**A Feasible Framework for software defined network to self-heal the broken nodes in the clusters**

**Abstract:**

Software defined network dynamic model for data broadcasting and self healing is a big issue. So in this we propose 4 novel model approaches to overcome this issue. First all nodes will be divided into 4 clusters by using Random non uniform Manet formation. And after framing by using expect maximization algorithm all clusters will be allocated energies and the node from each cluster which is having highest energy will act as a node head. Once the broadcast starts central repository checks the source cluster and destination cluster and source cluster path will be framed with nearest nodes which are above aggregated energy and the end node will be edge(cluster head) for source cluster and other end receiver will be destination cluster head before data reaches to exact destination.

**Introduction:**

Software-Defined Networking (SDN) is an emerging networking paradigm that gives hope to change the limitation of current network infrastructures.

First, it breaks the vertical integration by separating the network’s control logic (the control plane) from the underlying routers and switches that forward the traffic (the data plane).

Second, with the separation of the control and data planes, network switches become simple forwarding devices and the control logic is implemented in a logically centralized controller (or network operating system1), simplifying policy enforcement and network (re)configuration and evolution.

**Literature Survey:**

1. **On Swarm Intelligence Inspired Self-Organized Networking: Its Bionic Mechanisms, Designing Principles and Optimization Approaches**

In this paper, we survey different aspects of bio inspired mechanisms and examine various algorithms that have been applied to artificial SON systems. The existing well-known bio-inspired algorithms such as pulse-coupled oscillators (PCO)-based synchronization, ant- and/or bee-inspired cooperation and division of labour , immune systems inspired network security and Ant Colony Optimization (ACO)-based multipath routing have been surveyed and compared. The main contributions ofthis survey include 1) providing principles and optimization approaches of variant bio-inspired algorithms, 2) surveying and comparing critical SON issues from the perspective of physical layer, Media Access Control (MAC)-layer and network-layer operations, and 3) discussing advantages, drawbacks, and further design challenges of variant algorithms, and then identifying their new directions and applications.

1. **Artificial Intelligence-Based Techniques for Emerging Heterogeneous Network: State of the Arts, Opportunities, and Challenges**

A detailed taxonomy of the related AI-based techniques of Het Nets is also shown by discussing the pros and cons for various AI-based techniques for different problems in Het Nets. Opening research issues and pending challenges are concluded as well, which can provide guidelines for future research work.

1. **A Survey of Self Organization in Future Cellular Networks**

This article surveys the literature over the period of the last decade on the emerging field of self organization as applied to wireless cellular communication networks. Self organisation has been extensively studied and applied in adhoc networks, wireless sensor networks and autonomic computer networks; however in the context of wireless cellular networks ,this is the first attempt to put in perspective the various efforts inform of a tutorial/survey. We provide a comprehensive survey of the existing literature, projects and standards in self organizing cellular networks. Additionally, we also aim to present a clear understanding of this active research area, identifying a clear taxonomy and guidelines for design of self-organizing mechanisms. We compare strength and weakness of existing solutions and highlight the key research areas for further development. This paper serves as a guide and a starting point for anyone willing to delve into research on self-organization in wireless cellular communication networks.

**Existing System:**

* **No proper cluster framing mechanism with proper energies allocation.**
* **Cluster head mechanism with shortest path finding is not there.**
* **Making the cluster head as edge node is applicable**
* **Shortest path with respect aggregated energies threshold is rare.**

**Proposed System:**

* **Cluster framing with dynamic range of energies allocation with proper central repository control.**
* **Shortest path is 2 ways one is to make source-cluster head(edge)-cluster head(edge other side cluster)-destination.**
* **Edge nodes are always cluster heads.**
* **Based on the threshold shortest path should be made according to the aggregated threshold. So the node which satisfies only will come into shortest path(this is second model)**

**Architecture:**

****

**Machine Learning**

**s🡨 source node**

**d🡨 destination node**

http://www.sciweavers.org/upload/Tex2Img_1524547628/render.png**🡨source cluster**

http://www.sciweavers.org/upload/Tex2Img_1524547662/render.png**🡨 destination cluster**

http://www.sciweavers.org/upload/Tex2Img_1524547758/render.png**ch🡸 GETCLUHEAD(**http://www.sciweavers.org/upload/Tex2Img_1524547899/render.png**) // cluster head**

