Implementing a Fault-Tolerant System
Based on Replication and Access Control

Jian Zhuang
MSc in Distributed Multimedia Systems

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student) _______________________________
Summary

The aim of the project is to create a fault tolerant system based on replication and security technology. All requirements for the project have been achieved.

A literature survey reviews the existing papers in this domain and various theories in fault tolerance and algorithms of secure communication are explored. Primary-backup fault tolerant model, ACL for security and communication using SSL are implemented. And graphic user interfaces are built to present the output of the system.
Acknowledgements

I would like to thank the following people for their support and advice during my work on the project.

Prof. Xu, my project supervisor, for his invaluable help and guidance.

Mr. Owen Johnson, my assessor, for his invaluable advice.

Last, but by no means least, I would like to thank my classmates and friends, who gave me everlasting support.
## Contents

**Chapter 1 Introduction** ........................................................................................................... 1  
1.1 The aims ................................................................................................................................. 1  
1.2 Overview of the system .......................................................................................................... 2  
1.3 Required Research ............................................................................................................... 3  
1.4 Objectives ............................................................................................................................. 3  
  1.4.1 Minimum Requirements ................................................................................................. 3  
  1.4.2 Intermediate requirements ............................................................................................ 4  
  1.4.3 Advanced requirements ................................................................................................. 4  
1.5 Project Plan ............................................................................................................................ 5  
  1.5.1 Methodology .................................................................................................................. 5  
  1.5.2 Schedule ........................................................................................................................ 6  

**Chapter 2 Background Research** ........................................................................................... 7  
2.1 Fault Tolerance ...................................................................................................................... 7  
  2.1.1 Failure Models ............................................................................................................... 7  
  2.1.2 Replication ...................................................................................................................... 8  
2.2 Security .................................................................................................................................. 11  
2.3 Analysis and Testing ............................................................................................................. 13  
2.4 Java RMI .................................................................................................................................. 14  

**Chapter 3 Design** .................................................................................................................. 15  
3.1 System Architecture ............................................................................................................. 15  
  3.1.1 Problem Description ....................................................................................................... 15  
  3.1.2 System Use Case ............................................................................................................. 16  
  3.1.3 System deployment Diagram .......................................................................................... 17  
3.2 System Modules .................................................................................................................... 18  
  3.2.1 Proxy Server .................................................................................................................... 18  
  3.2.2 Simulated Sensor ............................................................................................................ 19  
  3.2.3 Processing Node ............................................................................................................. 19  
  3.2.4 Voter System ................................................................................................................... 20
### Contents

3.2.5 Client .............................................................................................................. 20
3.2.6 Monitor ........................................................................................................ 20
3.3 Communication .................................................................................................. 20
  3.3.1 Process communication ............................................................................... 21
  3.3.2 Security Issue .............................................................................................. 22
3.4 Concurrent Programming .................................................................................. 23

#### Chapter 4 Implementation

4.1 Choice of Programming Language ................................................................. 24
4.2 Choice of Platform ........................................................................................... 25
4.3 Development Environment ............................................................................ 25
4.4 XML Library .................................................................................................... 25
4.5 Communication ................................................................................................ 26
  4.5.1 Java RMI ....................................................................................................... 26
  4.5.2 Using SSL over RMI .................................................................................... 28
4.6 System Modules ............................................................................................... 30
  4.6.1 Simulated Sensor .......................................................................................... 30
  4.6.2 Processing Node ........................................................................................... 32
  4.6.3 Fault Tolerant Voter System ........................................................................ 35
  4.6.4 Proxy Server ................................................................................................ 38
  4.6.5 Client & Monitor Programs ......................................................................... 40
4.7 Compile and Execution .................................................................................... 41

#### Chapter 5 Evaluation

5.1 Requirements .................................................................................................... 42
5.2 Evaluation ......................................................................................................... 42
  5.2.1 Debug ........................................................................................................... 43
  5.2.2 Functionality testing .................................................................................... 43
  5.2.3 Fault-injection testing .................................................................................. 45
5.3 Experimental results ......................................................................................... 46
5.4 Analysis .............................................................................................................. 47
  5.4.1 Performance ................................................................................................ 47
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4.2 Recovery</td>
<td>48</td>
</tr>
<tr>
<td>Chapter 6 Conclusions and Future Directions</td>
<td>49</td>
</tr>
<tr>
<td>6.1 Developments</td>
<td>49</td>
</tr>
<tr>
<td>6.1.1 Fault tolerant part</td>
<td>49</td>
</tr>
<tr>
<td>6.1.2 Security</td>
<td>50</td>
</tr>
<tr>
<td>6.1.3 Performance issue</td>
<td>50</td>
</tr>
<tr>
<td>6.1.4 Scalability</td>
<td>50</td>
</tr>
<tr>
<td>6.1.5 User interface</td>
<td>50</td>
</tr>
<tr>
<td>Bibliography</td>
<td>51</td>
</tr>
<tr>
<td>Appendix A Personal Reflection</td>
<td>53</td>
</tr>
<tr>
<td>Appendix B Objectives and Deliverables Form</td>
<td>54</td>
</tr>
<tr>
<td>Appendix C Marking Scheme and Mid Report header sheet</td>
<td>56</td>
</tr>
<tr>
<td>Appendix D Source Code Guide</td>
<td>59</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 The aims

Any hardware or software failure in a computer system can cause system crash. If a computer system is used in some critical areas, such as operation of airplane, control of nuclear power plant, it is obvious that the crash of the computer system could be result in a catastrophe. So how to develop a reliable system is a hot topic, and a lot of effort has been taken to design and implement a reliable system which provides continuous, correct and secure services.

The term “Fault tolerance” is described the ability of a system that can respond gracefully to an unexpected hardware or software failure. There are many levels of fault tolerance, the lowest being the ability to continue operation in the event of a power failure, the higher being the ability to ensure strictly proper behavior despite certain types of fault. Many fault-tolerant computer systems replicate all operations, which means that every operation is performed on two or more duplicate systems, so if one fails the other can take over. In [1] this method is defined as Replication which is a key to the effectiveness of distributed systems, it can provide enhanced performance, high availability and fault tolerance.

Distributed systems often provide their services through insecure networks such as the Internet, in which the attackers can attack the systems using many measures such as eavesdropping, tampering, etc. So security is an important issue in design of a distributed system.

The aim of the project is to design and implement a fault-tolerant and secure system. The system is required to be implemented based on the object replication technique to achieve fault-tolerance and access control to enforce access policies to different services provided by the system. All communication between objects is via a secure channel using SSL (Secure Sockets Layer).
1.2 Overview of the system

The system consists of a series of simulated sensors that return values within a given range, depending on parameters sent to them via the proxy server program. These sensors each pass their readings on to an individual channel in a replicated processing system, which processes the sensor readings and passes output on to a fault-tolerant, multiple voter system. Acceptance testing or voting is performed by voter system on the results provided by the n-copy system and any results are logged. An adjudicated result is then passed back to the proxy server. Figure 1-1 shows the outline of the system architecture, which is adapted from the supposed architecture in the project requirements. Clients, sensors, Processing Nodes, Voting nodes are processes that can be run in a single machine or in many different machines.

Figure 1-1 The system architecture
1.3 Required Research

This project involves a thorough understanding of the two key areas, namely fault tolerance and security. Awareness of different fault types and secure communications is important when attempting to implement the system. Analyzing existing systems reveals which models and algorithms are known to be successful and which are not. Details will be discussed in Chapter 2.

1.4 Objectives

The primary objective of this project is to design and implement a fault tolerant system based on replication and access control.

The second objective is to evaluate the implemented system against the requirements with experimental analysis and testing.

1.4.1 Minimum Requirements

The minimum requirements for this project are:

1. **Implement a fault tolerant system**
   A workable communicating system is to be implemented that consists of sensor services, processing services, a voter service, a proxy server and a client. The communication of the components is via Java RMI.

2. **Implement identity based access control**
   The request from the client program should be of the form: `<userName, actionList>`. Each user of the system should have a unique username. The action list contains a set of sensor reading requests, such as “readSensor1, readSensor2”.

   An access control list is a mapping between a username and allowed actions of the user. Assume there are two user names, Alice and Bob, in the system. Table 1-1 shows an example of access control list used by the proxy server. In the example, Alice is specifically prohibited from reading the data produced by sensor 2 but is allowed to read the data produced by sensor 3.
1. No other requirements about sensor 3 are specified. Therefore, any requests from Alice to read sensor 2 will be rejected, while other requests from Alice will be accepted. On the other hand, since there are no entries for Bob in the table, all requests from Bob will be allowed.

<table>
<thead>
<tr>
<th>Identity</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Allow to read the data produced by sensor 1</td>
</tr>
<tr>
<td>Alice</td>
<td>Not allow to read the data produced by sensor 2</td>
</tr>
</tbody>
</table>

Table 1-1  An example table shows the data stored in an access control list of proxy server.

### 1.4.2 Intermediate requirements

The intermediate requirements for this project are:

1. **Implement password based authentication**
   
The capability of checking password is added. Apart from sending simple requests, the client program can also send other parameters, such as the user’s password, with the request. The proxy server can use this password to check the user’s identity.

2. **Implement secure communication**
   
The secure communication between the client and the proxy server is implemented by Java RMI over SSL.

### 1.4.3 Advanced requirements

The advanced requirements for this project are:

1. **Enhance fault tolerance part**
   
   - A multiple-voter system is implemented.
   
