SURVIVABILITY THROUGH DYNAMIC RECONFIGURATION

BY

SUBRAMANIAM R. STHANU

Submitted to the Department of Civil and Environmental Engineering
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1998

ABSTRACT

Information survivability encompasses many aspects of security and reliability for computers, communication networks, and information systems in general. In this research, we propose the use of dynamic reconfiguration as a diversity technique to build robust systems capable of surviving denial of service attacks. Dynamic reconfiguration can be of different kinds. Here we focus on dynamically switching among different configurations, enabling an information system to continue to operate despite successful attacks against individual network nodes or particular protocols. To demonstrate this concept, we have developed a prototype collaborative planning tool with a shared drawing area, a text-based chat area, and voice communication, implemented using the Java™ programming language. The prototype focuses on a particular situation in which our proposed survival strategy can be used and demonstrates a means of implementing robust systems using our concept of dynamic reconfiguration.

Thesis Supervisor: Prof. Steven R. Lerman
Title: Director, Center of Educational Computing Initiatives and Class of 1922 Professor of Civil and Environmental Engineering

Thesis Supervisor: Dr. Thomas M. Parks
Title: Technical Staff, Information Systems Technology Group, MIT Lincoln Laboratory

™ Java is a registered trademark of Sun Microsystems Inc.
Dedicated

To

Amma & Appa

On my

23rd B’Day.
Acknowledgments

While "a cast of thousands" may be an overstatement, this document would not have come into being but for many people – and a few deserve special mention. First of all, I would like to thank my thesis supervisor, Dr. Thomas M. Parks, for his guidance and the time he spent towards this document, which I can be proud of for the rest of my life. His invaluable advice and ideas laid the foundation for this research and steered it towards completion.

I would like to thank my co-supervisor, Prof. Steven R. Lerman for his valuable comments on the work and for the support he extended towards me throughout my stay at MIT. Working at CECI, has been a memorable experience. I would like to thank the staff and students at CECI, and especially Dr. Judson Harward for his encouragement and suggestions during my stint at the CECI.

I would like to thank Dr. Clifford J. Weinstein, Director of the Information systems Technology Group at MIT Lincoln Laboratory, for supporting this research and urging me towards the completion of this project work. MIT Lincoln Lab was definitely a gorgeous place to work and a great learning experience. I would like to thank the staff and students of Group 62 for setting a great work environment.

I would like to thank all my friends at MIT whom I met during the two wonderful years spent at Boston. I would like to specially mention my roommate Chandra for proof reading this document, Gangu for the technical discussions, and Rajul for all the care and encouragement. I would also like to thank Prof. John R. Williams, Cynthia, Jesse and Linda for their advice and help.

I would like to thank my parents and my brother for their ever-reliable love and affection. Last but most definitely not the least – “Thank you God for everything”.

-4-
# Table of Contents

ABSTRACT .................................................................................................................. 2  

ACKNOWLEDGEMENTS .......................................................................................... 4  

TABLE OF CONTENTS ............................................................................................. 5  

LIST OF FIGURES ..................................................................................................... 7  

LIST OF EXAMPLES ................................................................................................. 8  

1. INTRODUCTION .................................................................................................... 11  
   1.1. Definition of Survivability ............................................................................ 13  
   1.2. Need for Information Survivability ............................................................... 15  
   1.3. Denial of Service Attacks ............................................................................ 17  
   1.4. Difference between Survivability and Traditional Security/Fault Tolerance __ 18  
   1.5. Dynamically Reconfigurable Multimedia Conferencing Tool ................. 20  

2. RELATED WORK .................................................................................................... 21  
   2.1. Related Work in Information Survivability .................................................... 21  
   2.2. Java-based Conferencing Tools ................................................................... 25  

3. SURVIVABILITY THROUGH DYNAMIC RECONFIGURATION ......................... 27  
   3.1. Dynamic Reconfiguration ............................................................................ 27  
   3.2. Description of Configurations .................................................................... 31  
       3.2.1. Unicast Peer-Peer Configuration .......................................................... 32  
       3.2.2. Client-Server Configuration ............................................................... 35  
       3.2.3. Multicast Peer-Peer Configuration ...................................................... 37  
   3.3. Attack Scenario ............................................................................................ 38  
   3.4. Summary ...................................................................................................... 40
4. PROTOTYPE - A SURVIVABLE CONFERENCING TOOL

4.1. Components of CollabTool

4.1.1. Class CollabComponent

4.1.2. The WhiteBoard - wb

4.1.3. The ChatBoard - cb

4.1.4. The AudioTool - at

4.1.5. The ConfigurationManager - manager

4.1.6. CollabTool

4.2. Design of the Connections in CollabTool

4.2.1. InputChannel and OutputChannel Interfaces

4.2.2. Class SocketInputChannel and SocketOutputChannel

4.2.3. Class PipedInputChannel and PipedOutputChannel

4.3. Design of the Client-Server Configuration

4.3.1. Class Server

4.3.2. Class Handler

4.3.3. Class ClientConfiguration

4.4. Design of the Peer-Peer Configuration

4.4.1. Class MultiInputChannel and MultiOutputChannel

4.4.2. Class PeerConfiguration

4.5. Design of the Dynamic Reconfiguration

4.5.1. Class SwitchInputChannel and SwitchOutputChannel

4.5.2. Class ConfigurationManager (contd.,)

4.6. Summary

5. CONCLUSIONS

5.1. Research Summary

5.2. Future Work and Extensions

REFERENCES
List of Figures

Figure 3.1: Logical Configuration of a Unicast Client-Server System 28
Figure 3.2: Physical Configuration of a Unicast Client-Server System 29
Figure 3.3: Black Box Representing the Connection Abstraction 30
Figure 3.4: Peer-Peer Connection between Two Processes 33
Figure 3.5: Fully Inter-Connected Peer-Peer Connection among Five Processes 34
Figure 3.6: Unicast Routing in a Peer-Peer Configuration 35
Figure 3.7: Client-Server Configuration 36
Figure 3.8: Multicast Routing in a Peer-Peer Configuration 38
Figure 3.9: Server Attacked and Disabled 39
Figure 3.10: Reconfigure to Fully Inter-Connected Peer-Peer Configuration 40
Figure 4.1: Screen Shot of the WhiteBoard, \textit{wb} 46
Figure 4.2: Screen Shot of the ChatBoard, \textit{cb} 48
Figure 4.3: Screen Shot of the AudioTool, \textit{at} 50
Figure 4.4: Screen Shot of the ConfigurationManager, \textit{manager} 53
Figure 4.5: Initial GUI of CollabTool 55
Figure 4.6: CollabTool Awaiting Connection 56
Figure 4.7: CollabTool GUI after Connections 57
Figure 4.8: Client-Server Configuration among CollabTools 64
Figure 4.9: Peer-Peer Configuration between Two Processes 68
Figure 4.10: Design of MultiInputChannel 70
List of Examples

