CHAPTER 5

FAULT TOLERANCE IN QOS AWARE P2P NETWORK

5.1 INTRODUCTION

In distributed computing technique, an application consisting of different components and objects are located on different computers connected to a network. Fault tolerance mechanism in a network allows the system to work even if one of the middleware fails. Fault tolerance in CORBA is achieved through entity redundancy, and CORBA object replicas are maintained as “logical singleton” composite objects as proposed by Balachandran et al (2000). CONFIA architecture that comprises of two layers where the upper layer contains replication management function and the lower contains database schema extensions for storing and processing transaction is mainly used to improve the availability of information through data replication as given in Armendariz et al (2006).

Distributed system built with the following factors make programming in heterogeneous distributed environment more difficult (1) Complex data exchange, (2) Different encoding of data types, (3) Synchronization and real parallelism, (4) Requirement for atomic sequence operations. To overcome these problems a middleware, which is a set of business-unaware services that allows interaction between application and end users in a network is needed. A middleware system is tuned and controlled in a predictable manner along with improved QoS metric by
selectively filtering out a “Magical 1%” of the raw observations of various
metrics by Tudor & Priya (2005).

Jiyong et al (2006) suggested a middleware that has been designed
based on Message-Oriented Middleware (MOM) structure for automotive
software satisfies requirements such as resource management, fault tolerance,
a global time base, and communication model specialized for automotive
network and resource frugality. A QoS based middleware architecture called
Frame-sensor adapter control (FSA-Ctrl) has been proposed to provide QoS
level between communication and control layer by (Jose et al 2009) using a
structure called “logical namespace tree” and a set of control processes called
“logical sensor graph”. Distributed system is made fault tolerant by using
techniques such as primary-backup replication and active replication that are
implemented by group-communication infrastructure which in turn provides
multicast primitives as stated by Rachid & Schiper (1996). Secured execution
of application in ambient intelligent distributed systems might be done
through remapping of failed application with redundantly deployed hardware
based on code migration scenario in literature by Diana et al (2003). Tudor &
Priya (2005) have articulated that strict predictability is hard to achieve in FT-
middleware since unpredictability is unavailability even in the absence of
failures.

One of the effective measures to access data effectively in a
geographically distributed environment is replication. Replication is one of
the most widely studied phenomena in a distributed environment. Replication
is a strategy in which multiple copies of some data are stored at multiple sites.
The reason for such a widespread interest is due to following facts: increased
availability, increased performance and enhanced reliability. In distributed
system DBMS, reading and writing of replicated data items are managed by
the replica control protocol known as Read-One-Write-All (ROWA) protocol as given in literature by Robin (2006).

Fault tolerance in real-time object oriented distributed computing system is achieved through a scheme called time-triggered, message triggered object programming scheme in Kane (2001). While working in a distributed system, some of the issues which make distribution of data a complex one are extra coding, transactions and reports, consistency maintenance and operational difficulty. Also, in a distributed environment middleware’s are distributed in more than one system and hence if failure occurs, services are continually provided from other middleware or server. When providing transparent service in a distributed system, care has to be taken to avoid failures in serving the clients. Yoonhee et al (2006) have stated that a communication system performance relies on QoS policies such as real time support, information optimization, message delay control and a paradigm which provides faster message transfer, component hiding and efficient information distribution.

Normally, middleware analyses parameters such as CPU usage, bandwidth, queue length of waiting instructions of the server and priority of the task to ascertain resource availability and to assign the task to appropriate server. Also, middleware scrutinizes remote method calls to direct to the appropriate server hosting the remote object that encapsulates the method. The proposed system framework comprises both fault tolerance middleware and data-replication on server side to improve reliability and QoS of distributed systems. Also, the proposed system takes advantage of “log file” and “middleware monitoring server” to analyze various parameters in each of the available middleware and identify the one which finishes the current task in short span of time. Fault tolerance in the proposed system is maintained by default because middleware that consumes above normal computation time
are not assigned any task. The parameters that are being analyzed are load, objects memory usage, objects function execution duration and CPU usage of the middleware in handling remote method calls.

