ISSN No:-2456-2165

A Framework for Monitoring Network Node Failure using Mobile Agents

M. O Lawal[*1]; K. G Akintola[2]; O. K Boyinbode[3]; N.C Onyeka[4]
[1]Department of Computer Science, Federal Polytechnic Ede
[2]Department of Computer Science, Federal University of Technology, Akure
[3]Department of Computer Science, Federal University of Technology, Akure
[4]Department of Computer Science, Federal Polytechnic Ede

Abstract:- Fault detection is an essential aspect of conducting fault diagnosis for computer networks. It comprises of two phases: fault detection and fault localization. The use of mobile agents for detecting faulty nodes on a network is a concept aimed at ensuring the proper functioning of networks. This research aims to design a fault detection framework for a network system using a mobile agent. Light Weight Agent (LWA) travels within the nodes to detect nodes that are down on the network and returns true or false along with other information as the status of each node visited. The system is designed using software agents. This subsystems of the system include the Agent Controller, Server Agent, Client Agent, Check Status and the database. The Agent Controller allocates and determines the agent functions using a unique identification number. The server agent controls the activities of the client agent by monitoring the migration of each of the probing agents to each node on the network. The system is implemented using the Java Application Development Environment (JADE) platform. It was tested on a network with twenty nodes, for five hours per day for twenty days. The system achieved a reliability rate of 100% for the highest and 47% for the lowest. This research work will be beneficial for testing the reliability of a networking system to ensure optimal functioning. Future research will focus on using mobile agents to diagnose faulty nodes on a network.

Keywords:- *Mobile Agent, System Reliability, Computer Network, JADE, Fault Detection.*

I. INTRODUCTION

Nowadays computer networks is becoming very large, covering the vast majority of geographical locations. Network Management Applications (NMA) designed to manage network tasks such as, maintenance and administration of the network were also designed to manage traditional client/server networks. However, as computer networks expand, the size and complexity of client/server models are faced with the problem of scaling and flexibility [6].

Researchers in the field of software mobile agents are now focusing their attention on Network management systems. However, if there is a malfunction, the issue of information overwhelming the network becomes especially severe, particularly since a quick solution is imperative. Swift diagnosis and resolution of the problem either through automated means or by informing and guiding a human operator on the appropriate course of action becomes crucial. Devices such as routers, hubs, servers, and more are monitored by the manager and when there are faults within the network, the application manager within the network notifies the network manager in real-time.

Operators working with large networks must remotely interact with numerous devices from their management workstation. To cater for the diverse range of network components, management applications feature a plethora of interfaces and tools. However, network management systems are often designed as large monoliths, making them challenging to maintain.

Automatic discovery is a crucial aspect of network management systems, with various objectives depending on the scope of the system. At its most basic level, discovery aims to locate all devices present within the network. However, an expanded version of this function involves constructing detailed views that encompass additional information, such as the services offered by each devices that meet specific criteria. As the process of identifying the problem becomes more complex, it becomes harder to implement using traditional client/server methods.

This research emerged from the exigency to use an agent to detect network faults/failures using intelligent decision-making agents. It also came from the reading literature reviews of previous researchers such as [6] on how to solve the problem of a complete recovery mechanism in case of fault/ failure within a network without simulation. The study by Jian Hu et al. (2008) enables users to define their own Management Information Base (MIB) tables, but this also results in increased system complexity, as mobile agents must communicate directly with the managed system, which may impact system compatibility. The primary objective of this research is to leverage mobile agents in managing today's large and diverse networks. Mobile agent software objects are autonomous and can move from one node to another, carrying logic and data to perform tasks on behalf of the user. The network management software objects based on mobile agents will be equipped with agents possessing network management capabilities that will enable

them to issue requests to managed devices or nodes once they migrate to these nodes.

II. REVIEW OF RELATED WORK

In this section, reviewed literature related to network faults, network fault detection, Mobile Agent, system reliability and Network reliability are as follows: Mobile Agents in [1], [2], [3], [4], and [40]. Network fault detection in [9], [10], [11] [12], [34], [40], [43], [49]. System reliability [11], [17], [45], [48]. Characteristics of Mobile Agents are as presented in [4], [6], and [30]. Network reliability in [12], [15], [28], [29], [31], [33]. Network management and monitoring in [33], [34], [35].

➢ *Fault Identification*

Fault identification is used to understand the elemental failure mode, ascertain the margin of the fault, and find the core cause. Fault identification methods may differ, but the strides to follow are mainly identical.

- A physical fault is a type of network failure that is related to hardware issues.
- Port faults typically fall into two categories: unstable ports and port failures..
- When switches or routers break down, it's often due to equipment damage resulting in abnormal network behavior.
- Network card faults are considered to be a type of host hardware failure and are a frequent reason for network problems.

➢ *Fault Detection*

Fault detection is the process of locating the existence of a fault in a network before it presents itself in the form of network failure and breakdown. It is the most important stage of network fault detection (NFD) as all of the subsequent processes depend on its accuracy. If the equipment is unable to identify the proper failure mode (or if detection is incorrect and triggers false alarms), then the separation, identification, and appraisal will also be ineffective.

III. MOBILE AGENT

Mobile agents are programs designed to function automatically moving from node to node. They can perform a task on behalf of users and allow difficult tasks to be shared amongst the agents [1], [2], [3], [4]. The primary goal of using mobile agents in the management of telecommunication networks is reducing network traffic by using load balancing and building scalable and reliable distributed network management systems. Some of the advantages of using agent technology in telecommunication networks are as follows:

- Addresses the handling of a large volume of data that agents can explore, gather, and filter.
- Facilitates the utilization of more intelligent techniques to manage a network, integrate different services into value-added services, and negotiate quality of service.
- Promotes the development of higher-level communication and organization within a network.
- Demonstrates reactivity, as agents can promptly respond to local events, such as link failures.
- Exhibits robustness, as agents can perform their duties to some extent, even when parts of the network are temporarily inaccessible. This is particularly crucial in mobile computing, where links can be expensive and unstable.
- Distributes management code to Simple Network Management Protocol (SNMP) agents to reduce bandwidth consumption in a wireless network.
- Decentralizes network management functions by allowing mobile agents to autonomously and proactively carry out administrative tasks, thereby reducing the amount of management traffic required.
- Dynamically adjusts network policies, as mobile agents can modify the underlying rules of network management periodically.