Example 4.1: connect method of class CollabComponent 45
Example 4.2: mouseDragged method of class Whiteboard 47
Example 4.3: run method of the WhiteBoard thread 47
Example 4.4: actionPerformed method of class ChatBoard 49
Example 4.5: run method of ChatBoard thread 49
Example 4.6: mousePressed method of class AudioTool 51
Example 4.7: mouseReleased method of class AudioTool 51
Example 4.8: Construction of Recorder 52
Example 4.9: run method of the Recorder thread 52
Example 4.10: run method of Player thread 52
Example 4.11: actionPerformed method of class ConfigurationManager 54
Example 4.12: itemStateChanged method of class ConfigurationManager 54
Example 4.13: run method of ConfigurationManager thread 54
Example 4.14: Interface InputChannel 58
Example 4.15: Interface OutputChannel 58
Example 4.16: Construction of SocketInputChannel 59
Example 4.17: readObject method of SocketInputChannel 60
Example 4.18: Construction of SocketOutputChannel 60
Example 4.19: writeObject method of SocketOutputChannel 60
Example 4.20: Creation of PipedInputChannel 62
Example 4.21: readObject method of PipedInputChannel 62
Example 4.22: Creation of PipedOutputChannel 62
Example 4.23: writeObject method of PipedOutputChannel 62
Example 4.24: run method of Server thread 65
Example 4.25: run method of Handler thread 65
Example 4.26: broadcast method of class Handler 66
Example 4.27: connect method of class ClientConfiguration 67
Example 4.28: writeObject method of class MultiOutputChannel 69
Example 4.29: addChannel method of class MultiInputChannel

Example 4.30: run method of MultiInputChannel thread

Example 4.31: connect (String) method of class PeerConfiguration

Example 4.32: connect (Socket) method of class PeerConfiguration

Example 4.33: setChannel method of class SwitchInputChannel

Example 4.34: readObject method of class SwitchInputChannel

Example 4.35: setChannel method of class SwitchOutputChannel

Example 4.36: writeObject method of class SwitchOutputChannel

Example 4.37: setConfiguration method of class ConfigurationManager

Example 4.38: connect method of class ConfigurationManager

Example 4.39: setChannel method of class ConfigurationManager
Chapter One

1. INTRODUCTION

In the realm of networked information systems, the escalation of offensive threats versus defensive counter measures makes it difficult to build a system that is totally invulnerable to attack. Attacks on systems can be either internal or external. Sometimes accidents have an adverse effect on the system similar to the effect of an attack. Three kinds of analysis can be performed on systems prone to attacks: vulnerability analysis, lethality analysis and survivability analysis.¹ Vulnerability analysis addresses the ease of attacking or neutralizing the system. For example, in the case of information systems, in a client-server configuration, vulnerability can refer to the ease of implementing a denial of service attack on the server. Lethality analysis addresses the impact of an attack on the system. For example, when the server is subject to a denial of service attack there could be varying degrees of damage done to the client-server system. Survivability aims at the

¹Report on the Background of DARPA’s research in the area of Information Survivability -- http://www.darpa.mil/ito/research/is/index.html
robust operation of systems when they are subject to such attacks. Information survivability involves study of the survivability of large-scale information systems.

Information Survivability is a challenging and relatively new area of research. It deals with the design of robust information systems that are capable of surviving attacks and providing continued service in whole or in part, despite intentional or accidental incapacitation of significant portions of the system. This thesis acknowledges the need for survivable systems and presents a strategy for building survivable information systems. A proof-of-concept prototype has been implemented.

In the remainder of this chapter, we discuss the importance of survivability, the difficulties involved in building survivable systems, and the differences between traditional security and survivability. In the last section we introduce our prototype conferencing tool.

Chapter 2 provides an insight into related research in the area of information survivability. We also examine other available conferencing tools and distinguish them from our prototype conferencing tool.

Chapter 3 explains our strategy of dynamic reconfiguration for building survivable information systems. An example scenario, where dynamic reconfiguration can be used as a survivability strategy against denial of service attacks and the relevant configurations
are described. The prototype we have built, a survivable conferencing tool, implements the dynamic reconfiguration.

In Chapter 4, the elements of the prototype conferencing tool, its design features and the classes designed to implement dynamic reconfiguration are explained.

In Chapter 5, we conclude with a summary of our work and discuss directions for future work.

1.1. Definition of Survivability

Survivability has been a difficult term to define. One proposed definition is: the ability of a system to complete its mission, in whole or in part, in a timely manner, despite the incapacitation of significant portions of the system by attack, failure or accident. The US Army defines survivability as the ability to avoid or withstand the effects of enemy action and continue mission requirements. The term system is used in a broad sense here, and includes networks and large-scale information systems. The term mission refers to a set of very high-level or abstract requirements or goals. Missions are not confined to military settings, because any project has an objective

---

Information Survivability is the study of survivable information systems.

For example, if a financial system shuts down for 12 hours during a period of widespread power outages caused by a hurricane, the system should preserve the integrity and confidentiality of its data and resume its essential services after the period of environmental stress is over. If this occurs, then the system can reasonably be judged to have fulfilled its mission. However, if the same system shuts down unexpectedly for 12 hours under normal conditions and deprives its users of essential financial services, the system can reasonably be judged to have failed its mission, even if data integrity and confidentiality are preserved.

Information Survivability is more than just computer security, safety and fault tolerance; it is a combination of all three. It encompasses many aspects of security and reliability for computers, communication networks, and information systems in general.

Survivability is not an all or nothing phenomenon. There are varying degrees of surviving. Survivability involves designing fault tolerant systems in which some faults are caused intentionally by attacks. Survivability has remained a muddled mixture of different ideas such as reliability, fault-tolerance, safety and availability. From a measurement standpoint, survivability is no easier to quantify than any of the
other characteristics listed above—probably less so, since survivability is a composite of some or all of the above mentioned ill-defined terms [VMG97].

1.2. Need for Information Survivability

Research in the area of Information Survivability involves developing technologies that create strong barriers to attack, detect malicious activities, isolate and repel such activities, and guarantee minimum continued operation of essential and critical system functions in the face of concerted information warfare attacks.

The need for research in the area of Information Survivability has increased for two main reasons. First, systems are increasingly interconnected using Internet technology. Vulnerabilities in this technology or in any connected and networked system can be exploited from anywhere in the network. Most of the security and survivability practices to date have been based on a bounded system paradigm that assumes administrative control over all of the systems’ computational and communication resources. This approach does not support the design of systems that must survive in an unbounded network domain. By unbounded network domain we refer to networks such as the Internet that have no central administrative control, no unified security policy and those in which it is difficult to control the number and nature of nodes connected to the network. Most of the systems developed today have to serve in such unbounded domains. A public Web server and its clients may exist
within many different administrative domains on the Internet. There is no central authority that requires all the clients to be configured in a particular fashion as expected by a Web server. For example, a Web server supporting an E-commerce application may require some plug-in to be installed on the client in order to support a secure transaction. Owing to the unbounded nature of the environment, there might exist a previously installed plug-in that could corrupt, subvert or damage the Web server [EFLLM97]. Survivability should focus on preserving essential services in unbounded environments, even when the systems in such environments are penetrated or compromised [AHH97].

Second, these systems increasingly make use of commercial hardware and software. Because these commercial products were not designed to be secure, they are easy to penetrate. Due to the wide use of a few products, most systems are vulnerable to the same attacks, and many of these attacks are implemented in tools freely available on the Internet. Popular commercial and public domain software components offer an attacker a ubiquitous set of targets, with well-known and typically unvarying internal structure. This lack of variability allows a single attack strategy to have a wide-ranging and potentially devastating impact. For example, an incident that occurred in the fall of 1988 illustrates the fact that all systems and networks are, to some degree, susceptible to attack by destructive programs. On November 2, 1988, a computer "worm" was introduced into the Internet. It replicated uncontrollably for several days by taking advantage of a flaw in the E-mail program sendmail. Eventually, it infected over 6,200 computers nationwide, and overwhelmed the
processing capabilities of many infected machines until they failed completely [JN95]. Thus, commercial technology is not engineered to levels of security and robustness adequate for building critical information systems.

