UNIVERSITY OF CALIFORNIA
Santa Barbara

Trustworthy Distributed Search and Retrieval over the Internet

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

by

Yung-Ting Chuang

Committee in Charge:

Professor P. Michael Melliar-Smith, Chair

Professor Louise E. Moser

Professor Timothy P. Sherwood

Professor Volkan Rodoplu

June 2013

The dissertation of
Yung-Ting Chuang is approved:

Professor Louise E. Moser

Professor Timothy P. Sherwood

Professor Volkan Rodoplu

Professor P. Michael Melliar-Smith, Committee Chairperson

May 2013

Trustworthy Distributed Search

and Retrieval over the Internet

Yung-Ting Chuang

To my parents, sister, and brother:

Shu-Hao Chuang, Fang-Mei Tai, Huan-Ching Chuang, and

Po-Yao Chuang

Acknowledgements

When I was a Master's student in Fall 2007, I took ECE254A from Professor Melliar-Smith. In that course, Professor Melliar-Smith believed that I had research potential, and he encouraged me to pursue the Ph.D. degree. I am now glad that I made the decision at that time to continue my studies for the Ph.D. degree.

First of all, I would like to express my gratitude to my advisors, Professor Melliar-Smith and Professor Moser, for their patience, support, encouragement, guidance and help throughout my M.S. and Ph.D. studies. This dissertation would not be possible without their advice and guidance. When I was having difficulty with my research, I initiated meetings to discuss my problems with them. These meetings cleared up my confusion, and provided new ideas and new directions for my research. Professor Moser and Professor Melliar-Smith taught me how to explore new ideas from existing papers, how to create new ideas that would not occur to most people, how to perform experiments and solve problems, and finally how to write conference and journal papers. As a foreign student, writing papers in English is a particular challenge for me. I really appreciate their help in improving my writing and presentation skills, and their great efforts to transform me from a student to an independent researcher.

In addition, I want to thank Professor Sherwood for inspiring me to pursue Ph.D. research. When I took his CS 254 class in Spring 2008, I was one of

the two students who had achieved the best performance, with the fewest errors, in the branch predictor project. I felt so much satisfaction; that was the moment I realized my enthusiasm for being a researcher. I am grateful to have Professor Sherwood serve on my Ph.D. Dissertation Committee because of his helpful comments and suggestions to improve my research work.

Moreover, I want to thank Professor Rodoplu for inspiring me to become an effective instructor. When I took his ECE152A class in Fall 2004, I was motivated by the way in which he taught the class. He is a very clear instructor and prepares all the class materials in a very organized way. His class is one of the few classes in which I understood the concepts immediately without preparation beforehand. ECE152A is a fun and challenging course, and I enjoyed attending every one of his classes. Furthermore, I am grateful to have Professor Rodoplu serve on my Ph.D. Dissertation Committee and for very helpful discussions.

I especially want to thank Isai Michel Lombera for being such a great and very helpful research partner. Isai has a lot of practical experience and is willing to help me on a regular basis. Whenever I got stuck on any technical-related problems (such as the PHP scripting language, Apache Web Server, Unix/Linux commands, cURL/libCURL scripts, PHP PECH, and SQLite), I immediately wrote him an email to seek his help. He always replied to my email by first introducing some background related to these problems, and then following up with some suggestions. Definitely, my research would not

have been so successful without his help. I enjoyed our collaboration and hope to have more collaboration opportunities in the future.

I also want to thank the other collaborators in our iTrust research group. The group meetings and discussions were very inspiring and helpful. I truly acquired lots of new ideas and thoughts from these meetings. I thank Christopher Badger, who created and developed the de-clustering algorithm for iTrust; Andre Abrahamian, who helped with the iTrust development using SMS; Wei Dai, who developed the local reputation system for mobile ad-hoc networks; Balasubramaniam Muthuvelu, who implemented authentication using asymmetric encryption for iTrust; and Jerry Peng, who implemented the ranking algorithm for iTrust. I especially want to thank Christopher Badger, who created a simulation program which I have used to generate simulation results quickly for my performance evaluations and Jerry Peng, who set up and created simulation scripts for Emulab which enabled me to compare my analytical results against the results obtained from Emulab.

I also want to thank all the faculty and staff members of the UCSB ECE and CS Departments, for all the wonderful undergraduate and graduate courses. In addition, I want to thank Wendy Ibsen for giving me a great opportunity to teach in the School for Scientific Thought program, and to Val De Veyra for all her help.