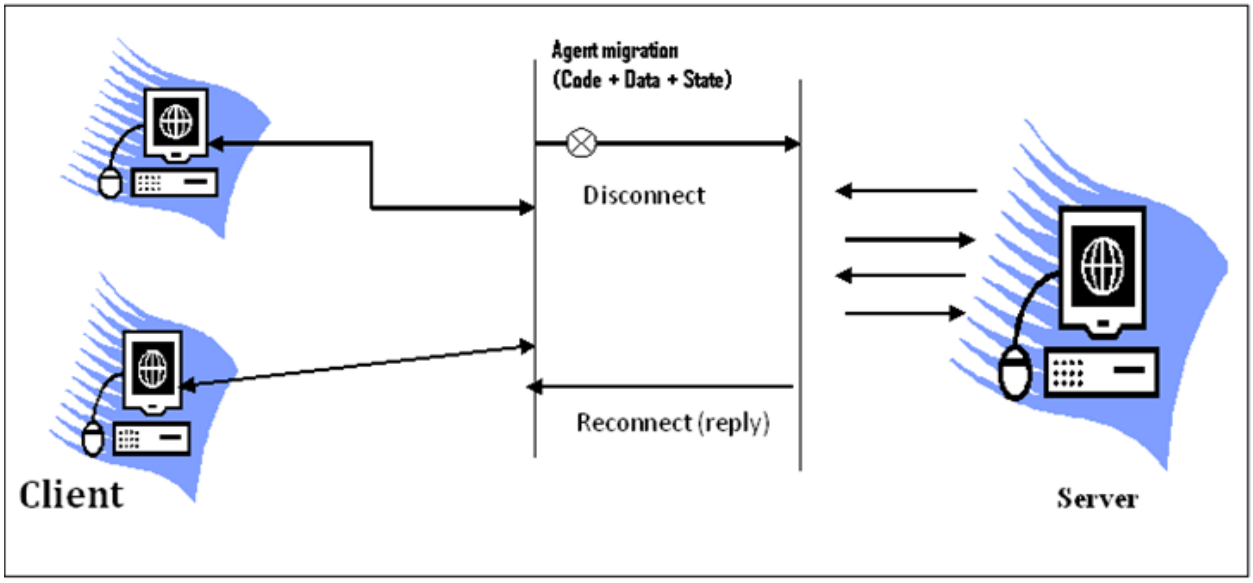

Fig 1 Mobile Agents Model (Singh1 et al., 2012)

ISSN No:-2456-2165

➢ *Mobile Agent System Architecture*

Network failure can be prevented if the concept of fault-detection design strategies is used. Fault tolerance is a way to head off a failure before it has a chance to happen. With the fault-detection technique, a problem such as a software defect in safety-network machines can be identified and prevented. The preemptive detection of node failure using mobile agents is the concern of this project work. If this fault is left uncared for, it could result in network failure and consequently network machine downtime.

In this research work, a fault-detection architecture which is based on a fault identification procedure is used and includes the following two steps: Fault detection and fault localization [7]. The initial step in fault diagnosis for computer networks is detecting the presence of any faults, which involves using detection tools. If any faults are detected, fault localization is then initiated to identify the location of the fault and the affected node. Therefore, fault detection is a crucial first step towards ensuring the normal operation of networks, and it is essential to employ fast and precise fault detection techniques. The proposed method of fault detection used in this research work is a non-deterministic environment [45], [49], [51]. The goal is to partition the detection process into several stages, with a small number of lightweight agents (LWAs) assigned to monitor particular network nodes at each stage. By the conclusion of numerous detection stages, every node in the network can be inspected. This approach guarantees that the traffic generated by probe agents during each detection stage is considerably less than conventional methods, although it may take more time to cover the entire network. The idea of the strategy to ensure this aforementioned is established in the following attributes:

- There will be an Agent Controller manager that identifies the agent type (Server/Client Agent).
- Each node should have an individualized fault detection mechanism (client agent) to ensure that its service is not impaired by any hardware failure or software fault.
- There is reliable and timely delivery of reliable messages from nodes to the Server Agent on the shared network.
- The individual node transmits at the appointed time slot at all times of the network machine's operation.

The essence of this work is to foresee the occurrence of a masqueraded fault and prevent it by providing a solution before the network fails using mobile agents. In this view, a proactive strategy to prevent faults resulting from software-defect or hardware defects is presented in this work [51] and [44].