The construction of large-scale information systems from off-the-shelf components whose internal structure is well known, combined with the complexity of designing and implementing software, suggests that no amount of hardening can guarantee the invulnerability of a system to external attacks [LL97]. This inability to build breach-proof systems triggers the need to build robust systems resistant to attacks.

1.3. Denial of Service Attacks

Denial of Service attacks can be defined as attacks interfering with the normal operation of a server, network, or other resources, reducing the capacity of the system to perform its intended functions. Denial of Service can occur due to an attack flooding a server with bogus requests. In some cases a server can also be accidentally flooded with more requests than it can handle, causing the same impact as an attack. Such attacks are very difficult to detect and thwart. For example, a user can send several requests to the server, exceeding the server's capacity to handle such requests. All requests could have false return addresses, so that the server is unable to find the user when it tries to send a response. The server waits, sometimes more than a minute, before closing the connection. When it does close the connection, the attacker
can send a new batch of forged requests, and begin the process again--tying up the
service indefinitely. On July 18th 1997, a Swedish site that challenged Internet users
to break its Macintosh Web server was hit by a denial of service attack. Hackers not
only blocked surfers from accessing the overseas site but also attacked a well-known
U.S. site – MacInTouch, http://www.macintouch.com.\(^3\) Even as recently as March 3\(^{rd}\)
1998, there were numerous customer reports of malicious network-based, denial of
service attacks launched against their networks.\(^4\)

1.4. Difference between Survivability and Traditional
Security/Fault Tolerance

Survivability is quite different from traditional security. It is important to realize the
difference between these two terms often used in the context of information warfare.
Computer Security has been traditionally used as a binary term suggesting that at any
moment a system is either safe or compromised [BCK98]. While traditional security
of a system mainly deals with aspects of identification, authentication and
confidentiality through cryptography, survivability of a secure system is its ability to
continue its secure operations in the event of an attack.

\(^3\) Hackers attack Mac sites - http://www.info-sec.com/denial/denial_072397a.html-ssi
\(^4\) Update on Network Denial of Service Attack - http://www.info-sec.com/denial/denial_030598a.html-ssi
Survivable systems are composed of components that collectively accomplish their mission despite active attacks that might damage significant portions of the system. Robustness under attack is the essential characteristic that distinguishes survivable systems from secure systems. Survivability can benefit from computer security research and provide a framework for integrating security with other disciplines that can contribute to system survivability [LL97].

Though the concept of survivability includes fault tolerance, it is not equivalent to it. Fault tolerance relates to the statistical probability of an accidental fault or combination of faults, not to malicious attack. For example, an analysis of a system may reveal that the simultaneous occurrence of three statistically independent faults will cause the system to fail. The probability of the three independent faults occurring simultaneously by accident may be extremely small, but an intelligent adversary with knowledge of the system’s internals can orchestrate the simultaneous occurrence of these three faults and bring down the system. A fault-tolerant system most likely does not address the possibility of the three faults occurring simultaneously, if the probability of occurrence is below a threshold of concern. A survivable system requires a contingency plan to deal with such a possibility [EFLLM97].
1.5. Dynamically Reconfigurable Multimedia Conferencing Tool

In this thesis, our primary concern is to design a distributed information system, such as a collaborative planning tool, that will gracefully degrade and survive denial of service attacks directed at individual network nodes. We prototype a multimedia conferencing tool that uses dynamic reconfiguration to achieve this functionality. In Chapter 3 we explain a scenario in which a system uses dynamic reconfiguration to switch over to a different configuration or protocol to avoid the effects of an attack. In Chapter 4 we describe our multimedia conferencing tool that consists of a shared white-board, a text-based chat area and a means of voice communication. This prototype, which we shall refer to as “CollabTool” has been built with the ability to dynamically switch its configuration from a fully interconnected peer-peer configuration to a client-server configuration. Our prototype could also switch among several other protocols, though this has not yet been demonstrated. CollabTool has been built using the object-oriented Java™ programming language [AG96] as a proof-of-concept implementation to test dynamic reconfiguration as a survival strategy.

™ Java is a registered trademark of Sun Microsystems Inc.
Chapter Two

2. RELATED WORK

In the following sections, we examine related research in the area of information survivability. We also distinguish other available Java based conferencing tools from our prototype conferencing tool.

2.1. Related Work in Information Survivability

Developing logically correct software and testing as a means for demonstrating correctness have been a focus of computer science research during the last two decades. Unfortunately, present day distributed systems, due to their complexity and size, preclude both of the above. Real systems have very few specified formal properties and are generally difficult to test. Developing and demonstrating survivable distributed systems remains an important research goal [VMG97].
Secure systems use cryptography to provide confidentiality, authentication and integrity for communication between systems. Current research in the field of computer system security takes a narrow view and focuses on hardening a system using firewall technology, or other security measures for host protection, in order to prevent a break-in or other malicious attack. It pays little attention to detection of and reaction to denial of service attacks. Previous work on operating system security has been sponsored by defense establishments and has focused on multilevel security that aims to prevent inappropriate access to various levels of classified information. This technology is not applicable to commercial applications because this concept of security is too narrow and rigid to be usable outside defense establishments.5

At present, systems are rarely designed with security considerations. Security is generally ignored during design stages and later achieved through post-design patches or other add-ons. Survivability at the design stage is the only viable approach that can withstand both the evolution of the system and the evolution of the networked environment to which the system belongs. Security mechanisms that can be customized for the varied needs of many sectors, such as financial, business, healthcare and defense, will encourage mainstream vendors to include them, rather than having to depend on a niche security industry offering add-ons and patches.

5 Report on the Background of DARPA’s research in the area of Information Survivability -- http://www.darpa.mil/ito/research/is/index.html
One approach to the design of survivable information systems has been to provide survivability through redundancy. Ensemble [H98] is a group-ware system being developed at Cornell University. The project is an effort to build a communication system with availability, reliability, fault-tolerance, consistency and security. It uses redundancy as its survivability strategy. But redundancy has its own drawbacks. Although it is an interesting approach to survivability, redundancy by itself is insufficient because backup systems will have identical vulnerabilities. A survivable system would require each backup system to offer equivalent functionality, but to exhibit significant variance from the others in its implementation. That would thwart any attempt to compromise the primary system and all the backup systems with a single attack.

Another approach to survivability has been to provide software diversity based on software mutation - an automated technique for modifying software at the source-code level. Reliable Software Technologies’ research on *Analytical Investigation of Software Mutation for Increased Information Survivability* aims at building an analytical framework describing the vulnerability of distributed programs to repeated attack. This is used to determine the characteristics that must be built into survivable software systems and then experiments using their techniques for software mutation at source-code level are tested.

---

The Multi-Agent Systems Laboratory at University of Massachusetts, Amherst, is investigating the use of distributed adaptive coordination to enhance survivability [BLZ98]. They are developing distributed algorithms to recognize and explain the cause of unacceptable performance of a distributed, multi-agent system. The explanation generated by the diagnosis algorithms is used by other components of the agent to reorganize processing and improve performance given current capabilities and resources. With this approach, they hope to achieve a higher degree of survivability when there are software errors, hardware malfunctions or hostile attacks.

Run-time reconfiguration (RTR) is another approach to developing scalable and reliable computing environments. The Reconfigurable Logic Laboratory in Brigham Young University is investigating development of new programming models and application interfaces that will support RTR [HW95].

The Infospheres project at California Institute of Technology is investigating compositional methods of obtaining high confidence dynamically reconfigurable scalable distributed systems. One focus of their work is the reconfiguration of objects and the connections between objects while they execute.