I also want to thank Mr. Shi-Kuo Li, for teaching me how to live and eat in a healthy way. My research would not have been so productive without adopting those healthy habits.

Moreover, I want to thank my friends for making my life more interesting and more enjoyable. In particular, I thank all of my friends in Taiwan, Ina Chou, Yi-Ting Chen, Chiu-Ge Ho, Feng-Chih Chen, Mei-Hui Hu, Maggie Wu, Stacy Cheng, Chih-Han Chan, Sophia Cheng, Vanessa Hung, Diana Lai, A-Boa Wang, Yu Wen, Guan Guan, Mini Lin, Jennifer Yu, Gui-Lin Ni, Lo Cheng, Jennifer Chou, and Sakana Hsieh, for sharing their lives with me. Moreover, I thank Esra Kucukoguz, Mamie Lih, and Szu-Pei Lu, for sharing their research lives with me. In particular, I want to thank Pei-Chen Chang for all of her support and help. She has been a very close friend since 2007, and was the one who helped me to discover that I had the potential to pursue Ph.D. studies. She was always there to share her research experiences with me and to listen to me whenever I needed her. I am thankful for all of her suggestions and comments. I would not have achieved so much without her encouragement.

In addition, I want to thank my uncle and aunt, Young-Chang Lu and Hsiu-Ling Lu, for letting me live with them for the last year of my Ph.D. studies. They truly gave me a comfortable environment to enable me to concentrate on my research. In addition, I want to thank them for treating me like their daughter, and giving me their support and help whenever I needed it.

Last, but not least, I want to thank my father Shu-Hao Chuang, my mother Fang-Mei Tai, my sister Huan-Ching Chuang, and my brother Po-Yao Chuang. I especially want to thank my father because without his encouragement, I would not dare to try this Ph.D. study. Moreover, I want to thank my beloved and great sister, who is always there to accompany me and share her life during my research in the United States.

Curriculum Vitæ

Yung-Ting Chuang

Education

2013	Doctor of Philosophy, Electrical and Computer Engineering University of California, Santa Barbara, CA
2008	Master of Science, Electrical and Computer Engineering University of California, Santa Barbara, CA
2006	Bachelor of Science, Computer Engineering University of California, Santa Barbara, CA

Research Experience

Fields of Study	Distributed systems, computer networks, statistical inference performance evaluation, trust and security, e-healthcare
2012 – Present	Graduate Research Assistant Supervised by Professor Melliar-Smith and Professor Moser University of California, Santa Barbara, CA Investigated and developed an adaptive membership protocol for iTrust, which uses a few extra messages beyond those used in the iTrust messaging protocol, but allows each member to keep its local view of membership close to the actual membership, even with a high rate of churn.
2011 – 2012	Graduate Research Assistant Supervised by Professor Melliar-Smith and Professor Moser University of California, Santa Barbara, CA Collected and evaluated data using the Emulab network testbed. Investigated and developed novel statistical algorithms for detecting and defending against malicious attacks in the iTrust search and retrieval network. Developed the iTrust infrastructure and user interface for membership maintenance.
2010 – 2011	Graduate Research Assistant Supervised by Professor Melliar-Smith and Professor Moser University of California, Santa Barbara, CA Developed the iTrust information search and retrieval infrastructure for the Internet, with no centralized search engine and no centralized administration. Created the iTrust infrastructure and user interface for creation of requests, distribution of metadata, storing metadata and documents, matching keywords in requests against metadata, and providing responses to requests. Analyzed matching probabilities in the presence of unavailable

	nodes and nodes that appear to be malicious with random distribution of the metadata and the requests. Investigated distribution of the metadata and the requests to different numbers of nodes and adaptively adjusted the number of nodes to which to distribute metadata and requests.
2009 – 2010	Graduate Research Assistant Supervised by Professor Melliar-Smith and Professor Moser University of California, Santa Barbara, CA Developed a patient-centered health monitoring system that interfaces health monitoring devices to a laptop computer via Bluetooth wireless communication to promote patient-centered healthcare and self-management.
Jan-Jun 2006	Undergraduate Research Assistant Supervised by Professor Manjunath University of California, Santa Barbara Bioimage Informatics Research Project: Built software tools to track and organize biological images and metadata, including decoding image filenames, matching images to metadata files, and ensuring consistency between the images and metadata files.