* 🡸 **GETCLUHEAD(**http://www.sciweavers.org/upload/Tex2Img_1524548158/render.png**) // cluster head**

**Path = MAKEPATH[s,** http://www.sciweavers.org/upload/Tex2Img_1524548333/render.png, http://www.sciweavers.org/upload/Tex2Img_1524548364/render.png,**]**

**Analysis=check(PATH)**

**Expert Maximization**

* **C** 🡸 **FRAME(CLUSTER)**

http://www.sciweavers.org/upload/Tex2Img_1524568062/render.png http://www.sciweavers.org/upload/Tex2Img_1524568115/render.png**=0; http://www.sciweavers.org/upload/Tex2Img_1524568062/render.png**http://www.sciweavers.org/upload/Tex2Img_1524568185/render.png**=0;**

**For I in 1 to SIZE(C)**

* **[i]=MAX(C[i]); //Cluster head**

**http://www.sciweavers.org/upload/Tex2Img_1524568185/render.png**[i]=GET ENERGY(C{I})

END FOR

**if(available)**

**ML SUCCESS**

**DISPLAY**

**BROADCAST**

**Else**

**LEARN**

**UPDATE**

**BROADCAST**

**End if**

**Random Uniform Matrix Generation**

**t1=MAKE(C1)**

**t2=MAKE(C2)**

**t3=MAKE(C3)**

**t4=MAKE(C4)**

**n1🡸RANDOM(1 - 150)**

**n2🡸RANDOM(1 - 150)**

**n3🡸RANDOM(1 - 150)**

**n4🡸RANDOM(1 - 150)**

**C1=ASSIGN ENERGIES(t1,n1)**

**C2=ASSIGN ENERGIES(t2,n2)**

**C3=ASSIGN ENERGIES(t3,n3)**

**C4=ASSIGN ENERGIES(t4,n4)**

**Ch1🡸MAXENERGY(C1)**

**Ch2🡸MAXENERGY(C2)**

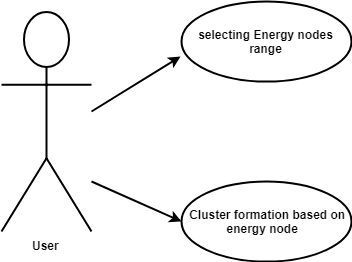
**Ch3🡸MAXENERGY(C3)**

**Ch4🡸MAXENERGY(C4)**

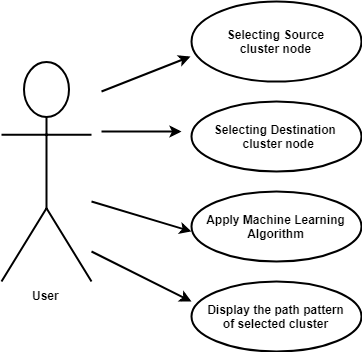
**DISPLAYACK(C1,C2,C3,C4)**

**Use case diagram**

**Use case 1:**

****

**Use case 2:**

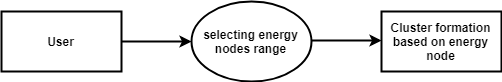
****

**Data Flow Diagram**

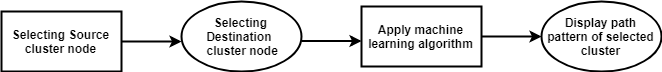
A data flow diagram is a graphical representation of the "flow" of data through an [information system](http://en.wikipedia.org/wiki/Information_system), modeling its *process* aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the [visualization](http://en.wikipedia.org/wiki/Data_visualization) of [data processing](http://en.wikipedia.org/wiki/Data_processing) (structured design).

The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of the input data to the system, various processing carried out on these data, and the output data is generated by the system.

