2012

Conformance testing of peer-to-peer systems using message traffic analysis

John Wesley Burris
Louisiana State University and Agricultural and Mechanical College

CONFORMANCE TESTING OF PEER-TO-PEER SYSTEMS USING MESSAGE TRAFFIC ANALYSIS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

In

The Department of Computer Science

By
John Wesley Burris
B.S., Louisiana Tech University, 2003
May, 2012

ACKNOWLEDGEMENTS

My grandfather once told me that I didn't have to go to school to be successful. The only key to success was doing what I loved. This work is the product of doing what I love and I dedicate it the memory of my grandfathers.

To the memory of Joseph H. Burris and Harvell "Mutt" Pierce.

One thing my grandfather didn't mention is the help I would need. It is my pleasure to thank my major professor, Doris L. Carver. My abilities as a writer and a researcher are the product of her constant guidance. I thank Christopher W. Branton for being a counselor, a collaborator, and a frequently used sounding board. I thank my committee, Jianhua Chen, Seung-Jong Park, Jian Zhang, and Ye-Sho Chen, for their comments, suggestions, and their time spent reviewing my research. I want to thank the members of the Software Engineering Lab who were there to discuss the interesting topics in our field and the lessons learned during their own experiences. These members include Coretta Douglas, Allyson Hoss, Yixin "Peter" Luo, Jian Guan, and Zhixiang "Jason" Shen.

I can't say how much I appreciate the love and support of my family. My wife, Natasha, has been there for me every step. My children gave away some "Daddy time" so I could pursue something they didn't understand. Finally, my parents supported me in every way possible. They helped with finances, babysat children, and even gave the occasional kick in the pants.

TABLE OF CONTENTS

ABSTRACT

Peer-to-Peer architectures are used by a large number of distributed systems; however, the challenges such as maintaining a reliable and stable peer-to-peer network can make such networks undesirable for distributed systems. Peer-to-peer architectures are designed to be executed on systems with diverse hardware configurations, distant geographic locations, and varied, unpredictable Internet connectivity that make the software testing process difficult. This research defines a method for conformance testing peer-to-peer content distribution systems called “Method for Conformance Testing by Analyzing Message Activity” (MCTAMA). MCTAMA uses a common representation for describing the behavior of nodes during both design and deployment. ATAMA generates, evaluates and filters test cases that help determine variation between the expected and observed behaviors. The focus on message traffic allows MCTAMA to be used at multiple stages of development and deployment while not being affected by the variations in the operating environment, availability of source code or the capabilities of a monitoring mechanism. As a part of MCTAMA, this research includes a method for combining sequence diagrams to create a description of the expected behavior of nodes in the system.

CHAPTER 1 INTRODUCTION

Software testing is the process of determining if a system's operational behavior is performing as required. In involves the execution of software with test inputs for which the output can be compared to a design specification or requirement. [Sommerville] The techniques used for testing can vary significantly as testing is done at different stages of development and for different types of systems. Conformance testing is the process of verifying that a software system meets the specifications of the design or standard that guided the development of that system. [Garstecki] Creating a high-quality software design is only useful if the characteristics of that design are followed during the implementation process as well. If design and code do not correspond, then a quality software design may not translate into a quality software product. [Pfleeger] Conformance testing is a form of dynamic testing, which means that the system is operated and observed so that the actual behavior of the system and the designed behavior of the system can be compared.

The testing process revolves around the test case. A test case defines scenarios for testing which include a sequence of actions to be performed on the systems, a set of conditions/inputs, and the expected outcome of the actions. Since test cases are used for many types of testing, the attributes of the test case can vary significantly. Regardless, all test cases should have a measurable expected outcome to produce a meaningful test result.

Peer-to-Peer (P2P) computing is the sharing of resources among interconnected systems. P2P content distribution systems utilize the bandwidth and storage of nodes within a network to perform the purpose of the system and distribute content over the network. This approach to

content distribution over a network is an alternative to the traditional client-server model that requires a single server to have sufficient resources to distribute the content. With the client-server model, if the server does not have sufficient resources, the network suffers from bottleneck and the performance of the network is drastically diminished. P2P content distribution systems vary by architectures, node discovery techniques, methods of locating content, and network configuration algorithms. Since the purpose of P2P is to distribute the functions of the system to the nodes of a network, it is difficult to perform a test on a deployed P2P system is challenging. Within a software system, "a software or hardware module that encapsulates a set of related functions, data, or responsibilities" is called a component. [Chigani] In peer-to-peer systems, the nodes are often built based on related functions and responsibilities. Therefore, components in distributed software systems include nodes of the system in addition to the other software and hardware components.

The tester who is performing conformance testing faces some common challenges. Variations in the operating environment, unavailability of source code and the capabilities of the monitoring mechanism limit the information available to the tester. The difficulty faced in testing distributed systems in different operating environments is partially caused by the variation in the behavior of the nodes within the system at design, simulation, and deployment. Most peer-to-peer systems are designed to be executed on systems with diverse hardware configurations, distant geographic locations, and varied levels of network connectivity. These factors are not easily accounted for in simulation where the system is run in a contained and observable environment. The lack of access to source code is a challenge to testing distributed systems since components of a distributed system may be commercial-off-the-shelf components. Any method that depends on the source code for information may make the method not applicable to

systems that contain such components. [Ghosh] Some distributed systems, are designed to run on nodes for which testers do not have significant access. If this access is required for gathering data to test the distributed system, the capabilities of the monitoring mechanism must exist outside of those nodes. Such limited access can be a challenge to designing test cases since the amount of information may be large and possibly create bottlenecks at the monitoring mechanism. [Dischinger]

The research described in this thesis investigates current methods for conformance testing peer-to-peer content distribution systems, the challenges that need to be addressed for the quality conformance testing of those systems, and a proposed method for conformance testing peer-to-peer content distribution systems. Current methods for testing distributed systems will be examined based on how each method addresses the challenges of testing Peer-to-Peer systems, and these methods will then be compared to the proposed method. The two research questions of this work are:

1) Can the behavior of components within a distributed system be described according to both the design of the system and the observed message traffic in a way that the conformance testing process will be enhanced?
2) Can the significance of inconsistencies between the designed and observed behavior be measured in a way that is meaningful for the testing process?

This work details the "Method for Conformance Testing by Analyzing Message Activity" (MCTAMA). MCTAMA is used for conformance testing of P2P content distribution systems. It uses a common representation for describing: (1) the behavior of nodes within a distributed

system as they are designed to behave, and (2) the observed behavior during simulation or deployment.

In addition to defining a new method for conformance testing, a key feature of MCTAMA is the creation of the common representation for describing the behavior of nodes at any step of design or deployment. By using a common representation for describing node behavior and a technique for generating test cases, MCTAMA creates test cases that identify variation between expected outcomes and observed behaviors. Another feature of MCTAMA is that the test cases it generates are not as vulnerable to the common challenges of testing distributed systems as other methods. The method focuses on the messages sent through the system, making MCTAMA applicable at multiple stages of development, since messages are readily obtainable during all stages of development. MCTAMA relies on relationships between different message types to provide information for the determination of the role nodes have within the system. MCTAMA is less affected than other methods by the variations in the operating environment, availability of source code or the capabilities of the monitoring mechanism because it does not rely on the sequence/patterns of messages or the interactions between specific pairs of nodes.

This research includes also presents a method for combining sequence diagrams that creates a single diagram for a distributed system while maintaining consistent identifiers for designed nodes. The resulting diagram is used to create a description of the expected behavior of nodes in the system across multiple diagrams. This research also provides a tool for sorting and filtering the results of testing where a large number of test cases are used. The generation of test cases for MCTAMA produces a large number of test cases that may include relationships that have no significance for the conformance testing of the system. The ability to filter out such relationships

as well as to identify results with the most significant variation is an important feature of MCTAMA.

This dissertation is organized as follows. Chapter 2 presents background information on software testing, conformance testing, and peer-to-peer software systems. Chapter 3 describes the literature review for testing distributed systems, the work related to individual steps of MCTAMA, the development of the *Taxonomy for Message Classification*, and the work related to performance analysis. Chapter 4 introduces MCTAMA, a method for combining sequence diagrams, and a tool for evaluating test results. Chapter 5 presents two experiments that serve as an example of system testing using MCTAMA. Finally, chapter 6 summarizes this research and suggests possible future work.

CHAPTER 2 BACKGROUND

2.1 Software Testing

In 2002, software sales within the United States reached approximately $180 billion. Yet according to a report by the National Institute of Standards and Technology, the costs incurred by faults in software reached an estimated $59.5 billion. The benefits of software to industry far outweigh this cost, but the cost is still tremendous considering that, according to the same report, over $22 billion of those costs could have been eliminated by identifying these faults through better testing during the development stages. Currently, more than half of all faults are found after the software is in use by the customer. [NIST]

Software testing is the determination of the quality of a software system. Software is an integral part of modern business. From the New York Stock Exchange to the barns of corn farmers, software has become an increasingly widespread benefit to commerce. As software has become more ubiquitous, the cost of damages associated with software failures has increased drastically. There has been no shortage of examples of major software system failures. On August 14, 2003, the northeast region of the United States experienced widespread blackouts due mainly to a software failure that triggered the failure of multiple electrical systems. While backup systems were in place, they suffered from the same failure as the primary systems. It took weeks of investigation to determine that a programming error was a major contributor to the scope of the blackout. [Jesdanun] It was later found that the economic damage related to the single software failure was an estimated $10 billion. [ICF] Another failure occurred in 2004 when the radio communication systems that allow air traffic controllers to communicate with planes suddenly shut down. The backup systems quickly failed as well. A unique aspect of this

failure is that the “bug” in the software was known. A counter used by the software would reset after 50 days of operation. The effect this reset would have on the system was unknown. The fault was addressed outside of software by having an operator reset the entire system. When the operator failed to reset the counter, the system failed. Luckily, the damage caused by the failure was not measured in human lives. This failure could have easily led to mid-air collisions. [Geppert] Even in 2009, a large number of Microsoft Zune media players became inoperable due to a bug in the internal clock that closely resembled the well-known Y2K bug.

Considering the damage that can be caused by software failure, users and customers need to know the risk that such a failure could occur. Software testing provides a quantitative measure of that risk. The purpose of testing is to find faults within the software by observing software failures, errors in code, or discrepancies between the software and requirements. The techniques used for testing a software system vary widely. One characteristic of a technique is when the technique is used during the development process. Traditionally, testing occurs most frequently after a software system has been developed. Faults that are found during such testing are either fixed and the software system retested or simply left in the software. Any fault left can cause the quality of the system to decrease and the risk of software failures to increase.

Failure and *Fault* are two important terms related to testing. A software *failure* occurs when software fails to behave as expected. The expected behavior can be taken from a requirement specification, which states what a system is required to do, or design document, which states how the system should perform the requirement. The errors that create the failures are called faults. *Faults*, also called “bugs” or “defects”, are errors in the specification, design or

software that causes software to behave unexpectedly. [Pfleeger] Not all faults lead to failures, but they do show inconsistencies, such as inconsistencies between the design and the software.

The difficulty of providing an accurate measure of software quality has increased as the complexity of software has increased. Where software used to be measured by the thousand lines of code, it is now common to see software systems measured according to million lines of code. In addition to the increased complexity, the amount of time that a software system remains in the market has drastically decreased. [NIST] The pressures to develop software quickly are often at odds with the demands of customers for quality software systems. Testing methods must now be able to determine the quality of extremely complex software while being practical for use in the current business environment. [Culbertson]

A defining characteristic of a software testing method is whether it uses a static or dynamic approach. The static testing of a software system involves analyzing the source code or design documents. It does not involve the running of the software. Dynamic testing is the act of operating a software system with the purpose of finding software faults. [Culbertson] Most types of dynamic testing relies on the creation and execution of test cases. As described in Chapter 1, the test case is a document that defines scenarios for testing which include a sequence of actions to be performed on the systems, a set of conditions/inputs, and the expected outcome of the actions. If the expected outcome is different than the observed outcome, then the test fails. Test failures indicate that there is a fault in the system.

Another important characteristic of software testing is white box and black box testing. Black box testing views software as a closed entity. The testing process involves the manipulation of inputs into the system and observing the outputs. White box testing involves any

type of testing which can exploit the structure or internal components of the software to test the system. White box testing offers powerful tools for determining the quality of a software system. Black box testing is generally not as powerful, but is not restricted by the internal structure of the software or affected by the unavailability of any components.

A key part of a test case is the availability of a mechanism for determining whether the software worked as expected. This mechanism is generally called a *test oracle*. Quality test cases have a measurable aspect of the outcome based on a requirement or specification.

Test case documents can vary widely. Some test cases are informal, written specifications that resemble a theatrical script with an identifier, a series of actions, and the expected result. Figure 2.1 shows such an informal test case:

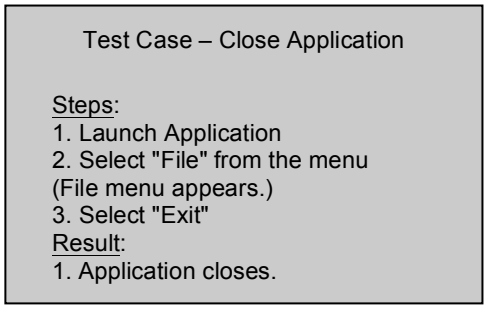

Figure 2.1 - Informal Test Case

Test cases can also be detailed and technical. For instance, the expected outcome could be a series of messages sent in a specific order as specified in a sequence diagram. Figure 2.2 shows one such test case written in TTCN-3, a testing language that bases tests on sequence diagrams. [Ebner]

```
module {
function testExecution1 () {
  verdicttype verdict ;
  integer i ;
  for ( i :=10; i>0; i:=i−1)
    execute ConnectionFailure ();
  verdict = execute
                ConnectionSuccess();
  if ( verdict == false )
      return ;
  execute TransmissionSuccess ();
  execute ConnectionRelease();
}
function testExecution2 () {
  verdicttype verdict ;
  Integer i ;
  for ( i :=10; i>0; i:=i−1)
    execute ConnectionFailure ();
  verdict = execute
              ConnectionSuccess();
  if ( verdict == false )
    return ;
  execute TransmissionFailure ();
}
control {
  testExecution1 ();
  testExecution2 ();
}
}
```

Figure 2.2 - Example of a test case written in TTCN-3 [Ebner]

2.2 Conformance Testing

Conformance testing is the process of verifying that a software system meets the specifications of the design or standard that guided the development of that system. Conformance testing is often associated with communication protocols, but it also applies to behavior design documents.

The Unified Modeling Language (UML) is a standardized language for designing software. It is made up of 13 diagrams which are divided into three categories: structural, interaction, and behavioral. [Booch] Conformance testing is often based on the interaction and behavioral design documents from UML. Structural design documents such as the class, component and deployment diagrams usually have a direct correlation between those documents

and the source code, so testing the conformance of an implementation to these diagrams is usually straightforward. Testing the conformance of a system to these UML models can be done effectively with a minimalist approach [Kaplan], but behavioral diagrams, specifically the sequence diagram, are considered an important document in distributed system design. [Ambler] In addition to having less correlation to source code, the behavioral and interaction diagrams present the challenge of requiring dynamic information to test the design since they describe how a system should perform during execution. Specific approaches to conformance testing behavioral diagrams will be discussed in Chapter 3.

2.3 Peer-to-Peer Software Systems

Peer-to-Peer software systems are becoming an increasingly useful and ubiquitous type of distributed software system. One of the most widespread uses of peer-to-peer model is the distribution of content across networks to eliminate bandwidth constraints that would be an issue in a traditional server-client model. Such systems have unique, yet related, functions for peer registry, content query, and content sharing.

Architectures for peer-to-peer systems have one defining characteristic: centralization. Centralized systems maintain some level of knowledge or control over the network through a server or series of servers. Centralized networks have an issue of scalability as the network grows beyond the computational or network capacity of the centralized servers. Decentralized systems which push activity to the edges of the network in order to better utilize available bandwidth are efficient, but can have stability and control issues as the system has no centralized control of nodes.

Peer-to-Peer software systems are volatile. Critical parts of the system are often spread across multiple computer systems, geographic locations and networks. Node failure is often common and can be difficult to replicate in a simulation environment. Testing peer-to-peer systems must account for the volatility either through the simulation of the different systems, locations, and networks or by testing the software system during early deployment. The latter approach is often preferable due to the difficulty of simulating the deployed environment. For instance, at least one major Internet Service Provider was discovered to be throttling peer-to-peer traffic for the purpose of disrupting the stability of a peer-to-peer network to alleviate the bandwidth usage within their network. [Dischinger] Such activity is difficult to account for during simulation since the method of throttling used by service providers has not been published.

The lack of access to source code is another significant challenge. Components of a peer-to-peer system may be so called commercial-off-the-shelf (COTS) systems for which the source code is usually inaccessible. [Ghosh] Any technique with a dependence on source code to test conformance to design would be inapplicable to systems that contain such components.

Some peer-to-peer systems are designed to run on nodes for which testers do not have any significant access. The architectures often push activity to the edge of the network to decentralize the resource requirements. The edge nodes of peer-to-peer systems are often users who join the network to consume the content as compared to nodes that join the network with the intent to distribute content. Detailed information about the behavior of specific remote consumer nodes in the system can be difficult to obtain. In cases where the nodes themselves are inaccessible, a monitoring mechanism between connected nodes must account for this lack of

access, but the amount of information may be large and possibly create bottlenecks at the monitor. [Dischinger]

2.4 Summary

Peer-to-peer systems present a unique set of challenges to testing. Variation in architectures, the availability of source code, the volatility of the network, and the accessibility to nodes all prevent many testing techniques inapplicable or impractical for testing peer-to-peer networks. In Chapter 3, we discuss existing techniques for testing distributed systems and the conformance testing of systems using design documentation.

CHAPTER 3 RELATED WORK

In the previous chapters, we have introduced the contributions of this work and given an overview of the areas of software testing, conformance testing, and peer-to-peer software systems. In this chapter, we will review related research in specific approaches to testing distributed systems, work that is related to each step of MCTAMA, and work that influenced the development of MCTAMA.

3.1 Testing Distributed Systems

The focus of MCTAMA is the conformance testing of peer-to-peer systems. Techniques for testing software run on a single computer are not necessarily inapplicable to peer-to-peer systems. However, methods specifically designed to test peer-to-peer systems often specifically address the unique challenges of ensuring the quality of the system. Some methods were designed to test software being executed on a single software system, but then extended to be applicable to software executed across multiple systems. [Long] Peer-to-peer systems are a specific type of distributed system in which the nodes of the system share a common task or workload. We begin with a review of related techniques for testing general-purpose distributed systems.

My research is focused on the conformance testing of peer-to-peer file distribution systems. As defined in Chapter 2, the term conformance testing means the verification that a software system meets the design or specification the guided the development of the system. What distinguishes conformance testing from the more general term, system testing, is that the verification is to a design or specification rather than a *requirement*. Since requirements can be

generated from design, MCTAMA is related to existing work in system testing in addition to conformance testing.

Conformance testing is important for distributed system testing since testing the conformance of the system to a protocol specification is a common need. Kang defines two different types of conformance testing in distributed systems. The first type is *Interface Conformance Testing* that includes the traditional testing of protocol conformance. Interface Conformance Testing is a test that verifies that two entities within the distributed system communicate as specified. This type of test is easier to develop and is useful for verifying protocol conformance. The second is *Entity Conformance Testing* that tests all of the interfaces to a single entity within the system. While more difficult to develop, entity conformance testing offers higher test coverage and is less susceptible to factors outside the tester's control. [Kang] MCTAMA is classified as an entity conformance testing approach since it analyzes all the connections at a single peer.

