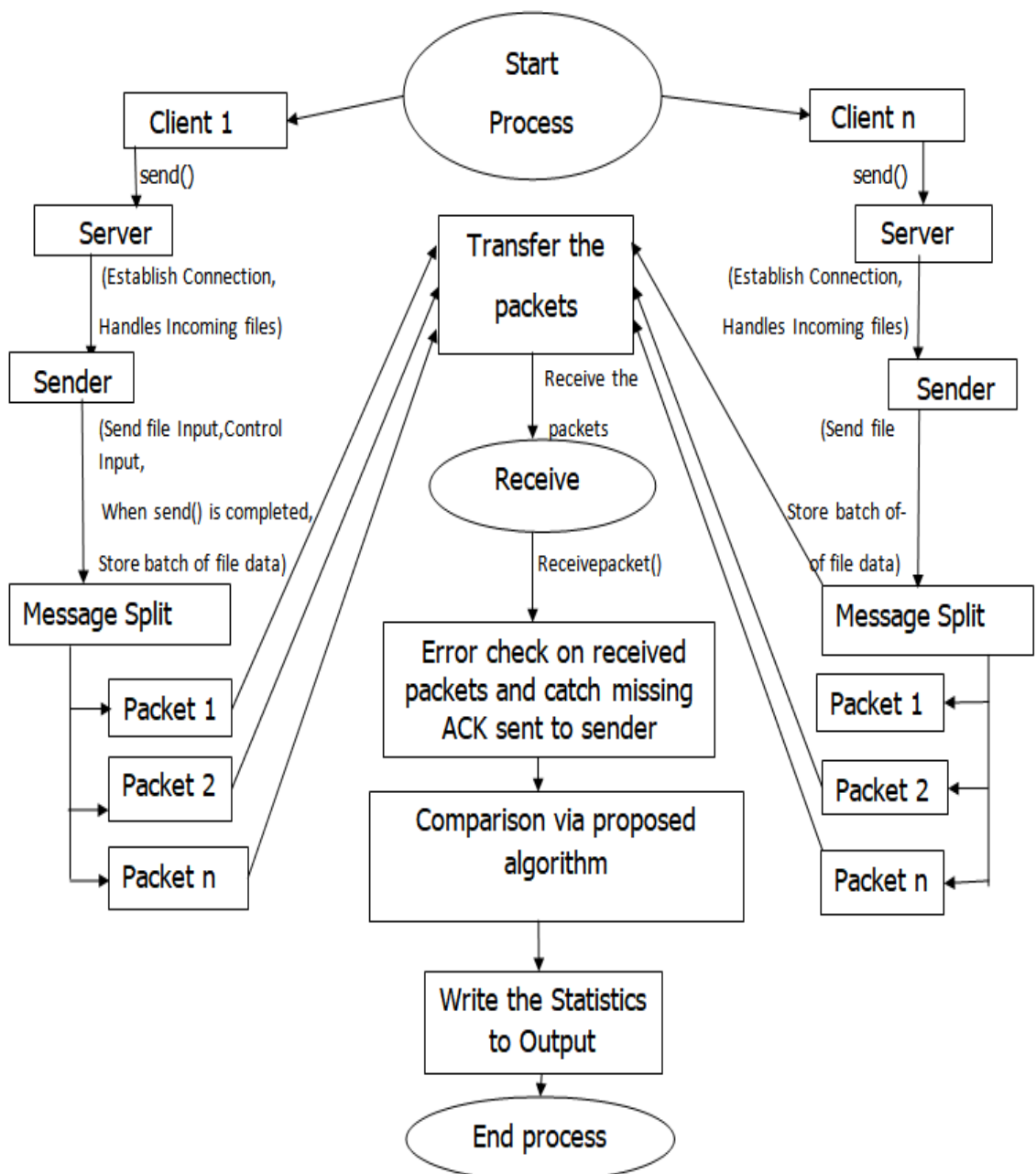


Figure 11: UDP Model Diagram

- Number of Modules=5
- Name of modules:
 - Server
 - Receiver
 - Message
 - Client
 - Sender

Initiation of connection starts from the Server class which prepares the initial packet to be received. It helps in capturing the connection data i.e. IP and Port to open a port for that DatagramSocket. The client for each DatagramSocket is handled by a thread.

A Receiver object is instantiated by the thread for the file being received from a client. The receiver is hoping for an initial packet containing file name and size. It helps in creating necessary buffers and output buffers. After that we keep track of bytes from incoming packets in a continuous loop.

With the help of an abstracted object called a Message packets are deserialized which serves as a container for the packet size, data, and segment ID. This loop continues until we reach bytes received to greater than file length.

Once the user generates the send command in the client side, an initial packet regarding the action of the user and connection details i.e. IP and Port is sent to the Server.

Now to handle the sending we will call the Sender.java class. In this class after setting up the streams, we start by keeping in check our position using current_pos and loop continuously until file length is smaller than current_pos. the packet timeout time is 2sec* the estimated connections delay so far following the original TCP RTT time algorithm. In the next step based on our buffer size read fixed amount of data from the file to be sent and make a Message object that contains the data, length of the data and segment ID. A packet is sent containing the Message object and RTT timing starts. Once we receive acknowledgement or reply that's the end time for RTT. But in some case we don't get

reply due to the time out that means packet has been lost. We just continue with our next packet without retransmitting the packet.

RTT(Round Trip Time):

The time it takes for a bundle to cross the web is a troublesome incentive to compute. In view of the Round-Trip Time (RTT), a clock is set. It is started when the bundle is sent, and is ended when the sender gets an affirmation from the beneficiary demonstrating the bundle's finished conveyance. We can diminish the measure of time squandered amid web correspondence by nearly evaluating this movement time. The first estimation calculation is named after its engineer Jacobson, and has been executed since the introduction of online information exchange.