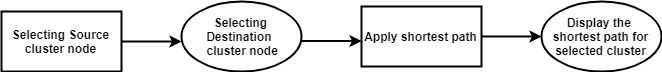
**DFD 0**

****

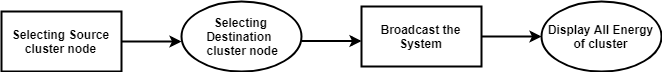
**DFD 1**

****

**DFD 2**

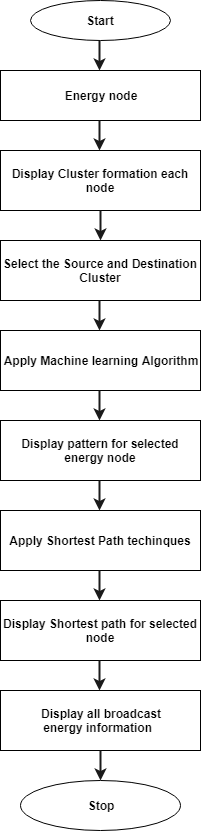
****

**DFD 3**

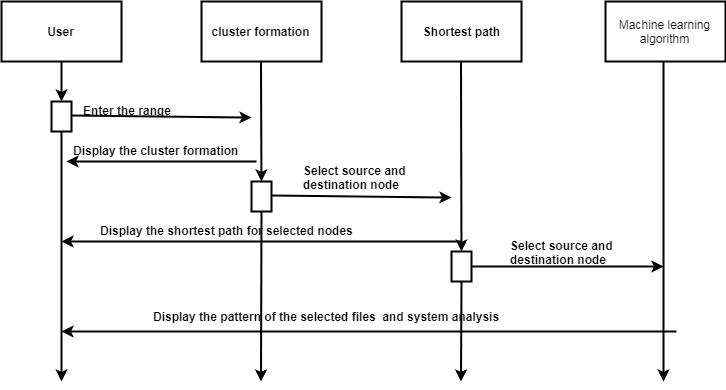
****

**Flowcharts**

A flow chart is a graphical or symbolic representation of a process. Each step in the process is represented by a different symbol and contains a short description of the process step. The flow chart symbols are linked together with arrows showing the process flow direction



**Sequence diagram:**

****

**SYSTEM REQUIREMENT SPECIFICATION**

To be used efficiently, all [computer software](http://en.wikipedia.org/wiki/Computer_software) needs certain [hardware](http://en.wikipedia.org/wiki/Computer_hardware) components or other software resources to be present on a [computer](http://en.wikipedia.org/wiki/Computer). These prerequisites are known as (computer) system requirements and are often used as a guideline as opposed to an absolute rule. Most software defines two sets of system requirements: [minimum](http://en.wikipedia.org/wiki/System_Requirements#Minimum_System_Requirements) and [recommended](http://en.wikipedia.org/wiki/System_Requirements#Recommended_system_requirements). With increasing demand for higher processing power and resources in newer versions of software, system requirements tend to increase over time. Industry analysts suggest that this trend plays a bigger part in driving upgrades to existing computer systems than technological advancements.

**Non functional requirements**

Non functional requirements are the functions offered by the system. It includes time constraints and constraints on the development process and standards. The non functional requirements are as follows:

* **Speed:** The system should process the given input into output within appropriate time.
* **Ease of use:** The software should be user friendly. Then the customers can use easily,

so it doesn’t require much training time.

* **Reliability:** The rate of failures should be less then only the system is more reliable
* **Portability**: It should be easy to implement in any system.

**Specific Requirements**

The specific requirements are:

* **User Interfaces:** The external users are the clients. All the clients can use this software for indexing and searching.
* **Hardware Interfaces:** The external hardware interface used for indexing and searching is personal computers of the clients. The PC’s may be laptops with wireless LAN as the internet connections provided will be wireless.
* **Software Interfaces:** The Operating Systems can be any version of Windows.
* **Performance Requirements:** The PC’s used must be atleast Pentium 4 machines so that they can give optimum performance of the product.

**Software requirements**

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application.

These requirements or prerequisites are generally not included in the software installation package and need to be installed separately before the software is installed.

* Java1.4 or higher
  + Java Swing – front end
  + Networking-Socket programming
* Windows 98 or higher-Operating System

**Hardware requirements**

The most common set of requirements defined by any [operating system](http://en.wikipedia.org/wiki/Operating_system) or [software application](http://en.wikipedia.org/wiki/Software_application) is the physical computer resources, also known as [hardware](http://en.wikipedia.org/wiki/Computer_hardware), A hardware requirements list is often accompanied by a [hardware compatibility list](http://en.wikipedia.org/wiki/Hardware_compatibility_list), especially in case of operating systems. An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application. The following sub-sections discuss the various aspects of hardware requirements.