3.1.1 Issues and Approaches to Testing Distributed Systems

The two primary issues in distributed system testing are controllability and observability. These two issues are well defined and specifically addressed in previous work. Controllability "refers to the ease of affecting the specified outputs". [Ural] Controllability is an issue since peer-to-peer systems are designed to be executed on systems with diverse hardware configurations, distant geographic locations, and varied levels of Internet connectivity. While easily solved by running the system within a simulation, simulations do not adequately recreate these challenges. Observability "refers to the ease of determining if specified inputs affect the outputs". [Ural] In a peer-to-peer system, it is much more difficult to determine the output for a

given input. Specifically, it is impossible for the system itself to know when to start or stop waiting for the output. It is also difficult for the tester to determine the state of the entire system. [Cacciari]

The most common approach to overcoming the issues of controllability and observability is to create a test-specific point of control and observation. However, these problems have also been addressed with methods for test construction and sequencing that alter the specification being tested. [Ural], [Khoumsi],[Bowman] In the method developed by Ural and Whittier, a system is tested by generating directed graphs from finite state machines given in the specification and inserting any needed coordination messages to complete the test. [Ural] A different approach produced by Khoumsi is to add timing constraints to methods within the distributed system even if the system is not real-time. This overcomes the issues of observability and controllability by setting limits on the amount of time a system tester would have to wait. [Khoumsi] Architectural frameworks have also been proposed to address these issues by having an architectural framework that integrates multiple points of control and observation to determine if the target system conforms to a specification defined for each point. [Bowman] [Almeida]

The controllability and observability problems are considered fundamental issues in testing distributed systems. Ghosh describes nine practical issues in testing distributed systems. They are:

1) ”Scalability of test adequacy criteria
2) Redundant testing during integration of components
3) Availability of source code

4) Heterogeneity of language, platforms and architectures
5) Monitoring and control mechanism in distributed software testing
6) Reproducibility of events
7) Deadlocks and race conditions
8) Testing for system scalability and performance
9) Testing for fault tolerance" [Ghosh]

Controllability and observability are addressed in the practical issues as "monitoring and control mechanism" as discussed previously. There are multiple approaches to addressing the practical issues of testing distributed systems. Some common approaches to simplifying the testing process are to narrow the focus of the test, to limit the inputs to the testing method, or to contain the test within a simulation of a network.

Narrowing the focus of the testing process addresses the challenges of testing distributed systems by focusing on a small window of time. The most common type of approach in this area is the use of behavioral design documents to determine the testable scenarios within a system. Methods by Pickin and Jéron generate tests based on UML diagrams. Picken developed a method for the generation of test scenarios based on UML sequence diagrams. [Picken] The method reduces the scope of the test to a small amount of observable time. Jéron establishes a framework that includes structural and behavioral diagrams that performs model checking throughout the software life cycle that includes simulation and execution behavioral testing. [Jéron] One approach to focusing the testing process takes what the author describes as a "minimalist approach". Kaplan uses the UML use case as a behavioral specification and creating a more comprehensive test by also including the structural specification and generating

verification sequences to determine if the structural and behavioral flow conform to specification. [Kaplan]

Limiting the inputs to a testing process is an effective approach to addressing the challenges of testing distributed systems by focusing on a smaller subset of the network. By limiting the inputs, the testing process is less susceptible to the practical issues such as scalability, heterogeneity of components, and deadlock/race conditions. One such method is the finite state machine-based testing developed by El-Fakih, that tests the interface conformance at a single node. [El-Fakih] Interface conformance testing has a reduced and simplified input. The combined finite state machines used to represent the single interface between two entities essentially determines the protocol specification. By focusing on a single connection, the issues of testing a complex, geographically dispersed systems are eliminated.

Another approach is to address the issues through the development of more robust simulations. Simulations allow the tester to have complete control and observation of the entire system. However, in two separate surveys [Baker], [Naiken], the existing peer-to-peer network simulators failed to adequately facilitate quality testing. The study by Baker and Lakhoo concluded that the usefulness of network simulators is not clear. The effort required to utilize the simulation made this type of approach impractical. [Baker] Whether simulations can truly test the scalability of a distributed system is unclear, but definitely limited by the processing capacity of the computer running the system. A recent approach to creating a peer-to-peer system simulation at the packet level produced a network of 8,192 simulated nodes across 16 physical machines. [He] However, it's been observed that a Gnutella network contains at least 400,000 observable nodes. [Ripeanu] At 2.048% of the observed network size, the simulation may be too

simplified to produce quality test results. However, He states that models of 100,000 or more nodes (25% of observed network size) are possible. [He]

Another approach to addressing the challenges of testing distributed systems is to create a middleware. This approach is related to simulation by the goal of the approach to contain the system into an observable space. The problem with middleware is that the middleware generally operates by brokering functions or communication within the system. This can create a bottleneck preventing communication in large-scale distributed systems. In an approach to test three-tier distributed systems, Yamany creates a middleware between the clients and a server. This traditional idea of a middleware isn't applicable to peer-to-peer systems where the clients do not communicate solely with the server, and are, in practice, much more likely to communicate with peers in the network. So the approach augments the middleware with different types of intelligent agents that observe the system at different locations and under different conditions. This approach increases the observation while limiting the affect of the observation on the network. [Yamany]

3.1.2 Testing Distributed Systems Using UML Behavioral Diagrams

The Unified Modeling Language (UML) is used to specify a software system by modeling the structure, processes, and behavior of the system. My research focuses primarily on the behavioral diagram, the sequence diagram. A sequence diagram "is used to show the interactions between objects in the sequential order that those interactions occur". [Bell] Figure 3.1 shows a sample UML sequence diagram. The diagram contains two objects labeled Object 1 and Object 2. MCTAMA focuses on message passing, so the two interactions represent each object sending and receiving one message.

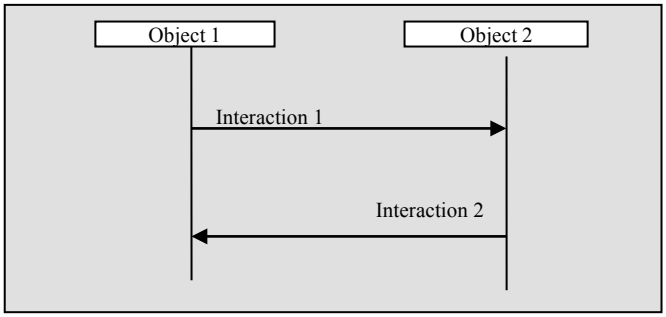

Figure 3.1 - Sample UML Sequence Diagram

The purpose of the sequence diagram is to describe how the system being designed is intended to behave. When compared to behavioral diagrams such as the use case, sequence diagrams can be used as a requirement of the system. [Bell] The testing process starts with requirements, so the sequence diagram is a logical foundation for testing software systems. The benefits of using the sequence diagram as a test case include the intuitive means of creating test specifications and the easy integration into the development process. The key limitation is that the sequence diagram does not model all parts of the software system, such a graphical user interfaces or code structure. [Fraikin]

System testing based on UML specifications has been established since the creation of UML. In a UML-based approach to system testing created by Briand and Labiche, a significant feature of UML is employed by integrating the models/specification throughout the software life cycle. [Briand] SeDiTeC is an approach to testing systems based on sequence diagrams that takes the approach that sequence diagrams can be altered to make them testable. [Fraikin] This alteration is essentially creating a test case from a sequence diagram.

UML behavioral diagrams are commonly used for specifying network protocols. The two diagrams that most frequently specify protocols are the UML sequence diagram and the UML state diagram. [Kim94] Two non-standard, behavioral diagrams are also common. The Message Sequence Chart (MSC) is a diagram for showing message communication in distributed systems. It is similar to sequence diagrams, but with the limitation that the objects must represent nodes or components in the system, and interactions must represent messages passed between the objects. [Ebner] The Finite State Machine is a behavioral diagram that is very similar to the UML state machine, and the differences between the two diagrams tend to be purely semantic.

3.1.3 The Generation of Test Cases for Testing Distributed Systems

Many approaches to testing software systems utilize design specifications (such as the diagrams provided by the UML) for test specification. These behavioral diagrams are useful for specifying a testable requirement because they already account for the inherent concurrency and asynchronous communication, and, by describing the system at a high level, the specification can be interpreted as a verifiable behavioral test pattern. [Picken]

The process of creating test cases and test data are labor-intensive. Automatic test generation creates test cases and test data directly from the design or specification. [Khurshid] Automating the creation of test cases offers many advantages to the testing process. The primary advantage is that the testing process is a by-product of design. By encouraging quality design, the testing process is simplified as well. [Boy] The automatic generation of test cases also alleviates the issue of updating test cases during software evolution where the specification or design is altered during the process of development. If the alteration is reflected in the design, it will be accounted for in the test specification. A practical advantage is that there is an overall

productivity gain by automating the creation of test cases as well as automating the completion of the test objective. [Pickin]

Sequence diagrams are not designed to be used as test specifications, so in order to use them in that way, some augmentation or alteration is usually required. SeDiTeC is an approach to testing traditional software systems by making sequence diagrams testable. [Fraikin] This type of approach has been extended to distributed systems by a number of methods.

Multiple methods for the automatic generation of test cases for distributed systems use Finite State Machines. In most cases, these approaches have a narrow focus and are not meant to verify overall system behavior. Pickin uses multiple diagrams to create scenarios for testing distributed systems. [Pickin] The use of finite state machines generally introduces a challenge to the testing process. It is difficult to determine the global system state is large systems. Pickin addresses this by using methods to simplify the global state to ease the application of the finite state machine and determine appropriate application of scenarios. In a derivative work, Jéron introduces a middleware to check models during the design process and create focused simulations. [Jéron] The method is not applicable as presented to verifying the system during execution. AGATHA is a tool for the formal verification of distributed systems using UML state charts. The diagrams are translated into a specification that AGATHA uses to create a series of sequence diagrams representing system behavior. The tool is able to detect some issues with deadlock and race conditions [Lugato], but remains a tool for verifying the system through simulation where the global state does not have to be determined. Kim and Song developed an approach focused on conformance testing the protocol of distributed systems that includes fours methods for generating test cases. Three of the methods are focused on determining the system

state. They show that the method which produces the lowest complexity test cases also provides the least amount of confidence that the system does not contain faults in the protocol implementation. Conversely, the test cases that provide the most confidence that the system does not contain faults require additional effort and complexity determining the system state to accurately test the transitions defined in the protocol. [Kim94] The work of Nguyen uses random test generation and execution to provide a compromise that offered better test confidence without requiring a complex testing process. [Nguyen] Extended Finite State Machines which provide a better coverage without additional effort to determine the system state have not been extended to distributed system testing [Kim99]

Many successful techniques for testing distributed systems exploit the nature of distributed systems to provide a focused test case. Garstecki developed a method for testing distributed languages by utilizing the uniqueness of programming languages that are created for parallel and distributed systems. [Garstecki] Henniger creates message sequence charts with a labeled event structure to describe the behavior of distributed systems. This structure shows the effect of all possible communications and prevents complex test specification by reducing possible system states. [Hinniger] The Tree and Tabular Combined Notation (version 3) (TTCN-3) also uses the information in message sequence charts to create scenario based testing that utilizes the communication methods of distributed systems to create points of observation and ordering. [Ebner]

3.2 Work Related to Steps of MCTAMA

MCTAMA contains two steps that are uniquely used together in a distributed systems testing technique. Combining sequence diagrams is an important step in the approach because it

creates a complete picture of an object's designed behavior. Classifying the messages sent through the system is important to making MCTAMA applicable to systems with system-specific message types. Together, these steps represent the transformation of multiple sequence diagrams into a single testable representation of node behavior.

Since the purpose of sequence diagrams is to describe the interactions between objects in a given scenario, the scope of each diagram is limited to one algorithm, behavior or function of the system. The scenario does not account for any interactions that take place outside of that single behavior, making testing each scenario difficult as the functions of the system are running in parallel and are frequently interrupted by network connection failures. In cases where behavioral diagrams have consistent objects, the process can be straightforward. [Köhler] Message sequence charts can be simplified by abstracting the details of transactions. Riva and Rodriguez demonstrate two types of abstraction. *Vertical Abstraction* combines similar interactions into a single representation. *Horizontal Abstraction* combines objects showing interactions between multiple objects. [Riva] Horizontal abstraction is an important part of MCTAMA's combination of sequence diagrams. For sequence diagrams to be utilized by MCTAMA, the objects must represent nodes in the system. In the system specification, multiple objects can represent components within a single node, so those objects must be combined. The technique for combining objects and horizontal abstraction are similar. An example of horizontal abstraction given by Riva is shown in Figure 3.2. In this example, Object 1 and Object 2 are combined into a single object. Interactions in between Objects 1 and 2 are no longer diagrammed. Interactions between Object 1 or Object 2 with Object 3 are shown as interactions with the combined object.

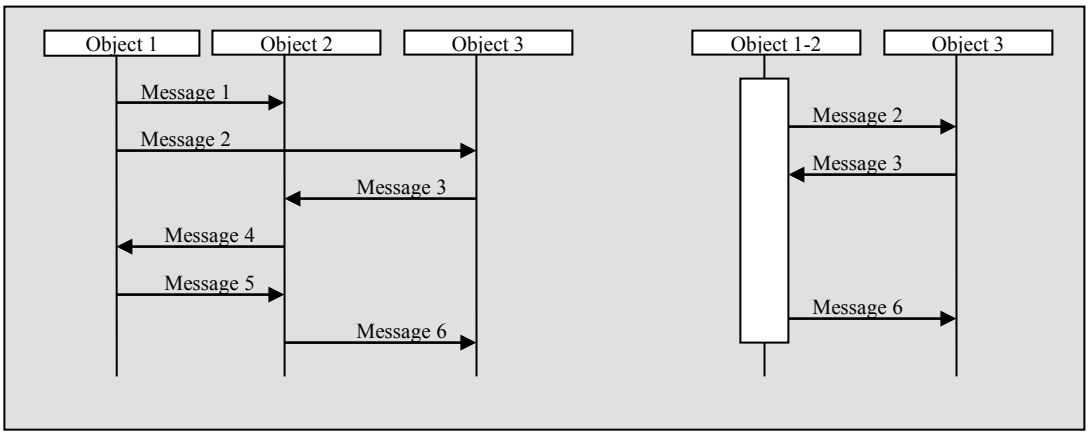

Figure 3.2 - Example of Horizontal Abstraction [Riva]

Choosing which objects to combine is not always described within the design. While some systems are designed to cluster similar peers [Ng], Fox introduces the idea of roles by relating organization theory to distributed systems. The most common architectures often resemble the organizational charts of companies. [Fox] This similarity between node types determines which sequence diagrams are combined to represent the behavior of the nodes within the system. In addition to creating roles, DeLoach uses sequence diagrams to define the communication that defines each role within the distributed system. [DeLoach]

The similarity of messages and the usefulness of this similarity are explored in previous work. In an SIP monitoring system, Acharya shows that focusing on detailed message traffic hides the important information about the system. By categorizing the messages being monitored, the monitoring system is more efficient and useful. [Acharya] Middleware that facilitates interoperability between peer-to-peer file distribution systems translate the protocols, by exploiting this similarity. [Cugola] [Pallikara] The middleware introduced by Lui and Kwok categorizes messages used by three popular peer-to-peer file distribution systems. Categories included “peer registry & discovery”, “query”, “data download protocol”, and “data transfer

mechanism for firewalled peers", but must be extended to more categories if more systems are included. [Lui]

3.3 Work That Influenced the Development of *The Taxonomy for Message Classification*

The classification of messages is a necessary feature of MCTAMA. By including a method for classifying messages, the approach to testing distributed systems is easier to apply to systems with system-specific message types. The definition of the classification used in this research was influenced by a classification taxonomy for software architecture recovery by Pollet. The classification uses intuitive axes and labeling to make the classification of methods simple while creating specific classifications. [Pollet] The similarity of peer-to-peer systems and their respective architectures has been studied in previous work. [Tsoumakos] [Kant] [Aleksy] Tsoumakos and Roussopoulos reviewed the search methods used by the most popular peer-to-peer systems and found that key behaviors of the these systems can be quite similar. [Tsoumakos] Kant developed a taxonomy for classifying peer-to-peer system according to five dimensions: resource location, control location, resource usage, global state control, and QoS constraints. This type of classification shows peer-to-peer file distribution systems that are similar according to any of those five characteristics. Kant included the classification of several widespread peer-to-peer software systems. [Kant] The design patterns of peer-to-peer systems have also been applied to some popular peer-to-peer systems to also show the similarity between them based on source code-level design patterns such as Role-Based Access Control, Key Management, Abstract Factory, etc. [Alesky]

The published similarities between peer-to-peer file distribution systems are key in developing a classification of a classification for message types. The methods found to be similar

are likely to have similar message classifications for messages used by those methods. MCTAMA uses a classification scheme called the *Taxonomy for Message Classification* that was developed from these findings. In order to develop a taxonomy, the protocol specifications of the most popular peer-to-peer file distribution systems were reviewed and the messages in the systems that were similar in the published work were classified. If there was no appropriate classification, a classification was added. By building the taxonomy from dissimilar methods and systems, the taxonomy can classify dissimilar message types.

3.4 Related Work in Performance Analysis

In the early stages of development, MCTAMA was envisioned as an approach to architecture design recovery using techniques developed for the performance analysis of peer-to-peer systems. Although MCTAMA became more focused on testing systems with known design, the foundations in performance analysis are still relevant. The idea came from observing that when modeled, the combined analyses indicated patterns in the overall behavior of the nodes within the system. In practice, the analysis of two nodes that have similar roles in the network would be expected to have similar results.

In an analysis by Gummadi, Dunn, and Saroiu, several behavioral characteristics of peers within a peer-to-peer content distribution system were observed that contained similar information to the information in an MCTAMA test case. One key analysis was of the ratio of file requests to file transfers for objects of various size. The result of the analysis was that the number of requests sent for small files was disproportionately high to the number of requests sent for large files. [Gummadi] Since the analysis was not a true test of the system, it was not determined if this behavior differed from the designed behavior of the system. The primary

inspiration of MCTAMA was to use such an analysis to test the system against its design. Other characteristics analyzed in this study that are similar to the characteristics analyzed by MCTAMA are the number of requests, the number of transactions, the average transaction size, the number of unique objects, and bytes transferred. The related feature of MCTAMA's analysis focuses of the ratio specific message types.

The efficiency and performance of peer-to-peer file distribution systems is critical to the usefulness of such systems. The challenges of peer-to-peer software systems must not affect a system to the point that it is no longer as effective as traditional system design.

Several methods exist for the performance analysis of peer-to-peer file distribution systems. The method published by Alouf, Dandoush, and Nain uses Markov chains to create an analytical model of the system to evaluate the performance of systems recovering lost data. A key feature of the work is that by using the framework developed, the system can be engineered to fulfill the requirements of the system. By recognizing these types of requirements, the method is described as a performance-testing framework. [Alouf]

Since most peer-to-peer file distributions share a set of similar methods for operations such as file transfer, network organization, file querying, and file storage techniques for analyzing each individual type of method have been developed. The work of Lin and Wan provides a method for the analysis of search (file query) performance. The method centers on the definition and testing of three characteristics of the system:

1) Query Efficiency, "the ratio of Query Hits to Messages per Node".
2) Search Responsiveness, "Success Rate / Hops Number".