TCP/IP is a five layer convention suite. It characterizes the trading of transmissions over the web, with the most essential conventions being TCP and IP. The convention most nearly connected with this venture is TCP. This is the vehicle layer in the convention suite and manages a higher level of information transportation, as opposed to the bits of information. In this layer, the whole bundle of data is contemplated. In light of the extent of this parcel, a RTT is evaluated. This is the time needed to make information go from a source to goal and after that get affirmation of the entry. The RTT has numerous applications to TCP. A standout amongst the most critical is the Retransmission Time-Out (RTO) clock. This clock is kept up with every association; what's more, if the sender does not get affirmation upon the termination of the clock, TCP will retransmit the information and restart the clock. This clock depends on the extent of the bundle. Since every bundle might be an alternate size, the clock can't be a settled esteem furthermore, thusly depends on a precise estimation of the RTT. Another use of RTT is computing the quantity of retransmissions of an information bundle, which in a perfect world is kept to zero guaranteeing unbroken correspondence. The time too measures the throughput and good put, which are the quantity of transmissions sent over the system and the quantity of positive transmissions recognized individually. In this manner, it is critical that the RTT be figured absolutely to or as near the genuine incentive as could be allowed.

Jacobson's Algorithm:

The estimation includes a straightforward four-advance process requiring just two info values and settled weights to ascertain the RTT and RTO (showed in figure 5). Basic math is included: expansion, subtraction, and augmentation. Figure 12 shows Jacobson's Algorithm:

1. error = measured RTT - prediction

2. new prediction = old prediction + $\frac{1}{8} \times$ error
= $\frac{7}{8} \times$ old prediction + $\frac{1}{8} \times$ measured RTT

3. new variation = $\frac{3}{4} \times$ old variation + $\frac{1}{4} \times$ abs(error)