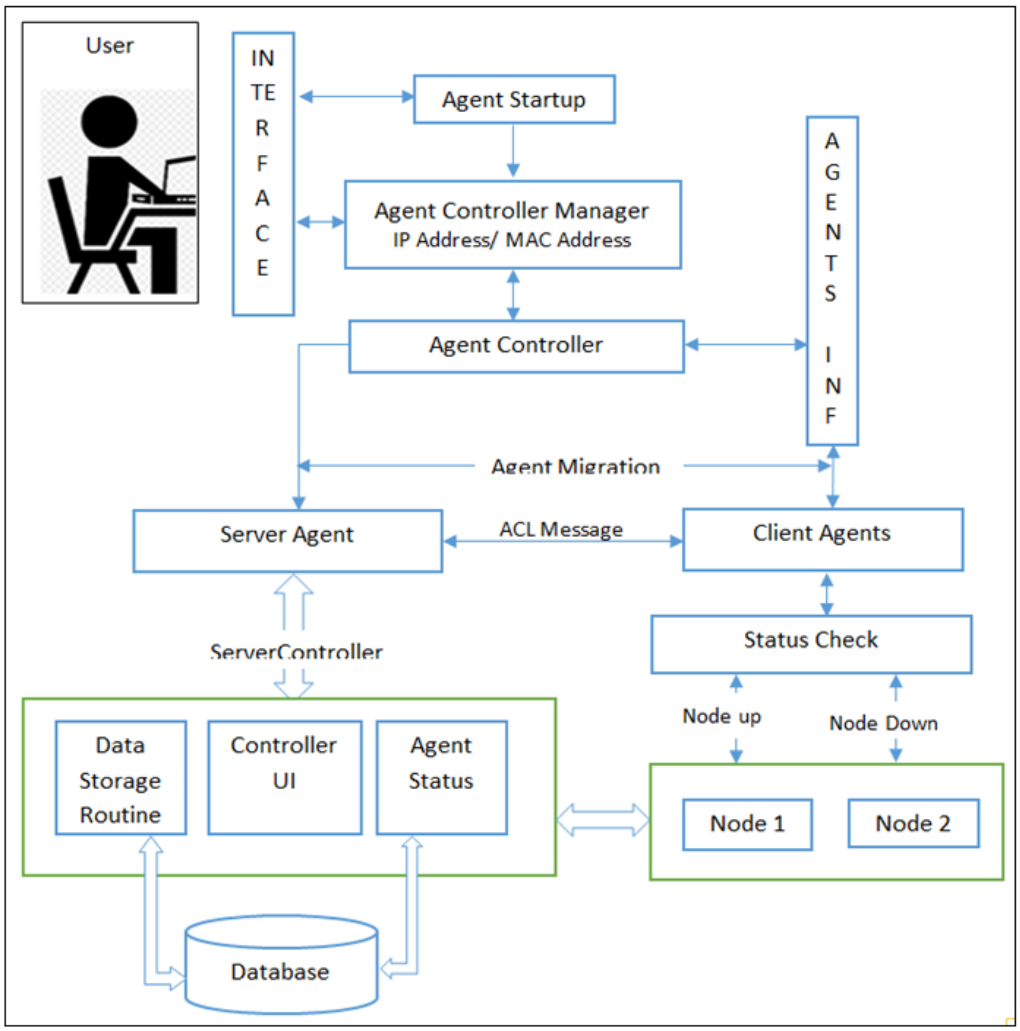

Fig. 2 Mobile Agent System Architecture

ISSN No:-2456-2165

IV. THE MATHEMATICAL MODEL

The main performance metric considered in this research work is reliability, which is a crucial aspect of engineering design and development. The field of reliability engineering encompasses all stages of a system's lifecycle, from design to fabrication, with the goal of minimizing the risk of equipment failure. Neglecting reliability can result in severe consequences, including the loss of critical information or the erosion of trust in the system. Moreover, acceptable levels of reliability may differ depending on the application environment [45] and [48].

The relationship between reliability parameters and probability theory can be expressed as follows: Suppose a fixed number N0 of identical items is being tested, and Ns is the number of items that survived after a certain time period t, while Nf is the number of items that failed during the same period. Then, for all t,

$$N0 = Ns + Nf. \quad (1)$$

If N0 is sufficiently large, the reliability R(t) of an item can be calculated as Ns divided by N0.

The failure rate function lambda (λ(t)) is defined as the instantaneous rate of failure at time t, which can be mathematically expressed as:

$$\lambda(t) = -1/R(t) * dR(t)/dt \quad (2)$$

Where R(t) is the reliability function of the system. The negative sign in front of the fraction indicates that lambda λ(t) is a decreasing function of time t, as the reliability function R(t) decreases over time. The failure rate function λ(t) is an important concept in reliability engineering and is used to estimate the probability of failure of a system over a given time interval.

$$R(t) = e^{-\lambda(t)dt} \quad (3)$$

The survival probability function, as defined in equation (2), is commonly known as the reliability function. This function represents the probability of an item not failing during the time interval [0, t]. When discussing the reliability of a system, it is often referred to as the probability of no occurrence of faults belonging to class F (i.e., the system survives) during time t.

The probability that a system will continue to operate without experiencing a fault of class F within a given time interval, t, is denoted by RF(t). This is also referred to as the system's reliability. It is defined as the probability that the time to the first failure, tf, is greater than t given that the system has operated successfully until time tinit. Mathematically, it can be expressed as:

$$RF(t) = P(tinit \leq t < tf \; \forall \; f \in F) \quad (4)$$

RF(t) is a probability function that represents the likelihood that a system will operate without a fault of class F occurring within a given time interval [t_init, t_f) for all f in the set F. In other words, RF(t) measures the probability that the system will survive without experiencing any faults of class F during the time interval [t_init, t_f) for all possible faults F that may occur. RF (t) can be calculated using the reliability function R(t) as:

$$RF(t) = R(tf \mid tinit \leq t)$$

Where R(tf | tinit ≤ t) is the conditional reliability of the system at time tf given that it has operated successfully until time t. It can be calculated as:

$$R(tf \mid tinit \leq t) = R(tf)/R(t)$$

Where R(tf) is the reliability of the system at time tf and R(t) is the reliability of the system at time t. Failure Probability $Q_f(t)$, is complementary to $R_f(t)$

$$R_f(t) + Q_f(t) = 1 \quad (5)$$

We can remove the subscript 'f' and write the equation as R(t) + Q(t) = 1, where R(t) is the reliability function and Q(t) is the probability of failure function.

If the lifetime of the system is exponentially distributed, the probability of no failure occurring in the time interval [tinit, t] is given by:

$$RF(t) = e^{\wedge}(-\lambda(t-tinit))$$

$$R(t) = e^{-\lambda t} \quad (6)$$

Where 'λ' is called the failure rate.

Since this research work is employing TTP/C (Time-Triggered Protocol/Clock-Synchronized) which makes use of the Time Distributed Media Access mechanism TDMA. The research hereby assumes that the operation of each node successively takes place as they take part in the schedule round at their allotted timestamp, except in the case of the fault being currently detected. This makes the serial reliability mathematical model suitable to abstract the networked embedded systems machine. The reliability for serial networked embedded systems is given as:

$R_k(t)$ is the reliability of a single component k:

$$R_k(t) = e^{-\lambda kt} \quad (7)$$

The overall system reliability $R_{ser}(t)$

$$R_{ser}(t) = R_1(t) \times R_2(t) \times R_3(t) \times \ldots \times R_n(t)_n \quad (8)$$

$$R_{ser}(t) = \prod R_i(t) \quad I=1\ n \quad (9)$$

The serial failure rate is given as $\lambda_{ser} = \sum \lambda_i$ (10)

Assuming that the failure rates of individual components are statistically independent, the overall reliability of a system can be calculated using equations (8) and (9). Equation (8) states that the system reliability Rser(t) at time t is equal to the product of the reliability of each individual component, denoted by R1(t), R2(t), R3(t),..., Rn(t), raised to the power of the number of components, n. Equation (3.9) provides a compact notation for this product using the product symbol, Π.

The serial failure rate of the system, denoted by λser, can be obtained using equation (10), which states that λser is equal to the sum of the failure rates of individual components, denoted by λ1, λ2, λ3,..., λn, summed over all n components.

➢ *Mobile Agent Migration Process*

The Server Agent (SV) consists of Servercontroller, Threads (TD), Resourcebundle (RB), Sqlconnection (SC), Messages (MSG), and Agents table (AT). SV communicates with all the nodes on the network using LWA and monitors the agent communication between the Server Agent and Client Agents. It creates threads for each client agent probe, monitors the agent thread, and collects responses in the form of messages which carries all the information representing the status of each client agent that migrated to each visited node. It creates Agent tables AT to store the list of all the probed nodes on the network and sends all the listed nodes and their respective statuses to the database using Servercontroller and Sqlconnection.