3) Search Efficiency, “Query Efficiency × Search Responsiveness”. [Lin]

These three characteristics provide an adequate analysis of different search methods. It is however, limited to algorithms for file querying. The work of Biersack, Rodriguez, and Felber creates a comparative, deterministic analysis of various peer-to-peer architectures. The primary comparison is done by comparing the time needed to serve a file as a function of the number of chunks transferred to the number of peers transferred to. By taking a simple approach to comparing peer-to-peer architecture, it is applicable to multiple operations used by peer-to-peer systems. For instance, the same approach is used to test the network organization operations by measuring the time and number of messages it takes to organize from a node departure. [Biersack]

A more comprehensive approach to the performance analysis of peer-to-peer networks was done by Qui and Srikant, but adds the limitation that it is only applicable to highly-decentralized systems. The analysis focused on four characteristics of the system that do not focus on any one operation, but rather a combination of operations. Peer evolution compares the arrival and departure of peers to the bandwidth at each peer. Scalability describes the effects of various network sizes. File sharing efficiency describes the ability of a peer to find the files it needs and to fully utilize the bandwidth of peer that have that file available. The final characteristic is the incentive to prevent free-riding, or peers that do not participate in the distribution of files on the network. The contribution of this work is that it creates a simpler model of behaviors so different types of peer-to-peer systems can be analyzed. The model is a numerical model that uses factors such as the number of downloaders, number of seeds, arrival rate of new requests, the upload bandwidth, the download bandwidth, the rate of downloaders

abandoning a download, and the rate at which seeds leave the system. The resulting analysis is a description of the scalability, stability and efficiency of a peer within the system. [Qiu] These descriptions are indications of the overall performance and behavior of individual nodes.

Performance analysis is called performance testing if the analysis is to determine if a peer-to-peer file distribution system meets a performance requirement. Performance testing is related to MCTAMA since it often focuses on the messages sent through the system as well as the tendency of performance testing to focus on how an individual node behaves in the system. MCTAMA is not focused on how well a system performs, but rather if nodes in the system behave as designed. However, many approaches to performance analysis are related to MCTAMA by their focus on captured messages and behavior during execution.

3.5 Comparison of Features to Related Work

Figure 3.3 shows a comparison of MCTAMA to all of the methods referenced in the literature review. The comparison is made based on the following features:

1) Generates test cases.
2) Tests the system according to UML diagrams.
3) Purpose - conformance testing of the system.
4) Applicable to distributed systems.
5) Utilizes sequence diagrams.
6) Considers all system behavior. (Combines sequence diagrams.)
7) Applicable to all levels of design.

	{1}	{2}	{3}	{4}	{5}	{6}	{7}
[Picken]	x	x	x	x			
[Ural]	x	x	x	x			x
[Khoumsi]		x	x	x	x		
[Almeida]		x		x	x	x	x
[Yamany]			x	x			
[Long]	x		x	x			x
[Jeron]	x	x		x	x		x
[Gunnadi]				x		x	x
[Qiu]				x		x	x
[Biersack]				x		x	x
[Lin]				x		x	x
[Alouf]			x	x		x	x
[Kim94]	x	x	x	x		x	x
[Lugato]	x	x	x		x	x	x
[Kim99]	x	x	x	x		x	x
[Nguyen]	x			x			
[Garstecki]	x					x	
[Henniger]	x	x	x	x	x	x	
[Ebner]	x	x	x	x	x	x	
[El-Fakih]		x	x	x			
[Briand]		x	x		x	x	x
[Fraikin]			x		x		
[Kaplan]	x	x	x		x		x
[Lui]				x			
[Acharya]			x	x			
[Fox]				x			
MCTAMA	x	x	x	x	x	x	x
[Kohler]*					x	x	
[DeLoach]*					x	x	X
[Riva]*					x	x	
[He]**				x			
[Baker]**				x			
[Naiken]**				x			

* - Sequence diagram combination method ** - Peer-to-Peer Simulation

Figure 3.3 - Comparison of MCTAMA to Related Work

The literature review was not limited to the testing of distributed systems, so some methods are only related to MCTAMA by a single feature. For example, the combination of sequence diagrams was not found in any existing approaches to testing peer-to-peer systems.

As the information in Figure 3.3 indicates, no other single method that we found after the extensive literature search provides all seven of these capabilities.

CHAPTER 4 METHOD FOR CONFORMANCE TESTING

This research defines a new a method to create and execute test cases for peer-to-peer file distribution systems. This method, named MCTAMA, gathers dynamic information about a system by capturing and analyzing the system's message traffic to determine the overall behavior, or "role", of nodes within the system. By focusing on messages, the method can be applied at all stages of design, development and deployment. The test cases generated by MCTAMA are sets of roles that together describe the behavior of individual, identifiable nodes as they are designed to behave within the system. The test is executed by observing the message traffic at specific nodes within the system, then comparing the observed role to the role given for that node in the test case.

MCTAMA relies on messages as the primary source of information rather than relying on the sequence of messages or the interactions between specific pairs of nodes. Thus, the method is not limited by variations in the operating environment, availability of source code or the capabilities of the monitoring mechanism. Given that messages are represented in design and are generated during testing, the same type of information is used to generate the test and to determine the test result. Therefore, a common representation can be used to describe them both.

Figure 4.1 provides a high-level overview of the steps of MCTAMA. The generation and execution of test cases are shown as two similar three-step processes. Both steps use messages as input and produce a set of roles. The final step (Step 3 of EXECUTING THE TEST CASES) gives a quantitative measure of the variation between the designed and observed behavior.

GENERATING THE TEST CASES

Step 1: Classify messages using *Taxonomy for Message Classification*

Step 2: Alter and combine sequence diagrams

Step 3: Describe the role of nodes in the system by analyzing messages used in design

EXECUTING THE TEST CASES

Step 1: Capture messages sent within the system

Step 2: Describe the role of nodes in the system by analyzing observed messages

Step 3: Analyze differences between assigned roles according to design and message capture

Figure 4.1 - Overview of testing process

4.1 Generating the Test Cases

The first step in MCTAMA is the classification of messages within the system. Each message type used in the design is classified according to the *Taxonomy for Message Classification* (shown in Figure 4.2). The design documents are augmented with these message classifications. The second step is the combination of the sequence diagrams used in the design of the system. The sequence diagrams are either taken from the design documentation or created using information from other types of design documents. The combined documents give a comprehensive description of the system's designed message traffic. The third step is the creation of a description of the roles each identifiable node has within the system. The roles resulting from this process form a test case for determining if a node behaves as designed in a peer-to-peer file distribution system.

4.1.1 Step 1: Classify Messages Using *Taxonomy for Message Classification*

Test case generation in MCTAMA begins with the classification of messages according to their purpose. The classification is used to describe the messages in intelligible, expressive terms that aren't specific to the system being tested. The goal of the classification is to simplify the subsequent steps of test generation by eliminating the variation in the system-specific description of message types. The expressive terminology used in the classification helps in the interpretation of test results and understanding the relationships between message types to determine if the relationships, and the test results related to the relationship, are significant.

In previous work, the messages used by peer-to-peer file distribution systems have been shown to have similar functions across multiple systems. This similarity has been utilized by middleware to facilitate interoperability between a relatively small number of systems by translating the protocols. [Cugola] [Pallikara] These middleware systems essentially remove the system-specific description of message types by classifying them. One such middleware categorizes messages used by three popular peer-to-peer file distribution systems (Napster, Freenet, and Gnutella). The message types were categorized as "peer registry & discovery", "query", "data download protocol", and "data transfer mechanism for firewalled peers". [Lui] The classification of messages for that middleware can be done with only four categories due to the considerable similarity of messages used by the three systems selected for the middleware. This research requires a more extensive classification system to account for the variation in message types used in a larger number of peer-to-peer systems.

To account for the variation between message types in general peer-to-peer file distribution systems, we developed the *Taxonomy for Message Classification* (shown in Figure 4.2) that is a hierarchal system for classifying messages based on their purpose. The supertype/subtype relationship provided by the hierarchal system allows message types to be given a classification with greater specificity while maintaining a categorical similarity to related message types. The development of the *Taxonomy for Message Classification* was detailed in Section 3.3 of this dissertation. The *Taxonomy for Message Classification* was developed by surveying current peer-to-peer file distribution systems. Dissimilar systems and architectures were analyzed to determine if the taxonomy contained an appropriate classification for each system's message types.

The supertypes of SENDER and RECIPIENT, shown in Figure 4.2, are used by MCTAMA to describe the type of nodes sending and receiving the message. This information is used to give a more complete description of the message traffic. The relationships analyzed can use this information to describe the behavior of a node as it relates to other types of nodes. For instance, with this information, a test case can relate the amount of data received from a server to the amount of data sent to peers.

The supertype of PURPOSE, shown in the *Taxonomy of Message Classification,* contains the five most common types of network algorithms used by peer-to-peer file distribution systems: (1) Peer Discovery, (2) Network Maintenance, (3) Data, (4) Server Administration, and (5) Security. The classifications for messages within each of these five subtypes are inclusive to dissimilar algorithms. This inclusiveness is much like the work by [Lui], that classifies message types according to their high-level functionality. The Data subtype includes only four subclassifications. These subclassifications were determined using a survey of search methods.

[Tsoumakos] Each algorithm performed a similar purpose in the system, using different approaches. Despite the differences between the algorithms, the message types used by the algorithms performed the same function. These similar message types varied by naming convention, so the message types were given a description based on the function of the message type. The message types used by the dissimilar search methods used by the Gnutella and G2 systems are classified in Chapter 5 of this dissertation.

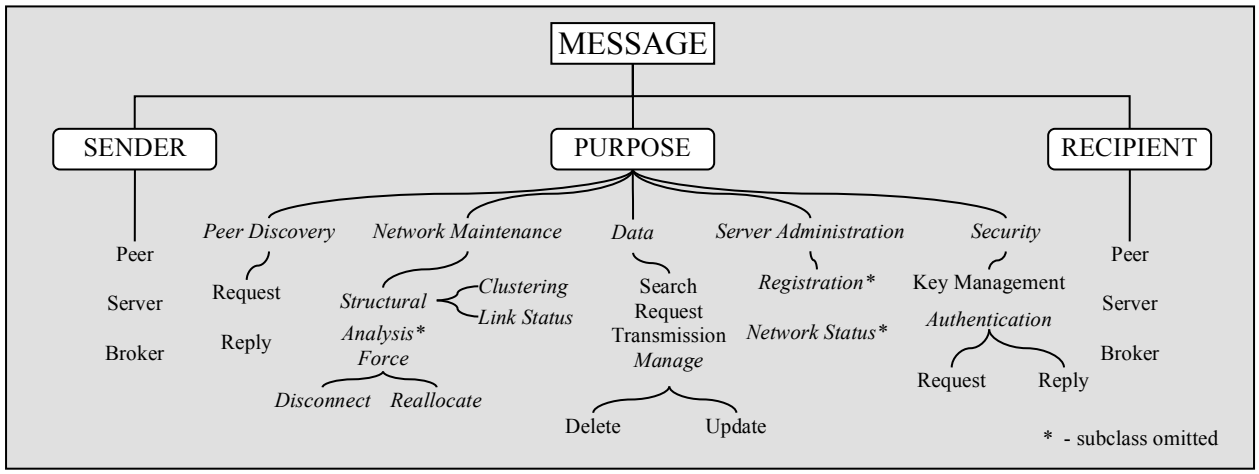

Figure 4.2 - Taxonomy for Message Classification

The classification of messages is done by human analysis of message types to determine their designed purpose. The system-specific name of messages can provide enough information to classify them, but the most common and useful source of information to this analysis is design documents. The protocol specifications, behavioral design documents, and architectural descriptions provide significant information about message types. Protocol specifications provide not only the message types used in the systems, but typically state the purpose of the message during certain sequences or algorithms. A protocol specification can provide enough information to classify all messages used by the system, but in cases where the protocol

specification is unavailable or inadequate, behavioral design documents, such as statecharts, sequence diagrams, and use case diagrams, give behavioral information that details the purpose of each message type. Architectural descriptions give a high-level description of how the system is designed. Architectural descriptions include the organization of nodes in the system and the roles that nodes are designed to have in the system. In some cases, these descriptions include textual descriptions of the algorithms used. This information adds context to the other design documents that aids the analysis.

The classification of messages does not rely solely on the availability of design documents. In cases where the design documents do not provide enough information to classify messages, the source code can be used. The analyst can determine the events that generate messages and the actions taken by the system when a particular message is received. This process of documenting source code in such a way has been accomplished in previous work. [Fraikin]

As an example of the message classification, consider the following description of a message type taken from RFC 913, the protocol specification for Simple File Transfer Protocol (SFTP):

"KILL file-spec

This will delete the file from the remote system.

Replies are:

+<file-spec> deleted

-Not deleted because (reason)"

This message described is named "KILL" within the system using SFTP. The purpose of the message is to delete a file on a remote system. One issue with this particular name is that its purpose isn't obvious from the name "KILL". In systems such as UNIX, kill has long been used

to mean the termination of a program, process or function. Having that in mind, the name kill could be more indicative of a command to terminate a current download. In this case, the above description provides enough information to classify this message and give it a more expressive description. Using the *Taxonomy for Message Classification*, the following classification process is applied:

1. Message most closely fits the DATA class since it applies to specific files.
2. Message most closely fits MANAGE subclass. It does not involve searching for data, requesting information about data, or transmitting data.
3. Message fits DELETE subclass by description.

The classification process classifies the SFTP KILL message as "SENDER, DATA > MANAGE > DELETE, RECIPIENT". This classification is more representative of the purpose of the message when comparing multiple systems and evaluating test results. SENDER and RECIPIENT are identifiers used later for identifying which nodes exchanged messages.

The *Taxonomy for Message Classification* classifies message types used by popular peer-to-peer file distribution systems. In cases where the taxonomy does not have a class that is specific to the message type being classified, it can be classified as the highest related class and a unique description. Using the example of KILL in SFTP, the message type could be classified as "SENDER, DATA > MANAGE > KILL, RECIPIENT" if the DELETE class did not accurately describe the purpose of the message type.

4.1.2 Step 2: Alter and Combine Sequence Diagrams

Test case generation in MCTAMA continues with the alteration and combination of sequence diagrams. Sequence diagrams are behavioral diagrams that describe the order of

messages sent during the execution of algorithms. These "scenarios" are essentially meant to describe the role of the objects during a period of time. By combining the scenarios into a single sequence diagram, MCTAMA creates a lifetime description of the objects. Sequence diagrams should show identifiable nodes as objects (vertical lines) and messages as interactions (horizontal arrows). Figure 4.3 gives an example of a sequence diagram used to generate test cases with MCTAMA.

Peer-to-peer file distribution systems are frequently designed using sequence diagrams. However, if the sequence diagrams are unavailable, such information could be included in the documentation within the protocol specification or source code. These documents contain the messages to be received by a node or node type and the expected responses. Such documents can provide enough information to create sufficient sequence diagrams as shown in previous work. [Fraikin]

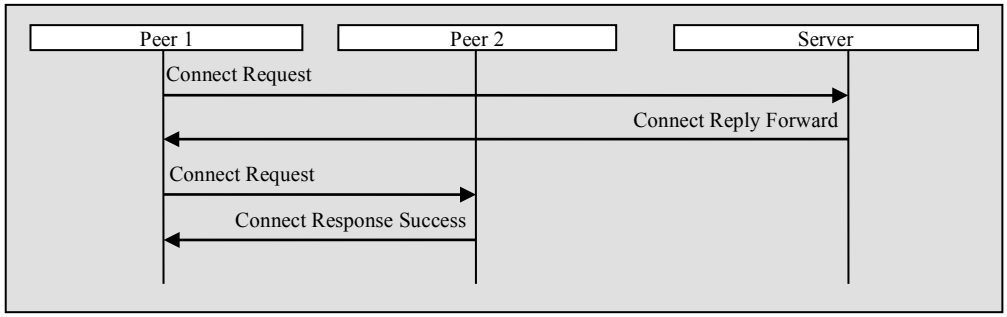

Figure 4.3 – Sample sequence diagram for a P2P file distribution system

The process for combining sequence diagrams based on interactions is given here in pseudocode as SD COMBINE. The following process assumes that objects are given the same identifier across the sequence diagrams being combined.

```
function SD COMBINE(SequenceDiagram A, SequenceDiagram B) :
    SequenceDiagram C := A ;                // copies contents of A into C
    for each object O in B :
        if O is not included in A :         // adds objects to A that are included
            add O to A ;                    // in B but not included in A
        end if ;
    end for ;
    for each interaction I in B :           // adds all interactions in B to A
        add I to A ;
    end for ;
    return A;
```

In set notation, sequence diagrams can be represented by a set of objects and a set of interactions. To combine sequence diagrams, the resulting sequence diagram is the union of these sets, and is written as follows:

$O.objects = \{ Object_1, Object_2, \ldots Object_N \}$

$O.interactions = \{ Interaction_1\{Object_1, Object_2\}, Interaction_2\{Object_1, Object_2\} \}$

$Let\ C.objects = \{ O_1.objects \cup O_2.objects \}$

$Let\ C.interactions = \{ O_1.interactions \cup O_2.interactions \}$

To illustrate these union operations, the following combination of sequence diagrams A and B is performed:

$A.objects = \{ Object1, Object2, Object3 \}$

$A.interactions = \{ InteractionA1\{Object1, Object2\}, InteractionA2\{Object1, Object3\} \}$

$B.objects = \{ Object1, Object2, Object4 \}$

$B.interactions = \{ InteractionB1\{Object1, Object2\}, InteractionB2\{Object2, Object4\} \}$

$Let\ C.objects = \{ A.objects \cup B.objects \}$

$Let\ C.interactions = \{ A.\ interactions \cup B.\ interactions \}$

```
C.objects = { Object1, Object2, Object3, Object4 }
C.interactions =  { InteractionA1{Object1, Object2}, InteractionA2{Object1, Object3},
                    InteractionB1{Object1, Object2}, InteractionB2{Object2, Object4} }
```

The sets of interactions are always disjoint since each interaction is distinct and unique to the sequence diagram. Therefore |A.interactions| + |B.interactions| = |C.interactions|. Objects can be included in multiple sequence diagrams, so |A.objects| + |B.objects| - (|A.objects| ∩ |B.objects|) = |C.interactions|.

The combined sequence diagrams provide a description of how nodes are designed to behave throughout the execution of the system. These diagrams provide an adequate representation of the roles individual nodes have within the peer-to-peer file distribution system for MCTAMA's test case generation process. A significant aspect of the combined sequence diagram is that it does not maintain the timing and ordering of messages that were present in the original design. MCTAMA does not require the sequence of messages for testing since the only relevant information is the messages likely to be sent and received by a node performing a specific role rather than a specific algorithm.

Each sequence diagram that represents an algorithm is included in the combined sequence diagram only once, but algorithms are executed at much different frequencies. For example, a node connecting to a peer-to-peer file distribution system may only connect once, but could download hundreds of files. The combined sequence diagram does not account for this type of variation. For example, MCTAMA's test case generation would create test cases that relate connection requests to download requests. In most of the systems surveyed, connection requests and download requests are not used in the same processes and were therefore unrelated. In a file

distribution system, a node connecting and downloading a single file and a node connecting and downloading multiple files would have much different ratios between the connect and download requests and this difference is considered a difference in node behavior. Despite these messages being unrelated, any variation in the relationship between these two message types would be reported as an inconsistency between design and the implementation if the allowed variation were unaccounted for in MCTAMA. This work includes an approach to account for the variation in algorithm frequency by the determination of which relationships are significant.