Diego & Donal (2003) have suggested a semantic layer in the overlay network which is used for organizing the topology based on the content of node. A distributed hash table is used for providing load balancing, query forwarding and look up tables. The semantic layer is above the transport layer in the overlay network. Searching of data and convergence in P2P network is guaranteed by a virtual topology built above the transport layer in the overlay network. The distributed nature of P2P network in the application layer makes each and every peer to communicate with other peers using the routing protocol in the layer as studied by Chonggang & Bo (2003).

In the case of P2P network, each peer knows the location of other peer and if any peer wants to communicate data with any other peer, then a direct edge is formed with other peers. There has been a continuous research on fault tolerance, and look up services since these two properties are the crucial for the P2P network. Since, the problems of fault tolerance are researched properly, it is necessary to provide a mechanism to discover the fault node and recover from it as given by Peter & Sobe (2012) and Boris & Peter (2007).

Any fault tolerant system must consists of the following two stages:

1. Fault Detection - This is an important step in fault tolerance process. The first step is to identify the fault nodes before recovering it, and this is first stage of fault tolerance mechanism.
2. Fault Recovery - Once faulty nodes are identified a mechanism has to be provided for recovering the network from the faulty nodes. There are three ways of recovering the network from failure i.e. (i) Proactive (ii) Reactive (iii) Hybrid approaches.

Routing faults in P2P networks are of two types they are:

Fail-Stop Faults - The main reason behind these types of faults is due to link failure or node failure. This is the simplest fault where the peers they themselves become aware of it. The entire network is made stable by isolating the faulty peers.

Byzantine Faults - The non-stop nature of a faulty node is the reason for this fault and these faults are also known as attacks. The attackers main job is to dismantle the entire network by corrupting the routing table and look up services by misleading the messages across the peers. Has identified that this fault has the ability to damage the entire network so it needs to be properly controlled.

Need for Fault Tolerance System in Peer-to-Peer Network:

1. Peer-to-Peer network offers symmetry in roles where a client may also be a server. Since it allows access to its resources by other systems, hence, in these situations the fault tolerance system is more crucial.

2. In order to maintain the robustness of the P2P network an efficient fault tolerance system is needed as given in the literature by Eng et al (2005).
3. Jagadish et al (2005) have studied that in a tree where every pair of nodes are connected by a single edge, requires a fault tolerant system when any link connecting between two nodes is unreliable.

4. Real time, Quality of Service and fault tolerance is the essential features of a Peer to Peer System as given in literature by Rolando et al (2008).

5. Fault Tolerant systems are used for creating Decentralized and P2P systems as proposed by Miguel et al (2002).

5.2 QOS DISTRIBUTED SYSTEM ALGORITHM

The system described in this section has set of clients \{C_1, C_2 and C_3\}, servers \{S_1, S_2 and S_3\} and middleware’s \{M_1, M_2 and M_3\} as in Figure 5.1. Here each middleware transmits “ACK/NAK” message to its neighbor’s middleware at regular intervals to know about ALIVE status. When the client sends a request, it will be received by the particular available middleware and the middleware in turn will forward the request to the available server based on the load information it has. This process ensures that the task is being handled by middleware with minimum load and best computing power. Since, the time required for computing a task by a middleware depends on its computing power, it is obvious that the middleware with best computing power and minimum wait time for instructions in a queue will have the highest priority to receive request from the client.
5.2.1 QoS Metrics

Standards to maintain middleware`s QOS: A set of standard has to be followed for successful middleware operation.

1. Fault tolerance: The middleware technique to tolerate faults in middleware parameters.

2. Replication: The mechanism of copying data from one middleware into the request server for improved availability, fault tolerance and data latency time reduction.

3. Response-time: Time taken by the middleware to serve the requests from the client and the server.

4. Monitoring: Process of maintaining and controlling the transmission and reception of data between servers in the system.

5. Self-organizing: The middleware checks for the faulty operations among the servers. In case of shutdown of one server the middleware assigns the control to another server for uninterrupted service.
6 Level of service: It is the measure of the middleware to determine, predict the requirement of the server and fulfill the requirement of the server is in safe state.

