NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

MAINTAINING HIGH AVAILABILITY IN DISTRIBUTED MOBILE SYSTEMS

by

Brad P. Boitnott

September 2009

Thesis Co-Advisors: Gurminder Singh
John H. Gibson

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 2009	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE Maintaining High Availability in Distributed Mobile Systems	5. FUNDING NUMBERS
6. AUTHOR(S) Brad P. Boitnott.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

Distributed Mobile Systems often require the ability to continue working, even when a major system component fails or there is a fault in the system. In some situations when a distributed mobile system is used, the difference between success and failure could mean the difference between life and death. Therefore, distributed mobile systems that require a high availability must be able to survive faults and resist failures.

A user of a distributed mobile system depends on the ability of the system to share information between the other users of the system. The information and its delivery may be the most important parts of the system, whether it is a data file, picture, or instant message.

A survey will be conducted of the different failsafe and fault-tolerant techniques available then grouped for potential effectiveness and cost efficiency. An experiment with a distributed mobile system and different combinations of failsafe and fault-tolerant techniques will be used to validate the effectiveness of the techniques. The expected result of the experiments is a higher availability rating than the system had before the experiments. TwiddleNet, a distributed mobile system with a high availability requirement, will be used as the platform for experimentation.

14. SUBJECT TERMS High Availability, Failsafe, Fault-tolerant, Distributed Mobile System, TwiddleNet	15. NUMBER OF PAGES 73
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

MAINTAINING HIGH AVAILABILITY IN DISTRIBUTED MOBILE SYSTEMS

Brad P. Boitnott
Major, United States Marine Corps
B.A., Saint Leo University, 2001

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2009

Author: Brad P. Boitnott

Approved by: Gurminder Singh
Thesis Co-Advisor

John H. Gibson
Thesis Co-Advisor

Dr. Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Distributed Mobile Systems often require the ability to continue working, even when a major system component fails or there is a fault in the system. In some situations when a distributed mobile system is used, the difference between success and failure could mean the difference between life and death. Therefore, distributed mobile systems that require a high availability must be able to survive faults and resist failures.

A user of a distributed mobile system depends on the ability of the system to share information between the other users of the system. The information and its delivery may be the most important parts of the system, whether it is a data file, picture, or instant message.

A survey will be conducted of the different failsafe and fault-tolerant techniques available, then grouped for potential effectiveness and cost efficiency. An experiment with a distributed mobile system and different combinations of failsafe and fault-tolerant techniques will be used to validate the effectiveness of the techniques. The expected result of the experiments is a higher availability rating than the system had before the experiments. TwiddleNet, a distributed mobile system with a high availability requirement, will be used as the platform for experimentation.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First, I would like to thank my family for supporting me through this rigorous endeavor. Your understanding of the late nights and early mornings helped me accomplish this goal and I could not have done it without you.

Secondly, I would like to thank my thesis advisors, Dr. Gurminder Singh and Professor John H. Gibson, for their continuous support and guidance. Your feedback on my writing and recommendations on where to go when I was stuck was instrumental to my finishing this thesis.

I would also like to thank Dr. Richard Riehle for lending me access to his extensive library. The books that you gave me access to helped me tremendously.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Distributed mobile systems are in use by military forces, fire departments, law enforcement agencies, emergency medical services, and in private organizations all across the world for reporting pertinent information between entities and nodes on that system. The information generated on these systems becomes vital to the success of the mission the system is supporting. The availability of that information could be the difference between life and death, for either the system users or the populace they support or defend. "Improving communications and providing critical information to emergency responders helps save lives," asserts Richard Mirgon, vice president of the Association of Public-Safety Communications Officials (APCO) (Vilaboy, 2009). APCO and Motorola cosponsored a survey that identified better communications, availability of data, and portable devices as a few of the greatest needs for first responders (Vilaboy, 2009).

A user of a distributed mobile system depends on the ability of the system to share information with the other users of the system, as well as with other interconnected systems. The information and its delivery certainly are the most important parts of the system, whether it is a data file, picture, or instant message. The availability of the system components may dictate the success or failure of the distributed mobile system. As an FDNY chief in the North Tower during 9/11 commented, "One of the most critical things in a major operation like this is to have information. Unfortunately, we didn't have a lot of information coming in" (Roemer, 2005). The limitations of the distributed mobile system used during 9/11 significantly affected the responders' abilities to coordinate actions to contain damage.

The availability of these distributed mobile systems are affected by the failures and faults attributed to the components, both software and hardware. By increasing the fault tolerance and failure prevention of the devices in a system, the availability of the data will increase and allow first responders to more

efficiently execute their mission and potentially reduce damage and save more lives. This thesis will survey the different failsafe and fault-tolerant techniques available and then group them for potential effectiveness and cost efficiency. An experiment with a distributed mobile system configured with different combinations of failsafe and fault-tolerant techniques will validate the effectiveness of the techniques. It is expected that the experiments will demonstrate a higher availability rating than the system had before the proactive measures were implemented. TwiddleNet, a distributed mobile system with a high availability requirement, will be used as the platform for experimentation.

A. FAILSAFE

To define failsafe for this thesis, it will first be defined what it means for a system to fail. If a failed component prevents a system from completing its purpose, then the entire system has failed. An example is a system that consolidates all of its data in one database, on one server, and that server crashes. This becomes a failed system because there is no access to that information; therefore, the purpose of that system is lost. A failsafe is required to ensure that the systems purpose is not lost (Storey, 1996).

A failsafe for a computer system is an automatic protection from failure of hardware or software that allows the system to continue with minimal interruption. Making a computer system failsafe can be accomplished in many ways, and it is not only when dealing with computers. However, for this thesis, the concentration will be on techniques for making computer systems failsafe.

B. FAULT TOLERANT

A fault tolerant system is a system that requires the ability to handle faults in the system while maintaining the purpose of the system, albeit at a potentially degraded state, often referred to as graceful degradation. This fault may be caused by the independent failure of a specific component such as a handheld

device or wireless access point, but the purpose of the distributed system is unaffected. Fault tolerance methods can be implemented to handle the faults and ensure the mission success.

Fault tolerance built into a system will ensure that most users continue to benefit from the performance of the system. Assuming that due diligence was done during the coding and testing phases, a deployed system can accomplish fault tolerance by a system design that minimizes single-points-of-failure using fault detection followed by targeted redundancy. Redundancy can be accomplished through hardware or software techniques, but since software fault tolerance cost can be comparatively quite costly, hardware redundancy is usually the best choice (Storey, 1996).

Hardware fault tolerance is required because component failure can happen at any time without warning. What are the primary reasons components, such as CPUs, hard drives, and power supplies fail? What is their failure rate? Answering these questions will provide a targeted list of components or additional resources that systems should apply to ensure fault tolerance in their systems.

C. OBJECTIVE

The objective of this thesis to identify and test methods for increasing the availability of distributed mobile systems. The following research questions will be addressed:

1) What are the different techniques for making a distributed mobile system more available?

2) What combination of techniques offers the best value and highest availability for a distributed mobile system with respect to cost, size, complexity, employment, and availability rating?

D. SCOPE

The scope of this thesis work is limited to failsafe capabilities for distributed mobile systems that utilize a database. The purpose is to develop a cost-efficient failsafe-design that meets the requirements of a distributed mobile system.

E. ORGANIZATION

The organization of the rest of this thesis is as follows: Chapter II–Background, Chapter III–Prevention Methods, Chapter IV–Method Combinations and Availability Ratings, and Chapter V–Conclusions and Future Work. Chapter II will discuss faults, failures, and system availability more thoroughly and introduce TwiddleNet as the test platform. Chapter III will discuss the methods for achieving a more failsafe or fault tolerant system with associated pros and cons. Chapter IV will discuss implementing the methods in combinations to determine the most cost-effective implementation. Finally, Chapter V will provide a conclusion and recommendations for future work.

II. BACKGROUND

A. HIGH AVAILABILITY DISTRIBUTED MOBILE SYSTEM

A High Availability Distributed Mobile System is a system that requires 24 hours/7 days a week of operating time over a given period. The critical period for the system may range from as little as 10 minutes for a short burst of activity, to an extended time, depending on mission needs, and the system must be available during that critical period. The system may be critical to protecting property and saving lives; without the system, a first responder may not be able to complete the mission. The users that require this type of sophisticated communications capability typically perform the roles of first responders, including law enforcement agencies, military forces, fire fighters, and emergency medical crews. While not essential to life or limb, other organizations and individuals, such as financial institutions or distribution management entities, may require 24/7 access to their data and peers for which they are willing to pay.

B. AVAILABILITY

Availability is a critical characteristic for determining whether a system is going to meet operational requirements. In other words, is it going to be ready when needed most? Because it can mean the difference between life and death, system availability is very important to first responders, who rely heavily on the situational awareness that a highly available system provides. Banking and business operations also need high-availability systems to prevent a financial crisis and revenue loss. For example, eBay suffered an outage that lasted almost 22 hours in June 1999 and cost the company approximately $5 million in revenues for that quarter and countless subscribers, which equates to untold future profits (Kawamoto, 1999).

The availability of a system as a function of time, $A(t)$, is the probability that the system is operational at the instant of time t. If the limit of this function exists as t goes to infinity, it expresses the expected fraction of time that the

system is available to perform useful work (Siewiorek, 1998). The formula for determining a probability of availability uses the Mean-Time-Between-Failure (MTBF) and the Mean-Time-To-Repair (MTTR) values, either estimated or experimental.

1. MTBF

The Mean Time Between Failures is the average of the lengths of time between consecutive failures, under stated conditions, for a stated period in the life of a functional unit. A more simplified MTBF definition for reliability predictions can be stated as the average time (usually expressed in hours) that a component works without failure (Relex, Reliability Prediction). The MTBF for hardware and software can only be an estimate if it is a new piece of hardware or a new software program. The manufacturer usually provides an MTBF by making an estimate from past performance of similar components—for hardware—but an actual MTBF calculation requires information gathered from reported failures of a particular component within a specific period.

In the real world, failure numbers are not steady over the lifetime of a set of devices. Hardware components tend to experience a high failure rate during an initial burn-in period. During this period, manufacturing defects lead to a large number of early system failures. During the operational lifespan of a set of devices, the MTBF numbers improve. Failures become much more rare as the systems experiencing early failure have been repaired or replaced. Eventually, the devices will begin to wear out. MTBF numbers will steadily grow worse until the devices are replaced with new equipment. (www.tech-faq.com, n.d.) Hardware component failures typically follow a “bathtub curve” like the one depicted in Figure 1. Failure frequency may also increase outside of the estimated MTBF by mishandling hardware or using them in harsh conditions outside of normal operating parameters, such as in first-responder situations and military uses.