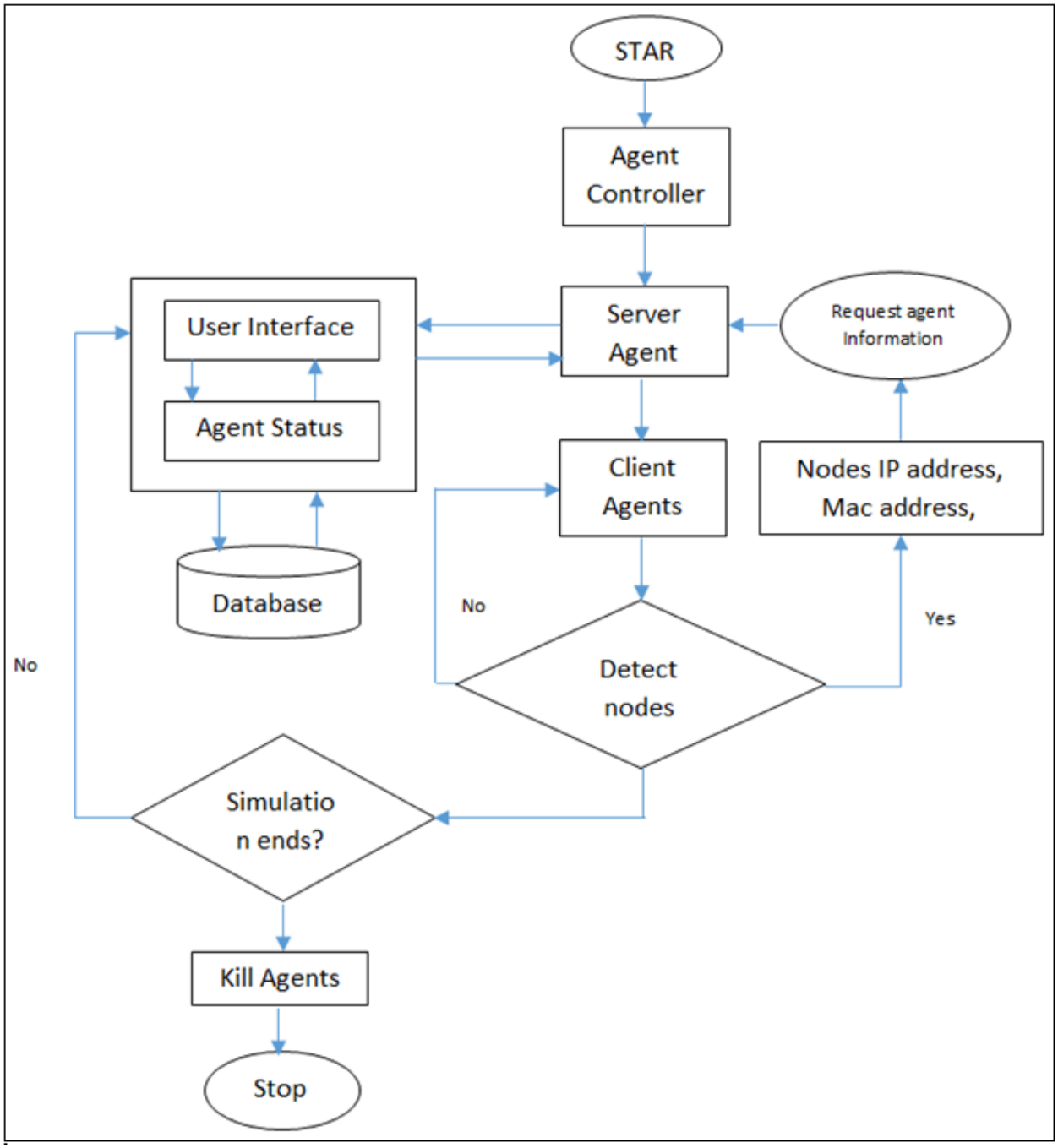

Fig. 3 Server and Client Agent

ISSN No:-2456-2165

V. IMPLEMENTATION OF THE MODEL

The local area network (LAN) in the Federal Polytechnic, Ede library was used for the implementation of the model. For testing of the framework on the LAN, twenty-day, five-hour-per-day test of the framework was done on a network with twenty nodes, consisting of one server node and nineteen client nodes. The operating platform used for the test is Windows 8 OS, SQLite-3.12.2-win64 for database, and Eclipse IDE for Java Developers 4.23. Eclipse allows the integration of JADE through plugins and it allows agent platform integration. Agent creation, starting, lunching, activation, and killing can be achieved on the platform. Since agents require multiple nodes, the start node and destination node must be compatible [42] and [43].

- *Performance Evaluation of the Model*

The research is composed of the Agent Controller, a single Server Agent, and several network nodes called the Client Agents (CA) and status checker. The Server Agent employs Light Weight Agents (LWA), which are special data packets sent to destination nodes (CA) to detect network faults. Each LWA can be identified by its destination node address, which includes the IP address and MAC address. Based on the results obtained from the LWA, the status of the nodes is determined and valued between 0 and 1. A value of 1 indicates 100% live components (Node Up) while a value of 0 indicates 0% live components (node down) on the LWA transmission paths.

As the reliability of this method heavily relies on the proper functioning of both the LWA and the destination nodes, it is assumed that these network components are always functioning correctly. In the active mobile agent technique that utilizes light weight agents for fault detection, the association between the agents and nodes must be taken into account. Researchers in fault detection have traditionally employed deterministic dependency information to model the network, assuming that the connections between the nodes and probe agents are well understood. This approach was adopted in prior studies by [52]. As illustrated in Figure 1.2, the nodes used to transmit LWA to destination node 1 are not deterministic when the Server Agent Controller (Manager) sends them. The reason for this is that any route from the Server Agent Controller (Manager) to destination node 1 can be chosen as the transmission path.

The table 1 below shows the failure frequency, Failure rate, Reliability, and Mean Time between Failures of each of the nodes. This figure shows the reliability rate of each of the nodes after the test period of four hours each day for twenty days. It also shows the mean time between failures for each of the nodes.

Table 1 Failure Rate, Reliability and MTBF for each day

Failure Rate, Reliability and MTBF for each day (T = 5 hours)				
Nodes	**Failure Frequency (f)**	**Failure rate** *(λ)*	**Reliability** *($R=e^{-\lambda t}$ per hours)*	**MTBF T/f** *(per hours)*
1	0	0	1	0
2	2	0.021	0.979	2.5
3	1	0.011	0.99	5
4	1	0.011	0.99	5
5	1	0.011	0.99	5
6	0	0	1	0
7	1	0.011	0.99	5
8	4	0.047	0.791	1.25
9	1	0.011	0.99	5
10	1	0.011	0.99	5
11	9	0.15	0.472	0.5
12	1	0.011	0.99	5
13	0	0	1	0
14	1	0	0.99	5
15	1	0.011	0.99	5
16	0	0	1	0
17	1	0.011	0.99	5
18	5	0.061	0.731	1
19	3	0.033	0.846	1.6
20	6	0.08	0.67	0.833

This table 2 shows the cumulative figures for all the test done on all the twenty nodes on the network for twenty days. It shows the total percentage reliability of all the nodes on the network. It also shows the reliability of the framework after all the test has been conducted for the twenty days.