7 Reconfiguration: The components in the distributed system undergo changes in the functions when the system is in safe state. This change must be validated once the update is competed.

5.2.2 Algorithm for Improving of Middleware

The proposed algorithm is efficiently based on middleware load, object memory usage and object’s function execution duration. The object requests processed by a middleware are maintained in a queue.

Assumptions:

\[ No \] - Number of objects in the queue of a Middleware

\[ S_{om} \] - Vector of Memory size of objects in the queue

\[ S_{NM} \] - Size of needed memory for the current object request

\[ T_{tot}[] \] - Vector of total time needed for the execution of Remote function calls of all the objects

\[ T_{ela}[] \] - Vector of elapsed execution time of remote function calls of all the objects

\[ T_{NO} \] - Time needed for the execution of current object request

The remaining time needed for the execution of remote function calls of all the objects is calculated using Equation (5.1) Rachid & Schiper (1996).
\[ T_{\text{rem}}[i] = T_{\text{tot}}[i] - T_{\text{ela}}[i] \] (5.1)

Then the vector \( T_{\text{rem}} \) is sorted in ascending order to get \( ST_{\text{rem}} \) and the corresponding indexes are stored in \( IT_{\text{rem}} \). The vector \( ST_{\text{rem}} \) consists of object requests in the order of their execution completion time. The memory freed up by each object’s function call after its completion, is then stored in the vector \( S_{\text{FRM}} \).

\[
\text{for each value of } ST_{\text{rem}} \\
S_{\text{FRM}} = S_{\text{OM}}[IT_{\text{rem}}]
\]

end

Then, a middleware is chosen to serve using Equation (5.2) the current object request based on the following algorithm:

\[
\text{for each value of } S_{\text{FRM}} \\
S = S + S_{\text{FRM}} \\
T = T + ST_{\text{rem}} \\
\text{If}(S > S_{\text{NM}}) \\
\text{Index = current index of } S_{\text{FRM}} \\
\text{break} \\
\]

end
The above algorithm finds the middleware which frees the required memory with minimum time. \( S_{FRM} \) of every middleware is processed to find the summation of size until it crosses the \( S_{NM} \). Then, the corresponding time needed is also calculated. The same process is repeated for all the middleware\'s. The time taken with minimum value is taken into account.

The following Equation (5.2) by Houida et al (2002) is applied to all the middleware. The middleware with least value is chosen for the next function call.

\[
3\log(T) - 2\log(U_{cpu}) - \log(T_{rem})
\]  

(5.2)

Where \( U_{cpu} \) is the CPU usage.

### 5.2.3 Improving Fault Tolerance of Middleware

In this section, main exposure is given for raising the fault tolerance aspect of middleware with an objective of improving QoS of the system. The algorithm proposed here identifies the apt middleware by scrutinizing the vital parameters and improving the fault tolerance. The parameters probed are the load, object\’s memory usage, object\’s function execution duration and CPU usage of the middleware in handling remote method calls.

Client\’s log file maintains all the requests from the client to the middlewares. The requests are maintained in the following format:

Date-time, Port, Server-Id, Object name, Function name.

This file is used by \( M_2 \) to identify and process the recent requests given to \( M_1 \). When \( M_1 \) is dead, for example consider a situation where a client sends a request at a time 09:01:01, which is logged in the client-logs file,
received and processed by $M_1$, for the next second 09:01:02 when client sends next request it is logged in client logs file as usual, but is not processed by $M_1$ since $M_1$ is dead at 09:01:01 and simultaneously $M_2$ receives NAK from $M_1$ and now $M_2$ reads the log file to process the recent request to $M_1$ to continue the process as shown in Figure 5.2. Data replication is a mechanism of replicating data’s port-number, object name, function name, server-Id in all neighbor middlewares after processing each request at regular time intervals, and also ACK and NAK messages are transmitted and received among neighbor middlewares to know their ALIVE status.