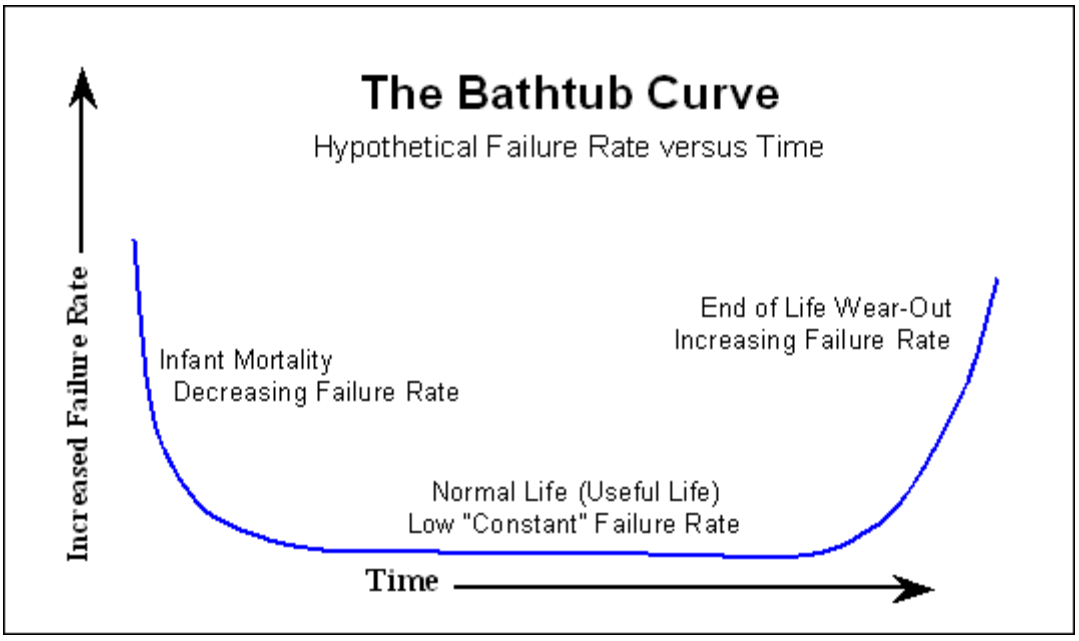

Figure 1. The Bathtub Curve (From Wilkins, 2002)

A Software MTBF is different from Hardware MTBF because software does not wear out as hardware does. The software MTBF rating is the result of a lot of analyzing, testing, and debugging during the software life cycle, to include the requirements phase and the design phase. Faults or "bugs," which are missing, extra, or defective code, accumulate in software programs and can potentially cause a failure (Friedman, Tran, & Goddard, 1992). Therefore, the majority of faults and bugs must be removed before releasing the software. It is almost impossible to remove all faults because some are revealed by events that happen over large periods and are not obvious to human evaluators. An estimate MTBF will be set after testing and debugging, but the final MTBF will be set after a reliability demonstration test (Darroch, 2006).

2. MTTR

MTTR (Mean Time To Repair) is the most common measure of maintainability. It is the average time required to perform corrective maintenance on all of the removable items in a product or system. This kind of maintainability prediction analyzes how long repairs and maintenance tasks will take in the event of a system failure (Relex, 2001).

Maintenance tasks can range from removing and replacing components to just rebooting a system. Additionally, the type of system oversight, whether it is onsite monitoring for 24 hours a day or remotely monitored from 8 a.m. to 5 p.m., makes the MTTR rating better to worse, respectively, as this oversight methodology impacts the responsiveness of outage reporting and maintenance queuing.

3. Calculating System Availability

Availability is a percentage of uptime in a given year while considering planned and unplanned maintenance. Planned maintenance is any maintenance that requires the system to be unavailable. However, it takes into account how it will affect users and is typically scheduled for low usage times. Unplanned maintenance is any maintenance that must be performed but was not expected. It generally involves removing and replacing hardware components or restarting software modules; nonetheless, it is seldom at an opportune time for the user. Calculating an availability score using the MTBF and the MTTR from above creates a percentage of time the system is operable. This is often referred to as the number of “nine’s” it produces, as depicted in Table 1.

Availability	Downtime
90% (1-nine)	36.5 days/year
99% (2-nines)	3.65 days/year
99.9% (3-nines)	8.76 hours/year
99.99% (4-nines)	52 minutes/year
99.999% (5-nines)	5 minutes/year
99.9999% (6-nines)	31 seconds/year

Table 1. Table of Availability and associated Downtime (From EventHelix.com, Reliability and Availability Basics, n.d.)

Individual component availability must be determined before system availability can be projected. Figure 2 illustrates the formula for determining the availability of an individual hardware component. Note that the operational availability also considers the delay in return to service due to logistics or administrative activities.

$$A = \frac{MTBF}{MTBF + MTTR}$$

Figure 2. Formula for Availability of an individual component (From EventHelix.com, Reliability and Availability Calculation, n.d.)

Once the availability is determined for a given hardware component, the value can be used with other component values to determine the availability of a series of components that are dependent on each other. Components that are in series actually reduce the availability of a system because each component must be operational for the system to be so. Figure 3 illustrates the formula for determining the availability of components in a series. The *A* represents the System Availability and A_X and A_y represent the components in the system.

$$A = A_x * A_y$$

Figure 3. Formula for Availability of components in series (From EventHelix.com, Reliability and Availability Calculation, n.d.)

Components that are in parallel are usually components of the same type of equipment and generally support the same functions in a system. By

supporting the same function and being in parallel, the availability of the system will actually increase with parallel components. This method adds a bit of fault tolerance at the hardware level. Figure 4 illustrates the formula for components in parallel. The formula calculates the availability of a module that is composed of two like entities in parallel. It can be extended, however, to reflect any number of entities operating in parallel, whether or not they are identical.

$$A = 1 - (1 - A_x)^2$$

Figure 4. Formula for Availability of components in Parallel (From Shooman, 2002)

Once the serial availabilities and parallel availabilities are determined, the system availability can be derived by taking the rolled up availabilities and calculating them together. After parallel components are calculated, they can be referred to as one component for calculation purposes; therefore, to determine the final system availability, use the equation for serial components, see Figure 3. Figure 5 serves as an example of determining system availability with serial and parallel components.

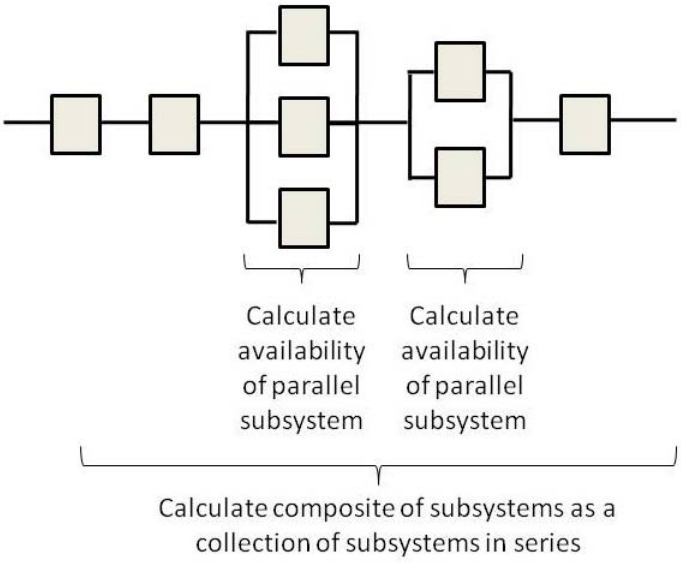

Figure 5. Determining System Availability with Serial and Parallel Components

The availability of a complex system is a little bit harder to determine because it is a marriage between a hardware system and specially created third-party software application, usually not specifically designed to work with each other. Together, the two make up the complex system, as the application cannot run by itself, nor will the hardware perform the required functions without the application. TwiddleNet, for example, is a complex system of software modules loaded onto hardware devices. Currently, the software is loaded on the hardware, mentioned in Section C, but the availability could change for better or worse when the two systems are combined. Additionally, the complex system availability may be higher or lower than the previous complex system availability if hardware components are changed. The availability of the new components may be different. To get the new availability, the hardware availability is multiplied with the application availability and the new complex system availability is created. Additional outside factors may affect overall complex system availability like network congestion but that is outside the scope of this thesis.

C. TWIDDLENET OVERVIEW

TwiddleNet is a distributed mobile system that has enormous potential. Professor Gurminder Singh (2008) states that TwiddleNet:

> … harnesses the power of pervasive edge devices, primarily smart phones, to enable 1) instant content capture and publish; 2) full owner control of content; and 3) search, view and download of content which was previously inaccessible. It exploits the multiple communication modalities available in modern smart phones (GSM/CDMA, GPRS/EDGE, Wi-Fi, Bluetooth—all in a single device) to provide a fail-safe and rapidly-deployable infrastructure which is so critical to first responders. (p. 1).

TwiddleNet is designed to serve the information sharing needs of first responders. It is a system of different components, hardware and software. The following is a description of the components and of the TwiddleNet operations.

1. Client/Handheld

The TwiddleNet Client/Handheld device is a Hewlett-Packard iPAQ hw6945 Smartphone. This device has a built-in camera for capturing events or actions that can be shared across its multiple communications options. The hw6945 has the ability to communicate via Wi-Fi on 802.11b, via a Personal Area Network using Bluetooth, or via GSM (Global System for Mobile Communications). It is also a very powerful device running on a Windows Mobile 5.0 operating system with a 416 Mhz Intel PXA270 processor and 64MB of RAM.

The TwiddleNet Client Software is loaded on the handheld device to enable the connectivity between that device and the other components in the system. The handheld software allows the device to be assigned to a user group to specify the information to be received.

The Client/Handheld has three primary functions: 1) to create metadata for new content captured and notify the Portal of its availability 2) to provide an interface for the user to discover and download new content and 3) to serve content to other Clients'/Handhelds (Glidden, 2009).

2. Portal

The TwiddleNet Portal is the brain of the entire TwiddleNet distributed mobile system. It is run on an OQO Ultra portable PC with a Windows operating system. This type of PC provides a highly mobile capability and helps to ensure a smaller footprint of equipment. This small but powerful PC has a 1.86 Ghz Intel processor and 2 GB of memory but has a very small display. It has the TwiddleNet Portal software installed as an application.

The purpose of the Portal is to handle the connectivity and the sharing of metadata that describes the content that the users have captured. It serves to notify users of newly captured content, provides links for downloading this content from the Clients, and allows users to search for specific content (Glidden, 2009).

3. Command Post

The TwiddleNet Command Post runs on a regular PC or laptop without special configurations. It contains the Command Post software that performs functions that are similar to the functions of the TwiddleNet Client Software but programmed to run on a PC instead of a handheld device.

The primary function of the Command Post is to receive alerts and download the content of each alert into a repository for future use. The content can be viewed via Web pages created by a Web server and data may be searched by keywords or phrases for specific content. "The Command Post is envisioned to be used at a command center or headquarters. It is intended to serve as a situational awareness tool providing real-time information to the commander of an operation to facilitate timely decision making" (Glidden, 2009).

D. TWIDDLENET OPERATION

1. Order of Operation

The mission of TwiddleNet is to allow clients to share information as soon as it is created or search and retrieve information when it is desired. The following is a detailed order of operation for TwiddleNet, which will be referred to later to identify single points of failure and potential fault locations. These single points of failure or fault locations could create a failure in the TwiddleNet system, rendering the system useless or, at the very least, extremely degraded.

The following operations are performed when setting up the TwiddleNet system, but some of them can be done concurrently. The administrator will create a database for the user groups and user names with their passwords on the Portal; this will allow the users to log in to the system and share the content they capture. Once the database is complete, the users can log in when necessary and share content. If a connection is lost during operation, the user will have to log in again to gain access to the content on the other clients, Portal, and Command Post.

The Command Post will also have to be set up to allow for the collection of content in a secondary location other than the clients. The Web server must be running on the Command Post to ensure that the content is saved for its users. The Command Post will also serve as the Dynamic Host Configuration Protocol (DHCP) Server for the devices on the network, if running a wireless local area network.

Once the users have logged in, they can start capturing content. When a user captures content and chooses to share it, the client automatically creates a notification message with the metadata describing the content and sends it to the Portal (Figure 5, Step 1). The Portal then sends a notification message to all clients in the originator's user group and the Command Post (Figure 5, Step 2). If the clients choose to download the content, the file will be downloaded directly from the originating client via an HTTP GET method (Figure 5, Step 3). The Command Post will also perform an HTTP GET method to download the content, store it, and make it available in the future to clients.