The approach to overcoming variations due to the frequency of algorithm execution is to calculate the likelihood that the relationship is significant. We consider the relationship between two message types significant if the message types are included in the same scenarios (sequence diagrams) prior to combination. In the previous example of connection requests and download requests, the processes that use these message types are shown in separate sequence diagrams since the design documents describe each process in separate sequence diagrams. The following algorithm calculates the likelihood that a relationship between message types is significant. The input to the algorithm is the set of all sequence diagrams used in design and the set of all message types used by the system.

```
function SD RELATIONS( SequenceDeagrams[], MessageTypes[] ) :
stats[MessageTypes][MessageTypes][2] = {0} ;
relationship[MessageTypes][MessageTypes] ;
for each SequenceDiagram SD in SequenceDiagrams :
    for each MessageType Msg1 in MessageTypes :
       for each MessageType Msg2 in MessageTypes :
              if Msg1 and Msg2 ∈ SD :
                 stats[Msg1][Msg2][0] := stats[Msg2][Msg1][0] += 1 ;
```

```
                stats[Msg1][Msg2][1] := stats[Msg2][Msg1][1] += 1 ;
            else if ( Msg1 ∈ SD ∧ Msg2 ∉ SD ) ∨ ( Msg2 ∈ SD ∧ Msg1 ∉ SD ) :
                stats[Msg1][Msg2][1] := stats[Msg2][Msg1][1] += 1 ;
            end if ;
        end for ;
    end for ;
end for ;
for each MessageType Msg1 in MessageTypes :
    for each MessageType Msg2 in MessageTypes :
        relationship[Msg1][Msg2] :=
            stats[Msg1][Msg2][0] / stats[Msg1][Msg2][1] ;
    end for ;
end for ;
return relationship ;
```

The result of SD RELATIONS is a matrix that indicates the significance of the relationship between message types as a number between 0 and 1. That number is the likelihood (as a percentage) that if a message type appears in a sequence diagram then the second message type will also appear. If two message types are more likely to be included in the same sequence diagram, the relationship between the two types will be greater than messages that appear in different diagrams. Since test cases are based on the relationship between multiple message types, the significance of the relationship indicates the significance of the test case.

4.1.3 Step 3: Describe the Role of Nodes in the System

Test case generation in MCTAMA ends with the description of the roles that each node is designed to have within the peer-to-peer file distribution system as determined by the message traffic. Roles are given as relationships between message types. Figure 4.4 shows two sample roles. ROLE 1 is based on the relationships between the message classifications SEARCH and

TRANSMISSION from the *Taxonomy for Message Classification*. ROLE 2 is based on the PEER LIST messages sent and received by a node.

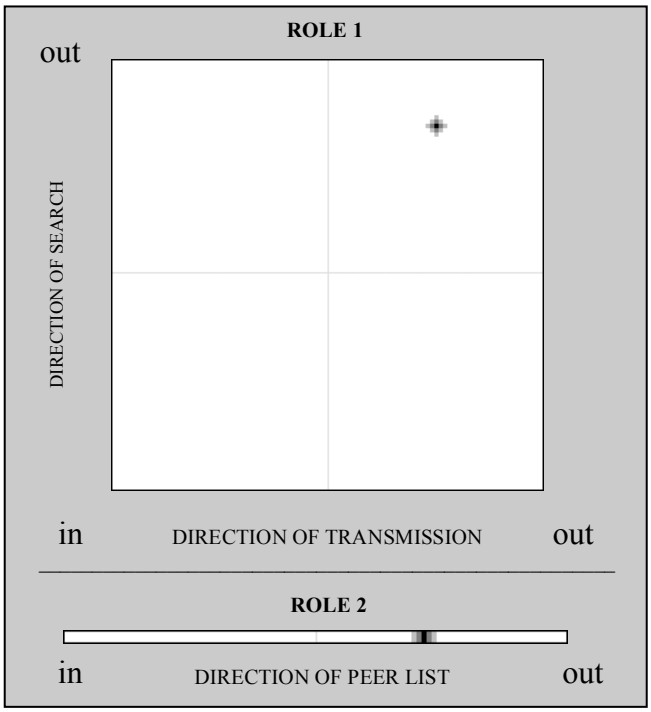

Figure 4.4 – Sample Roles for describing node behavior

The roles produced are given as a set of coordinates that describe a relationship between message types. The axes represent a characteristic of the message type. The coordinates are a point on these axes that show the expected values for each characteristic. The coordinates show the relationship between the two message types according to the characteristic being measured. This representation can be used to describe this relationship between messages generated from design and observed during execution. The expected values appear on the charts shown in Figure 4.4 as black points with a graphic emphasis to aid the interpretation of the results.

ROLE 1, shown in Figure 4.4, determines the role of a node based on two subclassifications of DATA messages and the direction they are sent (incoming or outgoing). To show the role for a particular node, a point is plotted at the coordinates that is based on average direction of both message types. Using the role as presented, the location of the point on the x-axis is determined by the average direction of DATA messages that are classified as TRANSMISSION and the location on the y-axis is determined by the average direction of DATA messages that are classified as SEARCH. As an example, a node designed as a "leaf node" consumes more data than it sends and is not an active participant in the structure of the systems, the expected behavior of the leaf node would be plotted as a point in the first quadrant of ROLE 1.

ROLE 2 shown in Figure 4.4 only considers the direction of a single classification. The construction of roles can vary significantly. Besides the number of message types (axes) considered by the role, the axes of roles can be associated with different characteristics as well. For example, a role could be created that plots the likelihood of a PEER_DISCOVERY message being classified as REQUEST or REPLY.

A completed test case contains a role based on a characteristic that can determine behavior according to message activity. A role is given as a set of coordinates and is a testable and consistent characteristic of the system. It is important that MCTAMA's test case generation create a comprehensive set of test cases since individual nodes can have multiple roles within the system. This set of roles describes the overall behavior of nodes within the system. As the size of the set of roles increases, the description of the behavior of the node within the system becomes more complete. This description is made up of multiple testable characteristics of the system. Therefore, MCTAMA generates a complete set of test cases for testing the behavior of nodes in the system according to roles.

4.1.4 The Completed Test Cases

The result of MCTAMA's test case generation is a set of roles that each node is designed to have within the system and a matrix giving the likelihood that two particular message types are related. Figure 4.4 shows an example of the generated test case. The set of test cases would include the relationships between all message types in the system.

The set of test cases contains the testable roles of nodes within the system as designed. This set of roles should match the roles observed during execution. The test is passed if all roles are consistent. Any inconsistency could mean a lack of conformance between design and implementation, so any such inconsistency results in a failure of the test. The significance of a failure is indicated by the likelihood that the message types used for that test are related as given in the matrix, SD RELATIONS.

4.2 Executing the Test Cases

The execution of test cases is a three-step process. As described in Figure 4.1, these steps are capturing the messages sent through the system, describing the roles of nodes in the system by analyzing observed messages, and analyzing the differences between assigned roles according to design and message capture.

4.2.1 Step 1: Capture Messages Sent Through the System

The first step in executing MCTAMA test cases is the dynamic capture of the messages sent through the peer-to-peer file distribution system. The capture of the messages can be facilitated through port sniffers, message brokers, or mechanisms within the application such as packet

dumps. Choosing the appropriate method of message capture depends on certain characteristics of the system. Two such characteristics are defined here.

Closed systems are systems for which the source code is unavailable. A frequently used type of closed system is a Commercial Off The Shelf (COTS) system. These systems are usually distributed as executable and are therefore considered *black boxes,* a system or component with limited or no knowledge of the inner workings of the software. [Comino] A system with significant access to the source code, components of the system, or documentation is an *open system*.

Energy-aware systems are distributed systems that have limited energy resources. A common type of energy-aware system is the sensor network but can include peer-to-peer networks when the location and resources of the nodes are utilized. These networks are often battery-powered and contain thousands of nodes. Any algorithm used by the system must be optimized for reducing energy usage to maintain the usefulness of the network. [Raghunathan]

4.2.1.1 Port Listening

Peer-to-peer systems may use an accessible form of communication to transmit messages between nodes. The vast majority of peer-to-peer systems use IP communication for which there are established methods of intercepting packets sent over a network. One such method for capturing messages in a system with accessible communication is to use a program that logs all messages received by listening to the ports used by the nodes of a peer-to-peer system. This type of program is called a “port listening” or “port sniffing” program and is readily available. The program must be installed on any physical node being tested in the distributed system and configured to listen to any port that the distributed system uses to communicate. [Vivo]

The benefit of using a "port sniffer" to capture messages is that it does not require the modification of source code and can therefore be applied to closed systems for which the source code is inaccessible. This approach is also ideal for large systems for which communication overhead for message capture can affect network performance.

A port sniffer cannot be used in cases where a peer-to-peer system's communication is inaccessible. The communication is inaccessible when the ports or communication media are restricted from external access. In addition, this type of message capture is not applicable when the messages are encrypted before they are sent through the network, and the tester does not have the ability to decrypt the messages. A port sniffer should not be used to capture the communication of energy-aware or resource-constrained systems since the port sniffer uses energy and resources to either store the log locally or transmit it over the network.

4.2.1.2 Brokered Communication

Brokered communication uses nodes called "brokers" to record the messages sent through a peer-to-peer system. Brokers do not act as an active part of the system, but rather act as passive intermediates in communication. These brokers could be switches that log all communication sent through the network or nodes that log messages before sending them to the intended recipient.

Brokered communication is useful for peer-to-peer systems that have resource constraints or limited physical access to the node. However, the brokers could create network bottlenecks in large or high volume networks with a limited number of brokers. To prevent such bottlenecks, multiple brokers would have to be used or the scalability of the method reduced. Therefore, the

ideal application of brokered communication is to relatively small-scale, resource constrained systems.

4.2.1.3 Source Code Modification

An efficient method for capturing messages in peer-to-peer systems with accessible source code is the modification of the software to create log files of all messages that are sent and received by the node itself. The effort required to modify the source code is minimal and also formats the messages for easier further use. In addition, this method can guarantee that all attempts to send or receive messages will be recorded in cases where network access is lost. For open systems that send encrypted messages through the network or use an inaccessible form of communication, source code modification creates a record of the messages that are sent before they become inaccessible or after they are retrieved.

However, source code modification requires recompilation and redistribution of the software to all nodes. This aspect of source code modification makes it less practical for large-scale systems. Another limiting characteristic of this method is that it increases the workload on the processor as well as the amount of storage space required on the node being tested. Therefore, source code modification should be used when MCTAMA is being used to test to relatively small-scale, open systems for which the nodes of the system have no significant resource constraints.

4.2.1.4 Summary and Comparison of Message Capturing Methods

The application of message capturing methods is described in Table 4.1.

Table 4.1 Comparison of message capturing methods

	Inaccessible communication	Closed System	Resource Constrained	Large-Scale
Source code modification	Ideal	Not Applicable	Not Ideal	Not Ideal
Port Listening	Not Applicable	Ideal	Not Ideal	Not Ideal
Brokered communication	Not Applicable	Not Ideal	Ideal	Not Applicable

MCTAMA can be applied to all systems except closed systems with inaccessible communication since there is no applicable method for capturing the messages. It should also be noted that there are no ideal methods for capturing methods sent through large-scale peer-to-peer systems. This is because the nature of large-scale systems prevents any method from being able to capture all messages sent though the system. However, MCTAMA can be used to analyze individual nodes within a large-scale system using source code modification or port listening.

4.2.2 Step 2: Describe the Role of Nodes in the System by Observed Messages

The second step in the process of test case execution is the analysis of the captured messages to determine the roles of the individual nodes within the system. (Step 2 of Figure 4.1) This step mirrors Step 3 of the process of generating test cases ("Describe the role of nodes in the system by analyzing messages used in design") with the exception that this step uses the messages sent and received during execution rather than the messages described in the design. By using the same process to describe behavior, the designed behavior and the behavior during execution are presented using a common representation for describing node behavior.

Step 1 of generating the test case is the classification the messages used by the peer-to-peer system according to the *Taxonomy for Message Classification*. In the execution of the test case, the same message classifications that are created in Step 1 of generating the test case will be applied to the observed messages. Any message types observed that are not accounted for in the design should be considered an error without any further investigation by MCTAMA. Step 2 of generating the test case details the modification and combination of design documents and is not applicable to the execution of the test case.

To create test results, MCTAMA creates a value for each role created during the test case generation. Each role created from design is a test case. The common representation for describing node behavior is a Cartesian coordinate system that shows the relationship between one or more message characteristics and the expected test results is a set of coordinates presented by a black dot. The actual values are created for each role by determining the value of those characteristics in the captured messages and shown on the chart as a red point with graphic emphasis.

The same sample roles as shown in Figure 4.4 have been updated in Figure 4.5 to include examples of actual values of the behavior of the nodes. The final step is to determine the Euclidean distance between the expected and actual values using the formula:

$$\sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

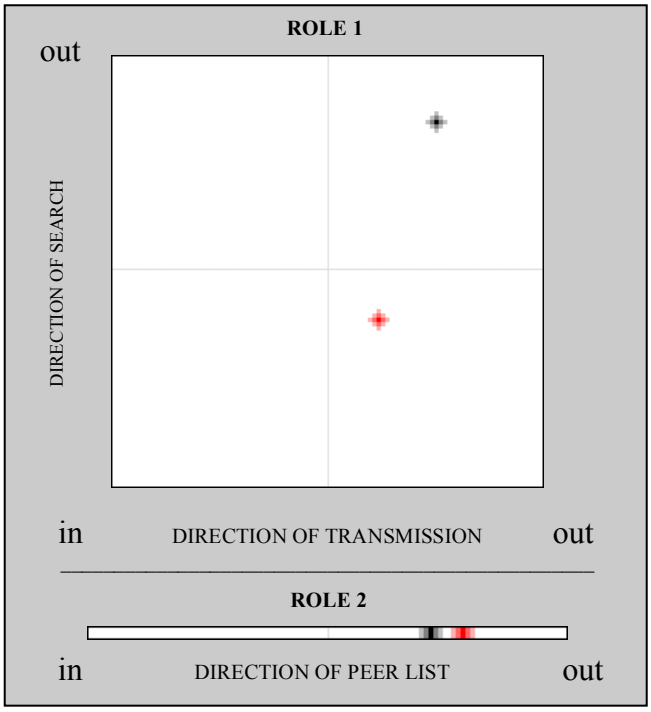

Figure 4.5 Sample Test Results

Since most test cases generated by MCTAMA will only analyze the behavior according to two relationships, the role will be presented in one or two dimensions. The equation for finding the Euclidean distance between points in one or two dimensions is simplified to:

$$|p_1 - p_2|, \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}$$

for one and two dimensional coordinates respectively.

This step produces a set of test results from the test cases and the messages captured during the system test. The test results are described as a both a Euclidean distance between the expected and actual results and a pair of coordinates showing the distance between the two points graphically.

4.2.3 Step 3: Analyze Differences Between Design and Message Capture

The final step of MCTAMA is to analyze the test results. (Step 3 of Figure 4.1) MCTAMA produces a large number of test cases and results to analyze since it analyzes all of the relationships between all message types for each node tested. MCTAMA does not produce a simple PASS/FAIL test result, but it considers any inconsistency between designed and expected behavior as significant.

A querying system was written in the server-side scripting language, PHP. It requires that all of the messages captured by MCTAMA are stored in an accessible MySQL database with the sender, recipient, and message type stored for each message captured. It generates graphical test results for all test cases with 1 or 2 message types.

For this aspect of the research, we developed a querying feature that facilitates the filtering of the results from the test cases. This querying feature helps determine the significance of the inconsistencies between the expected and actual values found by MCTAMA. It provides the selection options for the tester to create a query for viewing a specific test result. This querying feature allows the tester to analyze specific behaviors within the system. For example, if the tester want to analyze a specific feature such as connection or file transfer, the querying system allows the tester to specify that feature. The message type selection is the list of all message types captured by MCTAMA. The characteristic selection is the list of all MCTAMA-defined characteristics. Figure 4.6 shows the querying system generating the test result that was used in Figure 4.5.

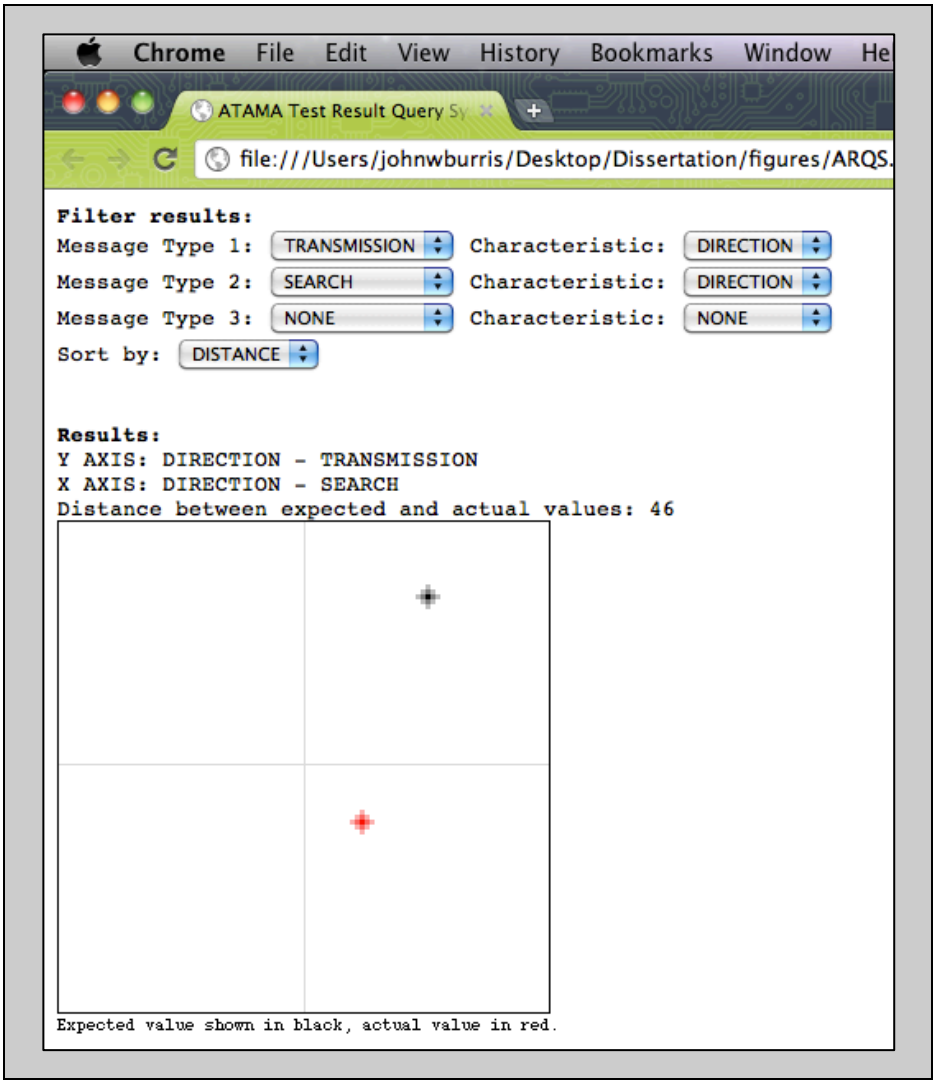

Figure 4.6 - Test Result Querying System

The querying feature can sort the results to assist the tester in determining which results are significant. The results can be sorted by the distance between expected and observed results or they can be sorted by the value given to the relationship in SD RELATIONS. This capability allows the tester to look for significant test results without querying a specific result, allowing the tester to analyze the significant inconsistencies that may not normally be analyzed due to the large number of possible test cases.