This file is used by $M_2$ to identify and process the recent requests given to $M_1$. When $M_1$ is dead, for example consider a situation where a client sends a request at a time 09:01:01, which is logged in the client-logs file, received and processed by $M_1$, for the next second 09:01:02 when client sends next request it is logged in client logs file as usual, but is not processed by $M_1$ since $M_1$ is dead at 09:01:01 and simultaneously $M_2$ receives NAK from $M_1$ and now $M_2$ reads the log file to process the recent request to $M_1$ to continue the process as shown in Figure 5.2. Data replication is a mechanism of replicating data’s port-number, object name, function name, server-Id in all neighbor middlewares after processing each request at regular time intervals, and also ACK and NAK messages are transmitted and received among neighbor middlewares to know their ALIVE status.
neighbor middlewares after processing each request at regular time intervals, and also ACK and NAK messages are transmitted and received among neighbor middlewares to know their ALIVE status.

Fault-detection is the mechanism of fault tolerance which helps the middleware to work continually even after partial failure. Figure 5.2 shows the design of fault-free middleware. Fault-Detection Protocol (FDP) is used to provide reliable communication between middlewares on the basis of ACK and NAK messages transmitted and received among the neighbor middlewares.

When a particular middleware is detected faulty, and if it fails to respond to the “are-you-alive” message from the neighbor middleware for $M_{MT}$ time in $M_{TO}$ period, then the detected middleware is shunned, where it is removed from the cluster and the cluster updates its view to balance the load and client interceptors thereby avoiding the dead node.

Assumptions:

$M_{TO}$ - Middleware’s maximum time to wait for response.

$M_{MT}$ - Middleware’s maximum number of tries.

$M_S$ - Middleware $M_i$ is removed from cluster.

for each middleware in $N_M$

If ($M_{TO} = 2000$)

$M_1$ has missed “are-you-alive” message $M_{MT}$ times

$M_1$ is shunt and it sends NAK to $M_2$ and dies.

end
The fault tolerant middleware mechanism describes the method of choosing the next middleware based on the shortest path between middlewares found using Dijkstra’s algorithm, the availability of individual middleware memory, required memory and $N_{FC}$ remote function calls of the objects. Figure 5.3 shows the design of the fault tolerant middleware system, where a client requests an object from server in which the request is logged in log file and after that the request is forwarded to the server through a middleware. The main objective of the middleware is to check the load balance of various servers and to find the appropriate one which is able to satisfy the request of the client. When a middleware is identified as fault based on fault detection protocol which monitors all the middlewares, then the process is taken over by the next neighboring middleware that is identified based on the fault tolerance protocol.

**Figure 5.3 Fault tolerance middleware systems**

Assumptions:

\[
\begin{align*}
N_M & \quad - \quad \text{Number of available middlewares} \\
S_{AM} & \quad - \quad \text{Size of available memory in a middleware}
\end{align*}
\]
\[ S_{RM} \quad - \quad \text{Size of required memory by the } N_{FC} \text{ remote function calls of the objects.} \]

\[ Md_{ij} \quad - \quad \text{Weight of edges between } i \text{ and } j \text{ middlewares.} \]

\[ MD_{ij}(m) \quad - \quad \text{Weight of minimum cost path between } i \text{ and } j \text{ middlewares.} \]

\[ \text{for each middleware in } N_M \]

// shortest path using Dijkstra's Algorithm

\[ MD_{ij}(0) = 0, \text{ for all } i, \quad MD_{ij}(0) = \infty \quad \text{for } i \neq j; \]

for \( m = 0 \) to \( N-1 \)

\[ \{ \]

\[ MD_{ij}(m+1) = 0, \text{ for all } i; \]

\[ MD_{ij}(m+1) = \min_{k \neq j} \{MD_{ik}(m) + Md_{kj}\} \text{ for } i \neq j \]

\[ \} \]

end

If (Middleware \( 1 == \) failure)

\[ \text{for each middleware in } N_M \]

\[ \text{IF}(S_{AM} > S_{RM}) \]

Call next Suitable \( M_2 \)

end

end

end
5.2.4 Data Replication at Server Side

The replication of data at server side is focused to improve the availability of data, reliability and the performance of distributed system. This will raise the fault tolerance aspect of data and QoS of the distributed systems. The algorithm proposed here replicates the data on the servers based on their access rates. The data which is not available in a server which is needed by a request processed by that server is fetched from the nearby server where the data is available. The access rates of files fetched from other servers are maintained in a vector. If the access rate of a file exceeds a predefined threshold, the corresponding file is replicated in the current server from the nearby available server to improve the availability, fault tolerance and to reduce the data latency time.