These approaches to survivability are vastly different from our proposed survival strategy of dynamically switching between different configurations.

---

7 Executive Summary of CalTech Infospheres Project -http://www.infospheres.caltech.edu/infospheres.html
2.2. **Java-based Conferencing Tools**

Since the advent of Java as a programming language [AG96] a number of distributed chat tools, shared whiteboard applications and other multimedia conferencing tools have emerged on the World Wide Web. But not all of these systems are survivable and robust. Most of them have been developed rapidly and lack the fault tolerance required of applications used to exchange valuable and confidential information. Most of them do not even include the simplest of security features in their design.

The Gamelan World Wide Web-site\(^8\) lists a number of such “Network and Communication” systems. Most of these applications are client-server applications, probably due to the ease of network programming and implementing server-side applications in the Java programming language [HHSW97]. In general, the conferencing tools available have a server actively awaiting connections and clients connect to this server in order to interact with other clients. Some of the well-known collaboration tools that follow the client-server architecture include *Habanero*\(^9\) - a Java based collaborative application developed at the University of Illinois at Urbana Champaign, and Shaking Hands Collaboration Tools & *WebCollab*\(^10\) - Java based applications developed at the IBM AlphaWorks.

---


An alternative to client-server design is a fully inter-connected peer-peer configuration. Each participant is connected to every other participant and a central server is not required. The *Personal Chat* application developed by Pinguin Software\textsuperscript{11} has been implemented using this architecture. The CalTech Infospehers Project group also describes the implementation of a Java based distributed system that supports peer-peer communication among processes spread across a network [CRSMR96].

The multimedia conferencing tool we have developed, CollabTool, is a much simpler tool, but has a survivability mechanism designed into it, which the above mentioned conferencing tools do not possess. CollabTool has the capacity to function using both of the above configurations for connectivity among participants and can also dynamically switch between these two configurations.

\textsuperscript{11} Pinguin Software - http://bewoner.dma.be/campbell/
3. SURVIVABILITY THROUGH DYNAMIC RECONFIGURATION

The following sections introduce our concept of dynamic reconfiguration for building survivable information systems. We present an example scenario in which dynamic reconfiguration can be applied to help a system survive denial of service attacks.

3.1. Dynamic Reconfiguration

The configuration of a system usually consists of three parts: the logical configuration, the physical configuration and the mapping of the logical configuration onto the physical configuration. A logical configuration defines a set of processes and their logical communication channels. Logical reconfiguration may involve the
addition or removal of logical processes or communication channels. A physical configuration consists of a set of processors and physical communication links such as the hosts, routers and network links of the Internet. Physical reconfiguration may involve the addition or removal of processors or communication links. The mapping between logical and physical configurations assigns the processes to processors and the logical communication channels to physical paths in the network. The following figures represent a logical configuration of a client-server system (Figure 3.1) and the physical configuration of the same system (Figure 3.2). In Figure 3.2, the server “S” is physically mapped on to the same processor as process “A” and routing details are also explicitly represented.

![Logical Configuration of a Unicast Client-Server System.](image)

*Figure 3.1: Logical Configuration of a Unicast Client-Server System.*
Reconfiguration is said to be dynamic if the act of modifying the configuration of a system occurs while the processes are executing and interacting [SG95]. Dynamic Reconfiguration is a term generally used in conjunction with physical configuration\textsuperscript{12}, but we are using the term with reference to the logical configuration or architecture of the system and its mapping onto the physical configuration of the system. Dynamic reconfiguration is a powerful technique for building robust and fault-tolerant systems.

In order to survive denial of service attacks; the proposed system must have the capability to dynamically reconfigure to avoid the effects of an attack. There might be a decrease in performance and efficiency in this new trade-off configuration, but

\textsuperscript{12} Reconfigurable Data Acquisition Platform, Acquisition systems - http://www.acqsys.com/
the system becomes more robust because it survives attacks that are deadly in the earlier configuration. The implementation can be such that the processes are oblivious to the changes made to the configuration.

We can consider our distributed system to be a set of processes connected together in some fashion. We can represent the connections by a *black box* as shown in Figure 3.3.

![Figure 3.3: Black Box Representing the Connection Abstractions.](image)

The system is designed such that the processes A, B, C, D and E continue interacting oblivious to the internal details of the *black box*. The *black box* could represent a fully inter-connected unicast or multicast peer-peer configuration among all of the
processes, or it could represent a client-server configuration. It could represent different protocols, or different versions of a protocol. Our system design allows dynamic switching of the contents of the black box in a way that is invisible to the active processes.

The system can be pre-programmed with several diverse communication configurations. When a disruptive attack is detected, the black box switches from its current configuration to another configuration where the impact of the attack will be less severe and the system can continue to function uninterrupted. For example, if the server in a client-server system becomes over-loaded or fails due to a denial of service attack, the system is reconfigured in reaction to the attack; clients disconnect from the server and use a fully inter-connected peer-to-peer configuration among themselves. But the denial of service attack has little or no effect on the system in this configuration, and so the system as a whole is able to survive attacks on the server.

3.2. Description of Configurations

Distributed information systems can be implemented using one of several different communication configurations. Each configuration can be implemented using one of several protocols. Below we describe the design of our distributed information system, which uses dynamic reconfiguration as a survival strategy. In order to
explain the idea of survivability through dynamic reconfiguration we design a robust conferencing application: a system that helps in collaborative planning and that continues to function when subject to denial of service attacks. Instead of building our conferencing tool in just one configuration, such as a typical client-server configuration, we design it with the ability to continue functioning in another configuration if required. In this example we discuss survival against denial of service attacks, but we believe that dynamic reconfiguration can be extended for design of other fault-tolerant systems and applications subject to different kinds of situations and attacks [SPL98].

3.2.1. Unicast Peer-Peer Configuration

Peers can be defined as communicating processes that are on the same protocol layer of the network. The connections between peer processes are peer-peer connections. In a peer-peer configuration, all communicating network nodes are equals, with no central control [BG91]. If there are N processes, then each of them has (N-1) connections to the other processes, requiring a total of N(N-1)/2 bi-directional connections. Examples of distributed programs in peer-peer configuration include the Unix utilities Talk and YTalk. YTalk is in essence a multi-user chat program.13

13 Ytalk man page – http://www2.uchicago.edu/ns-acs/asa/man/ytalk.html
Let us consider a collection of processes initially connected in a peer-peer configuration where there is full connectivity among all the nodes. Initially, there are just two processes, directly connected as peers as shown in Figure 3.4. As other processes join the ongoing session, more connections are established. The configuration evolves into a fully inter-connected peer-peer configuration in which every process is connected with every other process as shown in Figure 3.5.

*Figure 3.4: Peer-Peer Connection between Two Processes.*
Figure 3.5: Fully Inter-Connected Peer-Peer Connections among Five Processes.

This configuration is robust, for even if one of the processes is attacked and disabled, the others can continue collaborating uninterrupted because they do not depend on the attacked process for communication. As the conference proceeds, suitable recovery actions can be taken at the affected system and it can rejoin the session. But this configuration does not use bandwidth efficiently because multiple copies of the same data traverse a single network link as shown in Figure 3.6. Hence, this configuration does not scale to a large number of participants.
3.2.2. Client-Server Configuration

Our system is also designed to function in a client-server configuration. A client can be broadly defined to be a process that requests a service of another process. A server can be defined to be a process providing services for other processes connected to it through a network. In a client-server configuration, a server communicates with a group of clients. The clients depend on the server, and are subordinate to it [BG91]. Group communication is accomplished with multiple unicast connections between the server and the clients as shown in Figure 3.7. This requires only \( N \) bi-directional
connections to support N clients. The Microsoft NetMeeting\textsuperscript{14} is an application following the client-server configuration.