ISSN No:-2456-2165

Table 2 Cumulating (C) Failure Rate, Reliability, and MTBF

Cumulating (C) Failure Rate, Reliability, and MTBF			
		C = C/20	**C*100**
Failure rate *(λ)*	0.491	**0.02455**	**2.455**
Reliability*($R=e^{-\lambda t}$) per hours)*	18.389	**0.9195**	**91.945**
MTBF T/f *(per hour)*	57.683	**2.8841**	

Discussion: ServerAgent and the clientAgents are connected to the network which contains twenty nodes. The serverAgent is loaded on the single node while the rest of the nodes are loaded with the clientAgents. The system works on Client/Server architecture and each of the client nodes receives probes from the serverAgent which consistently monitors all the LWA sent to each of the client nodes. Figure 4 is a chart representation of data gotten from table 3.2 which shows the test period of twenty days containing twenty nodes on a network. The failure frequency (f) of each node per day (twenty days) of the test is also recorded as the corresponding nodes that failed during the test period. The node2 as 2 failures, node3, node4, node5, node7, node9, node10, node12, node14, node15, and node17 respectively have failed only one time within the twenty days test period. Node8 failed four times, node11 failed nine times, node18 failed five times, node19 failed three times and node20 failed 6 times respectively.

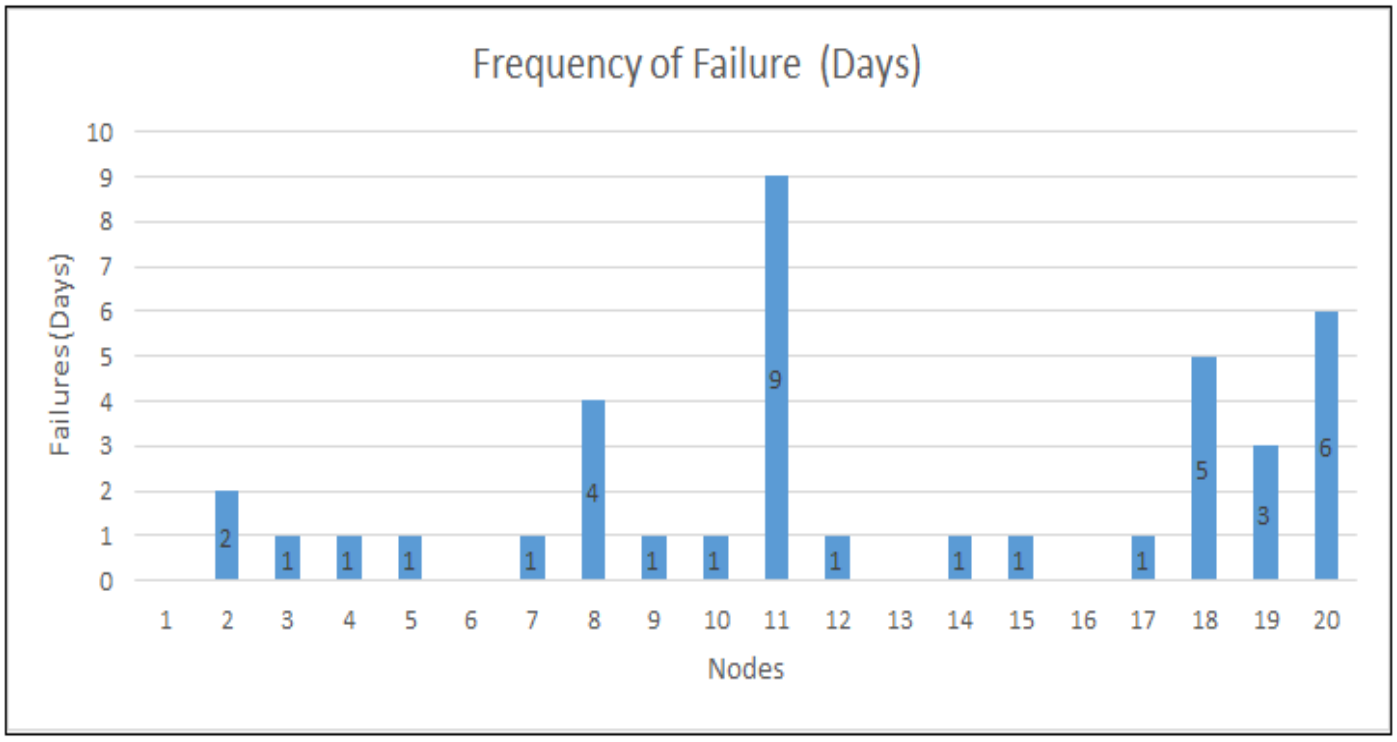

Fig 4 Number of Failures with Corresponding Nodes for Twenty Days Test

Discussion: The failure rate of a system is the frequency at which the system fails or malfunctions over a given time and it is usually expressed as the number of failures per unit of time. The measurement depends on the type of system and the data available. The data is usually obtained by monitoring the system over some time and recording the number of failures that occur. Figure 4 above shows the failure rate of each of the nodes on the network. The system was tested using twenty nodes for twenty days, and the data for the failure rate for each node was calculated from the failure frequency date in table 3.2. Figure 4.8 shows the Failure rate of each of the nodes, starting from *(λ) = 0.011 for* nodes that have the lowest failure rate to *(λ) = 0.15* for node(s) that have the highest failure rate.