   - The client program is adapted to show a graphical display of the results being obtained in real-time.

2. **Enhance interface and statistics**
   
   - A GUI based monitor window is developed to show the current status of the proxy server.
   
   - Statistics data is provided to help administrators comprehend the situation and raise alerts if necessary.
1.5 Project Plan

1.5.1 Methodology

In order to achieve the aim of the project which is to develop a fault tolerant distributed system, an appropriate software life cycle should be used. From the late of 1960 when people realized the software crisis in developing software to present, many software life cycle models were invented, including “Waterfall” model, Prototyping, Spiral Model, and Unified Process.

· “Waterfall” Model

This model is widely used and considered as a conventional software life cycle. It goes through the following steps: requirements analysis, design, implementation, testing, and deployment. But for some software projects, there have been a number of criticisms of the waterfall model. In [2], two significant problems are presented:

1) Software requirements, a key factor in any software development project, are not properly tested until a working system is available.

2) A working system becomes available on late in the life cycle, thus a major design or performance problem might not be detected until the system is almost operational.

· Prototyping

This approach can be used to develop software rapidly. It helps clarify user requirements and gets feedback from the user through demonstrating the prototype of the system.

· Spiral Model

This model involves iteration of four steps cycle, namely define objectives, analyze risks, develop product, and plane next cycle. It iterates the four steps until the system meets the client requirements.

· Rational Unified Process (RUP)

This model is a use-case driven approach that uses the UML notation. It consists of four phases, namely Inception, Elaboration, Construction, Transition, 9 workflows and their iterations. RUP is an object-oriented software process, and UML is standard language for designing and documenting the software system.
In this project, some thoughts from RUP were applied for development, and UML was used to manage and document the design and implementation of the system.

### 1.5.2 Schedule

The following schedule has been made to ensure that the project can be completed successfully.

- **Literature review**
  Investigate different technologies used in this project, including concepts on fault tolerance, security, evaluation, and Java RMI.

- **System design**
  Choose a suitable fault tolerant model and appropriate secure communication algorithm for the system. The design involves five components, namely client, proxy server, stimulated sensor node, processing node and a fault tolerant voting system.

- **System implementation**
  Implement the model and algorithms chosen to the system.

- **System Evaluation**
  Objectively evaluate the system components to prove the design and implementation.

- **Write up**
  Allow adequate time to write up the project.
Chapter 2

Background Research

The project involves the four kinds of concepts or technologies, namely fault tolerance, security, Java RMI, system evaluation. In the following sections, concepts on these four areas used in this project will be discussed.

2.1 Fault Tolerance

2.1.1 Failure Models

Failure model of a server

A fault-tolerant system is defined as a system that always guarantees strictly correct behavior despite a certain number and type of fault [1]. A system is said to fail when it cannot provide the services that it promises to clients. In order to design a fault tolerant system, it is necessary to investigate types of fault.

Failure models have been developed to fully understand what a failure actually is. They are discussed in [3] and [4]. Failure classification schemes described in [3] is shown in Table 2-1

<table>
<thead>
<tr>
<th>Type of failure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crash failure</td>
<td>A server halts, but is working correctly until it halts</td>
</tr>
<tr>
<td>Omission failure</td>
<td>A server fails to respond to incoming requests</td>
</tr>
<tr>
<td>Receive omission</td>
<td>A server fails to receive incoming messages</td>
</tr>
<tr>
<td>Send omission</td>
<td>A server fails to send messages</td>
</tr>
<tr>
<td>Timing failure</td>
<td>A server’s response lies outside the specified time interval</td>
</tr>
<tr>
<td>Response failure</td>
<td>A server’s response is incorrect</td>
</tr>
<tr>
<td>Value failure</td>
<td>The value of the response is wrong</td>
</tr>
<tr>
<td>State transition failure</td>
<td>The server deviates from the correct flow of control</td>
</tr>
<tr>
<td>Arbitrary failure</td>
<td>A server may produce arbitrary responses at arbitrary times</td>
</tr>
</tbody>
</table>

Table 2-1 Different types of failures

A crash failure occurs where a server or a process stops meeting its promise, but it is working
correctly until it halts. Only one solution which is to reboot the server or reset the process can handle this failure.

An omission failure occurs where a server or a process fails to send response to a client that requests. There are two kinds of omission failures, one is Receive omission which occurs when the server does not receive incoming message, and the other is Send omission which happens when the server fails to send results to requestor.

A timing failure is a kind of failure that a server’s response can not send to requestor in a specified time interval. If a server responds too fast may cause buffer overrun in recipient, but too late, it will decrease the performance.

When a server responds incorrect messages, it will cause a Response failure. There are two kinds of response failures, namely Value failure and State transition failure.

Arbitrary failures (Byzantine failures) are the most serious failure, it is difficult to detect.

Crash failures model and omission failures model are equivalent with respect to the consensus problem. For both failure models, consensus can be achieved in the presence of any number of faulty processors. [4]

In [4], Babaoglu presents that achieving consensus for arbitrary failures is considerably more expensive than crash failure and omission failure.

Communication failures

In distributed system, faults can not only occur on faulty processes, but also in communication channel. A communication channel may suffer crash, omission, timing, and arbitrary failures. In practice, when building reliable communication, the focus is on masking crash and omission failures [3]. In distributed systems, reliable communication is established on a reliable transport protocol such as TCP, or high-level communication protocol such as Remote Procedure Calls (RPCs) and Remote Method Invocations (RMIs).

2.1.2 Replication

The key technology for providing fault tolerance in distributed system is to use replication. It is also called redundancy in [3]. In [3], it presents that there are three kinds of redundancy, namely information redundancy, time redundancy, and physical redundancy. With information redundancy,
extra information generated for original information is sent with original, and then the recipient can get correct information although some errors occur when transmitting. With time redundancy, the request can be sent more than once if needed. With physical redundancy, duplicated computers or processes are added to tolerate the hardware or software faults. In this project the physical redundancy is used to build a fault tolerant system. It will be discussed deeply in following sections.

**Replication Model**

Two kinds of models of replication for fault tolerance are discussed in [1]. They are passive or primary-backup replication (Figure 2-1) and active replication (Figure 2-2).

![Figure 2-1 The Passive Model (Adapted from [1])](image)

The passive model for fault tolerance consists of four kinds of objects which are client, front end, primary replication manager, and backup replica manager. The client sends request to front end which relays the message to primary replica manager. The replica managers provide services to clients. In normal situation, the primary replica manager processes the requests. If a request updates the data of replica manager, the primary needs send the results to backup replica mangers. After synchronization of the data, the primary responds to the front end, which then forwards to
the client. When primary fails, one of the backups will take over primary according to certain election algorithm.

In the active model, the front end multicasts the requests to all replica managers. If one replica fails, the system can still work correctly since other replicas also can provide services. No data synchronizing like in passive model should be done in active model, because all replicas receive requests and responds. But the front end needs an algorithm to get correct result from all replicas. The simple way is to pass the first respond to client, and ignore the rest from the other replicas.

Each of passive replication model and active model has its own advantages and disadvantages. The active replication can tolerate Byzantine failure. In [1], it has been discussed that the active replication system can mask up to f Byzantine failures, if the service incorporates at least 2f+1 replica managers. The group communication in active replication model is complex. Although passive replication can not mask Byzantine failures, the group communication is easier than active model.
2.2 Security

The security is a very important issue in distributed system, because the components of the system could be located in the public networks such as the Internet. Many threats can occur when communication depends on a public channel. Before correct security measures or technologies are applied, it is necessary to assess the threats and risks in distributed system. Without analysis of threats, appropriate technologies and methods could not be chosen to build a secure application.

**STRIDE Model**

In many security literatures, such as [5] and [6], STRIDE model is used to classify the security threats. It consists of 6 kinds of threats:

- **Spoofing Identity**
  Identity spoofing occurs when an attacker has known how to replay authentication, then he impersonates the valid user to access the system. For example, authentication protocols that use passwords without encrypting is easy to attack.

- **Tampering with Data**
  Data tampering is the deliberate destruction or manipulation of data. The tampering may or may not be detected until some time in the future [7].

- **Repudiability**
  Repudiability is the notion of denying that an action occurred. Denying that you received an item, when in fact you did, is an example of repudiability [7].

- **Information Disclosure**
  Information can be eavesdropped when it transmits. For example, the information without encryption is easy to collect when it is sent over the Internet.

- **Denial of Service (DoS)**
  DoS is an attack that makes the application unavailable to provide services to its users. For example, flooding with traffic to a server is a kind of DoS.

- **Elevation of Privilege**
  An elevation of privilege occurs when a user obtains privileged access to portions of the application or data that are normally inaccessible to the user. For example, an attacker could
exploit a buffer overflow vulnerability to execute malicious code with the same security context as the application process [7].

**Security Techniques**

Encryption is the process of encoding a message in such a way as to hide its contents [1]. There are two main encryption algorithms which are symmetric algorithm and asymmetric algorithm. Symmetric algorithm uses same key to encrypt and decrypt, while asymmetric uses different keys. Data Encryption Standard (DES) is an example of symmetric algorithm and RSA is a kind of asymmetric one.