This configuration makes more efficient use of bandwidth than a peer-to-peer configuration. But it is not as robust because an attack on the server can slow down or even halt the entire system. The client-server configuration, being one of the most common configurations for distributed information systems, needs robustness against such attacks to be built into it for more reliable service. For example, the server could

\textsuperscript{14} Microsoft NetMeeting - http://www.microsoft.com/netmeeting/
be replicated, and the resulting multi-server configuration would be more robust. But replicated server systems are beyond the scope of this thesis.

3.2.3. Multicast Peer-Peer Configuration

As designed, our system can also function in a multicast configuration. Group communication can be accomplished by multiple conventional unicast (point to point) connections, with the sender establishing separate connections to each receiver, or by a multicast (multi-point to multi-point) connection. As long as the group is small, the unicast approach is reasonable. But as the group size increases, the inefficient utilization of bandwidth can affect performance. Figure 3.6 showed unicast connections with intermediate routers when they are connected in a peer-peer configuration.

Multicast is a more scalable approach in which data streams are replicated by routers within the network instead of requiring each sender to generate redundant data streams. This utilizes bandwidth better because no more than one copy of each data stream traverses any link in the network [T97]. Figure 3.8 shows multicast connections among the same group as in Figure 3.6. The MASH toolkit [M97] is a good example of an application built in peer-peer configuration using multicast connections between the nodes.
3.3. Attack Scenario

A robust system would be designed to function under both peer-peer configuration and client-server configuration using unicast or multicast routing. As explained, the client-server configuration is not very fault tolerant. If the server is attacked, (Figure 3.9) then the system fails and the conference is interrupted.
The design of the system allows us to switch to a fully inter-connected peer-peer configuration in the event of an attack on the server, as shown in Figure 3.10. The remaining processes can continue with their session uninterrupted. Thus we have a survivable system that can continue to function even if a part of the system is incapacitated by an attack. There might be a decrease in performance due to the reconfiguration but the design is a trade-off between efficiency and survivability.
Thus by switching between two different configurations, one of which is more fault-tolerant but less efficient than the other, the system can survive denial of service attacks.

3.4. Summary

In this chapter we described a survivable system that uses dynamic reconfiguration as a strategy to increase robustness. The system was designed to function under different configurations and protocols, and to switch between them. The next chapter
explains our prototype implementation of a system capable of dynamic reconfiguration.

In this thesis we concentrate on a system’s capability to survive attacks through dynamic switching of its configuration. There are other aspects of information survivability, such as the use of encryption for confidentiality, integrity and authentication, network monitoring and intrusion detection that are beyond the scope of this thesis. The design of the conferencing tool, explained in the next chapter, concentrates on the classes required to achieve dynamic reconfiguration.

In our prototype, the CollabTool, we switch between two configurations of peer-to-peer and client-server. We explain CollabTool and all its features in detail in the next chapter. CollabTool has been designed to test if the system can actually switch from one configuration to another and continue its function in an acceptable manner.
4. PROTOTYPE – A SURVIVABLE CONFERENCING TOOL

In Chapter 2 we introduced our prototype conferencing tool, CollabTool, as one that has survivability features not present in other conferencing tools. In the following sections, we describe the various components of the CollabTool and their function. We step through the design of the prototype with illustrations of a few of the critical methods in important classes and explain how the intended functionality is achieved. Detailed code for all methods and classes is not provided. A few screen shots of the prototype are included.

CollabTool is a conferencing tool built using version 1.1 of the Java programming language [AG96]. It uses dynamic reconfiguration as a survival strategy. It is designed to function under a fully interconnected peer-peer configuration and a client-server configuration, and has the ability to switch between these configurations.
The classes designed to implement this switching can be used by other applications to switch between various configurations and protocols.

4.1. Components of **CollabTool**

CollabTool consists of:

- **wb**, a shared whiteboard.
- **cb**, a text-based chat area.
- **at**, a platform dependent means of voice communication.
- **manager**, which manages the configuration and enables connections.

The conferencing tool has been designed to use objects that provide an abstraction of the underlying complications of network streams.

4.1.1. **Class CollabComponent**

CollabComponent is a base class from which all the components of the CollabTool are derived. It is used to set the InputChannel and OutputChannel for the components of the conferencing tool. Each of the components of our prototype uses the InputChannel and OutputChannel as
their basic interface to network communication. This provides an abstraction of the actual streams being used for communication. The InputChannel and OutputChannel are described later in Section 4.2.1. The connect method of the CollabComponent class sets the InputChannel and OutputChannel as shown in Example 4.1.

```java
public void connect(InputChannel in, OutputChannel out) {
    this.in = in;
    this.out = out;
}
```

*Example 4.1: connect method of class CollabComponent.*

### 4.1.2. The WhiteBoard – `wb`

A whiteboard is a simple drawing utility, commonly supplied as part of collaboration frameworks to allow users to share a common drawing space. Our whiteboard, `wb`, provides a means of graphically representing and sharing ideas. `wb` uses a canvas on which participants can draw with the help of a mouse as shown in Figure 4.1.
wb is derived from the CollabComponent class described earlier; it uses an InputChannel and an OutputChannel for communication with other whiteboards. A line object that is drawn on the shared canvas in response to mouse events forms the essence of the whiteboard design. The line object contains the "x" and "y" coordinates for the starting and ending points of the line segments to be transmitted to the shared whiteboard.

When the mouse is pressed, the initial position is recorded. When the mouse is dragged and released, new line objects are created and drawn on the whiteboard and also sent to all the other instances of the whiteboard through the

\[\text{Figure 4.1: Screen Shot of the WhiteBoard, } wb.\]
OutputChannel. For example, the mouseDragged method is implemented as shown in Example 4.2, where "out" is the OutputChannel of wb.

```java
public void mouseDragged(MouseEvent event) {
    Line line = newLine(event);
    board.paint(line);
    out.writeObject(line);
}
```

*Example 4.2: mouseDragged method of class WhiteBoard.*

The objects written to the OutputChannel by one whiteboard are read from the InputChannel by the other whiteboards connected to it. wb constantly reads line objects from the InputChannel and displays them on the canvas. The implementation is shown in Example 4.3, in which "in" is the InputChannel of wb.

```java
public void run() {
    while (true) {
        Line line = (Line)in.readObject();
        board.paint(line);
        Thread.yield();
    }
}
```

*Example 4.3: run method of the WhiteBoard thread.*
4.1.3. The ChatBoard – cb

The ChatBoard, cb is another of the components of CollabTool. cb represents a simple, text based chat utility that enables participants to exchange text messages.

![Screen Shot of the ChatBoard, cb.](image)

cb consists of a text input area and a separate text output area as shown in Figure 4.2. Our simple chat tool allows participants to type messages in the input area that are transmitted to all the participants and appear on the output area of cb.