ISSN No:-2456-2165

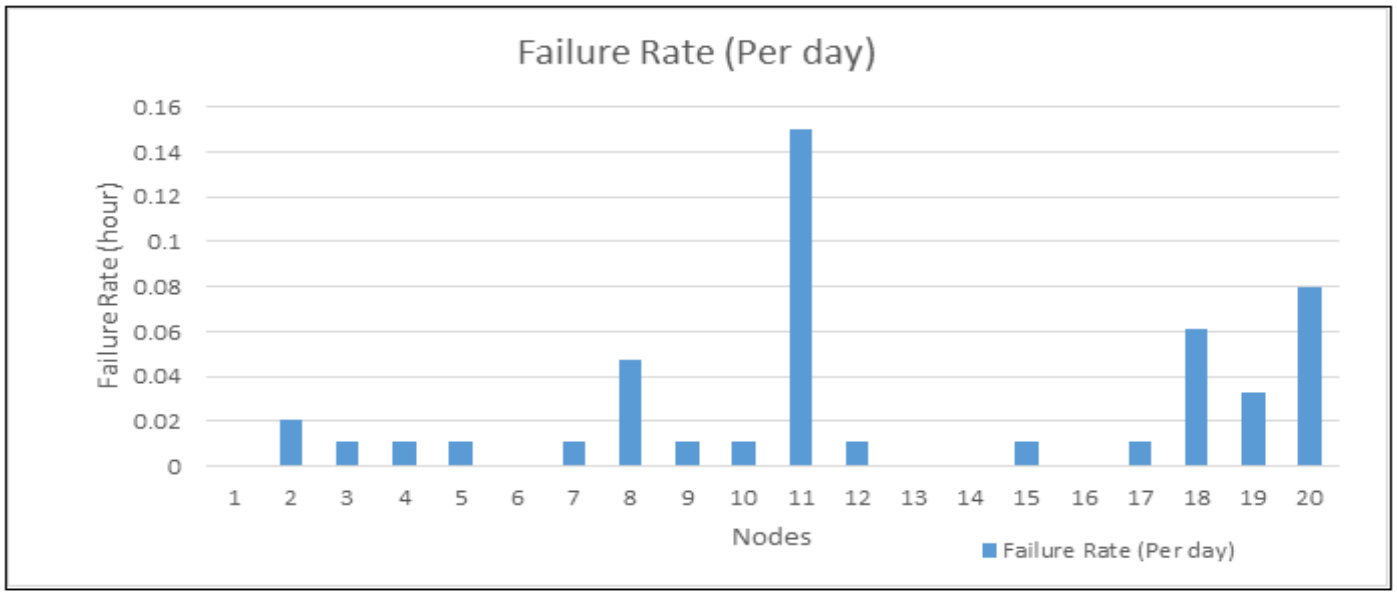

Fig 5 Failure rate of each of the Nodes on a Network

Discussion: measuring the reliability of a system is important for ensuring that it performs its intended function consistently and identifying potential problems before they occur. Figure 5 shows the reliability of each node in the system. The data used for the chart is from table 1 which shows the reliability rate of each of the nodes in the network. Each of the nodes tested for the twenty days with the system is shown and the corresponding calculated reliability rates are also shown. Nodes without any failure have a reliability rate of 1 which is 100% and also specify the probability that the node will not fail. The test shows the reliability rate of the nodes as node1 = 100%, node2 = 0.979 (97%), node3 = 0.99 (99%), node4 = 0.99 (99%), node5 = 0.99 (99%), node6 =1, node7 =0.99(99%), node8 =0.791 (79%), node9 =0.99 (99%), node10 =0.99 (99%), node11 =0.472 (47%). node12 =0.99 (99%), node13 =1 (100%), node14 =0.99 (99%), node15 =0.99 (99%), node16 =1 (100%), node17 =0.99 (99%), node18 =0.731 (73%), node19 =0.846 (84%), node20 =0.67 (67%). The node with the highest reliability shows a reliability rate of 100% while the lowest reliability rate as indicated above is 47%.

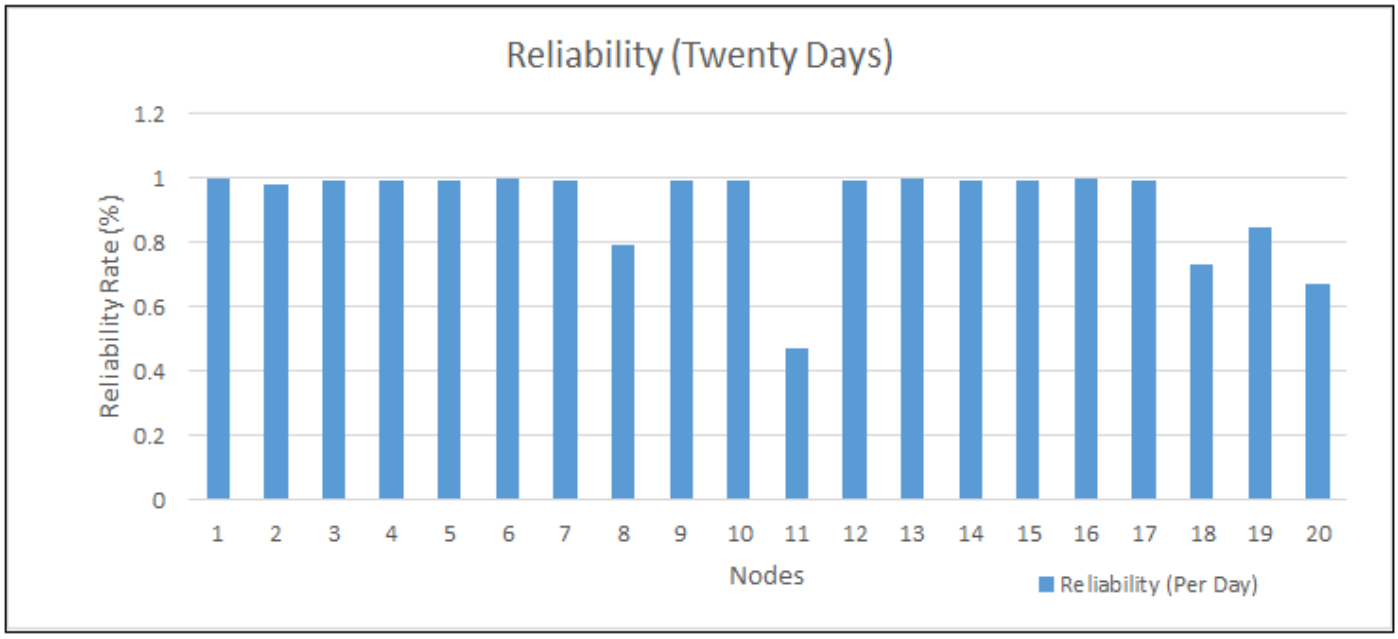

Fig 6 Reliability of each node for Twenty Days test

Discussion: The system testing was done for five hours every day and for twenty days, the cumulating results of the nodes from all the days are summed together and the average of the result is found. This is shown in figure 4.10 and shows the total failure rate, reliability, and MTBF. This can show the total test hours, the total number of failed nodes, and the total working nodes. Therefore, figure 5 can show how reliable the system is haven is gone through the five hours daily and twenty days test period.