Cryptography can be used to protect message integrity and confidentiality, provide authentication between pairs. A distributed system should offer a secure communication channel between processes. The secure channel should protect the communication against the threats such as Spoofing Identity, Tampering with Data, Information Disclosure, and Elevation of Privilege. Many technologies have been developed to establish a secure channel, for example, authentication methods, such as Kerberos authentication and client authentication certificate can be used to prevent spoofing identity. Sensitive information can be encrypted using hashes and digital signature and be transmitted using secure protocols, such as Secure Sockets Layer (SSL) and Internet Protocol Security (IPSec).

Another important issue in secure channel is to verify access rights. The verifying process is called **access control**. In [1] and [3], two approaches have been discussed: **Access Control List (ACL)** and **Capabilities**. In ACL approach a server maintains a list of the access rights of each user. When a user sends a request to a server, the server will check the access control list to decide whether or not to provide service to the user. But in capabilities approach no access control list is stored in a server, the user send a request with a certificate which decide the user can access the resource. The main benefit of using certificates is that a process can easily pass its certificate to another process, Certificates, however, have the drawback that they are often difficult to revoke [3].
2.3 Analysis and Testing

Analysis and testing of software is a way to ensure software quality. It covers many characteristics of a system, such as functionality, security, and performance. The analysis of the system could be divided into two types, static analysis and dynamic analysis [8]. The static analysis is based on the source code. The tester can inspect the program text to analyze the data and control flow of the system, and use some methods to mimic the execution of system. The static analysis methods that are mentioned in [8] include Graph-oriented code representation, levels of flow analysis, and configuration analysis. Some errors such as infinite loops, uninitialized variables, etc., can be found by static analysis technology, but many features of the system such as reaction time to a certain stimuli can not be gained. The only means to do this is use dynamic analysis to prove the behavior of the system. In [9], it presents that dynamic software test involves four phrases, namely modeling the software’s environment, selecting test scenarios, running and evaluating test scenarios, and measuring testing progress. In dynamic method, software monitoring techniques are utilized, such as logging events and data, check-pointing and message logging [8].

In order to effectively test distributed systems, certain test strategies should be used. In [8], some kinds of testing strategies are described, namely:

- **Statistical testing**
  
  It is based on the input data, which is not relevant to program functionality. The selected data should either special or random with uniform distribution [8].

- **Performance testing**
  
  The main aim of performance testing is to find the reasons of poor performance of a system in a given distributed system. There are three classes of performance evaluation approaches, based on measurement, simulation, and analytic models [8].

- **Fault-injection testing**
  
  Fault-injection test is often used to functionally test a prototype during the development cycle [10]. The injected faults should cover all errors that could occur in the distributed system. The errors include: [8]

  *Type of anomalies*, such as data, processing, or timing errors
Latency, referring to the time interval between insertion and observation of individual faults

Frequency, determining the average occurrence of a fault over a given time interval

2.4 Java RMI

Java Remote Method Invocation (Java RMI) enables the programmer to create distributed Java technology-based applications. Java RMI relies heavily on Java Object Serialization, which allows objects to be marshaled as a stream. Each Java RMI Server object defines an interface which can be used to access the server object outside of the current Java Virtual Machine (JVM). The interface exposes a set of methods which are indicative of the services offered by the server object. For a client to locate a server object for the first time, it looks up for a Server Object reference and invokes methods on the Server Object as if the Java RMI server object resided in the client's address space. In this project communication between the proxy server program, the sensors, the processing nodes and the voting nodes is all performed by Java RMI. Compared with Java Sockets, Java RMI hides the complexity of communication, but it is less efficient than Java Socket in performance.

In this chapter, many concepts of design and evaluation for building a fault tolerant system are discussed. In next chapter, how to utilize them to design an example system for this project will be discussed
Chapter 3

Design

The Unified Modeling Language (UML) is a graphical notation for drawing diagrams of software concepts [11]. The project employs UML to illustrate any decisions in design and implementation phase.

3.1 System Architecture

3.1.1 Problem Description

In the system, there are

- A set of simulated sensors. Each sensor generates random number within a given range.
- A set of processing nodes. Each processing node stores numbers from a specified sensor until it has 10 of them. It then sorts them using bubble sorting algorithm and calculate the average value of them.
- A fault tolerant voter system. It receives the averaged value from all processing nodes and forwards the consensus reached from average values to proxy server together with logging any results. Replication technology is used to make this subsystem fault tolerant.
- A proxy server. It is a communication channel between clients and other nodes, such as sensors, processing nodes, and voter nodes. It also enforces access control.
- A set of clients. The client can receive the readings generated by sensors
- A monitor. The monitor program can control the system and help administrator comprehend the situation of the system.

Each simulated sensor generates random numbers within a given range. The random generator is seeded with same number. The output of the each sensor is also affected by a parameter the fault rate which mimics the possibility of failure of the real sensors. Then random numbers are passed to a corresponding processing node which processes them and passed results to a fault tolerant voter system. The voter system tries to find correct values
although sensors could generate some readings out of range. Next the voter system forwards the consensus result to proxy server and logs any results. The client can read the values from a sensor via proxy server and the administrator can monitor the system status of the whole system.

### 3.1.2 System Use Case

There are two kinds of actors in the system: one representing the *User* who wished to read the values from sensors and the second representing the *Administrator* who is responsible to maintain the system. From the problem definition, the following use cases are identified:

- **Login.** The users and administrator must login the system before any operations.
- **Display readings.** The user chooses a sensor which he wishes to monitor and the readings will be displayed by a GUI program.
- **Configure Sensors.** The administrator initiates the simulated sensors.
- **Start/Stop Sensor.** The administrator can control the simulated sensors whether or not generate random numbers.
- **Monitor System.** The administrators can learn the status of the system from it.

The Figure 3-1 shows the system use cases.

![Figure 3-1 System actors and user cases](image-url)
3.1.3 System deployment Diagram

In order to explain the architecture of the system, the deployment diagram is used. The Figure 3-2 shows one possible deployment of the system.

It depicts how the five kinds of components which mentioned in section 3.1.1 can be deployed. Components can locate not only in one machine, but also in some different machines. The system has only one proxy server, but could have more than one copy of the simulated sensor, processing node, and voter node. The number of these three kinds of component depends on the hardware and software resources. In order to simplify the programming, the maximum number of each kind of component is 10.

All nodes are connected using RMI protocol and SSL is used for both authenticating user’s identity and encrypting data.
3.2 System Modules

3.2.1 Proxy Server

A proxy server is a server that sits between a client application and real server such as simulated sensor, a fault tolerant voter system. It intercepts all requests to the real server to see if it can fulfill the requests itself. If not, it forwards the request to the real server.

Proxy server has the following main purposes:

**Access control**: Proxy servers can be used to filter requests. The password and the allowed operation lists are stored in the proxy server. If proxy server receives the request from the clients, it will check the username, password, and Access Control List (ACL) to decide if the client has the permission to request a operation.

**Improve Performance**: Proxy servers can also improve performance for groups of users. This is because it saves the results from all sensors. Consider the case where both user X and user Y access the readings for simulated sensor1 through a proxy server. The sensor 1 only needs send the reading to proxy, and then the proxy server forwards the results to user X and user Y. If tens of users wish to receive results from a same sensor, this mechanism will improve performance dramatically.

**Transparency**: The proxy server stores the information of the all different nodes, including address, status, etc. So the client and monitor program do not need hold it. Because any nodes can join and leave the system any time, it is difficult for user program to keep the information fresh. When state of a node changes, it will be sent to proxy server. So the easiest way for clients to get the needed information is access the information center, proxy server.

**The elector**: The primary-backup mode is chosen to achieve the fault tolerant voter system (this will be discussed in the next section). The proxy also plays the role of the elector in the fault tolerant voter system. When a voter node joins the system, it passes its service name to proxy server. In the beginning stage, the proxy server selects a voter node as primary node when many voter nodes join the system at the same time. When the primary node crashes, the proxy will select a backup node as primary.

**Easy to maintain**: The proxy also plays a role of manager of the system. Each node in the
system sends the information of itself to the proxy server, when it starts up. Then the monitor program can get information of them from proxy server, not from all nodes in the system. So it makes the system easy to maintain.

In order to achieve the purposes of the proxy server, the following algorithms are need:

- **Access control**: As mentioned in section 2.2, two approaches, Access Control List (ACL) and Capabilities, can verify the access rights. This project employs the ACL approach. There are many methods to maintain the access list, for example, it can be stored in a database, or in a file. In this project, the access list is stored in an XML file.

- **Leasing algorithm**: Any node could crash at any time, so it is import for proxy server to know whether a node is alive or not. In this system, a leasing idea is used to achieve this purpose. The basic algorithm is this:
  
  1. A node calls the proxy server and requests a lease for period of time.
  2. The server responds back, granting a lease for a period of time
  3. During this period of time, the proxy server holds the reference of the client.
  4. When the lease expires, if the client hasn’t requested an extension, the proxy will set the reference of client unavailable and consider the client has crashed.

All nodes automatically try to renew leases as long as they are running. The proxy automatically checks the lease of the nodes. This is an easy way for proxy to maintain the information of system status.

### 3.2.2 Simulated Sensor

The main function of Simulated Sensors is to generate random numbers within a given range (50 -100) and then send the number to a corresponding processing node. The seed and the fault rate of the simulated sensor can be configured by system administrator via proxy server.

### 3.2.3 Processing Node

The processing node stores values coming form the sensor, until it has 10 of them. It then sorts this list of double numbers into ascending order using bubble sorting algorithm and calculates the average value of them. This value is then forwarded to the voters, together with
the list of 10 sensor values for logging purposes.