4. RTO = prediction + 4 \times variation

Figure 12: Jacobson's Algorithm

3.3.2 Class Diagram for the model: Figure 13 shows UDP class Diagram

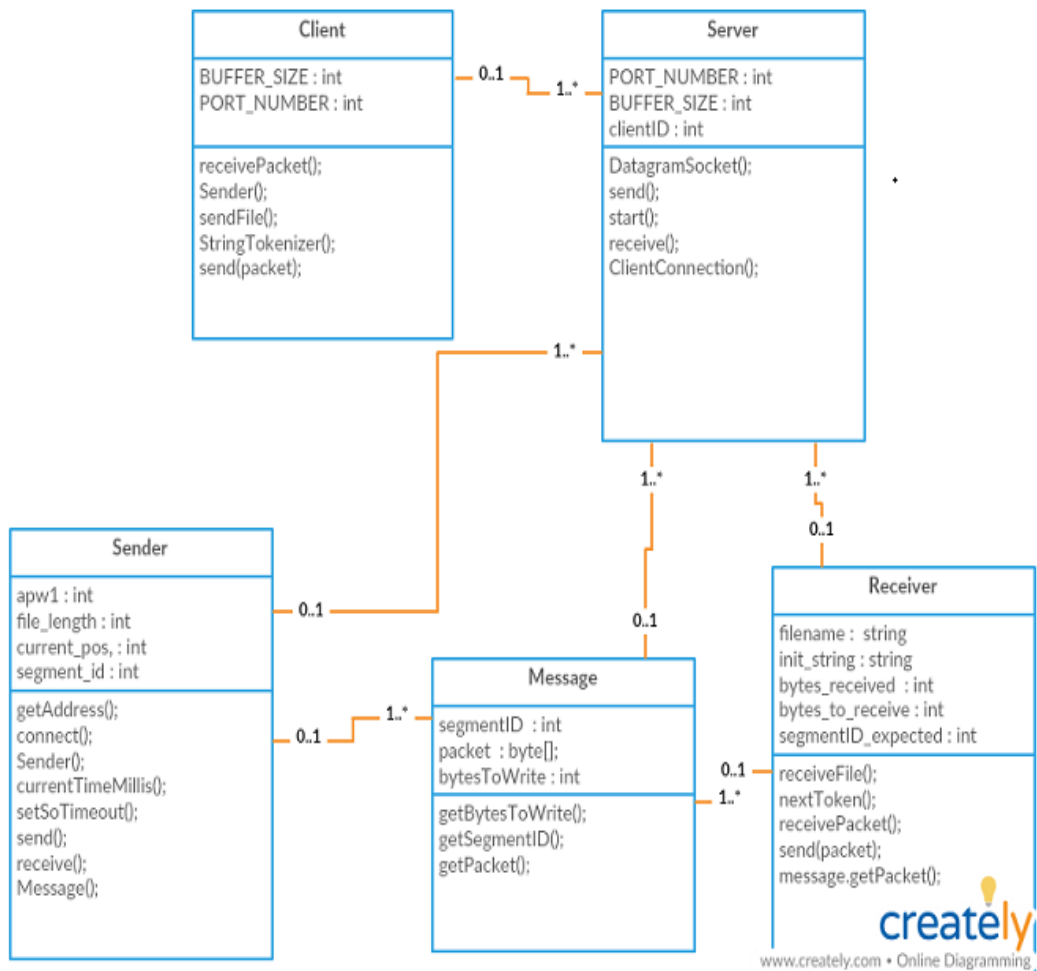


Figure 13: UDP class Diagram

3.3.3 Module Description for TCP: Figure 14 shows TCP Model Diagram

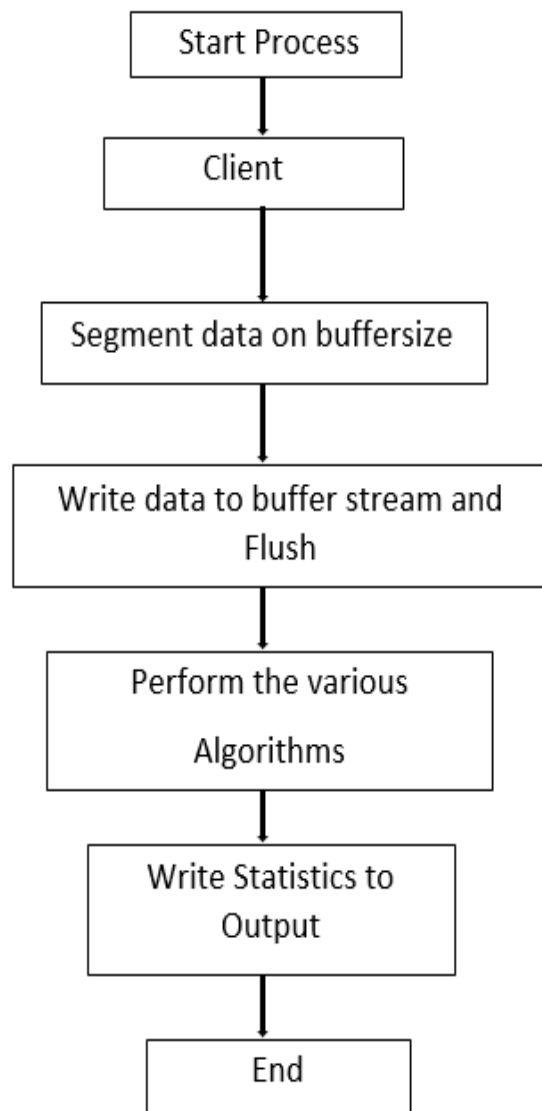


Figure 14: TCP Model Diagram

- Number of Modules=2

- Name of modules:
 - Client
 - Server

TCP architecture follows a basic and a concurrent model and Server-Client architecture. Its different in way that client.java which is a very convenient class (because it allows once to send not just send bytes but also java primitive types) sends file name and length information via a DataOutputStream. Data is segmented based on our buffer size which is 1024. This segment data is written out and is flushed. This operation is timed to RTT. Its timed this way as TCP sends a continuous stream of data.