Similar to the whiteboard, the cb is derived from CollabComponent; it uses an InputChannel and an OutputChannel for communication. A message object is read from the input area and displayed on the output area. cb writes the message object to the OutputChannel when the Return key is pressed, as shown in Example 4.4 in which “out” is the OutputChannel of cb, and “textfield” and “textarea” represent the input area and the output area respectively.
public void actionPerformed(ActionEvent event) {
    String message = textfield.getText();
    textarea.append(message);
    textfield.setText("";
    out.writeObject(message);
    out.flush();
}

*Example 4.4: actionPerformed method of class ChatBoard.*

Objects are read continuously from the InputChannel and displayed on the output area as shown in Example 4.5, in which “in” is the InputChannel of cb.

public void run() {
    while (true) {
        String message = (String)in.readObject();
        textarea.append(message);
        Thread.yield();
    }$
}

*Example 4.5: run method of ChatBoard thread.*

4.1.4. The AudioTool – at

The last conferencing utility of our prototype is a means of voice communication provided by the AudioTool – at (Figure 4.3).
at consists of a button that is released (default position) to listen to incoming audio and is pressed down while transmitting audio to other participants. Thus at either records or plays audio at any point of time.

at uses the Unix audio device file, /dev/audio for audio input and output and hence is not a platform independent implementation. The user must have access to /dev/audio to use at. The version of Java being used, JDK 1.1,\(^\text{15}\) provides support for playing audio only, so we must use /dev/audio to record and play the messages.

at uses an InputChannel and an OutputChannel for communication. The sound object is created using the AudioClip interface provided in the java.applet package of JDK1.1. When the audio button is pressed, sound is captured from the microphone and sent to the network through the OutputChannel. The mousePressed method is shown in Example 4.6. When the button is released sound arriving from the network through the InputChannel is played through the speaker. The mouseReleased method is shown in Example 4.7.

\(^{15}\)JDK 1.1.6 Documentation - http://www.javasoft.com/products/jdk/1.1/docs/index.html
public void mousePressed(MouseEvent event) {
    player.setEnabled(false);
    recorder.setEnabled(true);
}

Example 4.6: mousePressed method of class AudioTool.

public void mouseReleased(MouseEvent event) {
    recorder.setEnabled(false);
    player.setEnabled(true);
}

Example 4.7: mouseReleased method of class AudioTool.

at uses two threads, Recorder and Player, to constantly write sound objects to the OutputChannel and read sound objects from the InputChannel. The Recorder is created as shown in Example 4.8. The Recorder’s run method is shown in Example 4.9, in which “in” represents the audio device and “out” is the OutputChannel of at. The Player’s run method is illustrated by Example 4.10 in which “in” refers to the InputChannel of at, and “out” represents the audio device.
public Recorder(OutputChannel out) {
    enabled = false;
    this.out = out;

    in = new FileInputStream("/dev/audio");
    new Thread(this).start();
}

Example 4.8: Construction of Recorder.

public void run() {
    while(true) {
        in.read(buffer);

        if (enabled){
            SoundBite sound = new SoundBite(buffer);
            out.writeObject(sound);
            out.flush();
        }

        Thread.yield();
    }
}


public void run() {
    while(true) {
        SoundBite sound = (SoundBite)in.readObject();

        if (enabled) {
            sound.play(out);
        }

        Thread.yield();
    }
}

Example 4.10: run method of Player thread.
4.1.5. The ConfigurationManager – manager

The ConfigurationManager has a text input area for typing the IP address or name of the machine to which the connection is requested. It has a drop-down menu for choosing the configuration of the CollabTool as shown in Figure 4.4.

![Figure 4.4: Screen Shot of the ConfigurationManager, manager.](image)

As implemented there are two configuration choices: Peer-Peer and Client-Server. It has a Connect button that establishes connections using the selected configuration and supplies CollabTool with the InputChannels and OutputChannels required for the different components to communicate through the network.

manager maintains a list of configurations and switches to a new configuration that is selected, without interrupting the CollabTool components. It also sends a message to all the other connected managers when a new configuration is selected. This is achieved because the manager is also a CollabComponent. Thus it also accepts configurations sent by other instances over the network. Examples 4.11
and 4.12 illustrate the event handling of CollabTool. The ConfigurationManager constantly listens to new configurations being selected as shown in Example 4.13.

```java
public void actionPerformed(ActionEvent event){
    configuration.connect(address.getText());
}
```

*Example 4.11: actionPerformed method of class ConfigurationManager.*

```java
public void itemStateChanged(ItemEvent event){
    if (event.getStateChange() == ItemEvent.SELECTED) {
        String selection = (String)event.getItem();
        out.writeObject(selection);
        out.flush();
    }
    setConfiguration(selection);
}
```

*Example 4.12: itemStateChanged method of class ConfigurationManager.*

```java
public void run() {
    while(true){
        String selection = (String)in.readObject();
        choice.select(selection);
        setConfiguration(selection);
        Thread.yield();
    }
}
```

*Example 4.13: run method of ConfigurationManager thread.*
4.1.6. CollabTool

Class CollabTool creates instances of the conferencing components, the shared white-board, the chat area and the audio tool and sets up the graphical user interface. It also adds the ConfigurationManager area that controls the configuration being used and enables the connections.

Figure 4.5: Initial GUI of CollabTool.
The tool is initialized with all of its components disabled and with no default configuration selected (Figure 4.5). After a configuration is selected, the Connect button is enabled (Figure 4.6), and after a connection is established the individual tools are enabled (Figure 4.7).
4.2. Design of the Connections in CollabTool

In the previous section, we described the various components of CollabTool. For instances of such CollabTools to communicate with each other under different configurations, they require network connections to be established among them. In this section we describe the interfaces and classes that are used for setting up these connections.

Figure 4.7: CollabTool GUI after Connections.
4.2.1. **InputChannel** and **OutputChannel** Interfaces

The **InputChannel** and **OutputChannel** interfaces provide an abstraction to hide the details of streams and sockets from the application. They provide a simple interface that can be used by the components of the conferencing tool. **InputChannel** and **OutputChannel** interfaces are a subset of **ObjectOutput** and **ObjectInput** interfaces provided in the standard java.io package. **InputChannel** and **OutputChannel** prevent the user from writing and reading anything other than objects. This makes it possible to multiplex objects received from multiple sources into one **InputChannel** without damaging the objects. The **InputChannel** and **OutputChannel** interfaces have been defined as shown in Example 4.14 and 4.15.

```
public interface InputChannel {
    public abstract Object readObject();
    public abstract void close();
}
```

*Example 4.14: Interface InputChannel.*

```
public interface OutputChannel {
    public abstract void writeObject(Object object);
    public abstract void flush();
    public abstract void close();
}
```

*Example 4.15: Interface OutputChannel.*
The individual components, the whiteboard, the chatboard and the audiotool all use the InputChannel and the OutputChannel. But these are interfaces with a high level of abstraction and do not actually establish any network connections. The SocketInputChannel and the SocketOutputChannel classes establish socket connections.

4.2.2. Class SocketInputChannel and SocketOutputChannel

The SocketInputChannel and the SocketOutputChannel implement the InputChannel and the OutputChannel interfaces. The SocketInputChannel provides an InputChannel interface to a Socket (a class in the standard java.net package that provides a network connection) as shown in Example 4.16. It has a readObject method (Example 4.17) that reads objects from this InputChannel.

```java
public SocketInputChannel(Socket socket) {
    in = new ObjectInputStream(socket.getInputStream());
}
```

*Example 4.16: Construction of SocketInputChannel.*
public Object readObject() {
    return (in.readObject());
}

*Example 4.17: readObject method of SocketInputChannel.*

The SocketOutputChannel is similar, and provides an OutputChannel interface to a Socket as shown in Example 4.18 and has a writeObject method (Example 4.19) that writes objects to this OutputChannel.

```java
public SocketOutputChannel(Socket socket) {
    out = new ObjectOutputStream(socket.getOutputStream());
}

*Example 4.18: Construction of SocketOutputChannel.*
```

```java
public void writeObject(Object object) {
    out.writeObject(object);
}

*Example 4.19: writeObject method of SocketOutputChannel.*
```

Thus the SocketInputChannel and SocketOutputChannel provide the connections and streams required to establish a peer-peer connection between two instances of CollabTool, as shown in Figure 3.4, or client-server connection as shown in Figure 3.7, but these details are hidden and CollabTool deals only with an InputChannel and an OutputChannel.
The SocketInputChannel and SocketOutputChannel have to be extended further to achieve a fully interconnected peer-peer configuration among many instances of the CollabTool (as shown in Figure 3.5). This is achieved though the MultiInputChannel and the MultiOutputChannel, which are explained in Section 4.4.1.