ISSN No:-2456-2165

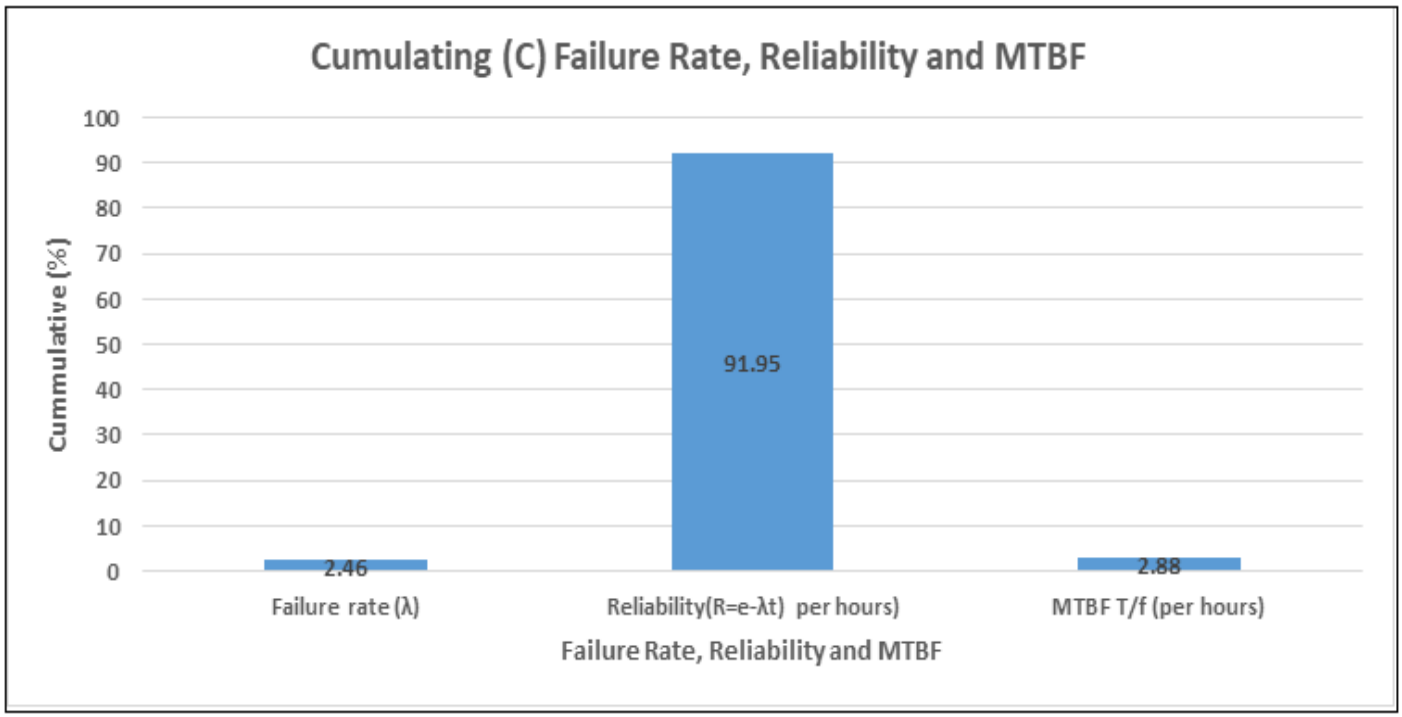

Fig 7 Cumulating (C) Failure Rate, Reliability, and MTBF

Discussion: Three agents (Agent Controller, Server Agent, and Client Agents) were used with twenty 20 nodes during the twenty days test of the framework. The test was done five hours every day for twenty days with the assumption that all the twenty nodes are in good condition. Figure 6, shows the total number of nodes used for the first test and it also shows the nodes that are alive and the ones that have failed during the five hours test. It shows the number of nodes that were alive throughout the test and each of the nodes that are still connected and stored in the database. The *failure rate (λ) of the system* for this test was computed and it shows the cumulated failure rate of *(λ)* = **0.02455** and the *reliability of R (t)* = 92%. This shows that the framework is reliable having computed the reliability rate of the system.

VI. CONCLUSION

This research work presents mobile agents as a solution for network fault detection systems. This work provides a system testing and a prototype implementation of a proactive fault detection solution using mobile agents. Correspondingly, the results from the test period of the fault-detection system model showed that the theory of reliability can be used to verify that this research can determine that the use of mobile agents is viable if properly deployed in an embedded network system. It is capable of providing a reliable, dependable, and always-available network detection system to organizations, industries, banks, and every other social sector that hinges on computer network in delivering their services.

REFERENCES

[1]. Zhang, J., Song, J., Hu, X., & Li, M. (2022). A mobile agent-based fault detection approach for wireless sensor networks. Ad Hoc Networks, 126, 103907.

[2]. Anand, V., & Devaraj, R. (2021). Mobile Agent based Fault Detection in Wireless Sensor Networks. Journal of King Saud University-Computer and Information Sciences.

[3]. Tripathi, D. S., & Gupta, S. (2020). Fault detection in wireless sensor networks using mobile agents. Wireless Personal Communications, 114(2), 737-756.

[4]. Singh, M., & Dave, M. (2020). Mobile agent-based fault detection and recovery mechanism for wireless sensor networks. Wireless Personal Communications, 114(3), 1319-1341.

[5]. Chen, Y., He, S., Li, L., & Li, Q. (2019). Mobile agent-based distributed fault detection in wireless sensor networks. Journal of Ambient Intelligence and Humanized Computing, 10(8), 3247-3261.

[6]. Choudhary, P., & Singh, A. K. (2018). A review on mobile agent based fault detection and recovery mechanisms in wireless sensor networks. In 2018 3rd International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU) (pp. 1-5). IEEE.

[7]. Maria Zubair and Umar Manzoor (2016). Mobile Agent based Network Management Applications and Fault-Tolerance Mechanisms. The Sixth International Conference on Innovative Computing Technology (INTECH 2016)