3.3.4 Class Diagram for the model: Figure 15 shows TCP Class Diagram

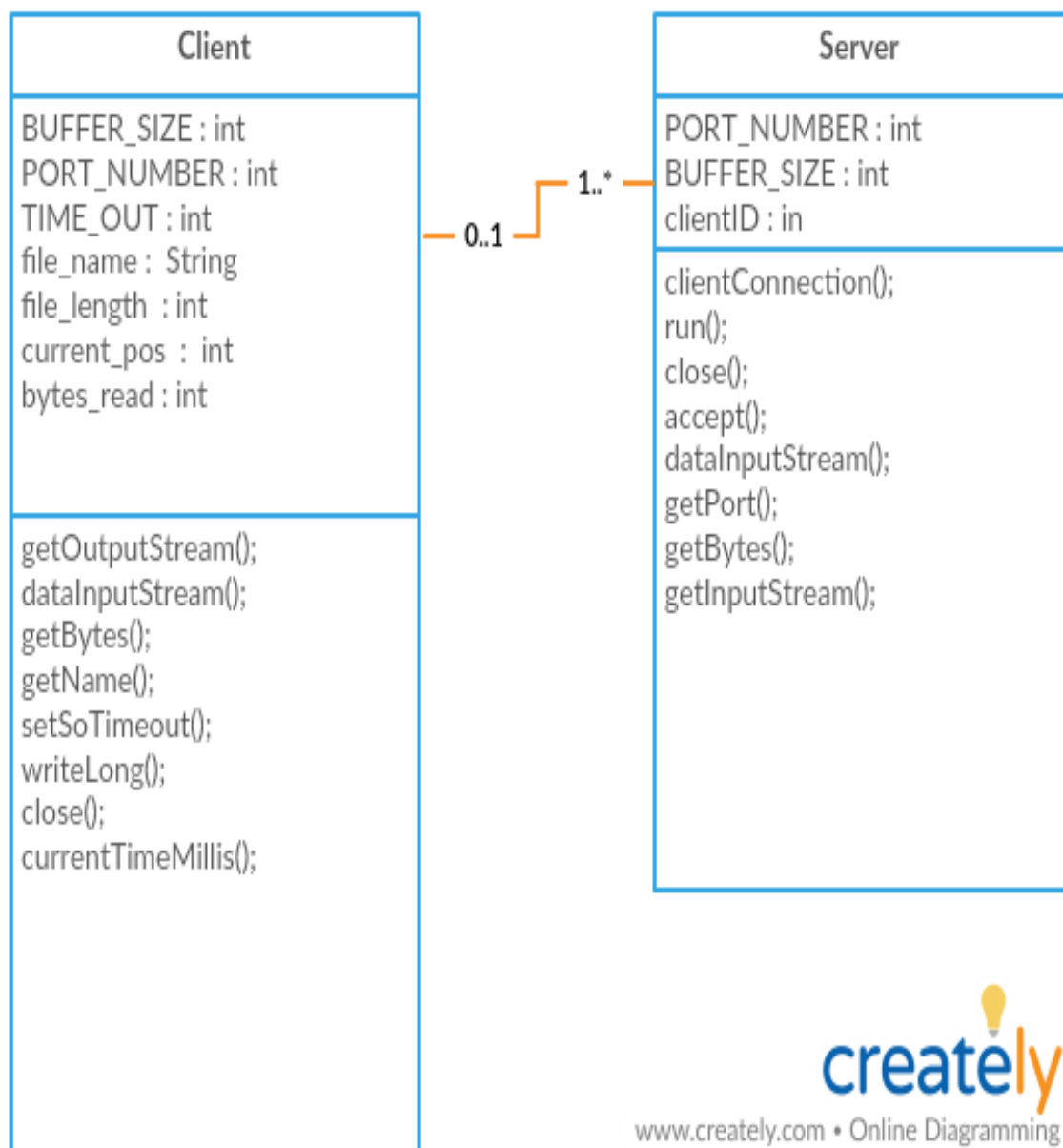


Figure 15: TCP Class Diagram

3.3.5 Activity diagram of the process: Figure 16 shows Activity Diagram

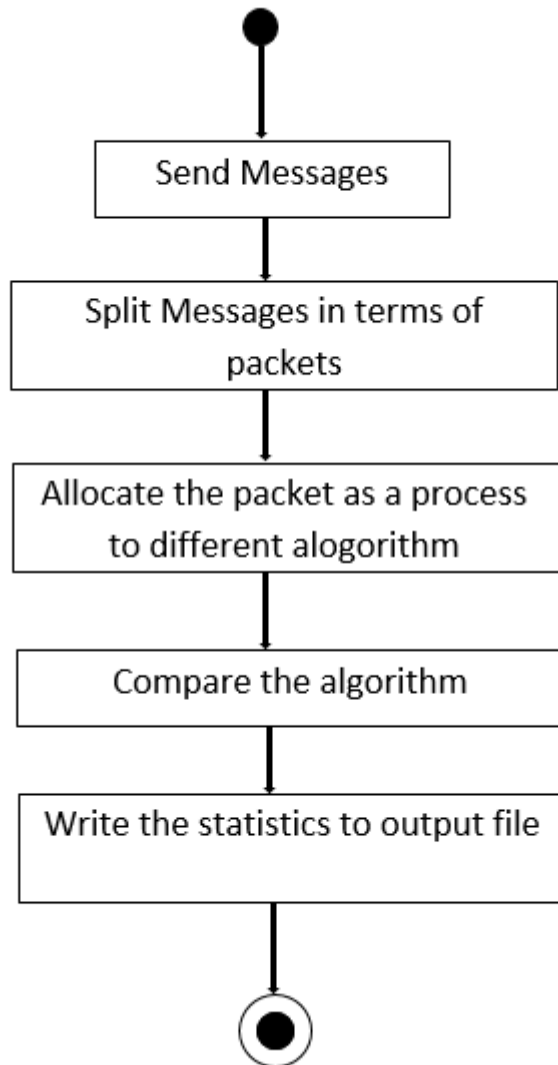


Figure 16: Activity Diagram

CHAPTER 4

EXPERIMENT RESULT AND ANALYSIS

4.1 Test Cases For UDP:

Table 10: Priority Scheduling Algorithm

Process	Burst time	Priority	Waiting Time	Turnaround time
P1	5	9	0	9
P2	3	10	9	11
P3	9	1	11	15
P4	2	5	15	20
P5	4	8	20	23
Average	-		11.0	15.6

Table 11: Proposed Algorithm

Process	Burst time	Arrival Time	Priority	Waiting Time	Turnaround time
P1	5	2	9	0.0	3.0
P2	3	3	10	12.17	17.17
P3	9	4	1	13.78	17.58
P4	2	1	5	10..17	12.17
P5	4	5	8	14.0	23
Average	-			9.98	14.58

Table 12: Round Robin Scheduling

Process	Burst time	Time Quantum	Waiting Time	Turnaround time
P1	5	3	11	16
P2	3	3	3	6
P3	9	3	14	23
P4	2	3	9	11
P5	4	3	16	20
Average	-		10.6	15.2

Table 13: Proposed Algorithm

Process	Burst time	Arrival Time	Priority	Waiting Time	Turnaround time
P1	5	2	9	0.0	3.0
P2	3	3	10	12.17	17.17
P3	9	4	1	13.78	17.58
P4	2	1	5	10..17	12.17
P5	4	5	8	14.0	23
Average	-			9.98	14.58

4.2 Experimental Result And Analysis for UDP:

- Figure 17 shows Transferring the packets and receiving the acknowledgement:

The screenshot displays an IDE with two windows. The left window shows the source code for a UDP client, and the right window shows the console output of the application.

```
Client.java Message.java Receiverjava Senderjava Serverjava
14         message = (Message) deserialize(received_packet.getData());
15     } catch (ClassNotFoundException ex) {
16         System.out.println("*** Message packet failed. ***");
17     }
18     } while (message.getSegmentID() != segmentID_expected);
19
20     segmentID_expected++;
21
22     // handles the last byte segmentID size .getBytesToWrite()
23     file_os.write(message.getPacket(), 0, message.getBytesToWrite());
24     System.out.println("Received segmentID " + message.getSegmentID() + " | fil
25     bytes_received = bytes_received + message.getPacket().length;
26
27     // Send ACK message (which is the segment id)
28     String ACK = Integer.toString(message.getSegmentID());
29     send(init_packet.getAddress(), init_packet.getPort(), (ACK).getBytes());
30 }
31 System.out.println("File transfer complete.");
32 file_os.close();
33 }
34
35 private DatagramPacket receivePacket() throws IOException {
36     buffer = new byte[BUFFER_SIZE];
37     DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
```

```
Client (1) [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (20-Mar-2018, 8:34:14 AM)
COMMANDS: send *file path*
          example: send C:/data.txt
udp> send C:\Users\HP\Desktop\eclipse\data.txt
*** Filename: data.txt ***
*** Bytes to send: 4533 ***

~~~~~ Sending segment 0 with 1024 byte payload.
+++ Received ACK to segment0 Delay: 1.0ms
~~~~~ Sending segment 1 with 1024 byte payload.
+++ Received ACK to segment1 Delay: 1.0ms
~~~~~ Sending segment 2 with 1024 byte payload.
+++ Received ACK to segment2 Delay: 0.0ms
~~~~~ Sending segment 3 with 1024 byte payload.
+++ Received ACK to segment3 Delay: 0.0ms
~~~~~ Sending segment 4 with 437 byte payload.
```

Figure 17: UDP Packet Transfer

- Figure 18 shows comparing the basic algorithms with proposed algorithm:

```

14 public class Client {
15     private static byte[] buffer;
16     private static final int BUFFER_SIZE = 1024;
17     private static final int PORT_NUMBER = 26433; // set port number to 2000 +
18     private static DatagramSocket socket;
19     private static BufferedReader stdin;
20     private static StringTokenizer userInput;
21     private static DatagramPacket initPacket, packet;
22     public static void main(String[] args) throws IOException {
23         socket = new DatagramSocket();
24         InetAddress address = InetAddress.getByName("localhost");
25         buffer = new byte[BUFFER_SIZE];
26         stdin = new BufferedReader(new InputStreamReader(System.in));
27         System.out.println("COMMANDS: send *file path*");
28         System.out.println("\t example: send C:/data.txt");
29         System.out.print("\udp> ");
30         String selectedAction = stdin.readLine();
31         userInput = new StringTokenizer(selectedAction);
32         try {
33             if (userInput.nextToken().equalsIgnoreCase("send")) {
34                 // send 'Send' command to server
35                 packet = new DatagramPacket((selectedAction).getBytes(), (selectedAction
36                 socket.send(packet);
37
38                 File theFile = new File(userInput.nextToken());
39
40                 initPacket = receivePacket();
41
42                 // create object to handle out going file
43                 Sender fileHandler = new Sender(socket, initPacket);
44                 fileHandler.sendFile(theFile);
45             }
46         } catch (Exception e) {
47             System.err.println("Not valid input " + e.toString());
48         }
49         socket.close();
50     }
51     private static DatagramPacket receivePacket() throws IOException {
52         DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
53         socket.receive(packet);
54         return packet;
55     }
56 }

```