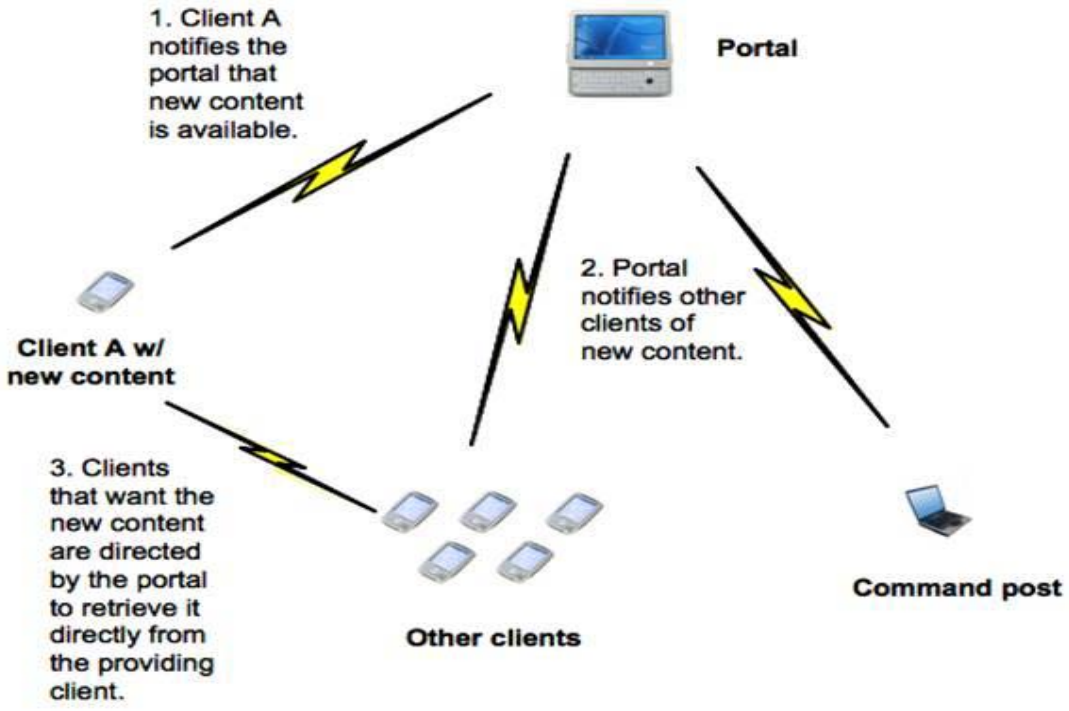

Figure 6. TwiddleNet Order of Operation (From Glidden, 2009)

2. Continued Operation

The TwiddleNet System can continue operating as long as there are clients connected to the Portal and the Portal and Command Post are operating correctly. However, a Client/Handheld can suffer a failure, but the result will be isolated and not affect the rest of the system. To resume operation, the user will have to get a new Client/Handheld then rejoin the system. Any content that has not been uploaded to the portal or to another device will be lost, however. The client software has some fault tolerance capabilities built into it, such as a battery level module that monitors the battery strength. As it decreases to a certain level, an automatic content push to the Command Post may be executed.

The Portal is a different story than the Client/Handhelds. If the Portal experiences enough faults, there could be a system failure because, without the Portal, the Clients'/Handhelds cannot connect with each other to share content. The current configuration of the Portal does not include any kind of failsafe or fault tolerant capability to prevent system failure.

The Command Post can suffer some faults but must continue to serve its mission. Like the Portal, the Command Post does not include any kind of failsafe or fault tolerant capability to prevent system failure. The DHCP Server requires redundancy, as does the data on the server. The failure of some of the other functions of the Command Post can be treated as just faults to the TwiddleNet System.

E. CHAPTER SUMMARY

To get to the optimal availability, software developers and hardware manufacturers must use techniques to limit the amount of faults and failures attributed to their components. The advances of fault-tolerant techniques have spread from the military and space sector to almost all commercial sectors. These techniques are used in individual modules and collectively in systems. Some advanced systems have hardware in triplicate to ensure fault-tolerance, some have software that monitors the health of systems with checks and

balances to detect erroneous outputs, and others have a combination of techniques. The most popular fault-tolerant techniques for making software and hardware more reliable will be discussed in the next chapter.

III. PREVENTION METHODS

Fault-tolerance techniques have been developed since the 1960s and 1970s. Fault-tolerant computing is a generic term describing redundant design techniques with duplicate component or repeated computations enabling uninterrupted (tolerant) operations in response to component failure (faults) (Shooman, 2002). The following techniques may be used alone or combined within a system to produce the desired level of reliability or tolerance. The required availability level can consequently have a major impact on development costs; therefore, in some systems the developers must balance the cost between software techniques, hardware techniques, and added availability.

A. SOFTWARE

1. Requirements Generation

The prevention of software faults starts long before any coding gets done. It starts during the first meeting to discuss what the system is going to do and what the availability of that system needs to be. Safety must be designed into a system and dangers must be designed out; careful consideration of requirements is critically important (Bowen & Stavridou, 1993). The requirements generation for a safety-critical system is crucial to success of the system because it is where the capabilities are coupled with their criticality. For example, a system requirement may be able to print reports to show how many planes landed between 4 p.m. and 5 p.m., but the criticality for that requirement is low. On the other hand, the requirement to track planes as they are on approach to land is very critical and if this requirement fails, the system fails.

A preliminary hazard analysis should be conducted to identify all of the requirements that pose a hazard to the safety of the data or users (Information Processing Limited, 1997). If it is a system like a child's computer game, then the hazard analysis may be less complicated. However, a preliminary hazard analysis on a safety-critical high availability system should produce a list all of the

components that need fault-tolerant techniques to protect them. Once the analysis is complete, a prioritized list should be generated in order to identify the requirements that are to be protected first, which will allow management to choose which lower priority requirements may go un-protected.

There are two philosophies when it comes to specifying and developing safety-critical systems. The first philosophy is that the specification and design will be correct the first time and no further changes will be required. Basically, a perfect program will be created without faults and can be tested and proven to be so. Although this could very well happen, generally a small, non-complex program—a category into which most safety-critical programs don't fall—is created. The second philosophy is that a very good specification and design will be created with no illusions of it being fault-free. There are expectations that faults may be present; therefore, error detection and recovery capabilities are included in the code (Information Processing Limited, 1997).

The requirements generation process can be a very trying process, as all sponsors of specific functionalities feel that their requirement is critical. A balance must be maintained between critical requirements and non-critical requirements because as the number of critical requirements rises so does the cost and complexity. Some systems, like flight control systems and space navigation systems, need to be extremely resistant to faults and a lot of money is spent to ensure it. However, even if a developer had all of the money required to cover the costs, the complexity of the system may be so severe that it will be practically impossible to determine the actual availability.

2. Coding Language with Subsets

Just as the foundation of a house is the most important part of the house, so is the coding language for a software program. The historical choice has been to use the programming language ADA because of its ability to make coding of safety-critical systems easier by removing the harmful functions and keeping sufficient. Experts agree that unsafe constructions exist in all known

assembly and higher order languages; therefore, Table 2 is provided to prompt questions that a project manager and programmer should ask when choosing a programming language.

1.	Wild jumps: can it be shown that the program cannot jump to an arbitrary store location (i.e., can the control flow be totally determined)?
2.	Overwrites: are there language features which prevent an arbitrary store location from being overwritten?
3.	Semantics: are the semantics of the language defined sufficiently for the translation process needed for static code analysis to be feasible?
4.	Model of maths: is there a rigorous model of both integer and floating point arithmetic within the language standard?
5.	Operational arithmetic: are there procedures for checking that the operational program obeys the model of the arithmetic when running on the target processor?
6.	Data typing: are the means of data strong enough to prevent misuse of variables?
7.	Exception handling: if the software detects a malfunction at runtime, do mechanisms exist to facilitate recovery? (e.g., global exception handlers, which may in themselves introduce hazards if used unwisely.)
8.	Safe subsets: does a subset of the language exist that is defined to have properties that satisfy these requirements more adequately than the full language?
9.	Exhaustion of memory: are there facilities in the language to guard against running out of memory at runtime? (e.g., to prevent stack or heap overflow.)
10.	Separate compilation: does the language provide facilities for separate compilation of modules, with type checking across the module boundaries?
11.	Well understood: will the designers and programmers understand the programming language sufficiently to write safety-critical software?

Table 2. List of questions for Project Managers and Programmers (From Cullyer, 1991)

According to Brosgol (2009):

> The language should not contain ‘traps and pitfalls,’ and it should provide features that promote early error detection (at compile time, if possible). The language should have an unambiguous definition so that the effect of any program is predictable (thus, no features with unspecified semantics). Further, the language features should facilitate automated analysis techniques through which the developer can show that the program does what it is suppose to do and does not do what it shouldn’t. (para. 2)

Out of all the questions, it is the subsets that are offered by some languages that make them the better choice for safety-critical systems requiring high availability.

Creating subsets is the perfect means for taking powerful and dynamic programming languages with extensive libraries down to a reliable and analyzable language. Numerous languages have done this, like a subset for ADA called SPARK and a subset for C/C++ called MISRA C/C++. The subsets are meant to remove the complexity that some programs claim as a benefit but can be detrimental to the analysis and auditing of a safety-critical system.

3. Source Code Auditing

Another important method in preventing faults and failures is to perform audits and inspections of the source code. Since the majority of responsibility for software reliability is entirely in the hands and conscience of the software developers, third-party auditing tools are used to assist in auditing (Information Processing Limited, 1997). There are two general types of inspections to accomplish this: static and dynamic testing. No matter which one a team does, or if they do both, the process will work to improve the reliability and availability of a high available system.

Static code analysis is the analysis of program source code before it is executed. The method itself can employ other methods too, like using human analysis to gain program understanding and/or automated analysis using software tools. Human analysis is a necessary step in the full understanding of the code but the analyzer must know the requirements of the code as well to ensure proper analysis. The use of software tools can provide an insight into the program code that a human could never achieve.

Software tools statically analyze the code more thoroughly using mathematical techniques through semantic and interpretation modeling. This means that the checking is done using rigorous mathematical methods which are significantly more successful at catching errors than the human analysis. It also

provides an unbiased examination of the source code. This thesis will not go into the different types of analysis because it is beyond the scope, but two such tools are SPARK Examiner for ADA code and Malpas. Malpas has five analyzers that check code for general problems like bad structure and inconsistent data, as well as pinpointing specific errors such as incorrectly implemented algorithms or inconsistencies with the specification. It supports the following languages: C, ADA, PL/M, ASM, and PowerPC assembly language (Atkins Limited, 2006).

Even with the advances in technology and software programming, it is still an “undecidable” problem to determine if a software program can detect if another software program is going to fail with 100% accuracy. This statement is backed by the works of Alan Turing, Alonzo Church, and Kurt Gödel in the 1930s. However, a significant increase in the level of assurance that the system will not fail is gained by the use of tools during validation.

Dynamic testing is the process of testing the program code while the code is executing. It is the mainstay of verification, extending from testing of individual units of code in isolation from the rest of the software, through various levels of integration, to system testing (Information Processing Limited, 1997). The only caveat to the dynamic testing is that the test must take place in the target environment to produce acceptable results, good or bad.

There are numerous tools that conduct various checks during testing, Insure++ and Holodeck, for example, help to prevent some common errors. They check for memory leaks, memory corruption, race conditions, and other errors. Some tools even produce sequence diagrams for the tester after conducting the testing. The profiling that these tools perform help to identify where a programmer should focus his time in repairing, replacing, or optimizing the code. Sometimes the tool can actually update the source code so that the programmer doesn’t have to go back into the code. Once the cycle of test-fix-test has proven no adverse results, the system can then be submitted for certification.

B. HARDWARE REDUNDANCY

When it comes to making the hardware portion of a system more reliable, which increases its availability, there are two general ways to do it: 1) purchase the components with the highest availability/reliability rating on the market (higher reliability is generally accompanied by higher cost) or 2) use redundancy of the critical components to increase reliability and availability (using lower end components may reduce costs and increase availability). Each method is situational dependant so the solution that is chosen must meet the requirements of the system.