The first criterion for sorting the results is the significance of the relationship between two messages analyzed in the test case. The significance of the relationship is stored in the relationship matrix SD RELATIONS that was created during test case generation. By sorting the test results by the significance of the relationship, the display of test results highlights the test cases that are more likely to warrant further investigation if an inconsistency has been found. This criterion does not assist the tester in analyzing test cases for a single message since the significance of the behavior of a single message type does not involve additional messages. The significance of the relationship between three or more messages at the present time is available only numerically, but the querying feature could be extended to three or more messages.

The second criterion for sorting the results is the distance measure between the expected and actual results. Sorting by significance allows the tester to view the most "critical" test cases. However, there are also cases where the relationship may not be as significant, but a large inconsistency was found. Displaying test results sorted by distance measure allows the tester to analyze inconsistencies with the largest variations. The tester must determine if the inconsistency warrants further investigation by determining if the two relationships are related in design. For example, the relationship between a connection message and a data transfer may not be significant if the two processes are separate in the system being tested. The relationship measure given by SD RELATIONS can aid the tester if the test result is based on the relationship between two messages.

4.2.4 The Completed Test Execution

The result of an MCTAMA test is a set of test cases that are determined to have significant behavioral inconsistencies between system design and behavior. These test cases aid software

developers in the development of software that has a strong conformance to design by highlighting specific areas of code that may cause a failure of the software to meet a behavioral specification.

CHAPTER 5 VALIDATION AND VERIFICATION

In this chapter, we describe two experiments using MCTAMA to test peer-to-peer file distribution systems. The first experiment runs an MCTAMA test of the gtk-Gnutella implementation of a file-sharing client using the Gnutella protocol. The second experiment tests two similar protocols, the Gnutella and Gnutella2 protocols, to determine if a known difference between the two protocols is identified by MCTAMA. The purpose of these experiments is to: (1) validate MCTAMA as a method for conformance testing peer-to-peer software distribution systems, and (2) to verify that MCTAMA is capable of describing node behaviors at design and execution and identifying inconsistencies in those behaviors.

5.1 MCTAMA Test of Gnutella

The Gnutella network was chosen as the peer-to-peer file distribution system to verify that MCTAMA is capable of finding inconsistencies between design and implementation and validate it as a method for identifying which inconsistencies in behavior are significant. Gnutella is a decentralized peer-to-peer file distribution network. The protocol used to communicate in the system is called the Gnutella protocol. The system is an open source project released under the GNU General Public License. Gnutella was chosen for this experiment for numerous reasons. Gnutella is a popular system with millions of users. There are only five message types in version 0.4 of the protocol. The system is a well-established system and the focus of multiple published articles and surveys. [Lui],[Ripeanu],[Anderson],[Zeinalipour-Yazti]

Multiple software projects exist for clients that can connect and participate in the system. The client chosen for this experiment was the gtk-gnutella client. Like Gnutella, the gtk-gnutella client is an open source project under the GNU General Public License. The client was chosen

for its accessible source code as well as a feature of the graphical user interface that shows the user a summary of the message traffic by both the message type and the message direction. Figure 5.1 shows the gtk-gnutella client being run in Mac OS X. Figure 5.2 shows the gtk-gnutella client displaying the message traffic summary.

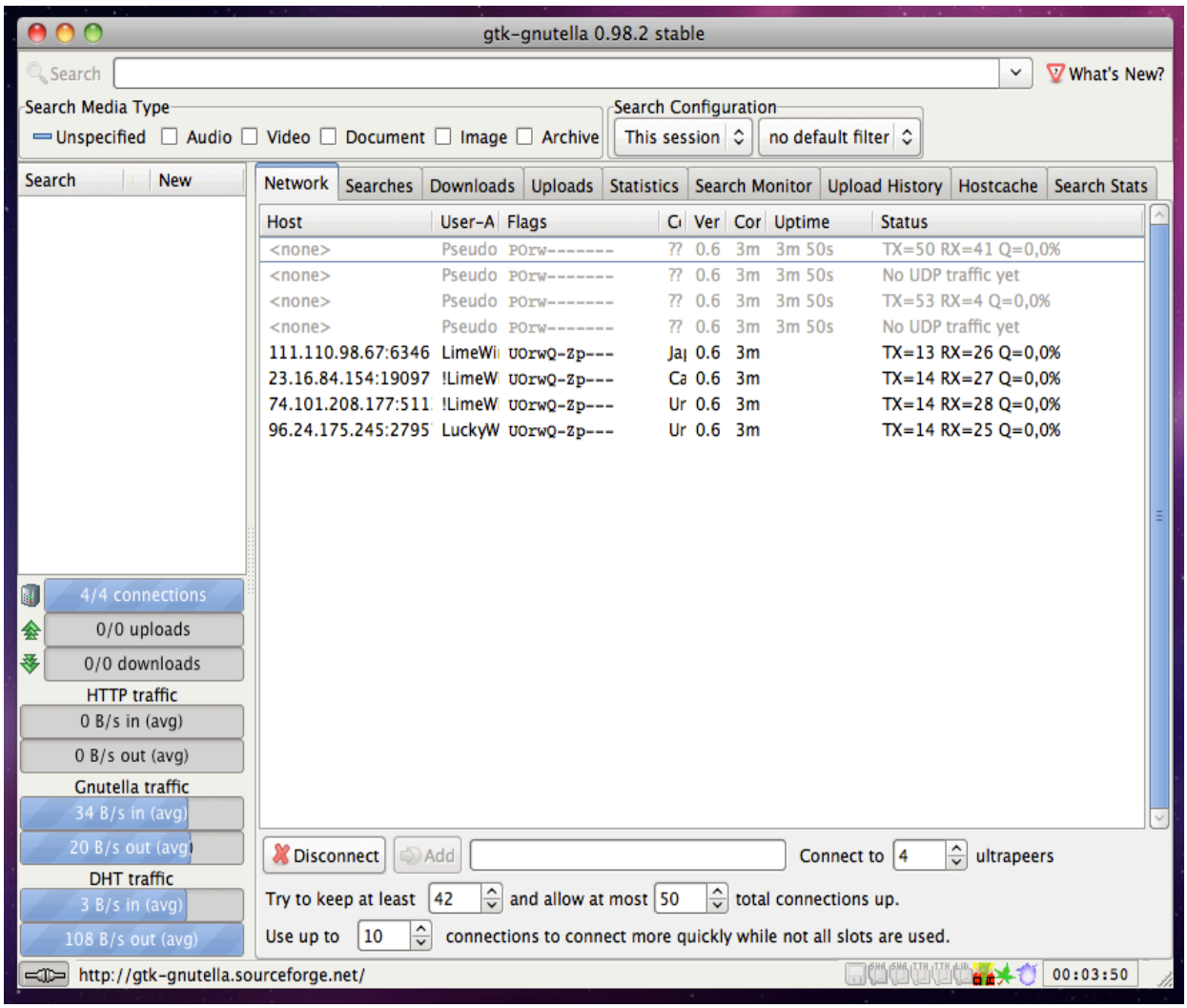

Figure 5.1 gtk-gnutella client at start-up.

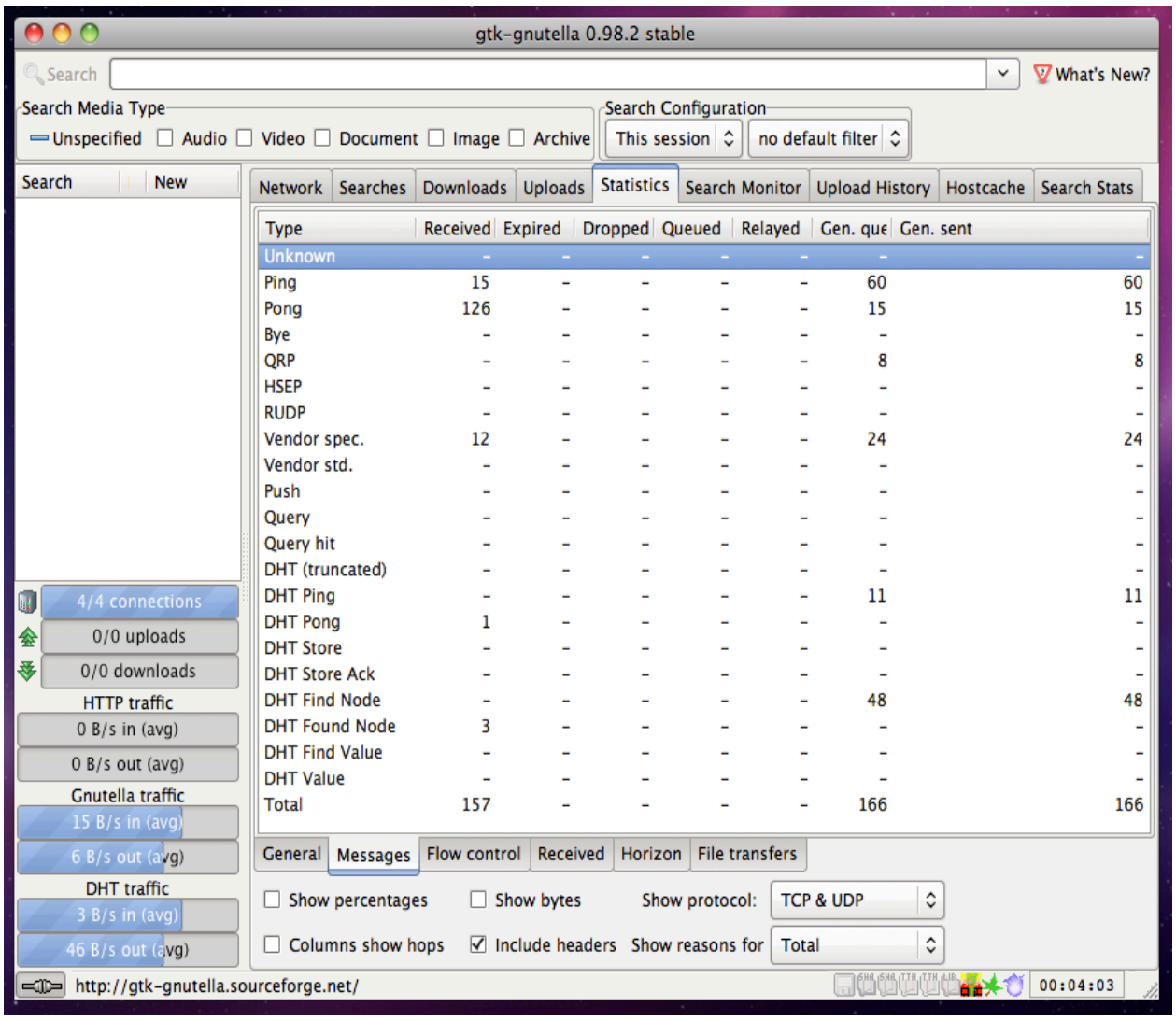

Figure 5.2 – gtk-gnutella client message traffic feature.

5.1.1 Step 1 of MCTAMA Test Case Generation

One key benefit of testing the Gnutella network in this experiment is the limited number of message types. The Gnutella Protocol Specification provides five primary message descriptors: Ping, Pong, Query, QueryHits, and Push. The messages were classified for this experiment based on the following description provided as written in the specification. Servents are the name given to nodes in the Gnutella network.

Ping

"A servent uses Ping descriptors to actively probe the network for other servents. A servent receiving a Ping descriptor MAY elect to respond with a Pong descriptor, which contains the address of an active Gnutella servent (possibly the one sending the Pong descriptor) and the amount of data it's sharing on the network." [Gnutella]

Pong

"Pong descriptors are ONLY sent in response to an incoming Ping descriptor. Multiple Pong descriptors MAY be sent in response to a single Ping descriptor. This enables host caches to send cached servent address information in response to a Ping request." [Gnutella]

Query

"A servent receiving a Query descriptor with a Minimum Speed field of n kb/s SHOULD only respond with a QueryHits if it is able to communicate at a speed >= n kb/s." ... "QueryHits descriptors are only sent in response to an incoming Query descriptor. A servent should only reply to a Query with a QueryHits descriptor if it contains data that strictly meets the Query Search Criteria." [Gnutella]

QueryHits

"QueryHits descriptors are only sent in response to an incoming Query descriptor. A servent should only reply to a Query with a QueryHits descriptor if it contains data that strictly meets the Query Search Criteria." [Gnutella]

Push

"A servent may send a Push descriptor if it receives a QueryHit descriptor from a servent that doesn't support incoming connections. This might occur when the servent sending the QueryHits descriptor is behind a firewall. When a servent receives a Push descriptor, it may act upon the push request if and only if the Servent Identifier field contains the value of its servent identifier." [Gnutella]

Based on those descriptions, the messages were classified as the following according to the *Taxonomy for Message Classification:*

1) Ping: Peer Discovery -> Request

2) Pong: Peer Discovery -> Reply

3) Query: Data -> Search

This message could be classified as a packet used for both searching and requesting data. The purpose of the message is to flood the peers with the query string and receive the data based on that query. The message was classified as Data -> Search because the message only serves the purpose of requesting data if the data being searched for is present at the node. All Query messages sent serve the purpose of searching for data.

4) Query Hit: Data -> Transmission

5) Push: Data -> Request

5.1.2 Step 2 of MCTAMA Test Case Generation

The Gnutella Protocol Specification version 0.4 gives three behavioral diagrams to describe the messages sent through the system. Figure 5.3 shows the process of connecting and finding peers. Figure 5.4 shows the process of transmitting data and the process of transmitting from firewalled peers.

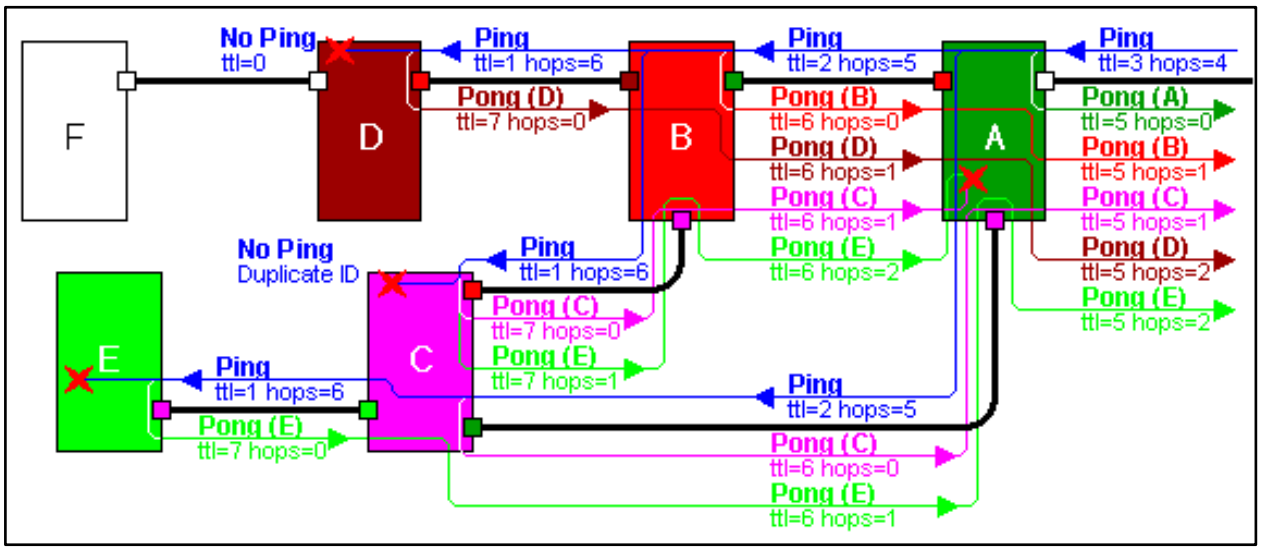

Figure 5.3 – Gnutella Connection and Peer Discovery as given in [Gnutella]

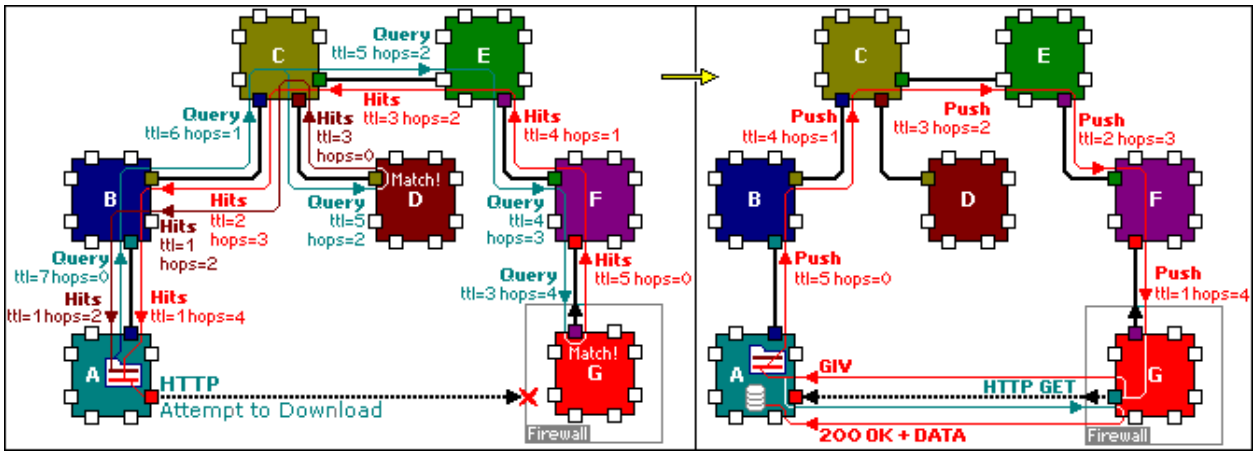

Figure 5.4 – Query/QueryHit/Push Routing as given in [Gnutella]

Since the specification was given in a diagram other than a sequence diagram, we constructed the sequence diagrams for use with MCTAMA. The diagrams given in the specification are not intended to describe a protocol, but rather provide an illustration of the processes. This is ideal for MCTAMA since it describes the expected behavior of the nodes in the system. The construction process was:

1) Create a sequence diagram for each component diagram.
2) Create an object for each component shown in the diagram.
3) Create an interaction for each message shown in the diagram.

The diagram shown in Figure 5.3 contains an additional "off-diagram" component with one connection to the component labeled "A". This component will be labeled as object "N" in the constructed diagram. Figure 5.5 is the constructed sequence diagram for Gnutella Connection and Peer Discovery given in Figure 5.3.

From Figure 5.3, each component, labeled "A"-"F" and "N", is added to the constructed sequence diagram as an object and is represented by a vertical line. The directed connections shown in Figure 5.3 are then added to the constructed sequence diagram as interactions. These interactions are represented as arrows between the source object and the destination object that correspond with the components with the directed connection in Figure 5.3. For example, components "C" and "E" in Figure 5.3 share two directed connections. The first is labeled as a Ping message from "C" to "E". The second is labeled as a Pong message from "E" to "C". These are added to Figure 5.5 as a pair of interactions between objects "C" and "E". The first is labeled as a Ping message sent from "C" to "E" and the second is labeled as a Pong message sent from

"E" to "C". This translation is repeated for all seventeen directed connections shown in Figure 5.3.