Replication Algorithm

Assumptions:

\[ \text{Thresh} \] - Threshold value for replicating a file

\[ N_S \] - Number of available servers

\[ AR_F \] - Vector of file access rate in a server

\begin{align*}
\text{for each value of } N_S \\
\quad \text{for each value of } AR_F \\
\quad \text{If } (AR_F \geq \text{Thresh}) \\
\quad \quad \text{Copy the file from the nearby available server} \\
\quad \quad \quad \text{to the current server.} \\
\quad \quad \text{end}
\end{align*}

end
5.3 QOS AND FAULT TOLERANCE IN TREE BASED P2P OVERLAY NETWORK

In the previous chapter, a caching technique is proposed for peer-to-peer file system. Now in this chapter, a fault tolerance approach using ALIVE packet for peer-to-peer network is given. In order to detect the faults, the ALIVE packet is used to indicate that a node and its associated link are reliable. When a requested peer wants to fetch data from another peer, it sends the ALIVE packet to all the nodes along the estimated path towards the destination. On receiving the ALIVE packet, the intermediate nodes will fill the corresponding fields of ALIVE packet and send back to the source node. The source examine the parameters of the nodes to detect if is there any possibility of fault occurrence. Based on this observation, it will mark that node as reliable or not reliable. If the nodes are reliable, then it begins the fetching of data from the destination. Once if the source detects that some nodes are not reliable, then it will immediately send a notification to its immediate parent node.

5.3.1 Architecture of P2P Overlay Network

Consider a tree $T$ with ‘r’ as the root. $N$ be the total number of nodes. Each node acts as a peer node. A Peer is an autonomous entity with a capacity of storage and data processing. In a computer network, a Peer may act as a client or as a server. A Peer-to-Peer is a set of autonomous and self-organized peers, connected together through a computer network. The purpose of a P2P network is the sharing of resources (files, databases) that are distributed on peers which avoids the appearance of a peer as a central server in this network.

This tree is a structured peer to peer network, where each node in the first level is connected with $\alpha$ links each subsequent links are connected
with $\alpha^2$ links. In this network, if any node is detected as failure node then its link will be replaced by its subsequent links.

Figure 5.4 Overlay Network
5.3.2  Fault Detection

Whenever a fault occurs, to detect the faults, the ALIVE packet is used to indicate that a node and its connected links are reliable. Whenever a data is requested from a peer and the data has to be fetched from another peer, the serving peer sends the ALIVE packet to all the nodes through the estimated path towards the destination peer.

Table 5.1 ALIVE packet

<table>
<thead>
<tr>
<th>A</th>
<th>BER</th>
<th>BO</th>
<th>ITI</th>
</tr>
</thead>
</table>

In Table 5.1, the ALIVE packet has four fields which are represented as followed:

A - This field is set to one to indicate that the transmitter node is ALIVE

**Bit Error Rate (BER)** - It is used to indicate the errors in the received signal.

**Buffer Overflow (BO)** - This is used to detect the loss of packets in the node

**Inter Test Interval (ITI)** - This is the time difference between the arrival times of two subsequent data.

Algorithm for Fault Detection

If (A source node S wants to send data to the destination node D)

Then

Retrieve a path from S to D node from the routing table.
Send ALIVE packet to all the nodes in the path.

For (All the nodes i in the path from S+1 to D)

If ((ACK not received) or (process (ALIVE) = Fault node))

Then

Node i is faulty

Recovery ();

Else

Continue

End If

End for

End If

The tree diagram of fault detection is shown in Figure 5.5. and it shows how faults are detected in a peer to peer overlay network. Initially, if the node 1 plans to send a data to node 14 and 12 then it will retrieve the path stored in the routing table and before transmitting the data. The node 1 will send ALIVE packet to all the nodes in the path i.e. the intermediate nodes between node 1 and two destination nodes 14 and 12.