Many systems have been developed with a narrow view of all of the potential pitfalls of that system. One major problem for most systems is the lack of forethought as to what may happen if something stops working in the system. When a single integral component, usually hardware, of a system unexpectedly stops working and causes the entire system to fail, single point of failure has been encountered.

The removal of a single point of failure can be accomplished by redundancy within the system. Redundancy means that there is more than one way to complete a requirement of a system. For example, a system with only one power supply may lose total functionality if the power supply fails, while a redundant power supply would allow the system to continue operating after failover.

One of the most common redundancy solutions is to simply double the instances of the required component. This technique has its drawbacks because the introduction of more hardware into a system means that the size or “footprint” of that system will get larger and the cost will rise.

Constraints may require the purchase of expensive components. For example, a flight control system may have a weight restriction on the component you are working on. The restriction requires a power supply with a high availability rating, but the space allocated for the power supply is limited to the

size of just one unit. Similar requirements force developers to choose the higher cost method to meet the constraints and satisfy the design.

If there were no constraints, a project manager would still need to make a decision as to which method to choose by analyzing the data. The following example (Data Center Design, 2007) will illustrate the decision making dilemma between the two methods:

> -Choice number 1 consists of 4 servers using cheap hardware with no internal redundancy. Each server costs $3,000. You estimate the availability of each server to be 75%.
>
> -Choice number 2 consists of 2 servers using expensive hardware with redundant hard drives and power supplies. Each server costs $20,000. You estimate the availability of each server to be 99%.
>
> -Additional information–you estimate the cost of downtime to be $500/hour, and you expect these servers to support your site load with a single server for the next three years, after which they will be replaced.
>
> -Details–using the above numbers, Solution #1 has an expected availability of 99.6% (obtained by the calculation $1-(1-\text{availability})^{s+1}$, where s=number of spares), at a cost of $12,000. Solution #2 has an expected availability of 99.99% (same formula) at a cost of $40,000. Solution #1 would experience 34 hours/year, or 102 hours over three years of downtime more than Solution #2. Over three years, this extra downtime would cost $51,000. So, by spending an additional $28,000 upfront for Solution #2, you would get a three year return on investment of 182%. Note that the model is only as good as your estimates. If the servers in Solution #2 only had 95% availability, then their combined availability would be 99.75%, which would only provide 13 hours less downtime per year. In this case, you would save $20,000 in downtime over three years for your $28,000 investment, so you would be better off with Solution #1.

The formula used in the above example was complete for the example but has additional capabilities. The complete formula actually includes the number of ways a component can fail, given the number of required nodes to be operational and the given number of spares. The complete formula is in Figure 7, where A is

the availability score, f is the number of ways there are for s+1 nodes to fail, and a is the estimated availability of the component.

$$A = 1 - f(1 - a)^{s-1}$$

Figure 7. Availability equation for an *n* node system with *s* spares (From Highleyman, 2006)

Another method for creating redundancy includes hardware and data redundancy. RAID technology allows users to capitalize on the use of low-cost and possibly less-reliable hard drives configured in a set to protect against failure and data loss. As RAID levels increase, the capabilities and features they offer also increase, creating a more robust and secure data storage. However, specific RAID levels may be more suited to some than others and the higher the level the more the implementation cost, as is generally the case.

The basics behind RAID are to combine two or more hard drives in a single system to provide a more capable storage system or a back-up to the primary hard drive. This provides a redundancy benefit for the hardware and more reliability in the data, too. This combination of benefits has made RAID configurations, not including RAID 0 (simply striping or logical combination of smaller memory devices into a single larger virtual device), a popular security feature for users. Table 3 is a list of the RAID levels and a description of the services they provide.

RAID Level Comparison

Features	RAID 0	RAID 1	RAID 1E	RAID 5	RAID 5EE	RAID 6	RAID 10
Minimum # Drives	2	2	3	3	4	4	4
Data Protection	No Protection	Single-drive failure	Single-drive failure	Single-drive failure	Single-drive failure	Two-drive failure	Up to one disk failure in each sub-array
Read Performance	High	High	High	High	High	High	High
Write Performance	High	Medium	Medium	Low	Low	Low	Medium
Read Performance (degraded)	N/A	Medium	High	Low	Low	Low	High
Write Performance (degraded)	N/A	High	High	Low	Low	Low	High
Capacity Utilization	100%	50%	50%	67% - 94%	50% - 88%	50% - 88%	50%
Typical Applications	High end workstations, data logging, real-time rendering, very transitory data	Operating system, transaction databases	Operating system, transaction databases	Data warehousing, web serving, archiving	Data warehousing, web serving, archiving	Data archive, backup to disk, high availability solutions, servers with large capacity requirements	Fast databases, application servers

Table 3. RAID Levels and descriptions (From Gatan Inc., 2006)

Table 3 details the highlights of the RAID Levels but does not indicate all of the levels. Levels 2, 3, and 4 have become obsolete because their capabilities have been consumed by RAID Level 5. Furthermore, there are some non-standard RAID Levels also not listed that are generally promoted by independent vendors but are not standardized. By analyzing Table 3, a program manager can quickly determine which RAID level his system fits into or which level within which he would like it to operate. Another helpful tool is the RAID level decision flow chart in Figure 8. This chart actually walks a person through the decision process and offers choices. Some of the choices are a little dated but, generally, all recommendations for Levels 2,3, and 4 can be replaced with Level 5.

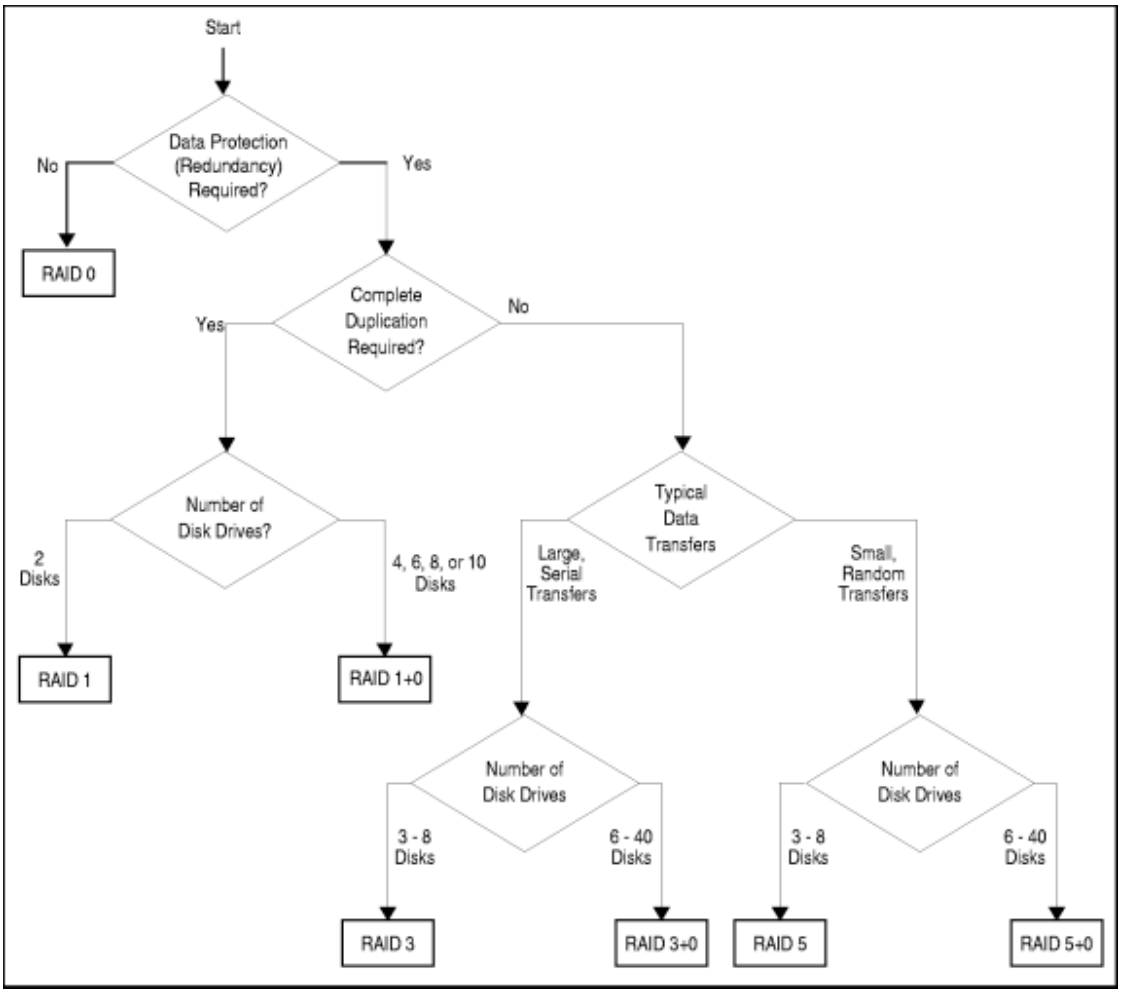

Figure 8. RAID Level Decision Flow Chart (From Hewlett-Packard Development Company, 2002)

Unfortunately, with redundancy comes an increase in complexity since there must be a mechanism for switching from failed component to back-up component. The mechanism can be in the form of an observer or watch officer who switches it over manually or a "watchdog" program that switches it over automatically. The watchdog program would perform the same functions as a human but would not tire nor require bathroom breaks. Balancing the system complexity, redundancy, and acceptance of risk will allow the system owner to come up with a combination of components he needs to ensure availability of the system.

C. DATA REDUNDANCY

Data availability is an essential element in most distributed mobile systems and that is one of the reasons why these systems require high availability. The system facilitates the sharing of the information, whether it is a banking system that allows its users to withdraw monies from two different ATMs in near simultaneous transactions or a military officer seeking up-to-date enemy intelligence. A lot of times, the information is perishable. Often, it is impossible for a user to recreate the information, so data redundancy techniques are paramount to mission success.

Many systems employ a database to store, manage, and organize their data so that users may find information more easily. However, if the database crashes, all the information may be lost. Database replication is a technique that can be used to back-up the data to prevent the loss of information. Database replication employs at least two servers, master and slave, and when data writes are sent to the master database server they are automatically replicated to the slave database server.

With database replication using a master and slave database server, a system may choose to make the slave database server a back-up server. When or if the master database fails or is taken offline for maintenance, the backup server will come online, perhaps automatically, and take over functions of the

master database server. This process is called failover. This scenario illustrates components in parallel and makes the entire system more available and reliable. Numerous database providers, such as Oracle, MySQL, and Microsoft, offer the features necessary to allow database replication and failover.

RAID systems also work to create data redundancy, either by mirroring the contents of one drive onto another or adding error recovery capability through parity-like check bits. This creates a virtual back-up of the data and accessible during a hard drive failure. Additionally, RAID Levels 3 through 7 (Level 5 being the most popular) include parity checking to allow the controller to determine the missing data block through an XOR computation in the case of a disk failure. Once the missing file is determined, the controller can actually rebuild the missing data but this takes considerable computing power from the processor. Both mirroring and striping have pros and cons associated with their implementation but the system administrator must decide what is in the best interest of his system: RAID 1–mirroring with full redundancy but potential higher cost due to reduced drive space, or RAID 5 (since it is most popular)–striping offers better performance but has high computing power requirements.

D. CHAPTER SUMMARY

The methods and techniques for creating a high availability and reliable system, like the availability rating process for software and hardware modules, work together in serial and parallel to create a better, more available system. The techniques were listed singularly but a combination of techniques is often the best method to achieve a successful implementation for a high-availability system. In Chapter IV, combinations of the techniques will be applied to the TwiddleNet System to try and elevate its inherent availability level.