### 4.2.3. Class PipedInputChannel and PipedOutputChannel

The PipedInputChannel and PipedOutputChannel implement the InputChannel and the OutputChannel interfaces respectively. They are very similar in function to the SocketInputChannel and the SocketOutputChannel except that they do not deal with network connections. Instead they build an ObjectInputStream and an ObjectOutputStream from a PipedInputStream and a PipedOutputStream respectively. This enables communication through memory rather than over the network. Examples 4.20, 4.21, 4.22 and 4.23 show the creation of the PipedInputChannel and PipedOutputChannel and their readObject and writeObject methods respectively.
public PipedInputChannel(PipedInputStream in) {
    this.in = new ObjectInputStream(in);
}

Example 4.20: Creation of PipedInputChannel.

public Object readObject() {
    return (in.readObject());
}

Example 4.21: readObject method of PipedInputChannel.

public PipedOutputChannel(PipedOutputStream out) {
    this.out = new ObjectOutputStream(out);
}

Example 4.22: Creation of PipedOutputChannel.

public void writeObject(Object object) {
    out.writeObject(object);
}

Example 4.23: writeObject method of PipedOutputChannel.
4.3. Design of the Client-Server Configuration

Individual instances of CollabTool described in section 4.1 can communicate among each other, making use of connections established using classes described in Section 4.2, to operate under different configurations. This section explains the configurations in which CollabTool can function. For simplicity, CollabTool has been implemented to communicate under two different configurations: Client-Server and Peer-Peer, but additional configurations can be implemented using this framework, if required.

The CollabTool has been implemented to function under the Client-Server configuration. In this configuration each of the instances of CollabTool is a client that establishes connections with a central server, as shown in Figure 4.8. There are 4 connections for each CollabTool, one for each of the CollabComponents. The Server is designed to accept all the objects from each of the clients and broadcast these objects to all the clients. The design of this Server is discussed below.
4.3.1. Class Server

The Server is designed to accept multiple connections from clients. It actively awaits new connections from clients and assigns a new handler object to each of the components of the client. Multiple servers are started, one for each component of the tool. Example 4.24 shows the run method of the Server thread, in which "server" is the ServerSocket listening for connections and "connections" is the Vector of connections maintained.
public void run() {
    while (true) {
        Socket client = server.accept();
        new Thread(new Handler(client, connections)).start();
    }
}


4.3.2. Class Handler

The handler thread adds the new OutputChannel to the list of OutputChannels. It reads and broadcasts the objects to all other connected clients. Example 4.25 illustrates the implementation of the Handler where “in” is the InputChannel, “out” is the OutputChannel, and “connections” is the Vector of connections being maintained. Example 4.26 illustrates the broadcast method of the Handler.

protected void broadcast (Object object) {
    Enumeration e = connections.elements ();

    while (e.hasMoreElements ()) {
        OutputChannel c = (OutputChannel) e.nextElement ();
        if (c != out) {
            c.writeObject (object);
            c.flush ();
        }
    }
}

Example 4.26: broadcast method of class Handler.

4.3.3. Class ClientConfiguration

ClientConfiguration is the class used to create the connections required to implement the Client-Server configuration. It creates the SocketInputChannels and SocketOutputChannels that the clients require to communicate with the Server. The array of SocketInputChannels and SocketOutputChannels are passed on to the manager – one element in the array for each component of CollabTool. The connect method of ClientConfiguration is shown in Example 4.27.
Example 4.27: connect method of class ClientConfiguration.

4.4. Design of the Peer-Peer Configuration

The other configuration that CollabTool has been programmed to operate under is a fully interconnected peer-peer connection among several instances of CollabTool. The SocketInputChannel and SocketOutputChannel are used by each of the components of CollabTool to achieve a peer-peer connection between two processes as shown in Figure 4.9, which is similar to Figure 3.4, but explicitly shows the separate connections for each of the CollabComponents.
4.4.1. Class MultiInputChannel and MultiOutputChannel

In a fully inter-connected peer-peer configuration, each instance of CollabTool must be able to read objects from several other instances of CollabTool and also write to all other instances of CollabTool as in Figure 3.5. MultiInputChannel and MultiOutputChannel have been designed to implement this functionality.

The MultiOutputChannel sends copies of the objects to each of several OutputChannels. It maintains an internal list of OutputChannels and adds new OutputChannels to this list when new connections are established. MultiOutputChannel has a writeObject method as shown in Example 4.28.
public void writeObject(Object object) {
    Enumeration e = channels.elements();
    while (e.hasMoreElements()) {
        ((OutputChannel)e.nextElement()).writeObject(object);
    }
}

Example 4.28: writeObject method of class MultiOutputChannel.

The writeObject method of MultiOutputChannel calls the writeObject method of each of the underlying OutputChannels. The MultiOutputChannel's close and flush methods are also passed on to the underlying channels in a similar manner.

The MultiInputChannel is designed to multiplex streams of objects from several sources into one InputChannel. Objects arriving from all the other processes in the conferencing session are presented to each CollabComponent as a single InputChannel. MultiInputChannel maintains a PipedInputChannel and a corresponding PipedOutputChannel connected to it. This pipe provides a buffer for objects that have been received from the network but not yet read by the application. Figure 4.10 illustrates how the MultiInputChannel works.
As new connections are established, a new thread is started to read objects from the network and place them in the pipe as illustrated in Examples 4.29 and 4.30. In Example 4.30, "in" represents one of the InputChannels and "out" represents the PipedOutputChannel. The readObject and close methods of the MultiInputChannel are passed on to the underlying PipedInputChannel.

```java
public void addChannel(InputChannel in) {  
    new Thread(new Copy(in, pipe)).start();  
}
```

*Example 4.29: addChannel method of class MultiInputChannel.*

```java
public void run() {  
    while (true) {  
        Object object = in.readObject();  
        synchronized (out) {  
            out.writeObject(object);  
        }  
    }  
}
```

*Example 4.30: run method of Copy thread.*
4.4.2. Class PeerConfiguration

PeerConfiguration is the class used to establish all the connections required to implement a fully inter-connected peer-peer configuration. Each time the Connect button is pressed a new connection is established by the connect(String) method as shown in Example 4.31.

PeerConfiguration uses an array of Sockets, one for each of the four CollabComponents and creates new SocketInputChannels and SocketOutputChannels for each socket in the array. Example 4.32 shows the addition of these SocketInputChannels and SocketOutputChannels to the MultiInputChannel and MultiOutputChannel that multiplex multiple channels into one in order to implement the peer-peer configuration. It passes on the channel arrays to the manager to establish the connections for each of the CollabTool components.

```
public void connect(String destination) {
    Socket socket[] = new Socket[4];
    for(int i = 0; i < 4; i++) {
        socket[i] = new Socket(destination, port+i);
    }
    connect(socket);
}
```

*Example 4.31: connect(String) method of class PeerConfiguration.*
protected void connect(Socket[] socket) {
    for(int i = 0; i < socket.length; i++) {
        addChannel(new SocketOutputChannel(socket[i]));
        addChannel(new SocketInputChannel(socket[i]));
    }
    manager.connect(in, out);
}

Example 4.32: connect(Socket) method of class PeerConfiguration.