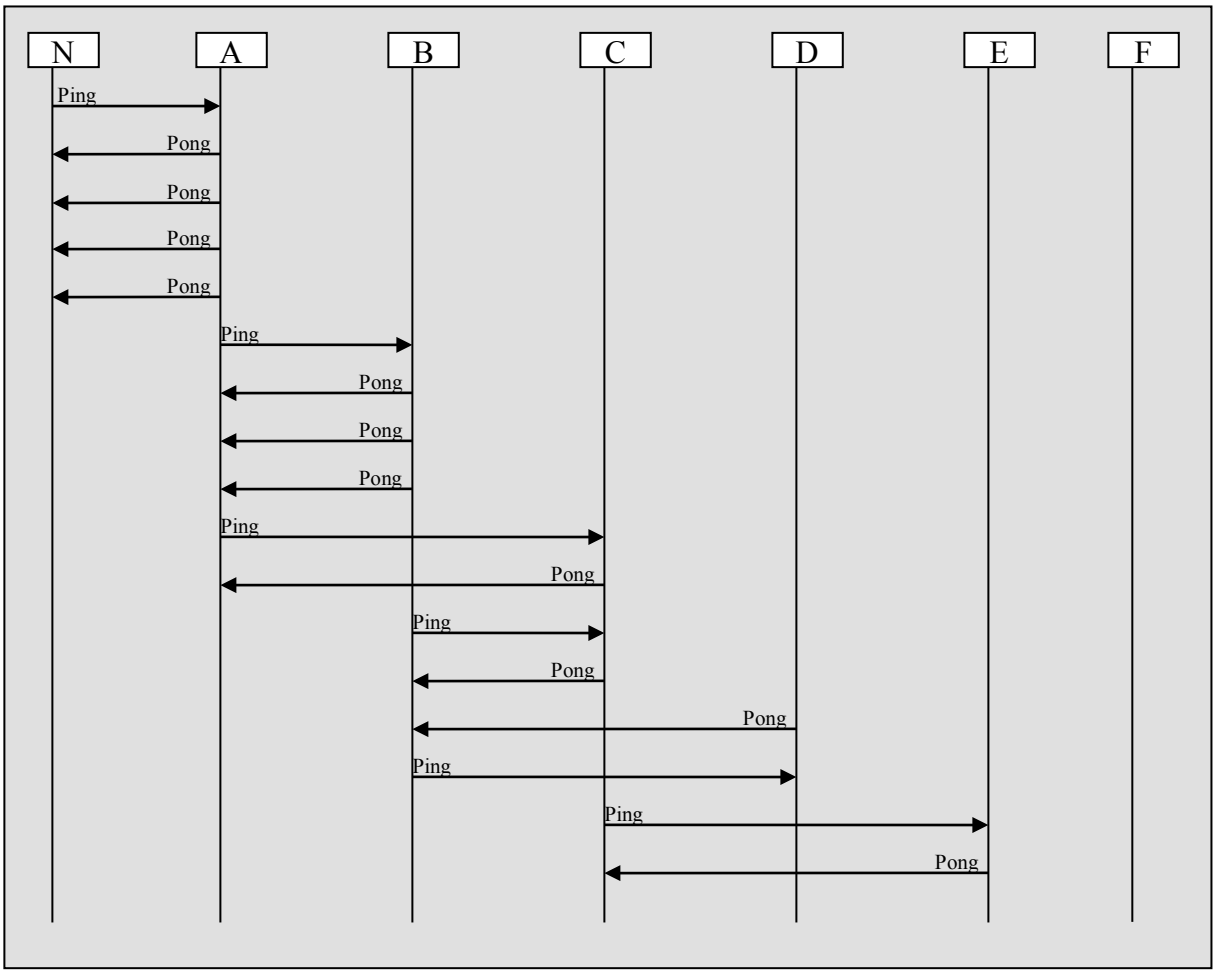

Figure 5.5 – Constructed Diagram for Gnutella Connection and Peer Discovery

Figure 5.6 is the constructed sequence diagram for the Query and Query Hit routing shown in Figure 5.4. The connection labeled HTTP in Figure 5.4 is omitted from the constructed sequence diagram since the dotted line denotes a connection not included in the search process. The connection was included for illustrative purposes to show that the process produces a connection for eventual file transfer. As with the previous construction, each component "A"-"G" that are shown in Figure 5.4 are added to the constructed sequence diagram as objects and

represented as vertical lines. The fourteen directed connections shown in Figure 5.4 are added to the constructed sequence diagram in Figure 5.6 as interactions. These interactions are represented as arrows between the source object and the destination object that correspond with the components with the directed connection in Figure 5.4.

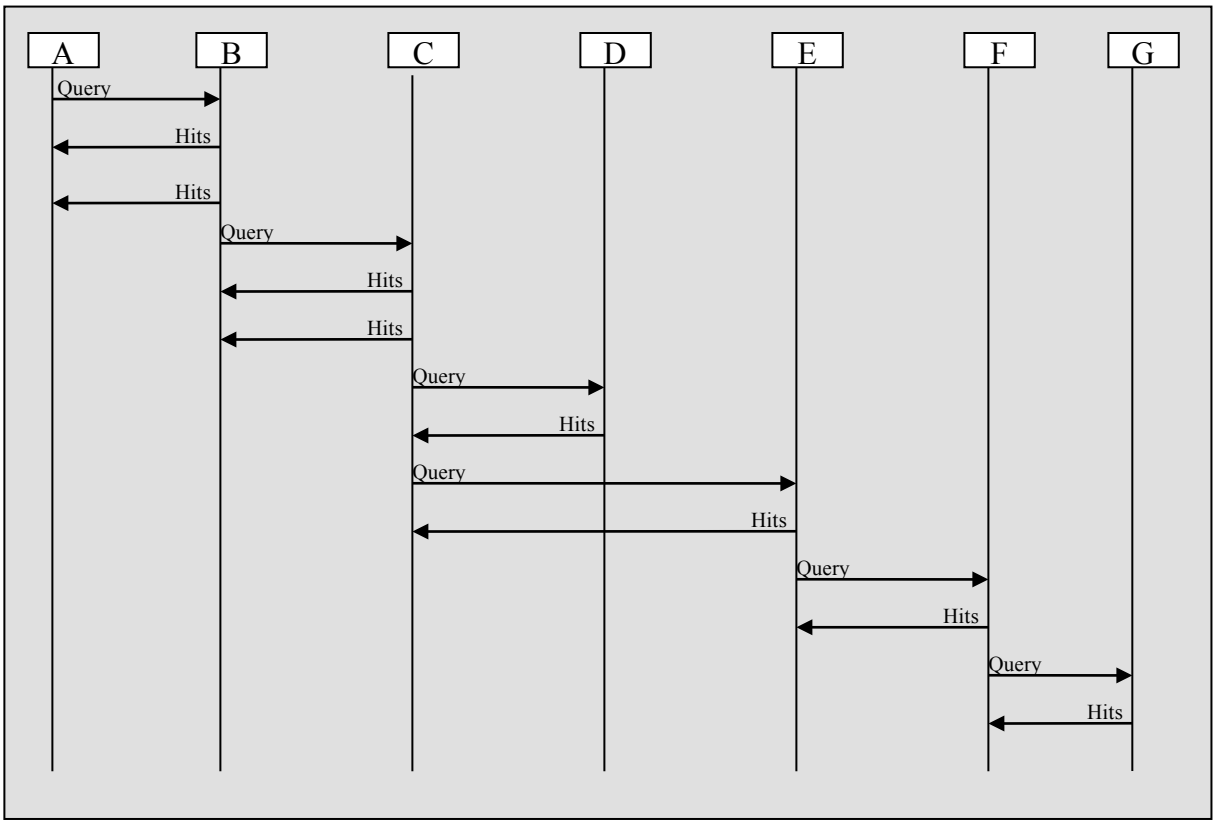

Figure 5.6 – Constructed Diagram for Query/QueryHit Routing

The Push routing diagram given in Figure 5.4 is not included in the constructed diagrams because the push message is only used for the transfer of data between a node and a firewalled peer and not for connection or search methods.

The sequence diagrams shown in Figure 5.5 and Figure 5.6 are then modified to use the message classifications. As determined in Step 1 of this experiment, the labels Ping, Pong,

Query, and Hits are replaced with “Peer Discovery -> Request”, “Peer Discovery -> Reply”, “Data -> Search”, and “Data -> Transmission” respectively.

The sequence diagrams shown in Figure 5.5 and Figure 5.6 are then combined to form a single sequence diagram. The union of objects from these sequence diagrams initially produces a set of nine objects labeled “A”-“G” and “N”. The union of interactions results in a set of thirty-one interactions since each interaction is distinct.

Further analysis of the objects represented in Figure 5.5 and Figure 5.6 show that the union produces a set with a single distinct object. The Gnutella Protocol, version 0.4 is a homogenous, decentralized system; so all peers behave the same. Therefore, a node within the network may assume the role of any object diagrammed, so the objects are not distinct. The combined sequence diagram shown in Figure 5.7 shows the single object labeled as “Gnutella Node”. The interactions are shown as loops since the source object and destination object have been combined as the same object. The combined sequence diagram is shown in Figure 5.7 with a single object and thirty-one interactions (multiplicity added).

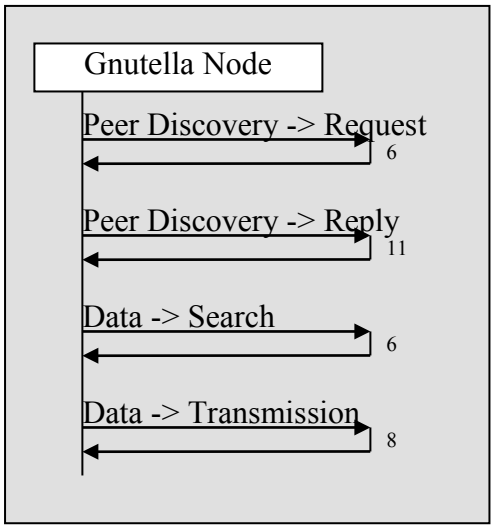

Figure 5.7 – Combined Sequence Diagram for Gnutella Node

An observation of MCTAMA can be made at this point in the experiment. In a homogenous network where all nodes are designed to behave the same, the relationship between the messages sent and the messages received for a single message classification will always show a 1:1 relationship due to SD COMBINE's reduction of the diagram to a single object. This result does not reduce the effectiveness of MCTAMA. In a homogenous network, a single node could assume the role of any object shown. A variation from the 1:1 relationship could show that the node being tested is more likely to assume the roles or avoid the roles of certain objects show in the original behavioral diagram.

The value of SD RELATIONS was computed and the results are given in Table 5.1. The results shown for SD RELATIONS show that the "Peer Discovery -> Request" and "Peer Discovery -> Reply" messages are closely related. In addition, the "Data -> Search" and "Data -> Transmission" messages are closely related. However, there is no significant relationship between the messages classified as the "Data" supertype or the "Peer Discovery" supertype.

Table 5.1 - SD RELATIONS for Gnutella System

	Peer Discovery -> Request	Peer Discovery -> Reply	Data -> Search	Data -> Transmission
Peer Discovery -> Request	1	1	0.000	0.000
Peer Discovery -> Reply	1	1	0.000	0.000
Data -> Search	0.000	0.000	1	1
Data -> Transmission	0.000	0.000	1	1

Step 2 of MCTAMA applied to the Gnutella file distribution system generated a combined sequence diagram for the Gnutella node as well as an indication of the significance of the relationships between the message types used in the design documents. The combined sequence diagram shown in Figure 5.7 provides the information to generate MCTAMA test cases in Step 3 of test case generation. The significance of the relationships provided in Table 5.1 determines the significance of the test results during Step 3 of test case execution.

5.1.3 Step 3 of MCTAMA Test Case Generation

The creation of the common representation for describing node behavior using the combined sequence diagram created in Step 2 completes the generation of test cases for the Gnutella file distribution system.

The test cases are provided for only the significantly related message types. According to Table 5.1, the only significant test cases are: (1) the relationship between the "Peer Discovery -> Request" and "Peer Discovery -> Reply" message types and (2) the relationship between "Data -> Search" and "Data -> Transmission" message types. Any test result based on the relationship between other message types would be disregarded by MCTAMA since SD RELATIONS determined that those relationships are insignificant.

Since Gnutella is a homogenous network, we focus on the relationships between the total numbers of messages that the node being tested sent and received of a single message classification. This type of relationship remains meaningful despite all interactions being sent and received by a single object (a looped message). The relationships that analyze the direction of the interaction are no longer meaningful since the ratio of a single message type's number of sent and received messages will always be 1:1.

These two factors reduce the total number of meaningful, significant test cases to two. The test cases are: (1) the ratio of "Peer Discovery -> Request" and "Peer Discovery -> Reply" message types and (2) the ratio of "Data -> Search" and "Data -> Transmission" message types. Any other test cases would be disregarded by MCTAMA as insignificant.

The test cases showing the expected relationship between "Peer Discovery -> Request" and "Peer Discovery -> Reply" is shown in Figure 5.8. The relationship between "Data -> Search" and "Data -> Transmission" is shown in Figure 5.9. The x-axis represents the total number of messages sent and received. The distance to the left and right of the expected value is the percentage of total messages that are sent and received of the labeled message type.

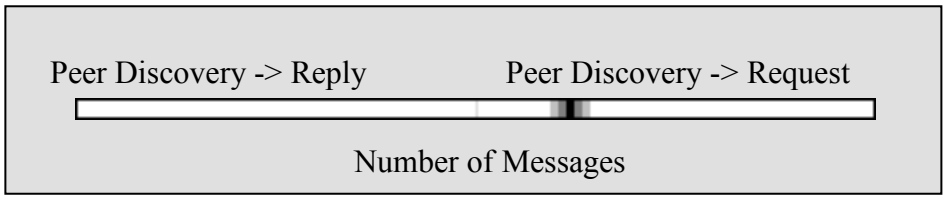

Figure 5.8 – Test case for Peer Discovery message classifications

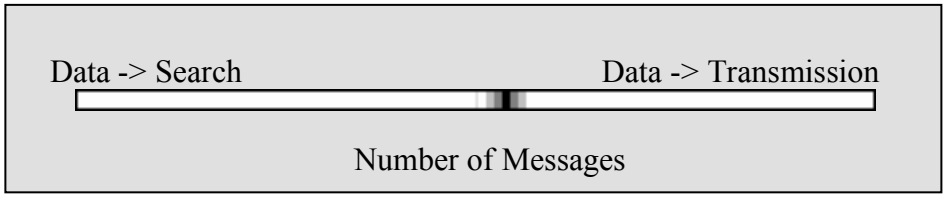

Figure 5.9 – Test case for Data message classifications

The test cases shown in Figure 5.8 and Figure 5.9 are the result of using MCTAMA to analyze and test the Gnutella peer-to-peer file distribution system. These test cases can be used to verify node behavior during the simulation and execution of the system. For the purposes of this test case, we use these test cases to test a single, active Gnutella node.

5.1.4 Step 1 of MCTAMA Test Case Execution

This experiment is the testing of the gtk-gnutella client against the Gnutella version 0.4 protocol specification. The execution of the test case begins with the capture of the messages sent through the system. MCTAMA does not test a node in a given scenario, but rather the behavior of a node throughout the execution and operation of the system. Therefore, a gtk-gnutella client is started and used to download a copy of the Linux operating system. All messages are captured until the program terminates. For this experiment, a method for message capture given in Section 4.2.1 was used despite the message-logging feature of the client. The source code for the gtk-gnutella client is available for download and alteration. The source code was modified to create a log of all messages sent and received by the node being tested. The following code includes the functions that required modification. Sections of the code have been removed and replaced with "[… code clipped …]".

The function "gmsg_dump" is included in the gtk-cnutella source code and logs the messages as hexadecimal strings in the filename provided to the function. The function "gmsg_init" provides the numerical identifier and string associated with each Gnutella message. The result of this capture method is a hexadecimal file that contains all messages sent and received by the node. The file can be parsed by a string processor or manually by the human tester. For this test case, the file was parsed using the search feature of a hexadecimal file editor, Hex Fiend. The search feature provided the number of occurrences of each message type.

```
 /**
 * Log an hexadecimal dump of the message `data'.
 *
 * Tagged with:
 *
 *     msg_type (payload length) [hops=x, TTL=x]
 *
 * to the specified file descriptor.
 */
static void
gmsg_dump(FILE *out, gconstpointer data, guint32 size)
{
       g_assert(size >= GTA_HEADER_SIZE);

       dump_hex(out, gmsg_infostr_full(data, size),
              (char *) data + GTA_HEADER_SIZE, size - GTA_HEADER_SIZE);
}

       [... code clipped ...]

/**
 * Initialization of the Gnutella message structures.
 */
G_GNUC_COLD void
gmsg_init(void)
{

[... code clipped ...]

              switch ((enum gta_msg) i) {
              case GTA_MSG_DHT:            w = 0;      s = "DHT"; break;
              case GTA_MSG_HSEP_DATA:      w = 0;      s = "HSEP"; break;
              case GTA_MSG_INIT:           w = 1;      s = "Ping"; break;
              case GTA_MSG_SEARCH:         w = 2;      s = "Query"; break;
              case GTA_MSG_INIT_RESPONSE:  w = 3;      s = "Pong"; break;
              case GTA_MSG_SEARCH_RESULTS: w = 4;      s = "Q-Hit"; break;
              case GTA_MSG_PUSH_REQUEST:   w = 5;      s = "Push"; break;
              case GTA_MSG_VENDOR:         w = VMSG_W; s = "Vndor"; break;
              case GTA_MSG_STANDARD:       w = VMSG_W; s = "Vstd"; break;
              case GTA_MSG_RUDP:           w = 6;      s = "RUDP"; break;
              case GTA_MSG_QRP:            w = 8;      s = "QRP"; break;
              case GTA_MSG_BYE:            w = 9;      s = "BYE"; break;

       [... code clipped ...]

}
```

5.1.5 Step 2 of MCTAMA Test Case Execution

The test cases generated by MCTAMA for the gtk-gnutella client, shown in Figure 5.8 and 5.9, focused on four message types used by the Gnutella system. These test cases do not differentiate between messages that were sent and messages that were received since all

interactions are sent and received at a ratio of 1:1 in a homogenous network. Therefore, the total numbers of messages sent and received of a message type were counted. The log file containing the messages sent and received by the node was parsed by counting the number of occurrences of the message identifier.

```
Total Messages of observed messages:      10,917      (100%)
Ping:                                      1,965      (18%)
Pong:                                      3,712      (34%)
Query:                                     3,821      (35%)
QueryHit:                                  1,419      (13%)
```

The result for a gtk-gnutella client after approximately 20 hours of execution was:

```
Total Messages of observed messages:      110,138     (100%)
Ping:                                      17,629     (16%)
Pong:                                      35,241     (32%)
Query:                                     53,960     (49%)
QueryHit:                                   3,308     (3%)
```

The message classifications created in Step 1 of the MCTAMA Test Case Generation process are applied to the observed messages. The relationships between these messages are then described using MCTAMA's representation for describing node behavior. The observed behavior of the node after approximately one hour of execution as shown by message traffic of Peer Discovery messages is shown in Figure 5.10. The ratio of "Peer Discovery -> Reply" to "Peer Discovery -> Request" is 18:34, or x=65. This is because 65% of Peer Discover messages sent or received by the observed node were "Peer Discovery -> Request". The observed behavior of the node after approximately one hour of execution as shown by message traffic of Data messages is shown in Figure 5.11. The ratio of "Data -> Search" to "Data -> Transmission" is

13:35, or x=73. This is because 73% of Data messages sent or received by the observed node were “Data -> Search”. The observed behavior of the node after approximately 20 hours is shown in Figure 5.12 and Figure 5.13 as shown by the message traffic of Peer Discovery and Data message classifications respectively.