The ALIVE packets are sent across both the paths but path between 1 and 12 is successful because the reply for ALIVE packet is received by the node 1 from all other nodes but the node 8 has not replied the ALIVE packet even after the threshold time so node 8 is fault but node 3 is indicated as it is nearing to be fault. The nearing fault of node 3 is determined after processing of ALIVE packet reply from node 3.
5.3.3 Fault Recovery

The fault recovery is much crucial for getting an uninterrupted service. Whenever a node fails and is identified faulty in a tree type network, the alternate parent-to-be nodes are sought to take the place of the failure node. A link is created between the node which accepts the request of
becoming the parent and the old node’s link is removed. A link is also created with the grandfather node to identify the other children of the grandfather and creating a link with them.

Figure 5.6 Fault Recovery

Figure 5.6 shows an example for fault recovery. Since, node 8 was detected as fault and node 3 was nearing to fault so the fault recovery algorithm makes node 9 as the parent node for node 10, 11, 14 and 15. According to the parent-to-be structure node 14, 15, 10 and 11 next parent is 9 so all these nodes send request which is then accepted by the parent node 9.
5.4 RESULT AND DISCUSSION

5.4.1 Result for FT-Middleware

The implementation of the proposed approach is carried out in Java for testing the efficiency. The system for simulation composed of few servers, middlewares and clients. The object’s memory consumption and method call’s time consumption were stored in the database.

![Figure 5.7 Client’s request to middleware](image)

![Figure 5.8 Current middleware’s process](image)
All the remote calls of methods were monitored and information like called object, method, total time, elapsed time and memory occupied were also stored in the database. Figure 5.7 shows one of the clients request to middleware through the log file. Figure 5.8 shows the process of current middleware in which the following details are displayed:

a. Time taken by the server to complete the current task and the server Id.

b. Based on above said details the server (8083) is selected.

c. Continuous update of server details such as Id, time consumption and memory capacity.

d. CPU usage value that is being updated for every 1000 milliseconds.

Figure 5.9 shows the performance of the current server’s operations based on the input from the middleware.

Java is used to implement the simulation with some servers, middlewares and clients. The memory consumed by the objects, the time consumed by every methods call was stored in the databases. All the remote
calls of methods were monitored and information like object, method, total time, elapsed time and memory occupied were also stored in the database. Whenever the function calls are over these entries will be removed automatically. Improving the fault tolerance algorithm takes these inputs to evaluate the next middleware qualified to serve the method call of the client. Finally, the testing of the software is done with three middleware with three clients. All the middleware almost worked with balanced load and time. The results of the time analysis for middleware are shown in Figure 5.10.

![Analysis of Middleware](image)

**Figure 5.10** Analysis of middleware

### 5.4.2 Result for FT- Peer-to-Peer

This section deals with the evaluation of the experimental performance of the proposed algorithms through simulations. NS2 simulator is used for testing the protocol. NS2, which is a general-purpose simulation tool, provides discrete event simulation of user defined networks.

The BitTorrent packet-level simulator is used for P2P networks. A network topology for the packet-level simulator is used. It is assumed that the bottleneck of the network is not at the routers but at access links of the users. Based on this assumption, a simplified topology is used in the simulations. The network is modeled with the help of access and overlay links. An
asymmetric link connects each peer to its access router. An overlay link connects all access routers directly to each other. This helps to simulate different upload and download capacities and different end-to-end delays between different peers as depicted on Figure 5.11.

![Figure 5.11 Simulation Topology](image)

**Figure 5.11 Simulation Topology**

**a. Based on Rate**

In the first experiment the transmission rate is varied as 100, 200, 300, 400 and 500 Kb.