All computer [operating systems](http://en.wikipedia.org/wiki/Operating_system) are designed for a particular [computer architecture](http://en.wikipedia.org/wiki/Computer_architecture). Most software applications are limited to particular operating systems running on particular architectures. Although architecture-independent operating systems and applications exist, most need to be recompiled to run on a new architecture.

The power of the [central processing unit](http://en.wikipedia.org/wiki/Central_processing_unit) (CPU) is a fundamental system requirement for any software. Most software running on [x86 architecture](http://en.wikipedia.org/wiki/X86_architecture) define processing power as the [model](http://en.wikipedia.org/wiki/List_of_microprocessors) and the [clock speed](http://en.wikipedia.org/wiki/Clock_rate) of the CPU. Many other features of a CPU that influence its speed and power, like [bus speed](http://en.wikipedia.org/wiki/Front_side_bus), [cache](http://en.wikipedia.org/wiki/CPU_cache), and [MIPS](http://en.wikipedia.org/wiki/Instructions_per_second) are often ignored. This definition of power is often erroneous, as [AMD](http://en.wikipedia.org/wiki/Advanced_Micro_Devices) [Athlon](http://en.wikipedia.org/wiki/Athlon) and [Intel](http://en.wikipedia.org/wiki/Intel) [Pentium](http://en.wikipedia.org/wiki/Pentium_%28brand%29) CPUs at similar clock speed often have different throughput speeds.

* + - * 10GB HDD(min)
      * 128 MB RAM(min)
      * Pentium P4 Processor 2.8Ghz(min)

**Overview of technologies**

The technologies used in secure and efficient data transmission is described as below:

**History of Java**

Java language was developed by James Gosling and his team at sun Microsystems and released formally in 1995. Its former name is oak. Java Development Kit 1.0 was released in 1996 to popularize java and is freely available on Internet.

**Overview of Java**

Java is loosely based on c++ syntax, and is meant to be Object-Oriented Structure of java is midway between an interpreted and a compiled language. The java compiler into ByteCodes, which are secure and portable across different platforms, compiles Java programs. These byte codes are essentially instructions encapsulated in single type, to what is known as java virtual machine (JVM), which resides in standard browser.

JVM is available for almost all OS. JVM converts these byte codes into machine specific instructions at runtime. Java is actually a platform consisting of three components:

* Java programming language.
* Java library of classes and interfaces.
* Java Virtual Machine

**Features of Java**

* Java is a simple language. It does not make use of pointers, function overloading etc,.
* Java is object-oriented language and supports encapsulation, inheritance, Polymorphism and dynamic binding, but does not support multiple inheritance.
* Everything in java is an object except some primitive data types.
* Java is portable.
* It is an architecture neutral that is java programs once compiled can be executed on any machine that is enabled.
* Java is distributed in its approach and used for Internet programming.
* Java is robust, secured, high performing and dynamic in nature.
* Java supports multithreading. Therefore different parts of the program can be executed at the same time.

**Java Database Connectivity (JDBC)**

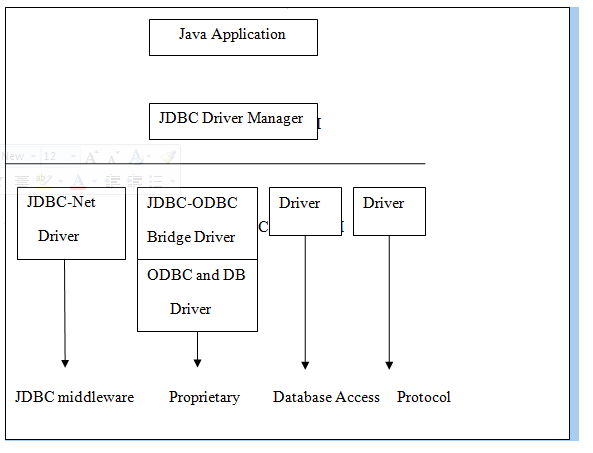
In an effort to set an independent database standard API for Java; Sun Microsystems developed Java Database Connectivity, or JDBC. JDBC offers a generic SQL database access mechanism that provides a consistent interface to a variety of RDBMSs. This consistent interface is achieved through the use of “plug-in” database connectivity modules, or drivers. If a database vendor wishes to have JDBC support, he or she must provide the driver for each platform that the database and Java run on.