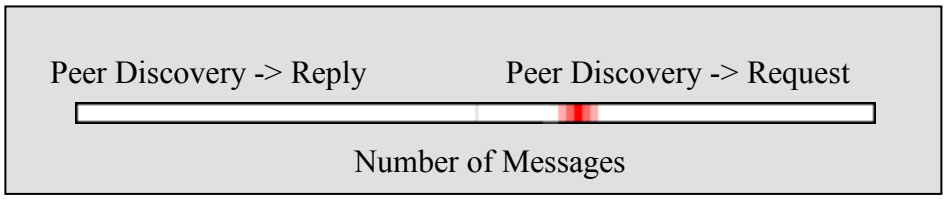

Figure 5.10 – Observed role according to Peer Discovery message classifications (1)

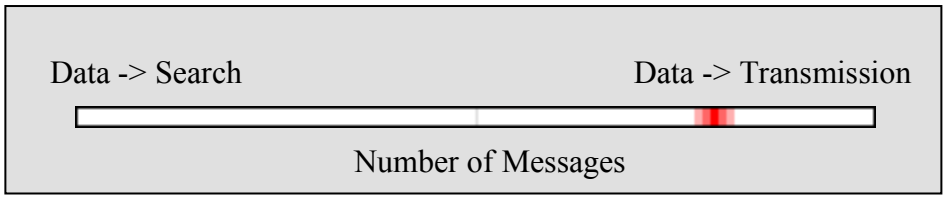

Figure 5.11 – Observed role according to Data message classifications (1)

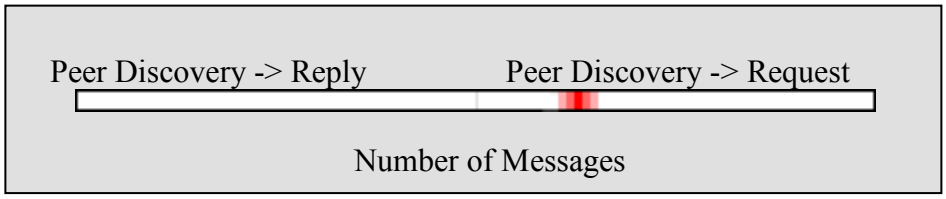

Figure 5.12 – Observed role according to Peer Discovery message classifications (2)

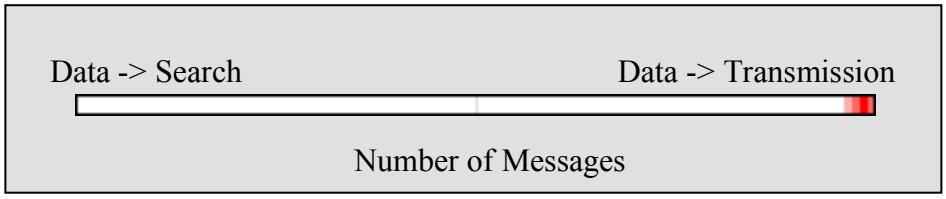

Figure 5.13 – Observed role according to Data message classifications (2)

The observed behavior of the node as shown by the number of Data messages after 20 hours of execution was significantly different from the behavior observed after only one hour of execution.

5.1.6 Step 3 of MCTAMA Test Case Execution

The final step of the testing process is to combine the roles provided by MCTAMA's analysis of the specification and message traffic. Figure 5.14 shows the test results for messages given the Peer Discovery classification. There was no significant difference between the observed values at one hour of execution and at 20 hours of execution, so the observer value at 20 hours is excluded. The point indicated by a black mark is the expected behavior of the node. The point indicated by a red mark is the observed behavior of the node. The variation (distance) between the two values is 0.6 out of a possible 100. The significance of the relationship is 1 out of 1. Despite the significance of the relationship, the expected and observed behaviors were nearly identical.

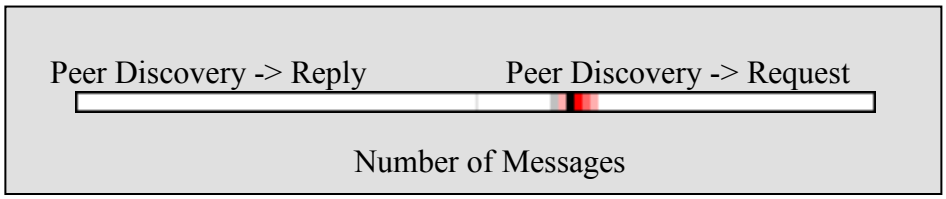

Figure 5.14 – Test result for the gtk-gnutella client

Figure 5.15 shows the test results for messages given the Data classification after 1 hour of execution. The variation (distance) between the two values is 15.77 out of a possible 100. The significance of the relationship is 1 out of 1. The high variation and significance of the relationship indicate a possibly significant test result.

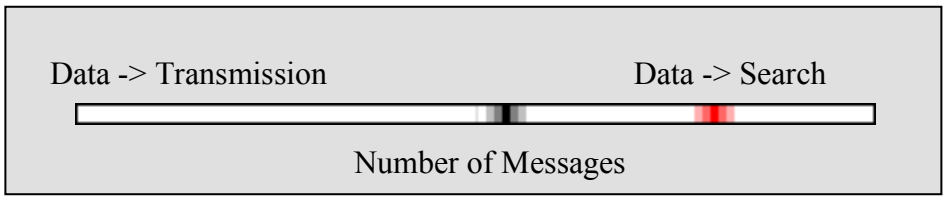

Figure 5.15 – Test result for the gtk-gnutella client at one hour

Figure 5.16 shows the test results for messages given the Data classification after 20 hours of execution. The variation (distance) between the two values is 37.08 out of a possible 100. This test result increases the significance of the test result shown in Figure 5.15. This type of variation combined with the significance of the relationship indicates a test result that warrants further investigation.

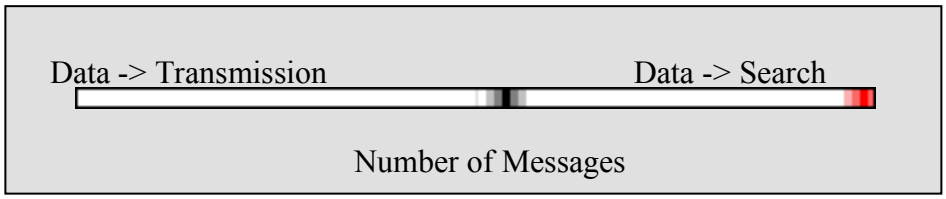

Figure 5.16 – Test result for the gtk-gnutella client at twenty hours

5.1.7 Verification of MCTAMA Test Results

The test results in Figure 5.15 and Figure 5.16 show a significant variation between the behaviors of a Gnutella node as designed in the protocol specification and the behavior of the gtk-gnutella client running as a Gnutella node. The significance of the relationship is 1. The existence of this variation was verified in published research. This experiment observed the ratio of Query messages to QueryHit message as 73:27 after one hour and 94:6 after 20 hours. Zeinalipour-Yazti and Folias observe the ratio of Query messages to QueryHit messages to be 88:12. [Zeinalipour-Yazti] Anderson observes the ratio to be 97:3. [Anderson] The reason for this variation has been determined to be the effects of the underlying Internet topology on the Gnutella network. These publications indicate that the observed inconsistency was a possibly significant lack of conformance between the specified behavior of nodes within the system and the implementation of the gtk-gnutella client. However, further investigation into this variation

showed that the behavior given in the specification was not accurate when the Gnutella node was being executed on an unstable and heterogeneous network. [Ripeanu] [zoomarchitecture] The variations shown in previous work were not the result of conformance testing the Gnutella system. These variations were found through performance analysis of Gnutella clients without the considering the design of the protocol.

5.2 MCTAMA Test of Gnutella2 Against Gnutella Specification

The second experiment of MCTAMA tests a peer-to-peer client using the Gnutella2 protocol. The Gnutella2 protocol is similar to the Gnutella protocol. Both protocols use the same basic message types and have the same network architecture. The two primary differences between Gnutella and Gnutella 2 protocols are the packet structure and search process. The similarity between the two protocols allows the test cases generated for Gnutella to be executed on a client using the Gnutella2 protocol. However, the variation in the search method used by the Gnutella2 protocol should alter the behavior of the node significantly.

The client chosen for this experiment was the Shareaza peer-to-peer client. Shareaza works across multiple protocols, but is primarily used for the Gnutella2 protocol. Shareaza was chosen due to the feature of the graphical user interface that is capable of logging the messages sent and received by the client. Figure 5.17 shows the Shareaza client being run on the Windows 7 operating system. Figure 5.18 shows the Shareaza client displaying the message logging feature of the client.

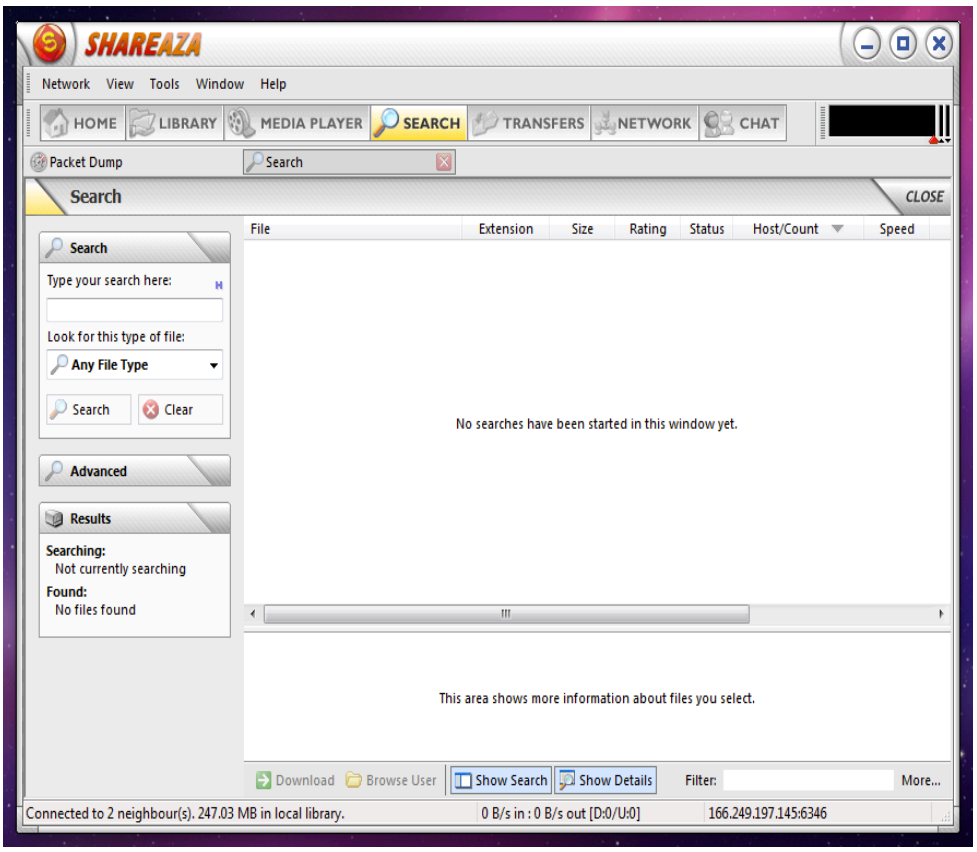

Figure 5.17 - Shareaza client in Windows 7

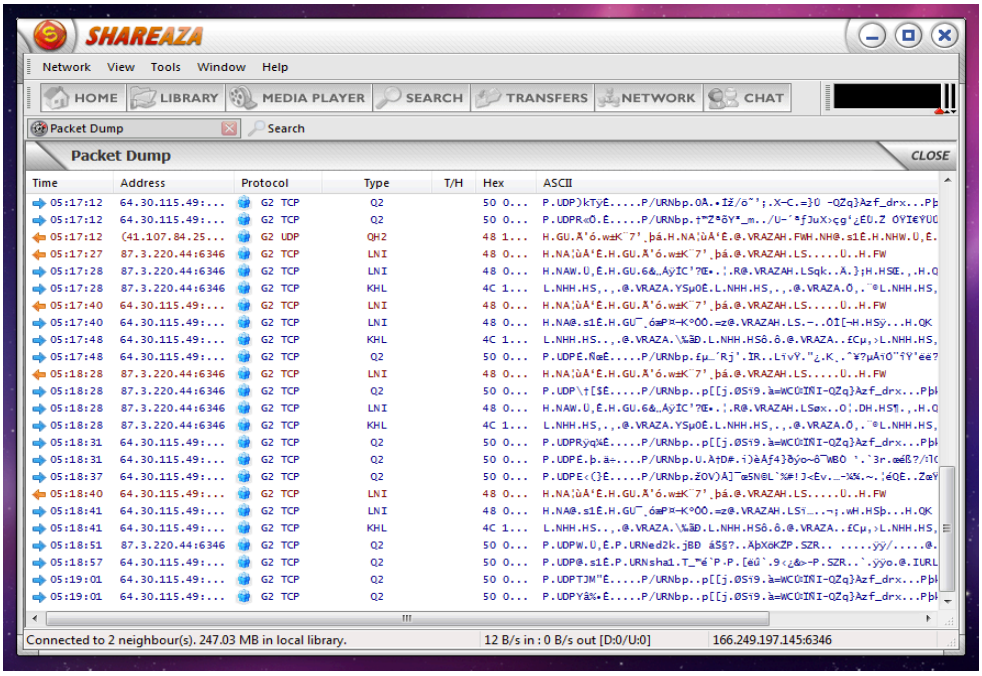

Figure 5.18 – Message logging in the Shareaza client

5.2.1 Test Case Generation

The generation of test cases is not repeated in this experiment because the test cases are generated from the Gnutella Protocol, version 0.4. This is the same specification used to generate test cases in the previous experiment. The steps for creating the test cases for the Gnutella specification are detailed in Sections 5.1.1-5.1.3. The message classifications for the four message types remain the same as the previous experiment. The Gnutella message names are Ping, Pong, Query, and QueryHit. The corresponding Gnutella2 message names are /PI, /PO, /Q2, and /QH2. The message classifications remain "Peer Discovery -> Request", "Peer Discovery -> Reply", "Data -> Search", and "Data -> Transmission" for both protocol's message types. The test cases generated in Section 5.1.3 will also be used for this experiment. These test cases are shown in Figure 5.8 and Figure 5.9. The values of SD RELATIONS also remain the same as shown in Table 5.1.

5.2.2 Test Case Execution

This experiment tests the execution of the Shareaza peer-to-peer client running the Gnutella2 protocol against the Gnutella Protocol specification version 0.4. In this experiment, the behavior of the client as determined by the Data message classifications should not conform to the specification due to the differences in the search methods used by the Gnutella and Gnutella2 protocols. However, the behavior of the node as determined by the Peer Discovery message classification should conform to the specification since the methods for peer discovery are consistent between the two protocols.

The Shareaza client was executed as a G2 client downloading a Linux distribution in the Windows 7 operating system for approximately one hour. The message types were manually recorded using the packet dump feature of the Shareaza client shown in Figure 5.18. The log file containing the messages sent and received by the node was parsed by simply counting the number of occurrences of the message identifier.

Total Messages of observed messages:	6,077	(100%)
Ping:	852	(14%)
Pong:	1,210	(20%)
Query:	2,727	(45%)
QueryHit:	1,288	(21%)

The messages were then used to create a description of the observed behavior using the common representation for node describing behavior. These descriptions were combined with the test cases shown in Figure 5.8 and Figure 5.9 to produce the test results given in Figure 5.19 and Figure 5.20.

Figure 5.19 shows the expected value of the ratio between “Peer Discovery -> Reply” and “Peer Discovery -> Request” to be 11:6, or x = 65 as given in Figure 5.8. The observed ratio is 20:14, or x = 59. The distance between the expected and observed values is 6 out of 100. The significance of the relationship is 1 out of 1.

Figure 5.20 shows the expected value of the ratio between “Data -> Transmission” and “Data -> Search” to be 8:6, or x = 57 as given in Figure 5.8. The observed ratio is 21:45, or x = 32. The distance between the expected and observed values is 25 out of 100. The significance of the relationship is 1 out of 1.

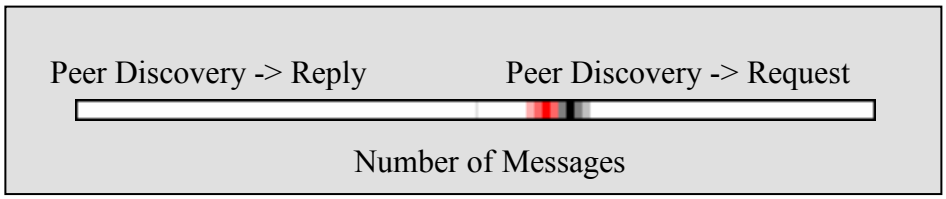

Figure 5.19 – Test result for the Shareaza client at one hour

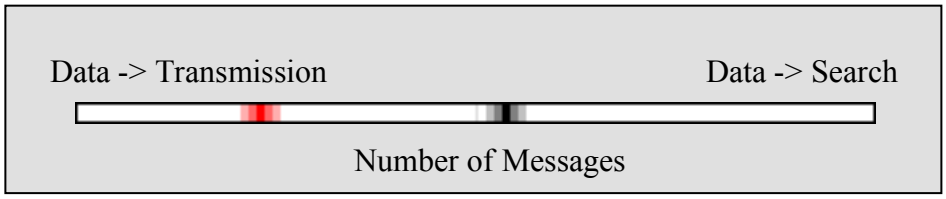

Figure 5.20 – Test result for the Shareaza client at one hour

5.2.3 Analysis of Test Results

The results of this experiment show a significant variation between the designed behavior of nodes according to "Data" message types using the Gnutella protocol specification and the behavior of nodes observed using the Gnutella2 specification.

The behavior of the node according to "Peer Discovery" message types remained consistent between the specification and the observation. The "Peer Discovery" test case was the control in this experiment since it is known that the Gnutella and G2 protocols have an identical connection process. The small variation found by MCTAMA (6 out of 100) verifies that MCTAMA generated a consistent representation of the behavior as designed and the behavior observed for a node with known consistent behavior.

The behavior of the node according to "Data" message types produced a significantly different representation of the behavior accord to design and the behavior during execution. The variation between the processes used for data search and transmission were known. The Gnutella

and G2 protocols used very different techniques to search the network for specific data. The Gnutella protocol uses a technique called “query flooding” that broadcasts “Data -> Search” messages to all peers”. The G2 protocol uses a more efficient approach to reduce the number of “Data -> Search” messages being sent. The behavior of the node according to the observed “Data -> Search” and “Data -> Transmission” messages show that the ratio of QueryHit messages to Query messages is substantially higher than expected based on the Gnutella protocol. This observation suggests that the message overhead for responses to a search query is drastically reduced. This characteristic of the Gnutella2 protocol has been established in previous work. [Tsoumakos] By observing a significant variation (25 out of 100) in a significant relationship (1 out of 1), MCTAMA successfully identified a known inconsistency between the designed and observed behavior of the system being tested. This experiment validates MCTAMA as a method for finding inconsistencies between a specification and the implementation of a peer-to-peer file distributions system.