### 3.2.4 Voter System

The voter system tries to reach the consensus result from the values that receive from all processing nodes, and forwards to proxy server. The algorithm for reaching consensus is straightforward. It just finds the majority of the averaged values received from all processing nodes. For example, assuming the system consists of 3 sensors, 3 processing nodes, if the voter gets two identical inputs and one error input, and the voter still can output the correct value. For n-copy system, if voter gets at least \( n/2 + 1 \) good value, it can reach consensus. If it can not reach consensus, the voter system will forward the first value received from processing node.

This subsystem achieves fault tolerance using primary-backup model of replication. When a voter node starts up, it sends service name (RMI object name) to proxy server. Then the proxy server selects one from running voter nodes as primary. The processing node can contact with proxy server to find service name of primary voter node. When the primary crashes, the proxy server will select a new one from backup nodes in 10 seconds. When processing nodes find the primary node has crashed, they will contact the proxy to find service name of new one.

### 3.2.5 Client

The client is a GUI application for user to interact with system. It can display the values from a sensor, and consensus value from voter system.

### 3.2.6 Monitor

The Monitor application is also a GUI interface for administrators to control the whole system, such as start / stop sensors, set up sensors, check the status of all running nodes.

### 3.3 Communication

Message passing between components located at networked computers is a main
characteristic of the distributed system. So it is obvious that communication is at the heart of all distributed systems.

### 3.3.1 Process communication

**Communication protocol**

The example system can be deployed in the local network or the Internet. The internet communication suite TCP/IP is one choice for building the system. It provides two alternative communication protocols can be used, one is unreliable protocol UDP which provides a simple message passing facility that suffers from omission failures but carries no build-in performance penalties, and the other is reliable protocol TCP which guarantees message delivery but at the expense of additional messages, and higher latency and storage costs [1].

RMI is short for Remote Method Invocation, a set of protocols being developed by Sun's JavaSoft division that enables Java objects to communicate remotely with other Java objects. RMI is a relatively simple protocol, but unlike more complex protocols such as CORBA and DCOM, it works only with Java objects. CORBA and DCOM are designed to support objects created in any language [13].

Java RMI allows one process to invoke a method on an object that exists in another address space. The other address space could be on the same machine or a different one. The RMI mechanism is basically an object-oriented RPC mechanism [14].

The process of gathering data and transforming it into a standard format before it is transmitted over a network so that the data can transcend network boundaries. In order for an object to be moved around a network, it must be converted into a data stream that corresponds with the packet structure of the network transfer protocol. This conversion is known as data marshalling. Data pieces are collected in a message buffer before they are marshaled. When the data is transmitted, the receiving computer converts the marshaled data back into an object [1]. Java RMI can do marshalling for the arguments and result which may be primitive data values or objects, so the programmer can focus on implementation of the functionality of the system.

Having discussed the UDP, TCP and RMI, the Java RMI is chosen as protocol for passing
message between distributed components.

**Synchronous and asynchronous communication**

There are two general strategies for communicating between processes: Asynchronous and Synchronous. Each has its advantages and disadvantages.

When the sender sends the information to receiver without waiting the response, it will continue to do its job. This method is called asynchronous communication. In short, the two ends do not synchronize the connection before communicating. Asynchronous communication is more efficient because the sender does not wait for the result.

In synchronous systems, when the sender sends the request to the receiver, it will block the processing until receiving the result. This method is easy to implement than the asynchronous communication.

In this project, asynchronous communication is used to pass message between different components.

The RMI objects should ensure an efficient use of the resources when asynchronous method is used. Ideally, the design should minimize the number of client RMI requests so the server can handle more users. An effective means of reducing the number of client requests is to implement a callback model for asynchronous state change notifications. Without using a callback model, clients would have to continuously query the server, thus wasting valuable network resources and server processing time.

Implementing an RMI application that supports callback model is straightforward. The server and client must declare interface for each other. The server-side interface defines the remote methods for the client and at least one other method that triggers the callback. The client-side interface defines the methods called by the server when it finishes the request of client.

**3.3.2 Security Issue**

Passing message between components via the Internet is vulnerable, which has been discussed in Section 2.2. In this system, SSL protocol is used to provide secure communication channel. SSL is a protocol that layered on top of ordinary socket connection.
In order to establish a SSL connection, a socket connection is first established. This socket connection is used by the client and server to negotiate an encryption algorithm to exchange information securely. After this negotiation process, often called the SSL handshake, the socket is used to transmit the encrypted information. The SSL handshake is necessary because SSL supports a variety of encryption algorithms (commonly referred to as \textit{cipher suites}) such as DSA, RSA, MD5 and SHA-1 [15].

Sun provides support for running RMI over SSL in its Java2 platforms. It offers not only encryption algorithms but also authentication algorithm. The details of implementation will be discussed in Section 4.5.2.

### 3.4 Concurrent Programming

Concurrency opens up design possibilities that are impractical in sequential programs. Threads liberate programmers from the limitations of code that invokes a method and then blocks, doing nothing while waiting for a reply [16].

In this system, each component needs to do more than one thing at a time. For example, proxy server need to wait for results from voter system when it is checking the status of all components. The processing node needs simultaneously perform a request from simulated sensor, and renew a lease. It is possible to develop such components using a single thread, but this it complicated and error-prone. It is easier to design and implement using multi-threads method.

Java 2 platform provides a few constructs and classes specifically designed to support concurrent programming [16]:

- The class \texttt{java.lang.Thread} (along with a few related utility classes), used to initiate and control new activities.
- The keywords \texttt{synchronized} and \texttt{volatile}, used to control execution of code in objects that may participate in multiple threads.
- The methods \texttt{wait}, \texttt{notify}, and \texttt{notifyAll} defined in \texttt{java.lang.Object}, used to coordinate activities across threads.
Chapter 4

Implementation

The implementation details for each component are described below together with UML diagrams illustrating the work where required. The source code can be downloaded from http://jzhuang.spymac.net/project/ft.rar.

4.1 Choice of Programming Language

There are many object oriented languages that can be used to develop the project, such as Java and C++. Important features that distinguish Java from C++ are presented below.

C++ is an object oriented programming language that has been widely used to build application in many areas, because the application written in C++ runs faster than that written by some other languages, such as Java. But it also has its disadvantages. For example, C++ provides pointer to operate memory directly, which could cause many problems, the worst is that system crashes.

Although the Java was derived from C++, it offers many features to avoid the disadvantages in C++. Objected oriented programming is compulsory in Java. Java provides a large class library enabling software to be developed quickly. Executing a Java program requires a run-time environment to be present so the program can be interpreted. The result is cross-platform independence at the cost of performance [17].

Generally, Java is more robust, via: [17]

• Object handles initialized to null (a keyword)
• Handles are always checked and exceptions are thrown for failures
• All array accesses are checked for bounds violations
• Automatic garbage collection prevents memory leaks
• Clean, relatively fool-proof exception handling
• Simple language support for multithreading
It can be seen that Java provides many useful features for programmer to build a robust multi-thread and cross-platform system. So Java is chosen to develop the system.

4.2 Choice of Platform

There are many possible platforms for the project development: UNIX, Linux and Windows. Because Java is selected as the programming language for the implementation of the system, all platforms can be used to develop and deploy the system.

4.3 Development Environment

The software is developed on an Inter P4 1800MHz laptop running windows XP with service pack 1.
Borland JBuilder X with Java SDK 1.4.2 is selected as IDE tool. It is the leading cross-platform environment for building Java applications.

4.4 XML Library

In chapter 2, it has been described that an XML file is used to store Access Control List and Password. Borland Jbuilder X provides several features and incorporates various tools to provide support for the Extensible Markup Language (XML).
In working with XML, JBuilder separates functionality into several layers [18]:

• Creation and validation of XML documents
• Presentation of XML documents
• Programmatic manipulation of XML documents
• Interface to business data in databases
With BorlandXML, the programmer can generate Java classes from a DTD, unmarshal (read) the data from XML file and convert it to Java objects, marshal (write) the Java objects back to the XML document.
So JBuilder XML library and tools make it easy to manipulate XML file programmatically.
The Jbuilder XML library used in this project is included in the package com.borland.XML.
4.5 Communication

4.5.1 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 [18].

The following code explains how to implement a RMI client/server application in which the server returns sum of two integer numbers sent by client. All components in the project are coded in this way.

• Define remote Interface

The remote interface RmiSample is defined as following:

```java
import java.rmi.*;
public interface RmiSample extends Remote {
    public int sum(int a, int b) throws RemoteException;
}
```

This interface defines a method that calculates the sum of two integer numbers. From above snippet, three properties of remote interface can be found. First, the interface must be public. Second, the interface must extend the interface java.rmi.Remote. The last is that every method in the interface must declare that it throws java.rmi.RemoteException.

• Implement remote interface

The implementation class must extend the java.rmi.UnicastRemoteObject class and implement the remote interface. Then objects of such a class can be invoked remotely. It can also have methods that are not in its Remote interface. These can only be invoked locally.

```java
import java.rmi.*;
import java.rmi.server.*;

public class RmiSampleImpl extends UnicastRemoteObject
    implements RmiSample {
    RmiSampleImpl() throws RemoteException {
```
super();
}

public int sum(int a,int b) throws RemoteException {
    return a + b;
}
}

- **Implement a server**

A server is a Java program that creates an instance of implementation class of remote interface and then registers the object with the Object Registry.