IV. METHOD COMBINATIONS AND RESULTS

The most important trait of the TwiddleNet system is data availability. This chapter will discuss methods detailed in Chapter III, and how the combination of appropriate methods increases the availability level of the TwiddleNet system.

A. CURRENT TWIDDLENET SYSTEM DIAGRAM

The current TwiddleNet system is the result of graduate research and, as is typical in research projects, it also did not follow rigorous software engineering methods for its development. The results of many theses have commented on the single points of failures, but prior to the research presented in this thesis, no work was done to prevent them. Figure 9 is a diagram of the TwiddleNet system design before implementing any redundancy techniques. The items with the red circles around them represent the single points of failure for the system. If any of these items fail, then the TwiddleNet system will fail.

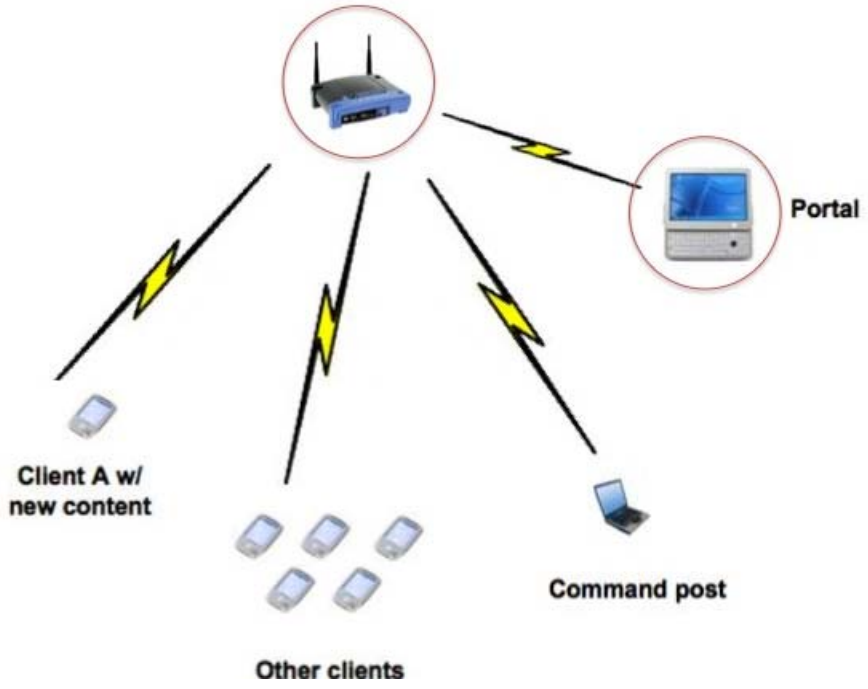

Figure 9. TwiddleNet system with red circles indicating single points of failure

The failure of any of the components circled in red in Figure 9 will result in a failure of the system. Additionally, there are sub-systems in those components that will cause a system failure as well, for example, the user database on the Portal. The wireless access point in Figure 9 can be substituted with which ever wireless means of communication is being used, but that, too, can be a single point of failure. The handheld devices are not considered a single point of failure because they are considered individual components of the system with a very low probability that all of the handheld devices fail at the same time. The Command Post is not considered a single point of failure because as long as the rest of the system is working properly, the Command Post is not needed for minimum functionality.

B. SOFTWARE DEVELOPMENT

The software development history of TwiddleNet does not lend well to developing a highly available system. Currently, there is very little documentation of historical lessons learned or requirements that show where tests were done to ensure proper development. TwiddleNet originated as a graduate research project and hence, did not follow a typical software development process followed in products. The developers of TwiddleNet were graduate-level computer science students, across multiple graduating classes who debugged the code as bugs were found and extended code to add new features to the system. There is no list of current bugs that still to be fixed or a list of bugs that has been fixed. Lists like these are paramount for retracing changes to find new bugs that may have been introduced, like the current memory leak in the Portal software. This is not to diminish the work that the students did, but to illustrate what should have been done for a high-availability system.

A product version of TwiddleNet must be created and maintained within a typical software product development cycle. The software development cycle is the application of a systemic, disciplined, quantifiable approach to development, operation, and maintenance of software which starts with the requirements

generation. Once the requirements are determined and the proper documentation has been created, the process continues to follow specific software development models like the Waterfall Model, Interactive “Chaotic” Model, and/or Spiral Model.

A very important decision in the creation of a high availability system is the language that is chosen with which to create the system. TwiddleNet is written in C# (C Sharp), which provides strong type checking, array bounds checking, detection of attempts to use uninitialized variables, and automatic garbage collection. The C# software language was reviewed against Table 2 in Chapter III and was found to meet the majority of the questions that program managers should ask prior to choosing a language. The most notable capabilities of the C# language are strong type checking, automatic memory management, and an easily understood language. Furthermore, it was designed specifically for use in Windows platforms but can work across platforms for Dynamic HTML, or in specific runtimes for Linux, Solaris, MAC OS X, and Windows, and on Microsoft Silverlight.

Source code analysis must be completed as part of the software development process to ensure that TwiddleNet meets the required specification and requirements. Past releases of TwiddleNet only saw static self-evaluation of the software code due to the sensitive nature of thesis work and the limited time available for student work. Future releases of this system software should include thorough static and dynamic testing to ensure compliance with requirements and testing for functionality. Maybe if more funding and time are available, some dynamic testing tools could be purchased for future projects and testing.

The rest of the methods will discuss combining techniques to make a system more reliable and available. A strong software development program is essential to meeting high reliability and availability; therefore, each of the following methods will assume a proper software validation and verification was

done prior to fielding. The current software components of the system can only realize a better availability score by maintaining a strict software version control program.

C. HARDWARE REDUNDANCY

The first piece of hardware to make redundant is the handheld devices. The handheld device is the key piece of equipment for the first responders in the field and they must be responsive and durable. The most vulnerable part of the HP iPAQ HW6945 is the battery life. The best way to combat a poor battery life is to minimize the use of the extra features like the camera and wireless radio but those are the principal functionalities of TwiddleNet and the reasons the smartphone is the best choice. Therefore, the procedures must be employed to mitigate battery life limitations. Additionally, there has been work with backpack solar cell technology that could be used to recharge backup batteries on the go but that technology will not be discussed in this thesis.

A standing operating procedure (SOP) needs to be in place to dictate that handheld devices get recharged whenever not in use and the user carry an additional battery. If the battery should die during operation, no data will be lost unless the user was in the middle of creating it, but depending on the situation, the recreation of non-instantaneous events may be possible at that time. Unfortunately, the current handheld device only provides an approximate 4-hour battery life without using the camera and wireless radio (Ableitor, 2008). Dirk Ableitor studied the life of mobile handheld batteries and how they suffer greatly as more transmissions are made with high files sizes (Ableitor, 2008).

The latest generations of smartphones are significantly better than the HP IPAQ HW6945 used in TwiddleNet. These newer devices come equipped with better batteries, more advanced power management, and lower power consuming hardware components such as displays, processor, and memory. By

porting the mobile client to the newer smartphones, the operational life of the client device on a single charge can be extended. This would also make the system faster.

The next step in making TwiddleNet more available is to increase the data availability with hardware by using a redundant data capability through a RAID setup on the Portal. The Command Post would benefit from a RAID setup as well. The current configuration only employs a single hard drive setup in each computer; which creates a single point of failure at each hard drive. By setting up a RAID service, whether it is a RAID 1 or RAID 5, the redundancy of the drive and data will create a higher availability score and provide piece of mind in the redundancy of the data.

In the current hardware configuration, the only computer of the two that can introduce a RAID system is the Dell Latitude, which the Command Center is running on. Being a mobile handheld PC, the OQO on the Portal does not have the space or external connections for a RAID setup. To setup a RAID system on the Command Center laptop, an external Serial ATA drive will be used in either a RAID-1 or a RAID-5 setup to increase the availability of the data. Unfortunately, the introduction of a RAID system is overshadowed by the poor availability rating of the Dell Latitude. An increase would be realized but it would not be significant by itself.

The TwiddleNet system availability would increase with the addition of a RAID setup on the Command Center PC but purchasing new computers would be the easiest technique to employ. The current configuration employs an ultra portable PC as the Portal and a Dell Laptop as the Command Center, both of which have an availability rating of 76% over a one-year period (Panasonic, 2008). Purchasing desktop computers like the XPC Shuttle P2 4800E, which can be preconfigured for RAID-1 or RAID-5, may also be a good solution. It includes 2 eSATA connections for additional drives and makes a perfect choice because of its high reliability and the RAID is built in. Unfortunately, they would each require a monitor so that increases the cost of the system.

In Figure 10, the addition of the new computers has made the components more reliable, which means they will be more available. The dotted red circle around the Portal means that a subcomponent of the Portal, the system database, can still fail and cause a system failure. The RAID system provides redundancy for the drives and the data on the drive but only changes drives during a failure and not during a database crash.

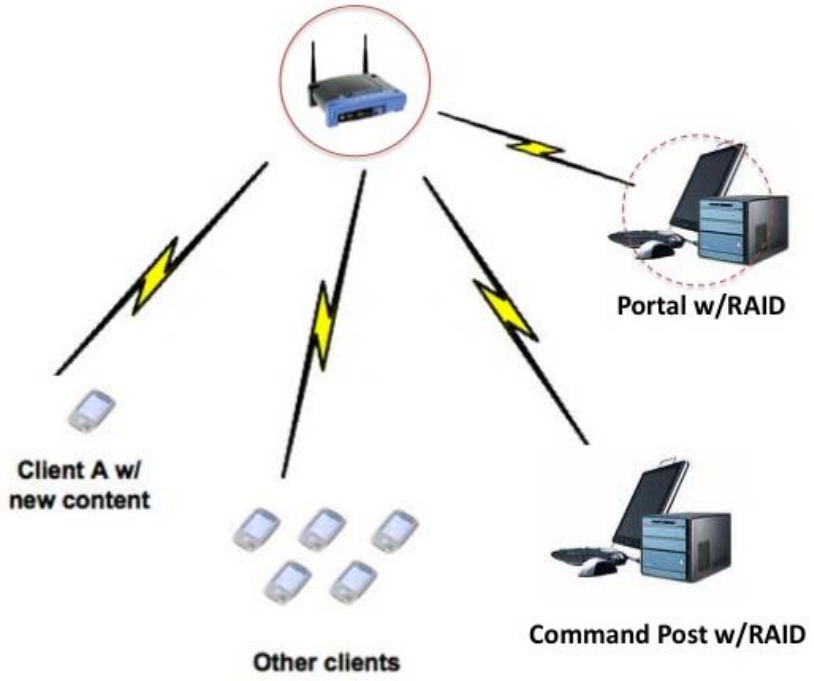

Figure 10. TwiddleNet System Design with XPC Shuttle with RAID

To save on size and cost, more robust computers could be purchased to replace the Portal OQO and the Command Center Dell Latitude. These computers come with higher availability scores, e.g., Panasonic Toughbook, has a 97% availability score (Panasonic, 2008).

A more costly solution, which may generate the highest availability score of all of the configurations, is to purchase primary and backup Panasonic Toughbooks for the Portal and potentially for the Command Post, too. This solution doesn't require an eSATA RAID solution because the redundant Toughbook acts as a redundant drive.

The primary and back-up computers will be connected via a back channel connection to pass the computer's "heartbeat" information and to share data. This connection can be made via crossover CAT-5 connection between the unused Ethernet ports since the computers will be on the wireless TwiddleNet network. The heartbeat report will tell the slave that the master is OK, but if the heartbeat does not respond within a certain time, the slave will take over primary functions.