To gain a wider acceptance of JDBC, Sun based JDBC’s framework on ODBC. As you discovered earlier in this chapter, ODBC has widespread support on a variety of platforms. Basing JDBC on ODBC will allow vendors to bring JDBC drivers to market much faster than developing a completely new connectivity solution.

**JDBC drivers**

There are four types of JDBC drivers. They are:

* JDBC-ODBC bridge plus ODBC driver
  + JDBC-Net all-Java driver
  + Native-API partly-Java driver
  + Native-protocol all-Java driver

Figure 7.7: JDBC driver types.

**Java RMI**

Java Remote Method Invocation (Java RMI) enables the programmer to create distributed Java technology-based to Java technology-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts. RMI uses object serialization to marshal and unmarshal parameters and does not truncate types, supporting true object-oriented polymorphism

**Java Socket Programming**

URLs and URL Connections provide a relatively high-level mechanism for accessing resources on the Internet. Sometimes your programs require lower-level network communication, for example, when you want to write a client-server application.

In client-server applications, the server provides some service, such as processing database queries or sending out current stock prices. The client uses the service provided by the server, either displaying database query results to the user or making stock purchase recommendations to an investor. The communication that occurs between the client and the server must be reliable. That is, no data can be dropped and it must arrive on the client side in the same order in which the server sent it.

TCP provides a reliable, point-to-point communication channel that client-server application on the Internet use to communicate with each other. To communicate over TCP, a client program and a server program establish a connection to one another. Each program binds a socket to its end of the connection. To communicate, the client and the server each reads from and writes to the socket bound to the connection.

**What Is a Socket?**

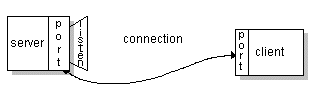
Normally, a server runs on a specific computer and has a socket that is bound to a specific portnumber. The server just waits, listening to the socket for a client to make a connection request.

On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.



Socket connection request

If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



Socket connection

On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server. The client and server can now communicate by writing to or reading from their sockets.

A *socket* is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent.

An endpoint is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints. That way you can have multiple connections between your host and the server. The java.net package in the Java platform provides a class, Socket, that implements one side of a two-way connection between your Java program and another program on the network.

The Socket class sits on top of a platform-dependent implementation, hiding the details of any particular system from your Java program. By using the java.net.Socket class instead of relying on native code, your Java programs can communicate over the network in a platform-independent fashion. Additionally, java.net includes the Server Socket class, which implements a socket that servers can use to listen for and accept connections to clients.

This shows how to use the Socket and Server Socket classes. If we are trying to connect to the Web, the URL class and related classes (URL Connection, URL Encoder) are probably more appropriate than the socket classes. In fact, URLs are a relatively high-level connection to the Web and use sockets as part of the underlying implementation. See Working with URLs for information about connecting to the Web via URLs.

**Packages**

One of the most innovative features of java is packages. The packages both a naming and a visibility control mechanism we can define classes inside a package that are not accessible by code outside the package.It can define the class members that are only exposed to the other members of the same package. Java uses file system directories to store packages. For example the .class files for any classes you declare to be part of My Package must be stored in the directory called MyPackage remember that cases significant and directory name must match the package name exactly.

A package hierarchy must be reflected in the file system of your java development system. For example the package declared as -package java.awt.image; needs to be stored in java\awt\image in a windows environment.

**Java.lang package**

The java package, java.lang contains fundamental classes and interfaces closely tied to the language and run time system which includes the root classes that form the class hierarchy, types tied to the language definition, basic exceptions, math functions, threading, security functions as well as some information on the underlying native system.

**Java.util**

Data structures that aggregate objects are the focus of the Java.util package included in the packet is the collections API and organized data structure hierarchy influenced heavily by design pattern consideration.

**Java .security**

It provides the classes and interfaces for security framework. It includes classes that implement an easily configurable, fine grained access control security architecture. The packages also supports a generation and storage of cryptographic public key pairs. Finally this package provides classes that support signed/guarded objects and secure random number generation.