The following code shows how to create and register the object.

```java
try {
    LocateRegistry.createRegistry(7000);
    SampleServerImpl Server = new SampleServerImpl();
    Naming.rebind("//localhost:7000/sample", Server);
} catch (RemoteException re) {
    System.out.println("Remote exception: " + re.toString());
}
```

- **Program a client**

The client calls Naming.lookup method to obtain an object handle from the Object Registry on the server and then invokes the remote method on the remote object.

The following code depicts the detail. It can be placed in any convenient class.

```java
try {
    String url = "/localhost:7000/sample";
    RmiSample RmiObject = (RmiSample)Naming.lookup(url);
    System.out.println("1 + 2 = " + RmiObject.sum(1,2));
} catch (RemoteException ex) {
    System.out.println("Error in lookup: " + ex.toString());
}
```

In this project, all components play the roles of client and server. It means that every component exports object to others and also call remote methods existing on others. This
technology makes the application more efficient. For example, if a user wishes to read numbers from a sensor, it sends request to proxy server and waits to display the values. Two ways can do this job; one is that the user application uses a background thread to send brief method calls to check whether or not the results can be fetched. Another is that user application exports a remote object to wait the proxy server to send the results without blocking. Obviously, the second callback method is more efficient.

4.5.2 Using SSL over RMI

Sun Java allowed RMI developer to replace the standard (cleartext) sockets that RMI uses with other sockets. With this mechanism an RMI application can use SSL socket instead of the default socket. The Figure 4-1 depicts how the applications communicate using RMI over SSL.

In order to use a custom SSL sockets to replace the default socket, Java provides support for running RMI over SSL in its Java 2 platforms using a custom RMISocketFactory [19]. And Java 2 SDK V1.4 also includes the Java Secure Socket Extension (JSSE) API which provides an implementation of SSL sockets.

An example of RMI over SSL is described in [19], which illustrates how to use SSL-based
connections for RMI calls to a remote object. In the source code, the main class is SecureRMISocketFactory.java with three utility classes: ServerSocketMonitor.java, SocketMonitor.java and InputStreamMonitor.java, it can be used to replace the default RMISocketFactory in the RMI application.

The following code fragment explains how to use the custom SecureRMISocketFactory in RMI applications.

```java
import common.SecureRMISocketFactory

// Registering secure RMI socket factory
System.out.println("Registering secure RMI socket factory ... ");
java.rmi.server.RMISocketFactory.setSocketFactory
  (new SecureRMISocketFactory());
```

The key point is to replace the RMISocketFactory with custom SecureRMISocketFactory before exporting any remote object. This code is put in main program of each component in the system.

The custom SecureRMISocketFactory not only encrypts the information when transmitting, but also provides authentication using certificate. Java provides a certificate management utility keytool to generate a so-called keystory which stores both a sequence of X.509 certificates, and an associated private key. The keystore is saved as a file which protects private keys with a password [20].

The following commands described in [19] illustrate how to generate a keystore for authentication between client and server, and how to execute the Java programs.

**Key Management**

1. Create a self-signed server and a self-signed client key each in its own keystore
   ```bash
   keytool -genkey -v -keyalg RSA -keystore server.keystore -dname "CN=Server, OU=Bar, O=Foo, L=Some, ST=Where, C=UN"
   keytool -genkey -v -keyalg RSA -keystore client.keystore -dname "CN=Client, OU=Bar, O=Foo, L=Some, ST=Where, C=UN"
   ```

2. Export the server's and the client's public keys from their respective keystores
keytool -export -rfc -keystore server.keystore -alias mykey -file server.public-key
keytool -export -rfc -keystore client.keystore -alias mykey -file client.public-key

3. Import the client's public key to the server's keystore, and vice-versa:
keytool -import -alias client -keystore server.keystore -file client.public-key
keytool -import -alias server -keystore client.keystore -file server.public-key

**Execution**

1. Start the server with the appropriate SSL configuration
   java -Djavax.net.ssl.trustStore=server.keystore -Djavax.net.ssl.keyStore=server.keystore
   -Djavax.net.ssl.keyStorePassword=server TestServer

2. Start the client with the appropriate SSL configuration
   java -Djavax.net.ssl.trustStore=client.keystore -Djavax.net.ssl.keyStore=client.keystore
   -Djavax.net.ssl.keyStorePassword=client TestClient

### 4.6 System Modules

The package common includes the classes that are used in other modules. For example, SecureRMISocketFactory is used to replace default RMI socket in all modules, class SystemConstants declares all constants for the system.

Source code guide can be found in Appendix D.

#### 4.6.1 Simulated Sensor

The purpose of simulate sensor is to generate random numbers and to provide remote methods to configure the parameters of itself.

The static structure of the Sensor program is shown in Figure 4-2. It implements a remote interface SensorInterfaceForServer which defines the remote methods for proxy server to control and configure the sensor. There are five methods are declared in SensorInterfaceForServer interface, namely, setSeed(), getSeed(), getFaultRate(), setFaultRate(), and startSensor(). setSeed() and getSeed() are responsible for getting and
setting the seed of random function. `getFaultRate()` and `setFaultRate()` are used to get and set
the value of fault rate of a sensor. `startSensor()` can start or stop the random generator.

It also creates two threads: RandomThread, and SensorThread. The first one generates the
random numbers per two seconds; the second renews the lease with proxy server.

![Diagram](image)

**Figure 4-2 Static structure of Sensor**

Figure 4-3 shows what happens when the sensor boots up. First it gets the IP address of proxy
server from command line arguments, and registers secure RMI socket factory for
establishing a secure communication channel. Then it constructs an instance of the Sensor
Class with arguments (50, 100) which define the minimal and maximal values that will be
generated by the random generator. After these initiation processes, it looks up the proxy
server and call connection method exporting by the proxy to get a serial number. The last
thing is to create two threads that have been mentioned above.
In order to pass random number to processing node, the Sensor needs to know where the processing node locates. Then RandomThread is also responsible for getting service name of an available processing node from proxy server.

### 4.6.2 Processing Node

The task of processing node is simple. It just waits for the number from the sensor node until receiving 10 of them and forwards the average value of the ten numbers to voter system.

The static structure of processing node program is illustrated in Figure 4-4.
Four classes are included in a java package processing. The ProcessingInterface class declares a remote interface for sensor to pass number. The ProcessingThread class and SortingThread class are two thread classes, the task of first one is to renew lease, and the second one is to sort the numbers received from sensor using bubble sorting algorithm and to calculate average of every 10 received readings. The reference of primary voter node is also gotten by ProcessingThread.

Figure 4-5 depicts what happens when the processing node boots up. First it gets the IP address of proxy server from command line arguments, and registers secure RMI socket factory for establishing a secure communication channel. Then it constructs an instance of the Processing Class. Next it looks up the proxy server and invokes connection method exporting by the proxy to get a serial number. Before creating two threads, it exports itself through

---

**Figure 4-4 Structure of Processing program**

In the image, a class diagram shows the structure of the Processing program. The diagram includes classes such as ProcessingNode, ProcessingThread, and SortingThread. The diagram illustrates the relationships and methods of these classes, highlighting their functionality in the processing system.
binding the instance to a given service name and passes the name to proxy server.

Figure 4-5 The workflow of ProcessingNode.main()

Multi-threaded and synchronous mechanisms of Java are employed to make the program more efficient. The Sorting thread is blocked until it is notified by the main thread of the processing program when it has received 10 numbers from sensor.

The following two methods show how the two threads are synchronized.

```java
public synchronized void passReading(int id, double value) {
    while (count == 10)  myWait(this);

```
buffer[inBuf] = value;
inBuf = (inBuf + 1) % sizeBuf;
count++;
sensorID = id;
if (count == 10) notify();
}

public synchronized void fetchReading() {
    while (count != 10) myWait(this);
    for (int i = 0; i < sizeBuf; i++) bufferSort[i] = buffer[i];
count = 0;
inBuf = 0;
notify();
}

These two methods are implemented in the main program of ProcessingNode. The passReading method is called by the sensor and fetchReading is called by sorting thread. They show the buffer as a circular array in which inBuf and outBuf incremented modulo sizeBuf to keep track of the slots for depositing and fetching numbers. Without the synchronized mechanism, the sorting thread must repeatedly check for the result.

4.6.3 Fault Tolerant Voter System

The primary-backup model is employed to build the fault tolerant voter system. Each node is an instance of Voter class.

The static structure of Voter class is depicted in Figure 4-6.
It implements a remote interface called VoterInterface which defines a remote method for processing node to pass result.

Figure 4-7 illustrates what happens when the voter node boots up. First it gets the IP address of proxy server from command line arguments, and registers secure RMI socket factory for establishing a secure communication channel like above two modules. Then it constructs an instance of the Voter Class. Next it looks up the proxy server, and invokes remote methods exporting by proxy to get a serial number and the number of sensors. After that it exports itself though binding the instance to a given address and passes the address to proxy server. The last thing is to create two threads, namely VoterThread, and PassResultThread. The
first one is used to renew the lease with proxy, the second one is to wait for the consensus and forward it to proxy server.

Get IP of Proxy server

Register secure socket

Create an Instance of Voter

Look up Proxy server

Get the number of sensors

Bind object

Pass bound name to proxy

Create a PassResultThread

Create a VoterThread

Figure 4-7 the workflow of Voter.main()

The PassResult thread is in charge of reaching a consensus from the values received from all sensors and logging results to a text file.