4.5. Design of the Dynamic Reconfiguration

CollabTool has been designed to work both in a fully inter-connected peer-peer configuration and in a client-server configuration. We use dynamic reconfiguration to switch between these two configurations to make CollabTool survivable. SwitchInputChannel and SwitchOutputChannel have been designed to implement the switching.

4.5.1. Class SwitchInputChannel and SwitchOutputChannel

The SwitchInputChannel and the SwitchOutputChannel implement the InputChannel and OutputChannel interfaces respectively. They are used to dynamically switch the communication channels used by an instance of the
CollabTool. They have a `setChannel` method that can be used for this purpose. The `setChannel` method can be used at any time to set the channel being used by CollabTool to communicate. This enables a dynamic reconfiguration from one configuration, such as a peer-peer configuration, to a different configuration such as a client-server configuration.

```java
public void setChannel(InputChannel in_channel) {
    in = in_channel;
}
```

*Example 4.33: setChannel method of class SwitchInputChannel.*

```java
public Object readObject() {
    return (in.readObject());
}
```

*Example 4.34: readObject method of class SwitchInputChannel.*

```java
public void setChannel(OutputChannel out_channel) {
    out = out_channel;
}
```

*Example 4.35: setChannel method of class SwitchOutputChannel.*

```java
public void writeObject(Object obj) {
    out.writeObject(obj);
}
```

*Example 4.36: writeObject method of class SwitchOutputChannel.*
4.5.2. Class ConfigurationManager (contd.,)

As introduced previously, the ConfigurationManager, manager, provides a graphical user interface to manage connections. In response to menu selections or commands received over the network, it creates a configuration object. If a configuration object is already present, then it switches from the old configuration object to the new one as shown in Example 4.37. Examples 4.38 and 4.39 illustrate the connect method and the setChannel method required for it.

protected void setConfiguration(String name) {
    connect.setEnabled(true);

    if (name.equals(peerName)) {
        configuration = peer;
    }
    else if (name.equals(clientName)){
        configuration = client;
    }

    if (configuration.isConnected()) {
        setCh(configuration.getInput(), configuration.getOutput());
    }
}

Example 4.37: setConfiguration method of class ConfigurationManager.
public void connect(InputChannel in[], OutputChannel out[]) {
    setChannel(in, out);
    tool.connect(switchIn, switchOut);
}

Example 4.38: connect method of class ConfigurationManager.

public void setChannel(InputChannel in[], OutputChannel out[]) {
    for (int i = 0; i < in.length; i++) {
        if (switchOut[i] == null) {
            switchOut[i] = new SwitchOutputChannel(out[i]);
        } else {
            switchOut[i].setChannel(out[i]);
        }
        if (switchIn[i] == null) {
            switchIn[i] = new SwitchInputChannel(in[i]);
        } else {
            switchIn[i].setChannel(in[i]);
        }
    }
}

Example 4.39: setChannel method of class ConfigurationManager.

4.6. Summary

CollabTool initially appears with all of the components disabled and without any connections (Figure 4.5). When the user selects a configuration, the Connect button is enabled (Figure 4.6). When the Connect button is pressed, the connect(String) method of the selected configuration is invoked (Example 4.31). The IP address or machine name in the text-input area of the manager
is passed as the argument to this `connect` method. The configuration's `connect` method opens sockets at four different ports, one for each component of the `CollabTool`. `InputChannel` and `OutputChannel` interfaces for these sockets are created (Example 4.32) and passed to the `ConfigurationManager`'s `connect` method (Example 4.38). The manager's `connect` method switches to the new channels using the `setChannel` method (Example 4.39) and invokes the `connect` method of the `CollabTool`. The `connect` method of the `CollabTool` in turn invokes the `connect` method of each of its components. At this point, the `CollabTool` can communicate with other `CollabTools` using the selected configuration. Once connections required for operation in a peer-peer and a client-server configuration are established we can toggle between the two configurations. This ability to switch between the configurations introduces survivability and fault tolerance into the information system. In this chapter, we discussed some of the important methods of the classes that implement the prototype. The classes used for setting up the communication, (the channels) and for reconfiguration could be reused to build the dynamic reconfiguration capacity into other such distributed applications and make them survivable information systems.
5. CONCLUSIONS

5.1 Research Summary

Information survivability encompasses many aspects of security and reliability for computers, communication networks, and information systems in general. In this research, we proposed the use of dynamic reconfiguration as a diversity technique to build robust systems capable of surviving denial of service attacks. We focused on dynamically switching among different configurations, enabling an information system to offer resistance to attacks against individual network nodes or particular protocols. The research did not focus on security issues such as privacy or intrusion detection. The primary concern was to create network applications that can survive denial of service attacks. To demonstrate this concept, we implemented a prototype collaborative planning tool using the Java programming language [AG96]. The
prototype collaborative tools were used to demonstrate and evaluate the concept of dynamic reconfiguration as a survivability strategy.

In the current prototype, dynamic reconfiguration is still a manual process. Failures and attacks are not detected automatically, and user intervention is required to switch configurations. This prototype demonstrates the capability to dynamically switch between peer-peer configuration and client-server configuration during an active group communication session. Initially the prototype was also implemented to use unreliable multicast. The final version of the software did not include a multicast protocol, as the collaboration tools need reliable multicast. Few Java implementations of reliable multicast are available [T97] but they were not incorporated due to time constraints. The capability to control the underlying network communication system in a manner that is transparent to the application, as we have done here, is the first step in surviving attacks and system failures.

5.2 Future Work and Extensions

Survivability is a relatively new area of research and a lot more can be done to build more robust and fault tolerant systems. We first discuss a few logical extensions of this research work.
The current prototype demonstrates a reconfigurable communication system. The reconfiguration has been implemented as a manual process. An intrusion detection component could be added to automate the reconfiguration process. Other aspects of network security, such as the use of cryptography for privacy and authentication, could be addressed to harden the system and make it difficult, but not impossible, to launch successful attacks. However, because attacks will still be possible, the ability to dynamically reconfigure is crucial.

Reliable multicast protocols could be added to the collection of configurations. We could also take advantage of the dynamic capabilities of Java to distribute objects that represent network connections. This would make it possible to dynamically update all nodes with code that implements a new protocol. Thus, instead of switching among several preset configurations, the application could be dynamically updated with objects that know how to transport themselves across the network with a new protocol. At present the prototype has been implemented to function in only two configurations. In future the ConfigurationManager could be implemented to exchange new Configuration objects defined by the user.

The collaboration tool could be enhanced by adding functionality such as loading images in the whiteboard, or by adding new components, such as video communication. Reliability could be enhanced to provide late joiners with data that they may have missed.
References


http://www.cert.org/research/isw97_hypertext/all_the_papers/no1.html


http://dis.cs.umass.edu/research/survive/publications.html


http://www.infospheres.caltech.edu/papers/chandy_etal/hpdc.html


http://www.cert.org/research/isw97_hypertext/front_page.html

http://www-mash.cs.berkeley.edu/dist/mash/papers/mash-nossdav97.ps.gz


http://monet.inria.fr/lrmp/lrmp_rtp.html


http://www.cert.org/research/isw97_hypertext/isw97.html