```

Client (1) [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (20-Mar-2018, 8:34:14 AM)
2.SHORTEST JOB FIRST method(non-preemptive)
3.SHORTEST JOB FIRST method(preemptive)
4.ROUND ROBIN method
5.PRIORITY method (non-preemptive)
6.PRIORITY method (preemptive)
7.EXIT
ENTER YOUR CHOICE
5
ENTER burst time for each process: p1
5
ENTER burst time for each process: p2
3
ENTER burst time for each process: p3
9
ENTER burst time for each process: p4
2
ENTER burst time for each process: p5
4
Enter the priority for p1
9
Enter the priority for p2
10
Enter the priority for p3
1
Enter the priority for p4
5
Enter the priority for p5
8
individual waiting time for process p3 is0
individual waiting time for process p4 is9
individual waiting time for process p5 is11
individual waiting time for process p1 is15
individual waiting time for process p2 is20

average waiting time is:11.0
turnaround time for process p3 is9
turnaround time for process p4 is11
turnaround time for process p5 is15
turnaround time for process p1 is20
turnaround time for process p2 is23

average turn-around time is:15.6

```

Figure 18: UDP Proposed Algorithm

- Figure 19 shows Calculating turnaround and waiting time for any algorithm(Round Robin)

```

14 public class Client {
15     private static byte[] buffer;
16     private static final int BUFFER_SIZE = 1024;
17     public static final int PORT_NUMBER = 26433; // set port number to 2000 +
18     private static DatagramSocket socket;
19     private static BufferedReader stdin;
20     private static StringTokenizer userInput;
21     private static DatagramPacket initPacket, packet;
22     public static void main(String[] args) throws IOException {
23         socket = new DatagramSocket();
24         InetAddress address = InetAddress.getByName("localhost");
25         buffer = new byte[BUFFER_SIZE];
26         stdin = new BufferedReader(new InputStreamReader(System.in));
27         System.out.println("COMMANDS: send *file path*");
28         System.out.println("\t example: send C:/data.txt");
29         System.out.print("udp> ");
30         String selectedAction = stdin.readLine();
31         userInput = new StringTokenizer(selectedAction);
32         try {
33             if (userInput.nextToken().equalsIgnoreCase("send")) {
34                 // send 'Send' command to server
35                 packet = new DatagramPacket((selectedAction).getBytes(),(selectedAction
36                 socket.send(packet);
37
38                 File theFile = new File(userInput.nextToken());
39
40                 initPacket = receivePacket();
41
42                 // create object to handle out going file
43                 Sender fileHandler = new Sender(socket, initPacket);
44                 fileHandler.sendFile(theFile);
45             }
46         } catch (Exception e) {
47             System.err.println("Not valid input " + e.toString());
48         }
49         socket.close();
50     }
51     private static DatagramPacket receivePacket() throws IOException {
52         DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

```

```

Client (1) [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (20-Mar-2018, 8:34:14 AM)
Which algorithm you want to compare the efficient algorithm to
1.FCFS method
2.SHORTEST JOB FIRST method(non-preemptive)
3.SHORTEST JOB FIRST method(preemptive)
4.ROUND ROBIN method
5.PRIORITY method (non-preemptive)
6.PRIORITY method (preemptive)
7.EXIT
ENTER YOUR CHOICE
4
ENTER burst time for each process: p1
5
ENTER burst time for each process: p2
3
ENTER burst time for each process: p3
14
ENTER burst time for each process: p4
9
ENTER burst time for each process: p5
16
enter time quantum
3

waiting time for p1 is:11
waiting time for p2 is:3
waiting time for p3 is:14
waiting time for p4 is:9
waiting time for p5 is:16

average waiting time is:10.6
turn around time is:16
turn around time is:6
turn around time is:23
turn around time is:11
turn around time is:20

average turn-around time is:15.2

```

Figure 19: Calculating Time

- Figure 20 shows finally calculating average waiting time and turnaround time for proposed algorithm:

```

22 socket = new DatagramSocket();
23 InetAddress address = InetAddress.getByAddress("localhost");
24 buffer = new byte[BUFFER_SIZE];
25 stdin = new BufferedReader(new InputStreamReader(System.in));
26 System.out.println("COMMANDS: send *file path*");
27 System.out.println("\t example: send C:/data.txt");
28 System.out.print("udp> ");
29 String selectedAction = stdin.readLine();
30 userInput = new StringTokenizer(selectedAction);
31 try {
32     if (userInput.nextToken().equalsIgnoreCase("send")) {
33         // send 'Send' command to server
34         packet = new DatagramPacket((selectedAction).getBytes(),(selectedAction).getBytes().length);
35         socket.send(packet);
36
37         File theFile = new File(userInput.nextToken());
38
39         initPacket = receivePacket();
40
41         // create object to handle out going file
42         Sender fileHandler = new Sender(socket, initPacket);
43         fileHandler.sendFile(theFile);
44     }
45 } catch (Exception e) {
46     System.err.println("Not valid input " + e.toString());
47 }
48 socket.close();
49 }
50 private static DatagramPacket receivePacket() throws IOException {
51     DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
52     socket.receive(packet);
53     return packet;
54 }
55 }

```

```

<terminated> Client (1) [Java Application] C:\Program Files\Java\jre-10\bin\javaw.exe (10-May-2018, 8:4
Harmonic Time Quantum calculated is:3.5856573705179287

Sequence of the packets will be:>
1
0
4
3
2

waiting time for p1 is:0.0
waiting time for p2 is:12.171314741035857
waiting time for p3 is:13.585657370517929
waiting time for p4 is:10.171314741035857
waiting time for p5 is:14.0

average waiting time is:9.985658

turn around time is:3.0
turn around time is:17.171314741035857
turn around time is:17.58565737051793
turn around time is:12.171314741035857
turn around time is:23.0