In the implementation of processing program, the synchronous mechanism for multi threads is also used in main thread of voter and passResult thread
4.6.4 Proxy Server

The proxy server is the core of the whole system. It is a bit more complex than the others. It must run continuously to accept remote calls and respond to requestor. It must maintain an Access List Control and authenticate identity of user.

The Figure 4-7 shows the static structure of proxy server program.

Figure 4-7 Structure of ProxyServer class
The ProxyServer class implements the remote interface ProxyInterface which declares all remote methods that can be invoked by other modules, such as sensors, processing nodes, voters, and clients.

The proxy plays a role of manager of the system. Every node needs to connect to server when it boots up and renew a lease per second when running. The checking thread is responsible for checking the lease of the each node. If the lease of a node expires (lease duration is defined as 10 seconds), which means the node crashes, the thread changes the status of this node to unavailable. If the primary node of voter system crashes, the server will elect a backup node as primary. Then the processing node can find the new voter node from proxy server when remote exception occurs in calling the voter system.

The ACL and password of users is stored in an XML file. The format of ACL is shown following DTD file.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT userlist ( user+ ) >
<!ELEMENT user ( name, password, allowedaction+ ) >
<!ELEMENT password ( #PCDATA ) >
<!ELEMENT name ( #PCDATA ) >
<!ELEMENT allowedaction ( #PCDATA ) >
```

The Borland JBuilder can generate several classes from the above document type definition for manipulating the XML file, include reading, writing. The next piece of code illustrates how to read the number of ACL items from XML file.