[8]. Ademaj, A. (2002). Slightly-Off-Specification Failures in the Time-Triggered Architecture. In Proceedings of the Seventh IEEE International Workshop on High-Level Design Validation and Test (HLDVT'02), Cannes, France, 7-12.
[9]. Ademaj, A., Sivencrona, H., Bauer, G., & Torin, J. (2003). Evaluation of Fault Handling of the Time-Triggered Architecture with Bus and Star Topology. In Proceedings of the 2003 IEEE International Conference on Dependable Systems and Networks (DSN'03), San Francisco, California, 123-132.
[10]. Adeosun, O. O. (2011). Development of an Enhanced Model for Internet System Availability Using Modular Redundancy (Unpublished doctoral thesis). Department of Computer Science and Engineering, Obafemi Awolowo University, Ile-Ife, 27-28.
[11]. Barborak, M., Dahbura, A., & Malek, M. (1993). The Consensus Problem in Fault-Tolerant Computing. ACM Computing Surveys, 25(2), 171-220.
[12]. Chandra, T. D., & Toueg, S. (1991). Unreliable Failure Detectors for Asynchronous Systems (Preliminary version). In 10th Annual ACM Symposium on Principles of Distributed Computing, 325-340.
[13]. Curtis, H., & France, R. (1999). Time-Triggered Protocol (TTP/C): A Safety-Critical System Protocol. EE382C Literature Survey, 10-24.
[14]. Dilger, E., Uhrer, T. F., Muller, B. M., & Poledna, S. (1998). The X-by-Wire Concept: Time-Triggered Information Exchange and Fail Silence Support by New System Services. SAE Conference.
[15]. Dobel, B., Hartig, H., & Engel, M. (2012). Operating System Support for Redundant Multithreading. In Proceedings of the Tenth ACM International Conference on Embedded Software, 83-92.
[16]. Elmenreich, W., & Ipp, R. (2001). Introduction to TTP/C and TTP/A. Vienna University of Technology, InstitutfürTechnischeInformatik, Vienna, Austria.
[17]. Fischer, M. J., Lynch, N. A., & Paterson, P. S. (1985). Impossibility of distributed consensus with one faulty process. Journal of the ACM, 32(2), 374-382.
[18]. George, A. R. (2008). Software modulated fault tolerance. (Doctoral dissertation, Princeton University).
[19]. George, A. P., & Barbara, J. P. (2003). Automotive vehicle safety. Technology and Engineer CRC Press, 9-10.
[20]. Helmer, G., Wong, J. S. K., Honavar, V., Miller, L., & Wang, Y. (2003). Lightweight agents for intrusion detection. The Journal of Systems and Software, 67, 109-122.
[21]. Kelvin, H. (2009). Introduction to TTP- Time-Triggered protocol (Seminar paper). Chemnitz University of Technology, pages 2-5.
[22]. Knoll, G., Suri, N., & Bradshaw, J. M. (2002). Path-based security for mobile agents. Notes in Theoretical Computer Science, 58(2), 16.
[23]. Kopetz, H. (2001). A comparison of TTP/C and FlexRay. Technische UniversitätWie, 5.
[24]. Kopetz, H. (2001). REAL-TIME SYSTEMS Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers, 1-5.
[25]. Kopetz, H., & Bauer, G. (2002). The time-triggered architecture. In Proceedings of the IEEE Special ISSUE on Modeling and Design of Embedded Software, 6.
[26]. Kopetz, H., & Bauer, G. (2003). The time-triggered architecture. In Proceedings of the IEEE, 91(1), 112-126.
[27]. Lamport, L., Shostak, R., & Pease, M. (1982). The Byzantine Generals Problem. ACM Transactions on Programming Language Systems, 4(3), 382-401.
[28]. Manju, N., & Jayanthi, J. (2013). An Effective Verification and Validation Strategy for Safety-Critical Embedded Systems. International Journal of Software Engineering & Applications (IJSEA), 4(2).
[29]. McCarthy, M. (2003). Fault-Tolerant. Tech Target, 3(1), 13-21.
[30]. Minsky, Y., Renesse, R., Schneider, F. B., & Stoller, S. D. (1996). Cryptographic Support for Fault-Tolerant Distributed Computing. Proceedings of the 7th ACM SIGOPS European Workshop, 109-114.
[31]. Olubosi, F. (2005). Recovering from Information System Failures. Vanguard Newspapers Online Edition, August 12.
[32]. Payal, B., & Mukesh, K. (2013). A Detailed Anatomization of Mobile Agents. International Journal of Science and Research (IJSR), 2(11), 1-7.
[33]. Pfeifer, H. (2000). Formal Verification of the TTP Group Membership Algorithm. In IFIP TC6/WG6 International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing, and Verification, FORTE/PSTV 2000 (pp. 3-18). Pisa, Italy.
[34]. Raman, C. V., & Atul, N. (2005). A Hybrid Method to Intrusion Detection System Using HMM. Proceedings of ICDCIT Workshop, 4-9.
[35]. Ross. C., Lee P. A. and Anderson T. (2003): "Providing Fault-Tolerant Call-Control in the IMS using the Rserpool Architecture." In Proceedings of the 8th International Workshop on Mobile Multimedia Communication, Munch, Germany, pp. 67-72.
[36]. Schlichting R.D and Schneider F.B. (1983): ''Fail-stop processors: an approach to designing fault-tolerant computing systems,'' ACM Transactions on Computer Systems, Vol.1, No.3, pp.222-238, ACM Press.
[37]. Seidel F. (2009): "X-By-Wire, Operation Systems", Presented at the Seminar on Transportation Systems, Chemnitz University of Technology.
[38]. Sunil G., Harsh K. V. and Sangal A. L. (2012): "Analysis and Removal of Vulnerabilities in Masquerading Attack in Wireless Sensor Networks", International Journal of Computer Technology and Electronics Engineering (IJCTEE), Vol.2, Issue 3.
[39]. Verissimo P. and Rodrigues L. (2001): "Distributed Systems for System Architects", Kluwer Academic Publishers, pp.1.

[40]. Wensley J. H., Lamport L., Goldberg J., Green M.W., Levitt K.N., Melliar-Smith P. M., Shostak R.E., and Weinstock C. B. (1978): ‘‘SIFT: Design and analysis of a fault-tolerant computer for aircraft control,’’ Proceedings of the IEEE, Vol.66, No.10, pp.1240-1255.
[41]. White T. and Pagurek B. (1998): “Towards Multi-Agent Problem Solving in Networks”, Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS ’98), pp.333-340.
[42]. Bathula, B. G., Sinha, R. K., Chiu, A. L., and Woodward, S. L. (2018). Routing and regenerator planning in a carrier’s core reconfigurable optical network. Journal of Optical Communications and Networking, 10(2), A196–A205.
[43]. Jhawar, R. and Piuri, V. (2017). Fault tolerance and resilience in cloud computing environments. In Computer and Information Security Handbook (Third Edition), Elsevier, pp. 165–181.
[44]. Wang, Z. (1989). Model of network faults. Integrated Network Management, Amsterdam: North-Holland.
[45]. Adetokunbo M and Ojieabu Clement E (2017). Automated Fault Detection and Identification System for Computer Networks.
[46]. Castro, M., & Liskov, B. (1999). Practical Byzantine fault tolerance. In Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI), pp. 173-186.
[47]. Ongaro, D., & Ousterhout, J. (2014). In search of an understandable consensus algorithm. Proceedings of the 2014 USENIX Annual Technical Conference, pp. 305-319.
[48]. Vukolić, M. (2015). The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. IEEE Communications Magazine, 53(9), 71-77.
[49]. Hopcroft, J. E., & Karp, R. M. (1973). An n5/2 algorithm for maximum matching in bipartite graphs. SIAM Journal on Computing, 2(4), 225-231.
[50]. Liu, Y., & Li, B. (2011). A survey of network fault diagnosis based on graph theory. Mathematical Problems in Engineering, 2011, 1-19.
[51]. Coates, M., & Marbach, P. (2002). Network tomography: Recent developments. Statistical Science, 17(4), 465-480.
[52]. Paul Barford, Yan Chen, Anup Goyal, Zhichun Li, Vern Paxson and Vinod Yegneswaran. “Employing Honeynets For Network Situational Awareness”, In Series: Advances in Information Security, Springer, 2009.
[53]. A. Dusia | A. S. Sethi (2018). Probe Generation for Active Probing. Article in International Journal of Network Management, Wiley, 2018.