average turn-around time is:14.585657

Average Waiting Time: 9.985658 | Average Turnaround Time: 14.585657

```

Figure 20: Calculation Time For Proposed Algorithm

4.3 Test Cases For UDP:

Table 10: Priority Scheduling Algorithm

Process	Burst time	Priority	Waiting Time	Turnaround time
P1	5	9	0	9
P2	3	10	9	11
P3	9	1	11	15
P4	2	5	15	20
P5	4	8	20	23
Average	-		11.0	15.7

Table 11: Proposed Algorithm

Process	Burst time	Arrival Time	Priority	Waiting Time	Turnaround time
P1	5	2	9	0.0	3.0
P2	3	3	10	12.17	17.17
P3	9	4	1	13.78	17.58
P4	2	1	5	10.17	12.17
P5	4	5	8	14.0	23
Average	-			9.98	14.59

Table 12: Round Robin Scheduling

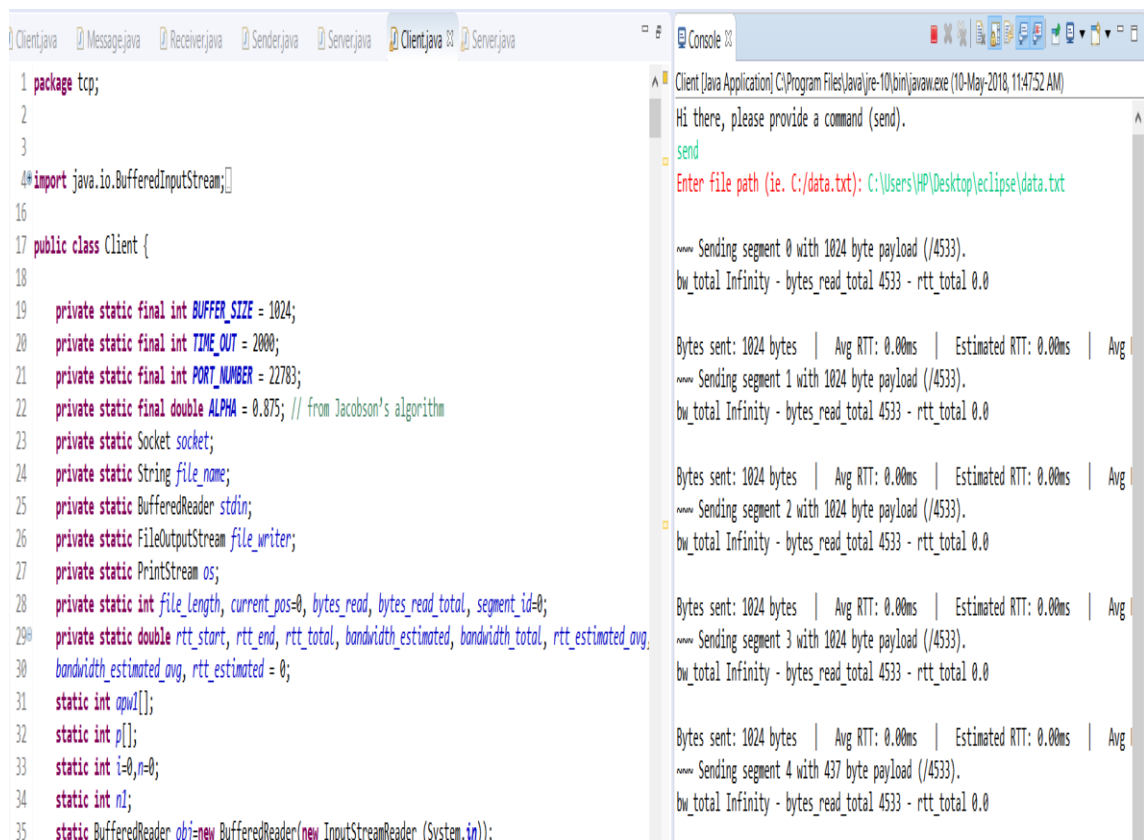
Process	Burst time	Time Quantum	Waiting Time	Turnaround time
P1	5	3	11	16
P2	3	3	3	6
P3	9	3	14	23
P4	2	3	9	11
P5	4	3	16	20
Average	-		10.6	15.3

Table 13: Proposed Algorithm

Process	Burst time	Arrival Time	Priority	Waiting Time	Turnaround time
P1	5	2	9	0.0	3.0
P2	3	3	10	12.17	17.17
P3	9	4	1	13.78	17.58
P4	2	1	5	10..17	12.17
P5	4	5	8	14.0	23
Average	-			9.98	14.58

4.4 Experimental Result And Analysis for TCP:

- Figure 21 shows Transferring the packets and receiving the acknowledgement:



The screenshot displays an IDE with a Java source file named 'Client.java' and a corresponding console window. The code defines a 'Client' class with various static variables and methods for handling network communication. The console output shows the execution of the program, including user input and network statistics for each segment sent.

```
1 package tcp;
2
3
4 import java.io.BufferedReader;
16
17 public class Client {
18
19     private static final int BUFFER_SIZE = 1024;
20     private static final int TIME_OUT = 2000;
21     private static final int PORT_NUMBER = 22783;
22     private static final double ALPHA = 0.875; // from Jacobson's algorithm
23     private static Socket socket;
24     private static String file_name;
25     private static BufferedReader stdin;
26     private static FileOutputStream file_writer;
27     private static PrintStream os;
28     private static int file_length, current_pos=0, bytes_read, bytes_read_total, segment_id=0;
29     private static double rtt_start, rtt_end, rtt_total, bandwidth_estimated, bandwidth_total, rtt_estimated_avg,
30     bandwidth_estimated_avg, rtt_estimated = 0;
31     static int opul[];
32     static int p[];
33     static int i=0, n=0;
34     static int n1;
35     static BufferedReader obj=new BufferedReader(new InputStreamReader (System.in));
```

Console Output:

```
Client [Java Application] C:\Program Files\Java\jre-10\bin\javaw.exe (10-May-2018, 11:47:52 AM)
Hi there, please provide a command (send).
send
Enter file path (ie. C:/data.txt): C:\Users\HP\Desktop\eclipse\data.txt

~~~ Sending segment 0 with 1024 byte payload (/4533).
bw_total Infinity - bytes_read_total 4533 - rtt_total 0.0

Bytes sent: 1024 bytes | Avg RTT: 0.00ms | Estimated RTT: 0.00ms | Avg
~~~ Sending segment 1 with 1024 byte payload (/4533).
bw_total Infinity - bytes_read_total 4533 - rtt_total 0.0