5.3 Summary of Experimental Results

Two experiments were completed to verify and validate the MCTAMA method for conformance testing peer-to-peer file distribution systems.

The first experiment tested the gtk-Gnutella implementation against the Gnutella Protocol, version 0.4. This experiment verifies that MCTAMA creates an accurate representation of node behavior based on both the design of the system and the observed message traffic. The results of the experiment validate the method by showing that MCTAMA identified a significant inconsistency between the behaviors as design and observed during execution. This significance is verified in previous work that investigated the behavior of Gnutella nodes.

The second experiment tested MCTAMA's ability to identify significant variations between design and execution by testing the Shareaza implementation of the G2 protocol against the Gnutella Protocol, version 0.4. Two relationships were analyzed. The connection process was known to be identical in both protocols. The processes for searching the network were known to be dissimilar. The results verify that MCTAMA identified the variation in search processes and determined that the variation was significant.

CHAPTER 6 CONCLUSION AND FUTURE WORK

Testing peer-to-peer file distribution systems is challenging due to the heterogeneity of the nodes within the system, the scale of the system, and the capabilities of a monitoring mechanism. In this research, we developed a method for the conformance testing of peer-to-peer file distribution systems that addresses these challenges. The motivation for this research was the need for new techniques to improve the quality of peer-to-peer distribution systems.

The "Method for Conformance Testing by Analyzing Message Activity" (MCTAMA) developed in this research adds to the state-of-knowledge in conformance testing by providing a new method for comparing the behavior of a system as designed to the actual behavior of the system at runtime. MCTAMA applies techniques commonly related to the performance analysis of distributed systems to the behavioral analysis of individual nodes within a peer-to-peer file distribution system. The same representation techniques are used to represent the behavior specified in the design. The same method to represent behavior in the design and the executed behavior provides a new way to test the conformance of a system to its design.

As stated in Chapter 1, the goal of this research was to answer the following research questions:

1) Can the behavior of components within a distributed system be described according to both the design of the system and the observed message traffic in a way that the conformance testing process will be enhanced?
2) Can the significance of inconsistencies between the designed and observed behavior be measured in a way that is meaningful for the testing process?

Relative to the first question, MCTAMA used behavioral diagrams to describe messages at the design level and message traffic at the execution level. By using the message as the common information to be represented, the method provides a new capability to conformance test a system. Thus, MCTAMA provides a new approach to conformance testing, thereby enhancing the conformance testing process. Relative to the second question, as described in Chapter 3, the method SD RELATIONS provides a method to measure the significance of inconsistencies based on relationships between messages.

These findings were validated through experimentation. One experiment tested the conformance of the gtk-gnutella client to the Gnutella Protocol Specification. MCTAMA was able to determine that there was a significant inconsistency between the behavior as described in the protocol specification and the behavior of the gtk-gnutella client. A second experiment tested the conformance of the Shareaza peer-to-peer client running the Gnutella2 protocol to the specification of the Gnutella Protocol. MCTAMA determined that the Shareaza client had a significant inconsistency in the file search behavior. These experiments show the capability of MCTAMA to identify inconsistencies and then characterize their importance.

6.1 Contributions

The primary contributions of this research to the area of testing peer-to-peer file distribution systems are described below.

- This research developed a new method for conformance testing of peer-to-peer file systems that determines the significance between message types and characterizes the importance level of inconsistencies uncovered during the testing process.

- MCTAMA's representation for describing node behavior can describe node behavior at the design, simulation, and execution levels. By reducing the description to a single graphical indication of the relationship between message types, a tester can analyze the variation between the designed and observed behaviors.
- The new Taxonomy for Message Classification as defined in this research enables techniques for testing peer-to-peer systems to be applied to systems with system-specific message types or clients that communicate using multiple protocols.
- The method for combining sequence diagrams to describe the behavior of nodes in a system as defined in this research provides the capability to specify the behavior of a node in a single diagram.

6.2 Future Research

This research can be extended in a number of ways:

- Both the Taxonomy for Message Classification and the common representation for describing node behavior may be applicable to the interoperability of peer-to-peer systems. Additional research is needed to determine whether the Classification can extend the capabilities of middleware to systems that use similar message types to communicate and to determine how the representation for describing node behavior could aid in the determination of compatible systems.
- MCTAMA requires the behavior of nodes to be described using a specific type of sequence diagram in which the tester provides descriptions for MCTAMA to determine the designed behavior of the node. Additional study is needed to determine whether

behavioral descriptions of nodes based on a wider range of specifications would improve the method.

- MCTAMA provides a concise description of node behavior that may be useful in the process of peer clustering. In peer clustering, "[p]eers work together to …detect unacceptable behavior" [Chen]. More research is needed to determine whether the approach used in MCTAMA for describing node behavior for conformance testing can successfully determine which nodes in the network are behaving in an unacceptable manner.

REFERENCES

[Culbertson] Robert Culbertson, Chris Brown, Gary Cobb. *Rapid Testing,* Software Quality Institute Series. Prentice Hall. 2002.

[Pfleeger] Shari Lawrence Pfleeger, Joanne M. Atlee: Software engineering - theory and practice (4. ed.). Pearson Education 2009: 1-782

[Sommerville] Ian Sommerville, Software Engineering, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1989.

[Chigani] Amine Chigani and Osman Balci (2012), "The Process of Architecting for Software / System Engineering," International Journal of System of Systems Engineering, to appear.

[Khurshid] Sarfraz Khurshid and Darko Marinov. TestEra: A Novel Framework for Automated Testing of Java Programs. Invited submission to Automated Software Engineering Journal, December 2002.

[Dischinger] Dischinger, M., Mislove, A., Haeberlen, A., and Gummadi, K. P. "Detecting bittorrent blocking". In Proceedings of the 8th ACM SIGCOMM Conference on internet Measurement (Vouliagmeni, Greece, October 20 - 22, 2008). IMC '08. ACM, New York, NY, 3-8. 2008.

[Gummadi] K. P. Gummadi et al., "Measurement, Modeling, and Analysis of a Peer-to-Peer File Sharing Workload," Proc. SOSP, Bolton Landing, New York, USA, Oct. 19–22 2003.

[Lin] T Lin and H Wang. "Search Performance Analysis in Peer-to-Peer Networks", *IEEE the Third Conference on Peer-to-Peer Computing*, 2003

[Qiu] Qiu, D., and Srikant, R., "Modeling and performance analysis of bittorrent-like peer-to-peer networks". In *Proceedings of ACM Sigcomm* (Portland, OR, Aug 2004).

[Biersack] E.W.Biersack, P. Rodriguez, and P. Felber, "Performance Analysis of Peer-to-Peer Networks for File Distribution", QOFIS'04, Barcelona, Sept 04

[Alouf] S. Alouf, A. Dandoush, P. Nain, "Performance analysis of peer-to-peer storage systems". *Lecture Notes in Computer Science*, 4516:642-653, 2007 (Proceedings of ITC-20 2007, Ottawa, Canada, June 2007).

[Cacciari] Cacciari, L, and O Rafiq. "Controllability and observability in distributed testing." *Information Software Technology* 41.11/12, 1999.

[Pollet] Pollet, D., Ducasse, S., Poyet, L., Alloui, I., Cimpan, S., and Verjus, H. Towards A Process-Oriented Software Architecture Reconstruction Taxonomy. *Proceedings of the 11th European Conference on Software Maintenance and Reengineering*. Washington, DC : IEEE Computer Society, 2007.

[Ural] Hasan Ural and David Whittier. 2003. Distributed testing without encountering controllability and observability problems. *Inf. Process. Lett*. 88, 3 (November 2003), 133-141.

[Booch] Grady Booch, James Rumbaugh, Ivar Jacobson. *The Unified Modeling Language User Guide*. Upper Saddle River, NJ : Addison-Wesley, 2005.

[NIST] The Economic Impacts of Inadequate Infrastructure for Software Testing, Special Planning Report 02-03, National Institute of Standards and technology, US Department of Commerce, May 2002 http://www.nist.gov/director/prog-ofc/report02-3.pdf

[Jesdanun] Jesdanun, Anick, "GE Energy acknowledges blackout bug", The Associated Press, February 2, 2004.

[ICF] ICF Consulting, "The Economic Cost of the Blackout: An Issue Paper on the Northeastern Blackout, August 14, 2003."

[Geppert] Geppert, L., "Lost Radio Contact Leaves Pilots on Their Own", IEEE Spectrum, Volume 41, Issue 11, Page 16-17, Nov 2004.

[Vivo] Marco de Vivo, Eddy Carrasco, Germinal Isern, and Gabriela O. de Vivo. 1999. A review of port scanning techniques. SIGCOMM Comput. Commun. Rev. 29, 2 (April 1999), 41-48. DOI=10.1145/505733.505737

[Comino] S. Comino and F. Manenti, "Open Source vs Closed Source Software: Public Policies in the Software Market," ssrn.com/abstract=469741, 2004.

[Raghunathan] V. Raghunathan, C. Schurgers, S. Park, and M.B. Srivastava. Energy-aware wireless microsensor networks. IEEE Signal Processing Magazine, 19(2):40–50, 2002.

[Khoumsi] Ahmed Khoumsi. 2002. A Temporal Approach for Testing Distributed Systems. *IEEE Trans. Softw. Eng*. 28, 11 (November 2002), 1085-1103.

[Kang] Sungwon Kang, "Relating interoperability testing with conformance testing," Global Telecommunications Conference, vol. 6, pp. 3768-3773, 1998

[Bowman] Bowman, H.; Derrick, J.; , "Testing and conformance within distributed systems," Software Testing, Reliability and Quality Assurance, 1994. *Conference Proceedings., First International Conference on* , vol., no., pp.73-77, 21-22 Dec 1994

[El-Fakih] EI-Fakih, K.; Yevtushenko, N.; Bochmann, Gv.; , "FSM-based incremental conformance testing methods," *Software Engineering, IEEE Transactions on* , vol.30, no.7, pp. 425- 436, July 2004

[Ghosh] S. Ghosh and A. Mathur, "Issues in testing distributed component-based systems," in *First ICSE workshop on testing distributed component-based systems*, 1999.

[Baker] M. Baker and R. Lakhoo, "Peer-to-Peer Simulators," ACET, University of Reading, Tech. Rep., 2007.

[Naicken] S. Naicken, A. Basu, B. Livingston, and S. Rodhetbhai, "A Survey of Peer-to-Peer Network Simulators," *Proceedings of The Seventh Annual Postgraduate Symposium*, Liverpool, UK, 2006.

[He] Qi He, Mostafa H. Ammar, George F. Riley, Himanshu Raj, Richard Fujimoto: Mapping Peer Behavior to Packet-level Details: A Framework for Packet-level Simulation of Peer-to-Peer Systems. MASCOTS 2003: 71-78

[Ripeanu] Ripeanu, M. and Foster, I.T. Mapping the Gnutella Network: Macroscopic Properties of Large-Scale Peer-to-Peer Systems. In *Proceedings of IPTPS*. 2002, 85-93.

[Kaplan] Kaplan, M, Klinger, T., Paradkar, A., Sinha, A., Williams, C., and Yilmaz, C. Less is More: A minimalistic approach to UML model-based conformance test generation. In *Proceedings of the 1st International Conference on Software Testing, Verification, and Validation*, Lillehammer, Norway, April 9-11, 2008.

[Ambler] Ambler, Scott, *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*. New York: Wiley, 2002.

[Picco] GP Picco and G. Cugola. PeerWare: Core Middleware Support for Peer-To-Peer and Mobile Systems. Technical report, Dipartimento di Electronica e Informazione, Politec- nico di Milano,2001.

[Jéron] Thierry Jéron, Jean-Marc Jézéquel, Alain Le Guennec: Validation and Test Generation for Object-Oriented Distributed Software. PDSE 1998: 51-60

[Pickin] Simon Pickin, Claude Jard, Thierry Jeron, Jean-Marc Jezequel, and Yves Le Traon. 2007. Test Synthesis from UML Models of Distributed Software. *IEEE Trans. Softw. Eng*. 33, 4 (April 2007), 252-269.

[Yamany] Hany F. El Yamany, Miriam A. M. Capretz, Luiz Fernando Capretz: A Multi-Agent Framework for Testing Distributed Systems. COMPSAC (2) 2006: 151-156

[Long] Brad Long and Paul Strooper. 2001. A Case Study in Testing Distributed Systems. In *Proceedings of the Third International Symposium on Distributed Objects and Applications* (DOA '01). IEEE Computer Society, Washington, DC, USA, 20-.

[Bell] Bell, Donald, "UML basics: An introduction to the Unified Modeling Language," The Rational Edge, June 2003

[Almeida] Almeida, E.C.D., Marynowski, J.E., Sunyé, G., and Valduriez, P. PeerUnit: a framework for testing peer-to-peer systems. In *Proceedings of ASE*. 2010, 169-170.

[Fraikin] Falk Fraikin and Thomas Leonhardt. 2002. SeDiTeC " Testing Based on Sequence Diagrams. In *Proceedings of the 17th IEEE international conference on Automated software engineering* (ASE '02). IEEE Computer Society, Washington, DC, USA, 261-.

[Briand] Lionel C. Briand and Yvan Labiche. 2001. A UML-Based Approach to System Testing. In *Proceedings of the 4th International Conference on The Unified Modeling Language,*

Modeling Languages, Concepts, and Tools, Martin Gogolla and Cris Kobryn (Eds.). Springer-Verlag, London, UK, UK, 194-208.

[Ebner] M. Ebner. TTCN-3 Test Case Generation from Message Sequence Charts. In *Workshop on Integrated-reliability with Telecommunications and UML Languages* (ISSRE04:WITUL), France, November 2004.

[Kim94] Chul Kim, J. Song, Test Sequence Generation Methods for Protocol Conformance Testing, *18th Annual International Computer Software and Applications Conference*, pp. 169-174 Nov. 1994.

[Lugato] David Lugato, Céline Bigot, Yannick Valot, Jean-Pierre Gallois, Sébastien Gérard, and Francois Terrier. 2004. Validation and automatic test generation on UML models: the AGATHA approach. Int. J. Softw. Tools Technol. Transf. 5, 2 (March 2004), 124-139.

[Boy] Boy, N., Casper, J., Pacheco, C., Williams, A.,. "Automated Testing of Distributed Systems", MIT 6.824, May 2004.

[Kim99] Y.G. Kim, H.S. Hong, S.M. Cho, D.H. Bae, S.D.Cha,. Test case generation from UML state diagrams, IEEE. Software, 1999, 46(4):187-192.

[Nguyen] Cu D. Nguyen, Anna Perini, Paolo Tonella, and Fondazione Bruno Kessler. 2008. Constraint-based Evolutionary Testing of Autonomous Distributed Systems. In *Proceedings of the 2008 IEEE International Conference on Software Testing Verification and Validation Workshop* (ICSTW '08). IEEE Computer Society, Washington, DC, USA, 221-230.

[Garstecki] Lukasz Garstecki. Generation of conformance test suites for parallel and distributed languages and apis. *In Eleventh Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pages 308-315. IEEE, 2003.

[Henniger] O. Henniger, M. Lu, H. Ural. Automatic generation of test purposes for testing distributed systems. In A. Petrenko, A. Ulrich, eds*., Proc. of the 3rd Internat. Workshop on Formal Approaches to Testing of Software*, Montréal, Québec, Canada, 2003. Springer (LNCS vol. 2931)

[Köhler] Hans J. Köhler, Ulrich Nickel, Jörg Niere, and Albert Zündorf. 2000. Integrating UML diagrams for production control systems. In Proceedings of the 22nd international conference on Software engineering (ICSE '00). ACM, New York, NY, USA, 241-251.

[Riva] Claudio Riva and Jordi Vidal Rodriguez. 2002. Combining Static and Dynamic Views for Architecture Reconstruction. In *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering* (CSMR '02). IEEE Computer Society, Washington, DC, USA, 47-.

[Fox] Mark S. Fox. 1988. An organizational view of distributed systems. In *Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA 140-150.

[Ng] C. H. Ng and K. C. Sia. Peer Clustering and Firework Query Model. In *Poster Proc. of The 11th International World Wide Web Conference*, May 2002.

[DeLoach] DeLoach, S. Analysis and Design using MaSE and agentTool. *Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conferece* (MAICS). 2001.

[Lui] Siu Man Lui and Sai Ho Kwok. 2002. Interoperability of peer-to-peer file sharing protocols. *SIGecom Exch.* 3, 3 (June 2002), 25-33.

[Acharya] Arup Acharya, Xiping Wang, Charles Wright, Nilanjan Banerjee, and Bikram Sengupta. 2007. Real-time monitoring of SIP infrastructure using message classification. In *Proceedings of the 3rd annual ACM workshop on Mining network data* (MineNet '07). ACM, New York, NY, USA, 45-50.

[Tsoumakos] D. Tsoumakos and N. Roussopoulos: Analysis and Comparison of P2P Search Methods. In *proceedings of the First International Conference on Scalable Information Systems* (INFOSCALE 2006), Hong Kong, May 30 - June 1 2006.

[Kant] Krishna Kant, Ravi Iyer, and Vijay Tewari. 2002. A Framework for Classifying Peer-to-Peer Technologies. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid* (CCGRID '02). IEEE Computer Society, Washington, DC, USA, 368-.

[Aleksy] Markus Aleksy, Axel Korthaus, and Christian Seifried. 2006. Design Patterns Usage in Peer-to-Peer Systems--An Empirical Analysis. In Proceedings of the 2006 IEEE/WIC/ACM international conference on Web Intelligence and Intelligent Agent Technology (WI-IATW '06). IEEE Computer Society, Washington, DC, USA, 459-462.

[Gnutella] Gnutella Protocol Specification Version 0.4, http://rfc-gnutella.sourceforge.net/developer/stable/index.html

[Anderson] Kelsey Anderson, "Analysis of the Traffic on the Gnutella Network", University of California, San Diego, March 2001.

[Zeinalipour-Yazti] Zeinalipour-Yazti, D., Folias, T.: "A Quantitative Analysis of the Gnutella Network Traffic," University of California, Department of Computer Science, Riverside, CA, June 17, 2002.

[zoomarchitecture] www.zoomarchitecture.fr/parsons_issuu.pdf

[Chen] Alvin Chen and Richard R. Muntz. 2006. Peer clustering: a hybrid approach to distributed virtual environments. In Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games (NetGames '06). ACM, New York, NY, USA, , Article 11 .

VITA

John Wesley Burris was born and raised in Franklinton, Louisiana. Raised on a farm, John was more acquainted with gardens, sheep, and cattle than a computer. Shortly after John's father moved the outdated Tandy 1000 from his law office to his home, John became fascinated with the possibility of computers. A BASIC Interpreter and a video game programming manual created an obsession. A decade later, John earned his bachelor of science in computer science from Louisiana Tech University.

John worked at Network Foundation Technologies under Michael O'Neal for two years as a programmer. While there, he developed the client and server software for a peer-to-peer video broadcasting system. His work at Network Foundation Technologies inspired him to pursue a graduate education at Louisiana State University.

While pursuing a doctoral degree at Louisiana State University, John researched under the supervision of Doris L. Carver and worked within the Software Engineering Lab. John worked primarily as a Teaching Assistant throughout his studies. This included instructing ethics in computing. John also pursued other opportunities such as directing a high school band, technology consulting, and owning a used clothing store during his time at LSU.

John is married to Natasha Provost Burris. They have three children: Grace Nichole, Lauren Ashley, and Luke Alan. He still raises livestock and plants a garden each spring.