![Figure 5.12 Rate vs Received Bandwidth](image)

**Figure 5.12 Rate vs Received Bandwidth**
From Figure 5.12, the received bandwidth of the proposed FT-tree method is better than the existing PSON-fault method. The received bandwidth of the proposed FT-tree method has increased by 6.03% when compared to PSON fault method. From Figure 5.13, it is seen that the throughput of the proposed FT-tree method is higher than the existing PSON-fault method. The FT-tree method shows the overall increase in throughput by 8.39% when compared to PSON fault method.

![Figure 5.13 Rate vs Throughput](image1)

![Figure 5.14 Rate vs Delay](image2)

From Figure 5.14, the delay of the proposed FT-tree method is less than the existing PSON-fault method. The delay of the proposed FT-tree method is half of the overall delay shown by the PSON fault method. Thus the FT-tree method reduces the delay by 50%.
Table 4.2 Comparison between FT-tree and PSON-fault

<table>
<thead>
<tr>
<th>Rate</th>
<th>FT-tree</th>
<th>PSON-fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>14.5192</td>
<td>28.9868</td>
</tr>
<tr>
<td>200</td>
<td>14.5212</td>
<td>28.9879</td>
</tr>
<tr>
<td>300</td>
<td>14.5252</td>
<td>28.9886</td>
</tr>
<tr>
<td>400</td>
<td>14.5274</td>
<td>28.9895</td>
</tr>
<tr>
<td>500</td>
<td>14.5296</td>
<td>28.9924</td>
</tr>
</tbody>
</table>

From Table 4.2, delay with respect to node for FT-tree and PSON-fault was observed. It clearly shows that FT-tree gives delay two times lesser than PSON fault.

b. Based on Time

In the second experiment, the performance is analyzed by using the time.

![Figure 5.15 Time vs Received Bandwidth](image)

**Figure 5.15 Time vs Received Bandwidth**

From Figure 5.15, the received bandwidth of proposed FT-tree method is better than the existing PSON-fault method. The PSON fault method showed
constant bandwidth though there were variations between 0 sec to 5 sec. The FT-tree method also showed different variations starting from 1.5 sec which was continuously increasing. Thus overall bandwidth of FT-tree is higher than PSON fault method.

![Time vs Throughput](image)

**Figure 5.16 Time vs Throughput**

From Figure 5.16, it is seen that the throughput of proposed FT-tree method is higher than the existing PSON-fault method. The FT-tree method showed a continuous increase in the throughput for every time period from 1.5 sec, where as the PSON fault protocol had a constant throughput after 5.5 sec, although initially FT-tree method provides lesser throughput.

![Time vs Delay](image)

**Figure 5.17 Time vs Delay**
From Figure 5.17, it is observed that the delay of the proposed FT-tree method is less than the existing PSON-fault method. The overall delay in FT-tree method is observed to be 167.16 sec. The PSON fault method showed a delay of 278.65 sec which is very high. Hence, the FT-tree method reduces the overall delay in the system and is better when compared to PSON fault method.

5.5 SUMMARY

In this research, an efficient tree based partial caching technique for peer-to-peer file sharing system is proposed. Concerning this technique, ‘k’ number of objects are placed using proactive segment placement approach during the deployment of the network. The left over objects are reactively regulated by partial caching technique. When the client sends the object request, the router calculates the hit ratio and access time of the object. Based on this calculation, the router estimates the popularity of the object. The router first searches the object in proactive cache proxy and then in partial cache proxy. When the router discovers more replicas for the same object, it selects the object that has higher popularity value. By simulation, it is shown that the proposed technique reduces the delay and improves the system performance considerably.

A mechanism for fault tolerance is also proposed. The fault tolerance algorithm involves two sub phases: (i) Fault Detection and (ii) Fault Recovery. An ALIVE packet is used to detect whether a node is fault or it is nearing fault. After the detection, a fault recovery algorithm is employed which uses proactive and reactive approach to recover the P2P network from faults. In a node if a fault occurs, then its child node will check its next parent node and it will send the connection request to the next parent node. If there is no reply, then the child node will use a reactive approach i.e. it will send request to the grandfather node to obtain a new connection. By simulation results, it is shown that the proposed approach reduces the packet drop and improves the throughput.