Bytes sent: 1024 bytes | Avg RTT: 0.00ms | Estimated RTT: 0.00ms | Avg
~~~ Sending segment 2 with 1024 byte payload (/4533).
bw_total Infinity - bytes_read_total 4533 - rtt_total 0.0

Bytes sent: 1024 bytes | Avg RTT: 0.00ms | Estimated RTT: 0.00ms | Avg
~~~ Sending segment 3 with 1024 byte payload (/4533).
bw_total Infinity - bytes_read_total 4533 - rtt_total 0.0

Bytes sent: 1024 bytes | Avg RTT: 0.00ms | Estimated RTT: 0.00ms | Avg
~~~ Sending segment 4 with 437 byte payload (/4533).
bw_total Infinity - bytes_read_total 4533 - rtt_total 0.0
```

Figure 21: UDP Packet Transfer

- Figure 18 shows comparing the basic algorithms with proposed algorithm:

```

1 package tcp;
2
3
4*import java.io.BufferedReader;
16
17 public class Client {
18
19     private static final int BUFFER_SIZE = 1024;
20     private static final int TIME_OUT = 2000;
21     private static final int PORT_NUMBER = 22783;
22     private static final double ALPHA = 0.875; // from Jacobson's algorithm
23     private static Socket socket;
24     private static String file_name;
25     private static BufferedReader stdin;
26     private static FileOutputStream file_writer;
27     private static PrintStream os;
28     private static int file_length, current_pos=0, bytes_read, bytes_read_total, segment_id=0;
29* private static double rtt_start, rtt_end, rtt_total, bandwidth_estimated, bandwidth_total, rtt_estimated_avg,
30     bandwidth_estimated_avg, rtt_estimated = 0;
31     static int apw1[];
32     static int p[];
33     static int i=0,n=0;
34     static int n1;
35     static BufferedReader obj=new BufferedReader(new InputStreamReader(System.in));
36* public static void main(String[] args) throws IOException {
37     try {
38         socket = new Socket("localhost", PORT_NUMBER);
39         stdin = new BufferedReader(new InputStreamReader(System.in));
40     } catch (Exception e) {
41         System.err.println("Cannot connect to the server, try again later.");
42         System.exit(1);
43     }
44     System.out.println("Hi there, please provide a command (send).");
45     os = new PrintStream(socket.getOutputStream());

```

```

Client [Java Application] C:\Program Files\Java\jre-10\bin\javaw.exe (10-May-2018, 11:47:52 AM)
5
ENTER burst time for each process: p2
3
ENTER burst time for each process: p3
9
ENTER burst time for each process: p4
2
ENTER burst time for each process: p5
4
Enter the priority for p1
9
Enter the priority for p2
10
Enter the priority for p3
1
Enter the priority for p4
5
Enter the priority for p5
8
individual waiting time for process p3 is0
individual waiting time for process p4 is9
individual waiting time for process p5 is11
individual waiting time for process p1 is15
individual waiting time for process p2 is20
average waiting time is:11.0
turnaround time for process p3 is9
turnaround time for process p4 is11
turnaround time for process p5 is15
turnaround time for process p1 is20
turnaround time for process p2 is23
average turn-around time is:15.6

```

Figure 22: UDP Proposed Algorithm

- Figure 23 shows comparing the basic algorithms with proposed algorithm:

```

Client.java 33  Message.java  Receiver.java  Sender.java  Server.java
22  socket = new DatagramSocket();
23  InetAddress address = InetAddress.getByAddress("localhost");
24  buffer = new byte[BUFFER_SIZE];
25  stdin = new BufferedReader(new InputStreamReader(System.in));
26  System.out.println("COMMANDS: send *file path*");
27  System.out.println("\t example: send C:/data.txt");
28  System.out.print("udp> ");
29  String selectedAction = stdin.readLine();
30  userInput = new StringTokenizer(selectedAction);
31  try {
32      if (userInput.nextToken().equalsIgnoreCase("send")) {
33          // send 'Send' command to server
34          packet = new DatagramPacket(selectedAction.getBytes(), selectedAction.getBytes().length, address, 8080);
35          socket.send(packet);
36
37          File theFile = new File(userInput.nextToken());
38
39          initPacket = receivePacket();
40
41          // create object to handle out going file
42          Sender fileHandler = new Sender(socket, initPacket);
43          fileHandler.sendFile(theFile);
44      }
45  } catch (Exception e) {
46      System.err.println("Not valid input " + e.toString());
47  }
48  socket.close();
49  }
50  private static DatagramPacket receivePacket() throws IOException {
51      DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
52      socket.receive(packet);
53      return packet;
54  }
55  }

```

```

<terminated> Client (1) [Java Application] C:\Program Files\Java\jre-10\bin\javaw.exe (10-May-2018, 8:4
Harmonic Time Quantum calculated is:3.5856573705179287

Sequence of the packets will be:>
1
0
4
3
2

waiting time for p1 is:0.0
waiting time for p2 is:12.171314741035857
waiting time for p3 is:13.585657370517929
waiting time for p4 is:10.171314741035857
waiting time for p5 is:14.0

average waiting time is:9.985658

turn around time is:3.0
turn around time is:17.171314741035857
turn around time is:17.58565737051793
turn around time is:12.171314741035857
turn around time is:23.0

average turn-around time is:14.585657

Average Waiting Time: 9.985658 | Average Turnaround Time: 14.585657

```

Figure 23: Calculating Time

- Figure 20 shows finally calculating average waiting time and turnaround time for proposed algorithm:

```

1 package tcp;
2
3
4 import java.io.BufferedReader;
5
6
7 public class Client {
8
9     private static final int BUFFER_SIZE = 1024;
10    private static final int TIME_OUT = 2000;
11    private static final int PORT_NUMBER = 22783;
12    private static final double ALPHA = 0.875; // from Jacobson's algorithm
13
14    private static Socket socket;
15    private static String file_name;
16    private static BufferedReader stdin;
17    private static FileOutputStream file_writer;
18    private static PrintStream os;
19
20    private static int file_length, current_pos=0, bytes_read, bytes_read_total, segment_id=0;
21
22    private static double rtt_start, rtt_end, rtt_total, bandwidth_estimated, bandwidth_total, rtt_estimated_avg,
23    bandwidth_estimated_avg, rtt_estimated = 0;
24
25    static int apw1[];
26
27    static int p[];
28
29    static int i=0, n=0;
30
31    static int n1;
32
33    static BufferedReader obj=new BufferedReader(new InputStreamReader(System.in));
34
35    public static void main(String[] args) throws IOException {
36
37        try {
38            socket = new Socket("localhost", PORT_NUMBER);
39            stdin = new BufferedReader(new InputStreamReader(System.in));
40        } catch (Exception e) {
41            System.err.println("Cannot connect to the server, try again later.");
42            System.exit(1);
43        }
44        System.out.println("Hi there, please provide a command (send).");
45        os = new PrintStream(socket.getOutputStream());