With this setup, the hardware is redundant and the database can be redundant, too. By having the secondary or slave computer, it provides the ability for the database to run a Master/Slave relationship to protect the data and increase the availability. The following setup is depicted in Figure 11. The database that is running on the TwiddleNet system is a MySQL database and it provides the ability to replicate data in a Master/Slave replication mode. MySQL recommends running the databases in a replication ring so that no data is lost if one database goes down (Dutta, 2005). Additionally, it makes it easier for the former Master to come back online as a slave.

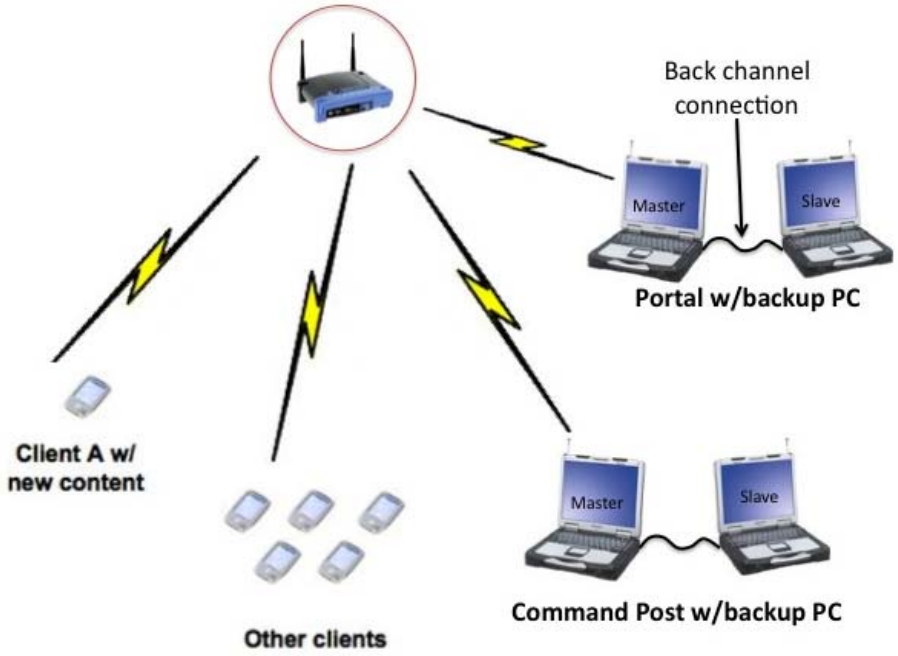

Figure 11. TwiddleNet System Design with redundant PCs and backup databases

D. DATA REDUNDANCY METHOD 1

The first type of data redundancy is via the introduction of a RAID setup on the Portal and potentially the Command Post computer as referred to in Figure 10. This redundancy does a lot for the redundancy of the data on the drives; however, if there is a catastrophic event that destroys or even hinders the Portal in any way, the TwiddleNet system could fail. This catastrophic event could affect the data on the Portal, so to protect the data, it is better to maintain a copy off of the Portal, as well. Another method to data redundancy is to spread the data out across all of the nodes on the system.

The Portal is responsible for numerous functions in the TwiddleNet system, but minimizing these functions could lessen the impact of failure. It is responsible for signing in the users and validating that they are authorized to be on TwiddleNet. This is a key function, and it would be best for it to stay on a more stable platform, such as a PC versus a handheld device, which may move in and out of a coverage area. However, an alternate method may be to not require login to the TwiddleNet system, since the user has already signed on to the wireless network, but to create an overlay network using a distributed hash table instead.

Next, currently the Portal creates a Recipients' Options Configuration that collects the group assignments and sends them back down to the clients. This allows the clients to send their information to only their group (default) or to whom they decide during setup. This includes sending information to the Command Post and/or all devices currently logged in.

The next two responsibilities may be able to be consolidated at the handheld device level to minimize Portal responsibilities. They are Notification of New Content and Alert Generation and Transmittal. The notification of new content comes from the handheld device that generated the content and the Portal currently receives it and begins the alert generation and transmittal process. At the same time, it caches the information for quicker access to the

users and stores the client's location in its database. Instead of the handheld device sending the notification of content to the Portal only, it can send the message to the other users in its group in addition to the Portal.

In this new design, the handheld could actually do more than just send the data to the users in its group; it could actually distribute chunks of the data to the other users to store for redundancy. This procedure would use an erasure encoding scheme, potentially a Hamming or Reed-Solomon encoding scheme, that would break the file into blocks. The scheme will pad the file to make it divisible by the distribution scheme. For example, if the distribution scheme was (7,5), then the file would need to be evenly divisible by five. Then it would distribute the blocks over seven different nodes, repeating some nodes, if necessary, to ensure recall but not placing more than two on any one node. Distribution scheme (7,5) means that the recall procedure only needs to find any five of the distributed seven blocks to recreate the file. The distribution scheme and locations of the blocks of the file are placed in a seed file called an .erasure file. The .erasure file would be stored at the Portal and then executed when the file needs to be recalled. Figure 12 illustrates the process of block dissemination and reconstruction. Additionally, the original file is left on the creators handheld unchanged and it is automatically requested by the Command Post for archiving when it receives a chunk of new data. This provides three methods for data retrieval: direct retrieval from the originator, retrieval from the Command Post, and finally, if all else fails, the data can be recreated by the Portal and hyperlink can be sent out to the requestor.

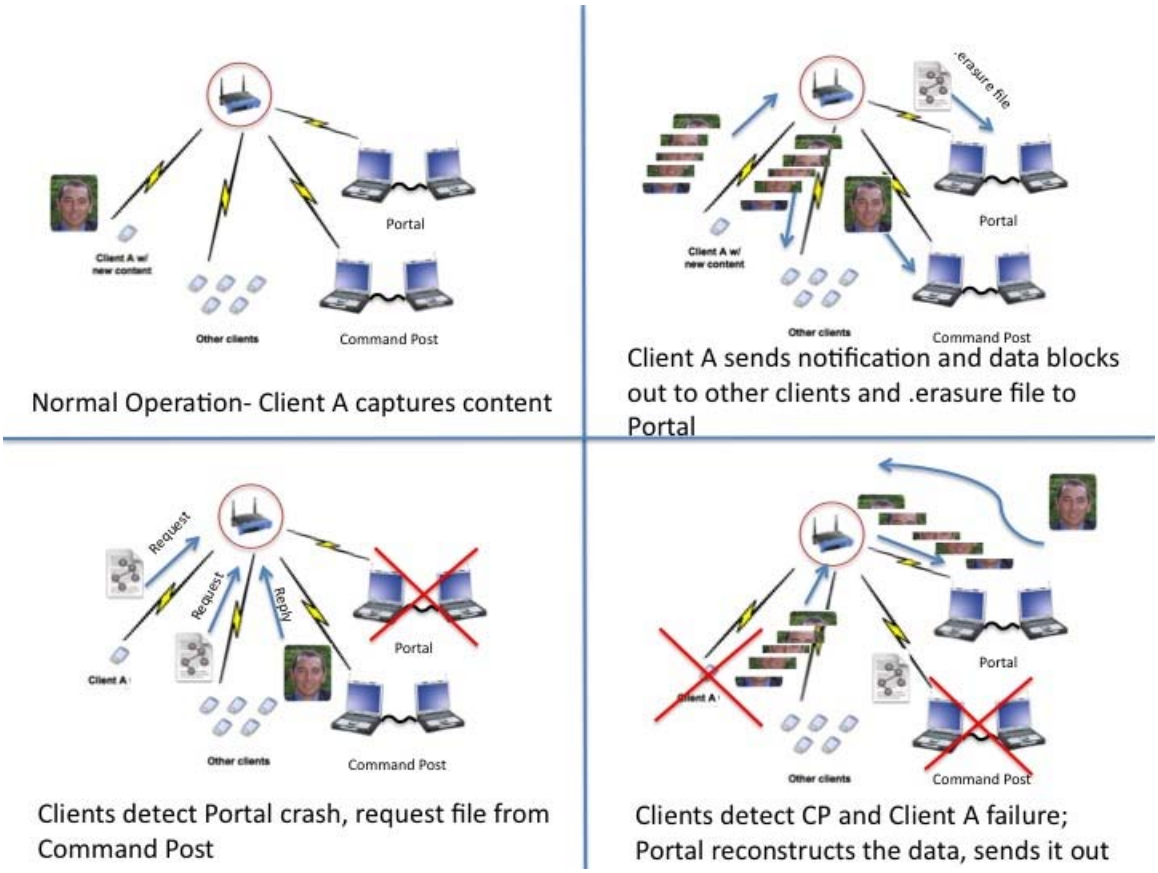

Figure 12. Block distribution, retrieval, reconstruction, and sending

The above detailed scheme allows for the failure of two out of three devices and the data to remain accessible. If the originator handheld fails, the content can be retrieved from the Command Post and vice versa. If both the originator handheld and the Command Post fail, the data can still be reconstructed at the Portal and then sent to the person seeking the data. This creates a triple redundancy of the data.

The choice to use the Portal as the keeper of the .erasure files was made to utilize its processing capability during the reconstruction of a file or data and the database information of the location information for each client. The burden of reconstruction on the handheld devices may have been too much given their current duties as clients.

When the client creates the new content and sends it out, it will send the information and .erasure file to the Portal. The Portal will still provide client connection information; therefore, routing requests to the proper IP address.

The negative aspect of this scheme is that there are no connection management or redirection capabilities within the network if the Portal fails. The users can retrieve archived information from the Command Post but this will make the Command Post a choke point or bottleneck. Furthermore, a failure of both the Portal and Command Post would leave the clients without any means of retrieving information or recreating it from the .erasure files.

E. DATA REDUNDANCY METHOD 2

TwiddleNet utilizes the Portal to maintain all of the connection data for the handheld devices currently connected to the system and it works really well. It actually keeps an up-to-date list of the locations of the devices, as some of the IP addresses may change as devices move through the wireless footprints. When a device wants a piece of data, it sends a request to the Portal to obtain the originator's current IP address so that it may talk to it directly. However, the Portal is susceptible to failure and if it fails, the entire TwiddleNet system fails.

TwiddleNet employs a centralized and non-redundant data management system. By instituting a personal mobile Web server on each handheld device, the TwiddleNet system will gain access and connectivity between its clients and their data during a failure of the Portal.

The data on the TwiddleNet system is inherently redundant in its current design because the Command Post retains a copy of all data obtained by each handheld device. Furthermore, any device that selected the download option on an alert message also holds a copy. The problem with the current configuration is that, without the Portal, users are unable to indirectly or directly access the information contained on other handheld devices. Without the Portal, the system becomes reduced to a collection of devices operating independently on a wireless network because nobody knows who else is on the network. Because

each handheld device has its own IP address, one device could communicate directly with another for the information as long as it knew its address, what data was on the other device, but only if the device was configured properly. With minor additions to the Portal software and the handheld device software, this can be setup to work during the normal operation of the TwiddleNet system.

First, each handheld already publishes its information to the Portal which keeps track of all handheld connection data in a database. In order to support operations without the Portal, each handheld device can also send its data content information with hyperlinks to its data. The Portal can then create a master file of the data on all the handhelds, with the associated links, and periodically push that file to each handheld device. This file would be called the Content Location File. This file would be updated periodically as new data is created or a change in the handheld devices IP address occurs.

When a handheld receives the new Content Location File via a push or pull operation, it compares the old data with the new data. All data that was not updated within a handheld device's submission is assumed to have been deleted and is then removed from the content location file. However, if a device does not submit an update after a significant predetermined period, the Portal will assume the handheld device has failed and it will configure the hyperlink to the Command Post. The Portal would update this master file periodically as new data is created and when connection information changes. This will all be normal operation with the Portal performing the brunt of the work and being able to handle it better than the handheld devices.