```java
java.io.FileInputStream in = null;
try {
    in = new java.io.FileInputStream(fileName);
    System.out.println("== unmarshalling \\
" + fileName + \" \\
" ==");
    userlist = Userlist.unmarshal(in);
    userlist.setSystemId("userlist.dtd");
    }catch (Throwable ex) {
        ex.printStackTrace();
    System.out.println("Total Number of Users read = "+ userlist.getUserCount());
```
4.6.5 Client & Monitor Programs

The client and monitor programs are graphical user interfaces developed by Java Swing. They both extend the ClientImpl class which implements a remote interface for proxy server to call back, for example, passing readings to client program and sending status of system to monitor program.

The monitor program can get information of available sensors and set parameters of sensors, including value of seed, and fault rate. It also can start or stop the sensor generating random. The figure 4-8 shows the graphic user interface.

![Figure 4-8 Setup dialogue](image)

It also can display the status of the system, such as how many sensors, processing nodes and vote nodes are in the system, how many nodes failed, how many results have been received.

The client program can read the values from a specified sensor and the consensus values from the proxy. Figure 4-9 illustrates the interface.
4.7 Compile and Execution

- **Compile**

All source codes are compiled by JBuilder X and all needed stub and skeleton files can be generated automatically. And all executable files are packed into a .jar file.

- **Execution**

In order to execute all programs, some parameters are necessary, namely:

1) `-Djava.security.policy`, it is a file that defines the socket permissions

2) `-Djavax.net.ssl.trustStore`, `-Djavax.net.ssl.keyStore`, `-Djavax.net.ssl.keyStorePassword`

These define parameters for SSL.

Detail of execution of the system is described in Appendix D.
Chapter 5

Evaluation

5.1 Requirements

The requirements of the project are described in Section (1.4.1). They are restated here:

· **Minimum Requirements**
  1) Implement a fault tolerant system
  2) Implement identity based access control

· **Intermediate requirements**
  1) Implement password based authentication
  2) Implement secure communication

· **Advanced requirements**
  1) Enhance fault tolerance part
     a) A multiple-voter system is implemented.
     b) The client program is adapted to show a graphical display of the results being obtained in real-time.
  2) Enhance interface and statistics
     a) A GUI based monitor window is developed to show the current status of the proxy server.
     b) Statistics data is provided to help administrators comprehend the situation and raise alerts if necessary.

5.2 Evaluation

The aim of evaluation is to prove whether or not the prototype meets the system requirements. The evaluation focused on two important aspects of the system requirements, namely, fault tolerance and security. In order to achieve this purpose, some testing strategies were applied to evaluate implemented system.
5.2.1 Debug

The most obvious anomalies in software can be discovered during the development cycle. The IDE debugger and application logs could be helpful. But more useful and easier method is to print the status of running application to console.

This application integrates four components. The real difficulty is trying to figure out whose fault it is. Debugging messages are built into all components, activated by a global DEBUG flag. When DEBUG is on, each component writes helpful internal state and activity nodes to console.

Java try-catch is also used to manage exceptional runtime conditions. When exception occurs, try-catch method can help to trace the errors, and discover causes.

The above two methods played an important role in debugging the software.

5.2.2 Functionality testing

The aim of functionality testing is to verify whether the module meets its function specifications. Testing scenarios were developed to test the functionalities of the system.

* Testing of Simulated Sensor

The purpose of simulate sensor is to generate random number depending on given parameters.

A simple test program was developed to help to test the program. It can call the remote methods that the sensor exports and simulate a processing node to receive the number from sensor. The following Table 5-1 lists all tests that have been done.

<table>
<thead>
<tr>
<th>Function</th>
<th>Test result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get seed</td>
<td>Pass</td>
</tr>
<tr>
<td>Set seed</td>
<td>Pass</td>
</tr>
<tr>
<td>Get fault rate</td>
<td>Pass</td>
</tr>
<tr>
<td>Set fault rate</td>
<td>Pass</td>
</tr>
<tr>
<td>Start / Stop</td>
<td>Pass</td>
</tr>
<tr>
<td>Connect to proxy server</td>
<td>Pass</td>
</tr>
</tbody>
</table>
**Find an available processing node** | **Pass**  
---|---
**Pass value to processing node** | **Pass**  

Table 5-1 Test of sensor

- **Testing of processing node**

Like testing of sensor node, a test program was used to verify the functionality of processing node. All tested functions are shown in Table 5-2.

<table>
<thead>
<tr>
<th>Function</th>
<th>Test result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect to proxy server</td>
<td>Pass</td>
</tr>
<tr>
<td>Find primary voter node</td>
<td>Pass</td>
</tr>
<tr>
<td>Receive values from sensors</td>
<td>Pass</td>
</tr>
<tr>
<td>Send results to voter</td>
<td>Pass</td>
</tr>
<tr>
<td>Bubble sorting and calculate average</td>
<td>Pass</td>
</tr>
<tr>
<td>Bind services for sensors</td>
<td>Pass</td>
</tr>
</tbody>
</table>

Table 5-2 Test of processing node

- **Testing of voter**

Table 5-3 depicts the functions tested by test program.

<table>
<thead>
<tr>
<th>Function</th>
<th>Test result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect to proxy server</td>
<td>Pass</td>
</tr>
<tr>
<td>Bind services for processing nodes</td>
<td>Pass</td>
</tr>
<tr>
<td>Logging</td>
<td>Pass</td>
</tr>
<tr>
<td>Voting</td>
<td>Pass</td>
</tr>
<tr>
<td>Pass results to proxy</td>
<td>Pass</td>
</tr>
</tbody>
</table>

Table 5-3 Test of voter

- **Testing of proxy server**

When sensor nodes, processing node, and voter have been tested, the next step was to test the proxy server. All tested modules were used to complete this task. All functions that have been tested are listed in Table 5-4.

<table>
<thead>
<tr>
<th>Function</th>
<th>Test result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept connection</td>
<td>Pass</td>
</tr>
</tbody>
</table>
Configure the parameters of sensor  Pass
Start / Stop sensor Pass
Elect primary node from voters Pass
Check the leases of all components Pass
Maintain information of all components Pass
Receive results from voter  Pass
Send results to client program Pass
Access Control Pass
Check identity of user  Pass

Table 5-4 Test of proxy server

- **Testing of Client and Monitor program**

After having tested core parts of the system, client and monitor program were tested to prove that whether or not they achieve their purpose.

## 5.2.3 Fault-injection testing

A major problem in the development of fault-tolerant systems is the accurate determination of the dependability properties of the system. The accepted solution to this problem is to inject the faults into system, and to observe the behavior of the system under the injected faults.

Different kinds of fault were injected to test the system. In the testing scenario, the system consisted of three sensors, three processing nodes, four voter nodes, a proxy server, a client, and a monitor program. Table 5-5 lists a set of faults that injected into the system for testing.

<table>
<thead>
<tr>
<th>Type of anomaly</th>
<th>Injection position</th>
<th>Injection approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor crash</td>
<td>Sensor node</td>
<td>State injection</td>
</tr>
<tr>
<td>Processing node crash</td>
<td>Processing node</td>
<td>State injection</td>
</tr>
<tr>
<td>Voter node crash</td>
<td>Voter node</td>
<td>State injection</td>
</tr>
<tr>
<td>Client crash</td>
<td>Client program</td>
<td>State injection</td>
</tr>
</tbody>
</table>

Table 5-5 A set of anomalies

State injection is a basic fault injection approach, which achieved through altering the state behavior of a running code. Modifications are made in the operating system, processors,
storages, dynamic states of running processes, messages, etc. [8].

Testing results:

Sensor crash: one of sensors was killed to mimic its crash. This fault affects the corresponding processing node and voter. Because the voting algorithm in voter system needs to know many sensors nodes are running. When sensor crashed, it didn’t renew lease any more. After 10 seconds (lease duration defined in proxy server), all voter nodes knew this situation. Primary voter reached consensus according the new parameter.

Processing node crash: when processing node crashed, a remote exception occurred in corresponding sensor. It didn’t renew lease. Then all voter nodes got the information from the proxy server.

Voter node crash: When primary node creased, all processing nodes could not send their results. They called a method in proxy server object to find a new voter. A backup node was elected as primary by proxy server in 10 seconds. And server name was passed to all processing nodes. Then all system worked normally again. When the voter system was recovering, the results produced by processing nodes were discarded.

Client crash: when a client crashed, it did not renew lease. The proxy-sever discovered this abnormal situation 10 seconds later and update the information maintained in a Java vector.

Fault injection was also utilized to test the security requirement. Any forms of error in the parameters of request can be detected. In an experiment, if there was any error in one of username, password, or action in the request parameter, the request was rejected.

5.3 Experimental results

In the experiment, 3 sensors, 3 processing nodes, 2 voter nodes, a proxy server, a client and a monitor program run on two PCs in MSc Lab. The fault rate of sensor 0 was set to 5 %, sensor 1 6 %, and sensor2 0%. Seed of all sensors were set to 12345. Each sensor generated 3000 readings respectively. All readings of sensors, averaged result, and consensus results were logged in the text file which can be downloaded from http://jzhuang.spymac.net/project/log0.txt.

The Table 5-6 shows the experimental results.
<table>
<thead>
<tr>
<th>Total readings</th>
<th>Total errors (Sensor 0)</th>
<th>Total errors (Sensor 1)</th>
<th>Total Errors (Sensor 2)</th>
<th>Total errors (voter)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>500</td>
<td>24</td>
<td>30</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>1000</td>
<td>50</td>
<td>61</td>
<td>0</td>
<td>21</td>
</tr>
<tr>
<td>1500</td>
<td>82</td>
<td>102</td>
<td>0</td>
<td>37</td>
</tr>
<tr>
<td>2000</td>
<td>114</td>
<td>125</td>
<td>0</td>
<td>44</td>
</tr>
<tr>
<td>2500</td>
<td>133</td>
<td>148</td>
<td>0</td>
<td>51</td>
</tr>
<tr>
<td>3000</td>
<td>163</td>
<td>173</td>
<td>0</td>
<td>58</td>
</tr>
</tbody>
</table>

Table 5-6 experimental results

Although sensor 0 generated 163 numbers out of range, sensor 1 173, only 58 average numbers didn’t reach consensus.

### 5.4 Analysis

It can be seen from functionalities testing, fault-injection testing, and experiment, that the implemented system has achieved the system requirements, both fault tolerance and security. But some important issues should be analyzed.

#### 5.4.1 Performance

The communication between different components is via Java RMI. Compared with TCP/IP, Java RMI is less efficient especially in a distributed environment. Java RMI is based on TCP/IP, and extra TCP/IP connection results in a decrease in performance. Java RMI is easier to use than TCP/IP socket, it hides many low level details and provides a set of high level interfaces for programmer. Figure 5-1 illustrates the RMI architecture.
If TCP or UDP socket was used to pass message between components, the performance would be better than that of the implemented system.

## 5.4.2 Recovery

As mentioned in chapter 2, replication is a key to providing fault tolerance. Two models of replication for fault tolerance which are passive model and active model have been discussed. Passive model is employed to achieve fault tolerance in this project. But passive model has a disadvantage, which is the system can not provide service to clients during it is detecting failure and electing a new primary when the primary crashed. In the implementation of this project, it takes 10 seconds (lease duration) for a backup replica to take over the primary. The parameter of recovering time could be reduced, which depends on conditions of traffic of network, workload of hosts, etc. In an experiment, it was reduced to 5 seconds, the system worked well. So in some fault tolerant system, in order to reduce the affection of network, processes hosted on different machines are linked by special channel such as RS232 to send heart-beat signal. Compared with passive model, active model can provide continuous service without interruption, but it is more complicated.
Chapter 6

Conclusions and Future Directions

All requirements stated in chapter 2, have been performed. The software developed may be considered as a prototype for future research and development for fault tolerant system based on replication technology. Main issues on this project are to improve the reliability, to make the system more secure, and to build a friendly user interface. There are several possible extensions and these are described in the following section.

6.1 Developments

6.1.1 Fault tolerant part

• Make proxy server fault tolerant

From the fault tolerance point of view, the current implementation of proxy server is a single point of failure for the system. For example, if proxy server fails all of its data will be lost. As a result, even though the other components are still alive they may not be accessible. Therefore, it is better to make proxy applications fault tolerant. Two kinds of fault tolerant models that mentioned in Section 2.1.1 can be used to achieve this purpose.

• Replicate data

The primary node in voter subsystem does not replicate received data to backup nodes, so each of them could not keep a complete log. This could make the administrator difficult to analyze the data of this system. In order to solve this problem, replication of data between primary node and backup nodes should be implemented.

The pairwise replication of data is not the best model for communication from one process to a group of other processes. A multicast operation is more appropriate. It can provide some useful characteristics, such as better performance.
6.1.2 Security

- Improvement of ACL

If a new user wishes to access the system, the username, password, and access lists should be added to the XML file by the administrator manually. The JBuilder provides some packages to manipulate the XML file, such as add, update. So a user interface can be developed to maintain the ACL.

6.1.3 Performance issue

- asynchronous logging

The voter node logs the results and any errors using synchronous method. It is not efficient because the process must be blocked until the logging has been done. Asynchronous logging provides a log service, allowing non-blocking operations. A logging thread can be created as log service. The work thread just passes the data that need to be logged into logging thread. The logging thread is responsible for writing the data to a file.

6.1.4 Scalability

The proxy server has so far been a small scale application. Because some arrays of fix length are used. To make the system more scalable, some data structure such as vector can be used instead of arrays.

6.1.5 User interface

- Improvement of Monitor program

It may be advantageous to design a more attractive graphical user-interface (GUI) to enhance the display of statistics data of the application.
Bibliography


[14] Java RMI Tutorial, URL: http://www.ccs.neu.edu/home/kenb/com3337/rmi_tut.html,
[Accessed date: 05-08-04]


[16] Doug Lea, Concurrent Programming in Java Design Principles and Patterns, Addison-Wesley
Publishing Company, 1997

[17] Bruce Eckel, Thinking In Java, Prentice-Hall, 2000

[18] Core Java, Java Remote Method Invocation (Java RMI)
URL: http://java.sun.com/products/jdk/rmi/, [Accessed date: 08-08-04]

[19] Alexander V. Konstantinou, Using RMI over SSL authentication for application-level access


[Accessed date: 03-09-04]
Appendix A

Personal Reflection

The programming involved in developing this project was a great challenge for me. Previously I have undertaken some database application projects but building this fault tolerant system was strange to me. My programming skills and project management skills have improved considerably through this project.

The research domains for this project focused on the fault tolerance and security. So, appropriate fault tolerance and security literatures were consulted and the problems and concepts of these areas became clear.

A lot of hands-on experience at designing and implementing a fault tolerant and secure system has been obtained from this project. For example, I did not know details of fault tolerant model, secure communication algorithms. After I finished the project, I have clear idea about them.

I am a novice of Java programming before I started this project. This project helped me gain a lot of knowledge from programming, especially in concurrent programming. I also have learnt how to evaluate a distributed system by different evaluation technologies.

The most difficult part of implementation of the system was to write secure communication program using Java RMI over SSL. I took three days to complete it. But I learned a lot in these three days.

In order to achieve a balance between background research, design, implementation and writing up, project management is necessary. I have learnt about planning, organising and managing my project to be able to complete it on time.

Overall, this project was every interesting, and many useful concepts, technologies, skills, and experiences have been gained from this project.
Appendix B

Objectives and Deliverables Form

School of Computing, University of Leeds

MSC PROJECT OBJECTIVES AND DELIVERABLES

This form must be completed by the student, with the agreement of the supervisor of each project, and submitted to the MSc project co-ordinator (Mrs A. Roberts) via CSO by 18th March 2004. A copy should be given to the supervisor and a copy retained by the student. Amendments to the agreed objectives and deliverables may be made by agreement between the student and the supervisor during the project. Any such revision should be noted on this form. At the end of the project, a copy of this form must be included in the Project Report as an Appendix.

Student: Jian Zhuang (ses3jo@comp.leeds.ac.uk)
Programme of Study: DMS
Supervisor: Prof. Jie Xu
Title of project: Implementing a Fault-Tolerant System Based on Replication and Access Control

External Organisation*:

*(if applicable)

AGREED MARKING SCHEME

<table>
<thead>
<tr>
<th>Understand the Problem</th>
<th>Produce a Solution *</th>
<th>Evaluation</th>
<th>Write-Up</th>
<th>Appendix A</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>40</td>
<td>20</td>
<td>15</td>
<td>5</td>
<td>100</td>
</tr>
</tbody>
</table>

* This category includes Professionalism

OVERALL OBJECTIVES:

To design and implement a fault-tolerant sensor system with replicated services and an access control mechanism.

To evaluate the implemented system with experimental analysis and testing

MINIMUM REQUIREMENTS:

Implementing a workable communicating system that consists of sensor services, processing services, a voter service, a proxy server and a client program.

SOFTWARE AND HARDWARE RESOURCES REQUIRED:

Java SDK, PCs

DELIVERABLE(s):

1. A project report.
2. Source code of the system

Signature of student: Jian Zhuang Date: 18-03-04

Signature of supervisor: Date:

Continued over
Amendments to agreed objectives and deliverables:

Date | Amendment

NOTE:

Students should discuss the requirements of the form with their supervisor prior to submission but it is the student's responsibility to submit it by the due date (not the supervisor's). Before submitting this form it is important that student and supervisor agree that the objectives, minimum requirements, and computer requirements are feasible.

The overall objectives of the project are the overall top-level goals. The project title should reflect these objectives.

The minimum requirements are components of the deliverables that constitute minimal acceptable work in producing a solution to the problem. They should be described in a quantifiable way, and in a way which gives the greatest opportunity to expand upon them and/or deliver them to a higher standard to gain extra credit. (At the mid-project stage you will be required to describe any enhancements that you intend delivering if time permitted or, failing that, you would recommend as future work in this area.) Note that the marks available for delivery of a solution relate specifically to your achievements in relation to the agreed minimum requirements.

Wherever possible we encourage students to use the software and facilities currently provided in the school. Whilst students are welcome to develop software on their home machines, they WILL have to demonstrate it to their supervisor and assessor at a progress meeting within the school. The issue of availability of facilities and the behaviour of software on school machines MUST be addressed at the outset. For instance, our security arrangements may prohibit the execution of programs and this will not be compromised to accommodate a demonstration. Students and supervisors should discuss computer requirements.

Requirements for software and tools which are not currently provided by the school have to have a technical justification and sensible requests will be considered. If you are in any doubt about current provisions or of your ultimate software needs please submit the request anyway. If you are doing a background research phase you should state what software you will be considering and give an estimated date by which you will have decided what you would prefer to use. Provide as much information about the request as possible e.g., operating system to work on, version to install, accessibility needs.

The school does not guarantee to honour requests for additional facilities.
Appendix C

Marking Scheme and Mid Report header sheet

School of Computing, University of Leeds

MSc MID PROJECT REPORT

All MSc students must submit an interim report on their project to the MSc project co-ordinator (Mrs A. Roberts) via the CSO by 9am Wednesday 28th April 2004. Note that it may require two or three iterations to agree a suitable report with your supervisor, so you should let him/her have an initial draft well in advance of the deadline. The report should be a maximum of 10 pages long and be attached to this header sheet. It should include:

- the overall aim of the project
- the objectives of the project
- the minimum requirements of the project and further enhancements
- a list of deliverables
- resources required
- project schedule and progress report
- proposed research methods
- a draft chapter on the literature review and/or an evaluation of tools/techniques
- the WWW document link for the project log to date

The report will be commented upon both by the supervisor and the assessor in order to provide you with feedback on your approach and progress so far.

The submission of this Mid Project Report is a pre-requisite for proceeding to the main phase of the project.

<table>
<thead>
<tr>
<th>Student:</th>
<th>Jian Zhuang (<a href="mailto:sca3jz@comp.leeds.ac.uk">sca3jz@comp.leeds.ac.uk</a>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programme of Study:</td>
<td>DMS</td>
</tr>
<tr>
<td>Title of project:</td>
<td>Implementing a Fault-Tolerant System Based on Replication and Access Control</td>
</tr>
<tr>
<td>Supervisor:</td>
<td>Prof. Jie Xu</td>
</tr>
<tr>
<td>External Company (if appropriate):</td>
<td></td>
</tr>
</tbody>
</table>

AGREED MARKING SCHEME

<table>
<thead>
<tr>
<th>Understand the problem</th>
<th>Produce a solution</th>
<th>Evaluation</th>
<th>Write-up</th>
<th>Appendix A</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20</td>
<td>40</td>
<td>20</td>
<td>15</td>
<td>5</td>
</tr>
</tbody>
</table>

* This includes professionalism

Signature of student: Jian Zhuang  
Date: 07-04-04

Supervisor's and Assessor's comments overleaf.

56
Supervisor’s comments on the Interim Report

This report is in a good shape. Some possible improvements are suggested as follows:

1) discuss in more detail how secure communication could be achieved, and analyse existing algorithms;

2) discuss fault types to be tolerated by the implemented system and understand the importance of an appropriate fault model;

3) have a carefully planned strategy for evaluating the implemented system, especially against the listed requirements for the projects;

4) Read more papers & books. Academic papers related to fault tolerance & security are many. The key papers should be discussed in your literature reviews.
Assessor's comments on the Interim Report

1. The methodology section does not lead to any conclusions. Need more evidence for valid conclusions.
2. “Minor” clarification such as the example point in Section I is quite clear. However, it should be included at the start. Using other people's words is not recommended — use your own.
3. You should develop (or continue to refine) a logical architecture (for example: information architecture or physical architecture). Both should be more effectively discussed — the logical architecture should be grounded in literature and the physical should reflect your technology choices and design decisions.

It is not clear with you intended physical implementation will be multi-platform and if so, how you intend to achieve it.
Appendix D

Source Code Guide

This appendix is a brief guide to the files of the source code, which can be downloaded from:

http://jzhuang.spymac.net/project/ft.rar

Source Tree

Package common:

SystemConstants.java                      System constants
InputStreamMonitor.java                    Utility for secure RMI Socket Factory
SocketMonitor.java                        Utility for secure RMI Socket Factory
ServerSocketMonitor.java                   Utility for secure RMI Socket Factory
SecureRMISocketFactory.java                Secure RMI Socket Factory
UserElement.java                          Data structure for storing user info in proxy
Result.java                               Data Structure used in Voter class
Reading.java                             Data structure used in ProcessingNode class

Package sensor:

SensorInterfaceForServer.java                Remote interface of sensor
SensorThread.java                         Thread for renewing lease
RandomThread.java                        Thread for generating random number
Sensor.java                               Main program of sensor

Package processing:

ProcessingInterface.java                     Remote interface of processing node
SortingThread.java                         Thread for processing readings
ProcessingThread.java                      Thread for renewing lease
ProcessingNode.java                         Main program of processing node

Package voter

VoterInterface.java                         Remote interface of voter
VoterThread.java                          Thread for renewing lease
PassResultThread.java  Thread for passing result to proxy
Voter.java  Main program of voter

**Package proxy:**

User.java  Generated by JBuilder for accessing XML file
Userlist.java  Generated by JBuilder for accessing XML file
Allowedaction.java  Generated by JBuilder for accessing XML file
Name.java  Generated by JBuilder for accessing XML file
Password.java  Generated by JBuilder for accessing XML file
ProxyInterface.java  Remote interface of proxy
SendThread.java  Thread for sending result to client
CheckThread.java  Thread for checking the status of the system
ProxyServer.java  Main program of proxy server

**Package client:**

Client.java  Program for testing
ClientInterface.java  Remote interface of client and monitor
ClientImpl.java  Implementation of remote interface
about.png  Image file
LoginDialog.java  Login dialogue of client
ClientThread.java  Thread for renewing lease
GUIClient.java  Main program of client
GUIClientFrame.java  Swing frame of client
GUIClientFrame_AboutBox.java  About dialogue of client
AdminLogin.java  Login dialogue of monitor
Monitor.java  Main program of monitor
MonitorFrame.java  Swing frame of monitor
MonitorFrame_AboutBox.java  About dialogue of monitor
MonitorThread.java  Thread for renewing lease
SelectSensor.java  Selection dialogue of client
SetSensorDialog.java  Configuration dialogue of monitor
Files:

- test_policy: Permission file
- server.keystore: Keystore
- client.keystore: Keystore

**Compilation and Execution**

The project was developed using Windows, but the system can run on any platforms that have installed Java VM 1.4 or above.

JBuilder provides tools to compile the program. The project file defines how to compile the application can be found in the source code archive. A file called FT-System.jar is the output.

To run the application, some parameters should be provided in the command line. For example, type the following command to run different components.

**Proxy server:**