```

```

<terminated> Client [Java Application] C:\Program Files\Java\jre-10\bin\javaw.exe (10-May-2018, 11:47:52 AM)
Harmonic Time Quantum calculated is:3.5856573705179287
Sequence of the packets will be:>
1
0
4
3
2
waiting time for p1 is:0.0
waiting time for p2 is:12.171314741035857
waiting time for p3 is:13.585657370517929
waiting time for p4 is:10.171314741035857
waiting time for p5 is:14.0
|
average waiting time is:9.985658
turn around time is:3.0
turn around time is:17.171314741035857
turn around time is:17.58565737051793
turn around time is:12.171314741035857
turn around time is:23.0
average turn-around time is:14.585657
Total Avg Estimated RTT: 0.00ms | Total Avg Estimated BW: InfinityKB/s
*** File transfer complete...

```

Figure 24: Calculation Time For Proposed Algorithm

CHAPTER 5

CONCLUSION

Different algorithms are compared on the basis of qualitative parameters in this report. Analysis is done by way of comparing several of scheduling algorithms with certain parameters. If either the AWT or ATT is required to be shortest, the SJF algorithm can be used. Also if either avg. CPU utilization or avg. throughput is to be minimized the FCFS can be used. Algorithm which is fair to every process is Round robin. The highly efficient and low scheduling overhead is achieved with multilevel feedback queue. Any type of simulation of any algorithm has very limited accuracy therefore a lot of effort and hard work has been put to make the algorithm fair and starvation free. This research establish that

- I. Fairness and starvation free qualities of execution of processes required a lot of effort.
- II. Dynamic behaviour of the time quantum should be considered if the goal is to minimize turnaround time and response time.
- III. We are trying to make new algorithm out of RR and PS by way of calculating time quantum and executing processes in that manner.
- IV. Highest priority is given highest chance.

Future Work: The future work includes implementing this algorithm in real time routers. Obviously it will need more research in the field of networks and existing applications of OS based scheduling algorithms in the packet transmission from any source to destination.

REFERENCES

1. A. Silberschatz, Galvin, and G. Gagne, "Operating System Concepts", 2009, Wiley Inc.
2. A. S. Tanenbaum, "Modern Operating Systems" 2009 Prentice Hall (book).
3. Al-Husainy, M.A.F., "Best-job-first CPU scheduling algorithm" 2007, Inform. Technol. J., Volume 6: No. 2, pp. 288-293
4. Adeeba Jamal and Aiman Zubair "A Varied Round Robin Approach using Harmonic Mean of the Remaining Burst Time of the Processes" 3rd International IT Summit Confluence 2012
5. Mohammed Abdullah Hassan Al-Hagery "A selective quantum of time for round robin algorithm to increase CPU utilization", 2011,IJCIS , Vol.3 No. 2, 2011
6. Abbas Noon¹, Ali Kalakech², Seifedine Kadry "A New Round Robin Based Scheduling Algorithm" for Operating Systems: Dynamic Quantum Using the Mean Average May 2011, IJCSI, Vol. 8,
7. Sanjay Kumar Panda, An Effective Round Robin Algorithm using Min-Max Dispersion Measure, Vol. 2, No. 12, December 2011.
8. Lalit Kishore, Dinesh Goyal, "Time Quantum Based Improved Scheduling Algorithm", Volume 3, Issue of IJARCSSE. ISSN: 2277 128X, 4, April, 2013.
9. KN Rout, G.Das,B.M. Sahoo,A.K. Agrawalla, ,Improving Average Waiting Time Using Dyanamic Time Quantum,IRAJ,2013.
10. Kiran, PolinatiVinodBabu, B.B Murali Krishna, Optimizing CPU Scheduling for Real Time Application Using Mean-Difference Round Robin (MDRR) Algorithm, Springer, 2014
11. KumkumPattanayak, PayalPansari, SubhashreePatra, SubhashreeMohapatra, "An Enhanced Round Robin (ERR) Scheduling Algorithm" using Dynamic Time Quantum,IJSAA,2013
12. HimanshiSaxena, Prashant Agarwal, Design and Performance Evaluation of Precedence Scheduling Algorithm with Intelligent Service Time , ICCMS, 2012
13. AashnaBisht, Mohd Abdul Ahad, Sielvie Sharma, "Enhanced Round Robin Algorithm For Process Scheduling Using Varying Quantum Precision", ICRIEST-AICEEMCS, 29th December 2013, Pune India
14. P.SurendraVarma, Design and Performance Evaluation of Precedence Scheduling algorithm with Mean Average as Time Quantum (PSMTQ), 2012
15. Sourav Kumar Bhoi, Sanjaya Kumar Panda and DebasheeTarai,Enhancing CPU Performance Using Subcontrary Mean Dynamic Round Robin (SMDRR) Scheduling Algorithm, JGRCS,2011
16. H. S. BeheraBrajendra Kumar Swain, "A New Proposed Precedence based Round Robin with Dynamic Time Quantum (PRRDTQ) Scheduling Algorithm" For Soft Real Time Systems,IJARCSSE, Vol. 2, Issue 6, 2012
17. C. Yaashuwanth and R. Ramesh, : A New Scheduling Algorithm for Real Time System, IJCEE, Vol. 2, No. 6, PP 1104-1106, December, 2010.

18. R. Bhaskaran and V.Parthasarathy“An improved performance analysis of priority scheduling algorithm” in modified ad hoc grid layer, International Journal of Distributed and Parallel Systems (IJDPS) Vol.3, No.1, January 2012
19. <http://www.informit.com/articles/article.aspx?p=31940>
20. <http://met.guc.edu.eg/OldOnlineTutorials/chapter1.aspx>
21. <http://www.newagepublishers.com/samplechapter/002302.pdf>
22. <http://www.math.uni-hamburg.de/doc/java/tutorial/getStarted/intro/definition.html>