In a Portal failure situation, a handheld device can now consult the content location file disseminated by the Portal and request the data directly from the handheld device that created it. When a handheld device detects that the Portal is not responding, it will change modes and automatically begin disseminating its data location file to the other handheld devices on the network. Figure 13 depicts the flow of the Content Data File and the Master Data File during a failed Portal situation.

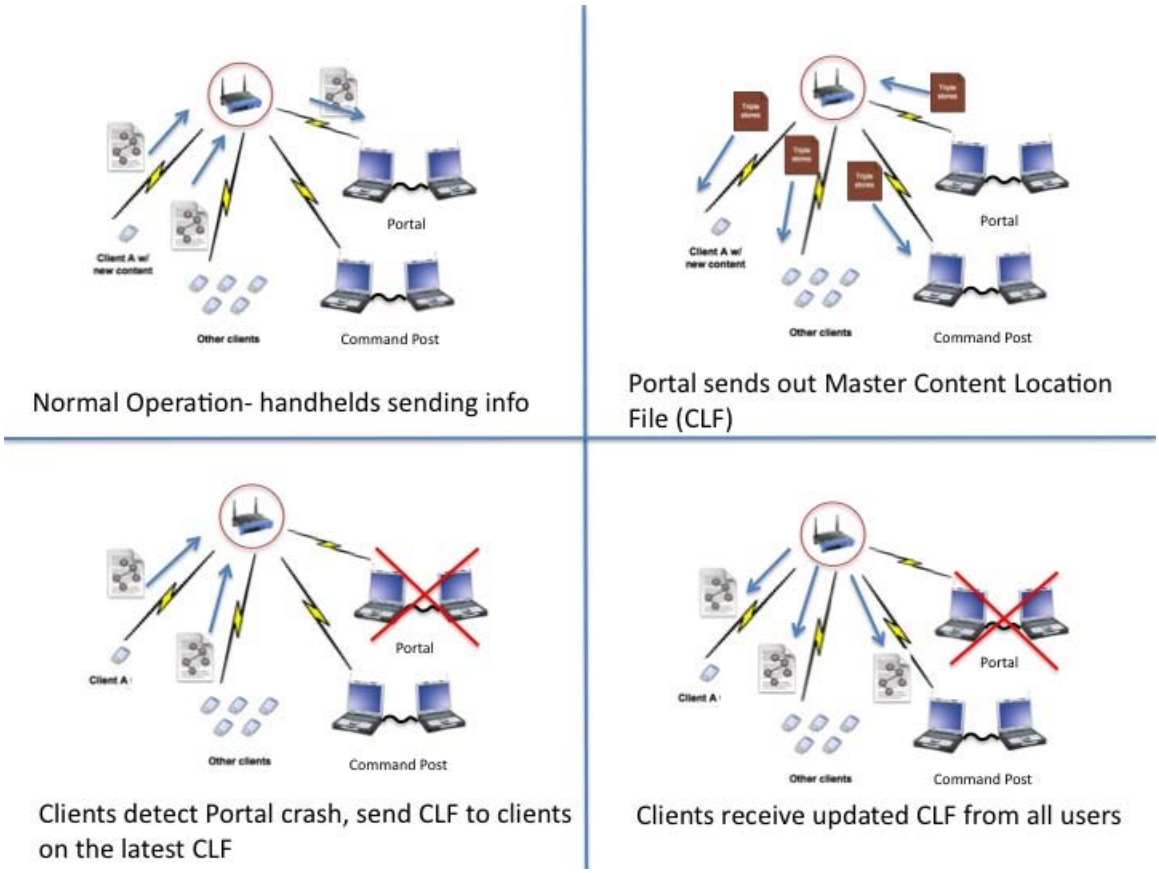

Figure 13. Content Location File Flow during failed Portal scenario

As new handheld devices join the system in this degraded state, it will transmit a multicast message to all devices on the network with its content location file so that the others may add its contact information for future transmissions. This multicast message would also prompt the existing handheld devices to transmit their data content list immediately to the new device. Unfortunately, this will create a lag on the system because all handheld devices will receive a data location file from every other client on the network, but at least the system will still be operational, albeit degraded. In an effort to reduce traffic on the network, the transmission time can be set so that a device only sends a new content location file when a certain amount of new data is created. The content location file would include the senders name, IP address, group identifier,

and hyperlinks to the data. The most probable format for the content location file would be an XML format for easy introduction into the current TwiddleNet system.

This method of communication between handheld devices will continue until the Portal comes back online. Since this is not the optimal means of disseminating the information, once the Portal is back online, all traffic will be routed as previously mentioned.

In order for this type of peer-to-peer communication to work when the Portal is offline, each handheld device will need to run a personal server application so that the others can connect directly to it and download the information. For example, Mobile Web Server 3.0, an application for handhelds and computers, can be downloaded to each handheld device and configured to start automatically when the device is powered on (SphinxSoftware). This Web-frontend interface can display the user's name and what files it currently has available for download.

The Mobile Web Server 3.0 offers many advantages, e.g., no special configuration required, no special programs for access, compatibility with any type of Web browser, any type of files accepted, and it can also display the pictures in a gallery for easy viewing in an Active Server Pages Compact installation. Furthermore, a great feature for first responders that have GPS location capabilities is the "FindMe" feature. It allows a visitor to your Web site to click on "FindMe" and view the latitude and longitude coordinates of that device. This software is for purchase but the functionality may be something that a thesis student could recreate in a programming project.

Active Server Page is a type of service running on specifically configured Microsoft-based Web servers, which allows it to perform additional functions over a Web page. When properly configured and operating, the Web server can interact with a database or file directory and automatically update the content of a

Web page. This feature should be carefully configured on the handheld devices in the TwiddleNet system because it can quickly consume the limited resources.

F. DATA REDUNANCY METHOD 3

To survive a failure of the TwiddleNet Portal, the system must have the ability to retrieve data and connection information without the use of the Portal. The current functions of the Portal and the TwiddleNet system as a whole may have to be diminished to achieve a more stable environment. A proposal would be to adopt a peer-to-peer scheme that focuses more on peer and file discovery. The system architecture is called WMP2P.

WMP2P is a proposal that claims to enable continuous resource discovery and file retrieval for mobile users in wireless mobile networks (Huang, Hsu, & Hsu, 2005). WMP2P tries to protect wireless P2P communications across different wireless networks but it definitely has the potential to support TwiddleNet requirements. It uses Mobile IP solutions to cover roaming customers going from one network to the next but although the connection did not drop the routing changed.

The shortfall of TwiddleNet with a failed Portal is the discovery and retrieval of peer information and data files. WMP2P has developed an algorithm that obtains a fresh status of peers that share files. This indicates that a current record of peers sharing records must be maintained and TwiddleNet does it at the Portal. Disseminating this information is key to using this algorithm so a process must be added to the Portal's software to send a peer sharing record down to the peers for archive. The algorithm is called the receiver-driven discovery control (RDC) algorithm. Figure 14 presents the RDC algorithm.

Algorithm: Receiver-driven Discovery Control (RDC)

Symbols definition:
$DiscoveryTime$: a time unit that defi nes the period of a peer sending discovery messages.
$RequiredTransRate$: a threshold determining a connection quality.
$CurrentTransRate$: the current transmission rate of an active connection that receives data packets.
β: a constant that defi nes the extend and shrink factor of discovery time ($\beta > 1$)
$Threshold_{boundary}$: a constant is defi ned to avoid too small $DiscoveryTime$
End definition

While $DiscoveryTime$ is reached **do**

//Step1: send discovery messages to obtain status of peers that share fi les;

PeerDiscovery();

//Step2: re-calculate the next discovery period;

If (CurrentTransRate > RequiredTransRate) **then**

DiscoveryTime = DiscoveryTime × β ;

Else If (CurrentTransRate < RequiredTransRate) **then**

DiscoveryTime = DiscoveryTime ÷ β ;

End If

//Step3: schedule the next discovery period;

Schedule(DiscoveryTime);

//Step4: avoid too small DiscoveryTime;

If DiscoveryTime < $Threshold_{boundary}$ **then**

Stop fi le transferring and wait a random time to discovery again;

End If

End While

Figure 14. The Receiver-based Discovery Control (RDC) Algorithm (From Huang, Hsu, & Hsu, 2005)

The RDC works to find the peers that shared files before and to update their present location. The algorithm is set in a “while loop,” so it will continue to search and update until the “DiscoveryTime” runs out. This discovery time is altered depending on how strong the bandwidth connection was during the previous selection, longer for stronger and shorter for weaker, which means you search more often to find a better signal. This feature would help TwiddleNet when it is in a degraded state, or in a future TwiddleNet edition, to limit the number of discovery requests sent to lessen bandwidth strain.

File sharing and retrieving is one of the main functions of the TwiddleNet system and the WMP2P Identical File Matching algorithm can help to ensure exact matches for the file requested. Figure 15 presents a decision making flow chart that the algorithm uses when comparing files for identical qualities. Qualities like filename, size, and CRC-32 value. Understandably, one can imagine the checking of the name with a secondary factor to insure uniqueness but just to be sure a CRC-32 value is computed. The cyclic redundancy check is calculated for a file on the file system it is stored and is consulted during retrieval as the seeker presents the same number. These factors are primarily used during interrupted transmissions, but if a peer goes down during an upload, finding an exact match could be difficult.

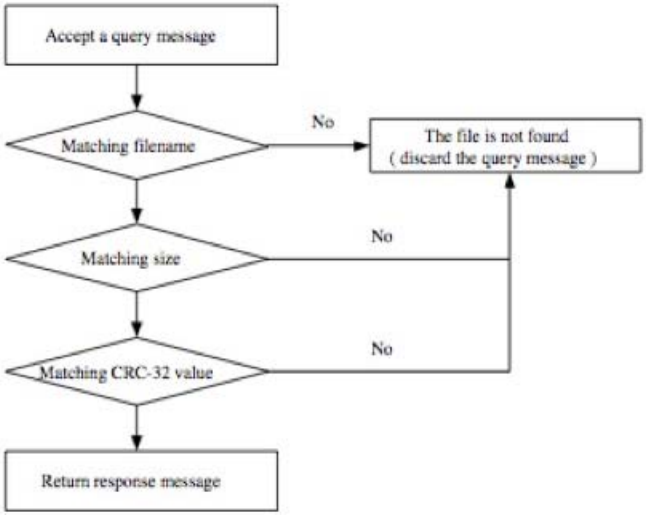

Figure 15. Identical File Matching flowchart (From Huang, Hsu, & Hsu, 2005)

The WMP2P scheme could really help the TwiddleNet system with providing a better identity to the Portal. It would allow the Portal to concentrate on working with the peers of the network and to act as a super-peer. Super-peer is defined as a selected node that provides functions for peers to locate a specific file (Huang, Hsu, & Hsu, 2005). The Portal already does this in one capacity but as the TwiddleNet peers become many and start to cross networks, the Portal will talk to other Portals for information.

G. NETWORK REDUNDANCY

Network redundancy is already incorporated into the TwiddleNet system design because the handheld devices, HP iPAQ HW6945 and HP iPAQ Data Messenger, both employ multiple radios to connect but primarily Wi-Fi or cellular radios. The primary mode of operation of TwiddleNet is to be run over a hastily formed network during emergency situations or disaster relief situations. The easiest to setup is a wireless network with the appropriate coverage.

The current TwiddleNet system only employs one wireless access point for the devices to connect to. This is another single point of failure in the system design as the access point could fail or the user could roam outside of the coverage footprint. The TwiddleNet software on the handheld device and Portal helps the user by maintaining connection information so that when the user comes back into the footprint the connection will automatically be reinstated.

Failure of the access point is the situation that must be remedied by redundant means. By providing a redundant wireless access point for the system, the availability for network devices will increase. Cisco Systems has a feature called access-point hot standby that can handle this situation. The two devices are configured using the same channel in the same coverage area. With only one access point active, the passive access point monitors the network and the primary access point. If the primary fails, the passive access point takes over to provide the network coverage (Oppenheimer, 2004).