**Swings**

Swing is a widget toolkit for Java. It’s a part of sun Microsystems Java foundation classes-API for providing graphical user interface for Java programs. Swing was developed to provide a more sophisticated set of GUI components than the earlier abstract window toolkit. Swings provide a native look and feel that emulates look and feel of several look and feel unrelated to the underlying platform. Swings introduced a mechanism that allows the look and feel of every component in an application to be altered without making substantial changes to the application code. The introduction of support for a plugable look and feel allows swing components to emulate for the appearance of native components while still retaining the benefits of platform independence. The above feature also makes it easy to make an application written in swing look very different from native programs if desired.

**Look and feel**

In software design look and feel is used in respect of GUI and comprises of its design, including elements such as colors, shapes, layout and typefaces(the “LOOK”) as well as the behavior of dynamic elements such as button, boxes and menus(the “FEEL”). The term look and feel is used in reference to both software and websites.

**TESTING**

Testing is a critical element which assures quality and effectiveness of the proposed system in (satisfying) meeting its objectives. Testing is done at various stages in the System designing and implementation process with an objective of developing an transparent, flexible and secured system. Testing is an integral part of software development. Testing process, in a way certifies, whether the product, that is developed, complies with the standards, that it was designed to. Testing process involves building of test cases, against which, the product has to be tested.

**Test objectives**

* Testing is a process of executing a program with the intent of finding an error.
* A good case is one that has a high probability of finding an undiscovered error.
* A successful test is one that uncovers a yet undiscovered error. If testing is conducted successfully (according to the objectives) it will uncover errors in the software. Testing can't show the absences of defects are present. It can only show that software defects are present.

**Testing principles**

Before applying methods to design effective test cases, a software engineer must understand the basic principle that guides software testing. All the tests should be traceable to customer requirements.

**Testing design**

Any engineering product can be tested in one of two ways:

**White box Testing**

This testing is also called as glass box testing. Inthis testing, by knowing the specified function that a product has been designed to perform test can be conducted that demonstrates each function is fully operation at the same time searching for errors in each function.

it is a test case design method that uses the control structure of the procedural design to derive test cases.

**Black box Testing**

Inthis testing by knowing the internal operation of a product, tests can be conducted to ensure that "all gears mesh", that is the internal operation performs according to specification and all internal components have been adequately exercised. It fundamentally focuses on the functional requirements of the software.

The steps involved in black box test case design are:

* Graph based testing methods
* Equivalence partitioning
* Boundary value analysis
* Comparison testing

**Testing strategies**

A software testing strategy provides a road map for the software developer. Testing is a set of activities that can be planned in advanced and conducted systematically. For this reason a template for software testing a set of steps into which we can place specific test case design methods should be defined for software engineering process.

**Any software testing strategy should have the following characteristics:**

* 1. Testing begins at the module level and works outward toward the integration of the entire computer based system.
  2. Different testing techniques are appropriate at different points in time.
  3. The developer of the software and an independent test group conducts testing.
  4. Testing and debugging are different activities but debugging must be accommodated in any testing strategy.

**Levels of Testing**

Testing can be done in different levels of SDLC. They are:

**Unit Testing**

The first level of testing is called unit testing. Unit testing verifies on the smallest unit of software designs-the module. The unit test is always white box oriented. In this, different modules are tested against the specifications produced during design for the modules. Unit testing is essentially for verification of the code produced during the coding phase, and hence the goal is to test the internal logic of the modules. It is typically done by the programmer of the module. Due to its close association with coding, the coding phase is frequently called “coding and unit testing.” The unit test can be conducted in parallel for multiple modules.