```java
java -classpath FT-System.jar -Djava.security.policy=test_policy
   -Djavax.net.ssl.trustStore=server.keystore -Djavax.net.ssl.keyStore=server.keystore
   -Djavax.net.ssl.keyStorePassword=server proxy.ProxyServer
```

**Sensor:**

```java
java -classpath FT-System.jar -Djava.security.policy=test_policy
   -Djavax.net.ssl.trustStore=server.keystore -Djavax.net.ssl.keyStore=server.keystore
   -Djavax.net.ssl.keyStorePassword=server sensor.Sensor
```

**Processing Node:**

```java
java -classpath FT-System.jar -Djava.security.policy=test_policy
   -Djavax.net.ssl.trustStore=server.keystore -Djavax.net.ssl.keyStore=server.keystore
   -Djavax.net.ssl.keyStorePassword=server processing.ProcessingNode
```

**Voter:**

```java
java -classpath FT-System.jar -Djava.security.policy=test_policy
   -Djavax.net.ssl.trustStore=server.keystore -Djavax.net.ssl.keyStore=server.keystore
   -Djavax.net.ssl.keyStorePassword=server voter.Voter
```

**Client:**

```java
java -classpath FT-System.jar -Djava.security.policy=test_policy
```

61
-Djavax.net.ssl.trustStore=server.keystore -Djavax.net.ssl.keyStore=server.keystore
-Djavax.net.ssl.keyStorePassword=server  client.GUIClient

Monitor:

java -classpath FT-System.jar -Djava.security.policy= test_policy
-Djavax.net.ssl.trustStore=server.keystore -Djavax.net.ssl.keyStore=server.keystore
-Djavax.net.ssl.keyStorePassword=server  client.Monitor