If cellular connectivity is primary means of communication, then a redundant means could be Wi-Fi, as redundant cellular capabilities is usually out of the control of the program manager.

H. RESULTS

The results of combining the techniques and methods show potential that there are true benefits in proper software development and redundancy. The software development methods and the hardware redundancy methods undeniably would produce a better TwiddleNet system. These are the easy

aspects of making the system more reliable because they follow a path of measurable steps and milestones and commercially purchased equipment. The tricky part is the data redundancy.

The data redundancy methods offer true potential in making the data in the TwiddleNet system more available. The three methods proposed each provide advantages and disadvantages to utilizing them in the TwiddleNet system.

The scheme using the erasure coding scheme offers redundancy and reliable in the data availability. Integrating its advantages into the existing TwiddleNet system design may require a significant overhaul of the code and functionality between the components of the system. Furthermore, the reconstruction a data file in the erasure scheme can become CPU intensive and that could seriously drain battery life faster on an already short battery life.

The Mobile Web Server offers the advantage of being a commercial-off-the-shelf product currently available for download. It is easily integrated into the current handheld device and worked on the first try. The author was able to download files off a peer handheld device and carry on a chat conversion as well. What has not yet been determined is how much strain on the processor and battery life this may cause and should be considered for future study.

The MWP2P scheme offers great potential, but like the erasure code requires great work to be integrated. A full software development evolution would need to be implemented to ensure the proper fusion of technology and technique. MWP2P, as an underlying scheme to a future TwiddleNet system, could fill voids in the current system for data availability and peer identification, but deployment of such a system could be a long way away.

I. CHAPTER SUMMARY

Recommendations for improvements and implementation of improvements are two very different concepts. For TwiddleNet to become a

more high availability system, drastic measures need to be taken to improve it. Currently, no significant failures have happened; however, the day that failure happens may be a critical day for some, like September 11, 2001, and the failed communications systems (Roemer, 2005). We should follow the motto of the Boy Scouts of America, and “Be Prepared.”

V. CONCLUSION AND FUTURE WORK

A. CONCLUSION

Our research shows that a distributed mobile system, like TwiddleNet, can improve its availability by implementing multiple redundancy techniques. The majority of the techniques should be implemented during the requirements development process, but hardware redundancy and replacement can be incorporated as a modification to a system that has already been deployed. Nonetheless, choosing the best techniques that meet the system's requirements can increase a system's availability.

The benefits of a proper software development program and system design could address many of the availability requirements mentioned in Chapter IV. If the techniques were not used during development, determining which is the best combination of techniques to choose with respect to system and data availability should be a program manager's next step. Since no evaluation of a system can provide the perfect answer to every situation, the program manager will have to determine if the choice recommended is the best for his system.

B. FUTURE WORK

This thesis highlighted several design limitations in the TwiddleNet system, which can be remedied by instituting the techniques and procedures detailed in Chapter IV. The following future work should be completed to create a TwiddleNet system with a higher availability score.

1. Software Development

The requirements for a high-availability system must be determined at the beginning of the planning process. After this is completed, the priorities, coding, and system design can be set to meet the system requirements and ensure a high availability. Any high-available system would benefit from this process; furthermore, no mission critical system would be certified without this process.

The TwiddleNet system should be reworked from scratch to cover a complete system development process. Students from the Software Engineering department and the Computer Science department at the Naval Postgraduate School could be the owners of the project. This group could be lead by professors from both departments and the project could be developed as a class project in both curriculums.

2. Network Devices

Wireless connectivity is generally the primary means of communication during an emergency response situation. The wireless access point needs a redundant capability to ensure the distributed handheld devices have a means to connect to the Portal, Command Center, and/or other handheld devices.

Future testing at the Naval Postgraduate School can also be performed on the LTE (Long Term Evolution) equipment being tested by students in the Computer Science department. LTE technology allows cellular devices to be added to a network and to share data via a 3G data connection. With this capability, the other radios on the handheld devices could be tested while operating the system.

3. Handheld Devices

The current TwiddleNet system handheld devices, HP iPAQ HW6945, are out of date and have out-of-date battery technology. New devices should be purchased to take advantage of new battery technology and advance operating system software. Some new devices (five HP iPAQ Data Messengers) have previously been purchased, and more handheld devices need to be purchased to provide redundant devices and back-up batteries.

Additional concentration may be given to backpack portable solar recharging system. These lightweight man-portable backpack systems have a solar panel and recharging capability fitting for numerous devices. I recommend

future work take a look into a man-pack system that includes this capability and extra space that can carry back-up supplies like secondary handheld or batteries.

Battery life and processor strain while running the Mobile Web Server and the TwiddleNet client at the same time should also be studied.

New TwiddleNet handheld software is being created for employment on the new handheld devices. Once written, analyzed, and thoroughly tested, the software can be implemented on the new device. This process should follow the proper software development process that has been discussed throughout this thesis.

4. Data Redundancy

The techniques listed in Chapter IV, erasure code schemes and WMP2P schemes should be evaluated more fully to determine if TwiddleNet can benefit from their encoding schemes and algorithms. This research may go hand-in-hand with the software development work that was recommended earlier. If a requirements generation is performed, then there should be a discussion about these two techniques.

5. System Design

The TwiddleNet system needs to be designed with redundant capabilities in place. The result of the combination of techniques illustrates the best method, that provides the highest availability, is to purchase new robust computers and a redundant computer for each of the Portal and Command Center. A cheaper option may be to institute a less expensive solution, such as the small form factor desktop computer with built-in RAID.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Ableitor, D. (2008). *Smart Caching for Efficient Information Sharing in Distributed Information Systems .* Naval Postgraduate School, Computer Science. Monterey, CA: Naval Postgraduate School.

Atkins Limited. (2006, January 20). *MALPAS in Static Analysis Tools*. Retrieved August 29, 2009, from www.testingfaqs.org: http://www.testingfaqs.org/t-static.html#Malpas

Bowen, J., & Stavridou, V. (1993, July 01). *Safety-critical methods and systems, formal standards* . Retrieved August 29, 2009, from IEEEXplore: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=225554&isnumber=5883?tag=1

Brosgol, B. (2009, June 01). *When less is more: Programming language technology for safety.* Retrieved August 29, 2009, from VME and Critical Systems: http://www.vmecritical.com/articles/id/?4030

Cullyer, W. G. (1991, March 01). *The Choice of Computer Languages for use in Safety-Critical Systems.* Retrieved August 29, 2009, from IEEEXplore: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=73717&isnumber=2482

Darroch, J. (2006). *Demonstrating Software Reliability.* Madison: Emerson Network Power.

Data Center Design. (2007, October 29). *In Search of Five 9s – Calculating Availability of Complex Systems*. Retrieved August 30, 2009, from edgeblog: Notes from the edge: http://www.edgeblog.net/2007/in-search-of-five-9s/

Dowling, J. A. (2007). *A Gradient Topology for Master-Slave Replication in Peer-to-Peer Environments.* Dublin: Springer-Verlag Berlin Heidelberg .

Dutta, P. (2005, July 26). *MySQL Forums::Replication:: Failover Master Slave Replication*. Retrieved September 17, 2009, from forums.mysql.com: http://forums.mysql.com/read.php?26,35499,36109#msg-36109

EventHelix.com. (n.d.). *Reliability and Availability Basics*. Retrieved June 30, 2009, from http://www.eventhelix.com/realtimeMantra/faulthandling/reliability_availability_basics.htm

EventHelix.com. (n.d.). *Reliability and Availability Calculation*. Retrieved June 30, 2009, from http://www.eventhelix.com/realtimeMantra/faulthandling/system_reliability_availability.htm

Friedman, M., Tran, P., & Goddard, P. (1992). *Reliability Techniques for Combined Hardware and Software Systems.* Rome: National Technology Information Service.

Gatan, Inc. (2006, January 01). *Optimising your PC Hardware for DSV Recording*. Retrieved August 30, 2009, from gatan gets it: http://www.gatan.com/knowhow/knowhow_15/optimizing.htm

Glidden, T. P. (2009). *Privacy for Mobile Networks Via Network Virtualization.* Monterey: Naval Postgraduate School.

Hewlett-Packard Development Company. (2002, January 01). *RAID Level Decision Flow Chart*. Retrieved August 30, 2009, from docs.hp.com: http://docs.hp.com/en/J6173-90007/ch01s04.html?btnPrev=%AB%A0prev

Highleyman, W. (2006, October 01). *Calculating Availability – Redundant Systems*. Retrieved August 30, 2009, from the Availability Digest: http://www.availabilitydigest.com./public_articles/0101/calculating_availability.pdf

Information Processing Limited. (1997, March 07). *An Introduction to Safety Critical Systems.* (I. P. Limited, Ed.) Retrieved August 29, 2009, from www.iplbath.com: www.iplbath.com/pdf/p0826.pdf

Kawamoto, D. (1999, November 9). *eBay spends to eliminate lengthy outages.* Retrieved July 10, 2009, from cNet News: http://news.cnet.com/eBay-spends-to-eliminate-lengthy-outages/2100-1040_3-232705.html

Oppenheimer, P. (2004). *Top Down Network Design Second Edition.* Indianapolis: Cisco Press.

Panasonic. (2008, June 30). *Why toughbook: Failure Rates*. Retrieved September 11, 2009, from Panasonic: Ideas for Life: http://www.panasonic.com/business/toughbook/why-toughbook-failure-rates.asp

Relex, S. C. (2001). *Maintainability - MTTR*. Retrieved July 9, 2009, from http://www.mttr.net/

Relex, S. C. (n.d.). *Reliability Prediction*. Retrieved July 9, 2009, from http://www.relex.com/resources/prediction.asp

Rennels, D. A. (n.d.). *Fault-Tolerant Computing.* Retrieved July 29, 2009, from UCLA Computer Science Department: http://www.cs.ucla.edu/~rennels/article98.pdf

Roemer, T. (2005, September 29). Putting First Responders First: Prioritizing Emergency Communications Spectrum. (H. E. Committee, Interviewer)

Shooman, M. L. (2002). *Reliability of Computer Systems and Networks: Fault Tolerance, Analysis, and Design.* New York: John Wiley and Sons, Inc.

Siewiorek, D. P. (1998). *Reliable Computer Systems Design and Evaluation Third Edition.* Natick: A K Peters, Ltd.

Singh, D. G. (2008, March 24). *Center for the Study of Mobile Devices and Communications.* Retrieved July 13, 2009, from TwiddleNet: Smartphones as Personal Servers: http://faculty.nps.edu/gsingh/CMDC/TwiddleNet/TwiddleNet_Page.htm

Storey, N. (1996). *Safety-Critical Computer Systems.* New York: Addison Wesley Longman Inc.

Vilaboy, M. (2009, July 7). *IP Business News.* Retrieved Julu 7, 2009, from www.ipbusinessmag.com: http://www.ipbusinessmag.com/departments/article/id/358/communications-a-first-priority-for-first-responders

Wilkins, D. J. (2002, November 1). *The Bathtub Curve and Product Failure Behavior.* Retrieved July 9, 2009, from Reliability Hotwire: http://www.weibull.com/hotwire/issue21/hottopics21.htm

www.tech-faq.com. (n.d.). *What is MTBF?* Retrieved July 27, 2009, from http://www.tech-faq.com/mtbf.shtml

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California

4. Director, Training and Education, MCCDC, Code C46
   Quantico, Virginia

5. Director, Marine Corps Research Center, MCCDC, Code C40RC
   Quantico, Virginia

6. Marine Corps Tactical Systems Support Activity (Attn: Operations Officer)
   Camp Pendleton, California