The Test cases in unit testing are as follows:

Table I: Unit Test Case 1

|  |  |
| --- | --- |
| Test Case ID | Unit Test Case 1 |
| Description | Nodes created successfully |
| Input | Select the energy range |
| Expected output | Cluster formation |
| Actual Result/Remarks | Got the expected output |
| Passed(?) | Yes |

Table II: Unit Test Case 2

|  |  |
| --- | --- |
| Test Case ID | Unit Test Case 2 |
| Description | Broad casting |
| Input | Select the source and destination node |
| Expected output | Display the energy info. |
| Actual Result/Remarks | Got the expected output |
| Passed(?) | Yes |

Table III: Unit Test Case 3

|  |  |
| --- | --- |
| Test Case ID | Unit Test Case 3 |
| Description | Apply machine learning algorithm |
| Expected output | Display the pattern for selected nodes |
| Actual Result/Remarks | Got the expected output |
| Passed(?) | Yes |

Table IV: Unit Test Case 4

|  |  |
| --- | --- |
| Test Case ID | Unit Test Case 4 |
| Description | Find out shortest path |
| Input | Select the source and destination cluster |
| Expected output | Display the shortest path |
| Actual Result/Remarks | Working as required |
| Passed(?) | Yes |

**9.5.2 Integration Testing**

The second level of testing is called integration testing. Integration testing is a systematic technique for constructing the program structure while conducting tests to uncover errors associated with interfacing. In this, many tested modules are combined into subsystems, which are then tested. The goal here is to see if all the modules can be integrated properly.

There are three types of integration testing:

* + - *Top-Down Integration*: Top down integration is an incremental approach to construction of program structures. Modules are integrated by moving downwards throw the control hierarchy beginning with the main control module.
    - *Bottom-Up Integration*: Bottom up integration as its name implies, begins Construction and testing with automatic modules.
    - *Regression Testing*: In this contest of an integration test strategy, regression testing is the re execution of some subset of test that have already been conducted to ensure that changes have not propagated unintended side effects.

**9.5.3 Functional test**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Table X: Functional Testing items

|  |  |
| --- | --- |
| Valid Input | Identified classes of valid input must be accepted. |
| Invalid Input | Identified classes of invalid input must be rejected. |
| Functions | Identified functions must be exercised. |
| Output | Identified classes of application outputs must be exercised. |

***Systems/Procedures:*** Interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

**9.6 Validation testing**

At the culmination of integration testing, software is completely assembled as a package; interfacing errors have been covered and corrected, and final series of software tests-validating testing may begin. Validation can be defined in many ways, but a simple definition is that validation succeeds when software functions in a manner that can be reasonably expected by customers. Reasonable expectation is defined in the software requirement specification- a document that describes all user visible attributes of the software. The specification contains a section title “validation criteria”. Information contained in that section forms the basis for validation testing approach

**9.7 Alpha testing**

It is virtually impossible for a software developer to forsee how the customer will really use a program. Instructions for use may be misinterpreted; strange combination of data may be regularly used and output that seemed clear to the tester may be unintelligible to a user in field.

When custom software is built for one customer, a series of acceptance tests are conducted to enable the customer to validate all requirements by the end user rather than system developer and acceptable test can range from an informal “test drive” to a planned and systematically executed series of tests. In fact, acceptance testing can be conducted over a period of weeks or months, thereby uncovering cumulative errors that might degrade the system over time. If software is developed as a product to be used by many customers, it is impractical to perform formal acceptance test with each one. Most software product builders use a process called alpha and beta testing to uncover errors that only the end user seems able to find.

A customer conducts the alpha test at the developer’s site. The software is used in a natural setting with the developer “Looking over the shoulder” of the user and recording errors and usage problems. Alpha tests are conducted in controlled environment.

**9.8 Beta testing**

The beta test is conducted at one or more customer sites by the end user of the software. Unlike alpha testing, the developer is generally not present. Therefore, the beta test is a “live” application of the software in an environment that cannot be controlled by the developer. The customer records all problems that are encountered during beta testing and reports these to the developer at regular intervals. As a result of problems reported during beta test, the software developer makes modification and then prepares for release of the software product to the entire customer base.

**9.9 System Testing and Acceptance Testing**

System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Include recovery testing during crashes, security testing for unauthorized user, etc.

Acceptance testing is sometimes performed with realistic data of the client to demonstrate that the software is working satisfactorily. This testing in FDAC focuses on the external behavior of the system

**Conclusion:**

Based on the current existing scenario with the current work the conclusion is to make the edge node as cluster head for sure for broad casting. The proper broadcasting is getting done only with shortest path with which nodes are having more than aggregated threshold value. So to reduce duplicate broadcast this work is updated with machine learning model for updation of past paths.