Professional Memberships

2002-2006	National Society of Collegiate Scholars
2009-Present	IEEE

Work Experience

2006-2010	Database Administrator and Web Developer Santa Barbara Mailworks, Santa Barbara, CA Developed Web pages that track the status of work orders using PHP and MySQL. Developed a Web-interface for a time-clock machine using PHP and MySQL.
2003-2006	Web Developer Office of International Scholars and Students Department, UC Santa Barbara Developed and maintained Web pages using Dreamweaver and HTML.
2003-2006	Database Developer Chinese Department of Davidson Library, UC Santa Barbara Maintained Web pages using HTML. Organized Chinese journals and books in the database.
1997-1998	Opera Accompanist Music Department, University of South Dakota

Teaching Experience

Sep-Nov 2012	Instructor at UCSB School of Scientific Thought
Spring 2012	Teaching Assistant, ECE 155B (Network Computing) University of California, Santa Barbara
Spring 2011	Teaching Assistant, ECE 155B (Network Computing) University of California, Santa Barbara
2009-2010	Instructor, Computer Architecture Far-East University
2009-2010	Instructor, Multimedia Production Far-East University
2009-2010	Instructor, Introduction to Security Chung-Hwa University
2004-2006	Chinese Teacher
1997-1998	Piano Teacher

Publications

I. Michel Lombera, L. E. Moser, P. M. Melliar-Smith, Y. T. Chuang, "Trustworthy mobile ad-hoc wireless networks over Wi-Fi Direct," in preparation.

Y. T. Chuang, P. M. Melliar-Smith, L. E. Moser, I. Michel Lombera, "Detection and defensive adaption algorithms for protecting against malicious attacks in the iTrust information retrieval network," in preparation.

Y. T. Chuang, P. M. Melliar-Smith, L. E. Moser, I. Michel Lombera, "An adaptive membership protocol for high churn networks in the presence of malicious attacks," submitted.

Y. T. Chuang, P. M. Melliar-Smith, L. E. Moser, I. Michel Lombera, "Membership management for the iTrust information retrieval network," submitted.

ICWMC13 I. Michel Lombera, L. E. Moser, P. M. Melliar-Smith, Y. T. Chuang, "Mobile ad-hoc search and retrieval in the iTrust over Wi-Fi Direct network," Proceedings of the Ninth International Conference on Wireless and Mobile Communications, Nice, France, July 2013, to appear.

ICWN13 W. Dai, L. E. Moser, P. M. Melliar-Smith, I. Michel Lombera, Y. T. Chuang, "The iTrust local reputation system for mobile ad-hoc networks," Proceedings of the 2013 International Conference on Wireless Networks, Las Vegas, NV, July 2013, to appear.

WPMC13 I. Michel Lombera, L. E. Moser, P. M. Melliar-Smith, Y. T. Chuang, "Peer Management for iTrust over Wi-Fi Direct," Proceedings of the 2013 Wireless Personal Multimedia Communi-

cations Symposium, Atlantic City, New Jersey, June 2013, to appear.

WEBIST13 B. Peng, L. E. Moser, P. M. Melliar-Smith, Y.T. Chuang, I. Michel Lombera, "A distributed ranking algorithm for the iTrust information search and retrievals system," Proceedings of the 9th International Conference on Web Information Systems and Technologies, Aachen, Germany, May 2013, pp. 199-208.

IMECS13 Y. T. Chuang, P. M. Melliar-Smith, L. E. Moser, I. Michel Lombera, "Discovering joining nodes and detecting leaving nodes in the iTrust membership protocol," Proceedings of the International Multi-Conference on Engineers and Computer Scientists, Hong Kong, China, March 2013, pp. 189-194.

ACM MONET I. Michel Lombera, L. E. Moser, P. M. Melliar-Smith, Y. T. Chuang, "Mobile decentralized search and retrieval using SMS and HTTP," ACM Mobile Networks and Applications Journal, vol. 18, issue. 1, February 2013, pp. 22-41.

WiCOM12 I. Michel Lombera, L. E. Moser, P. M. Melliar-Smith, Y. T. Chuang, "Decentralized search and retrieval for mobile networks using SMS," Proceedings of the IEEE 8th International Conference on Wireless and Mobile Computing Networking and Communications, Barcelona, Spain, October 2012, pp. 134-141.

JCSE Y. T. Chuang, I. Michel Lombera, P. M. Melliar-Smith, L. E. Moser, "Protecting the iTrust information retrieval network against malicious attack," Journal of Computing Science and Engineering, vol. 6, no. 3, September 2012, pp. 179-192.

MS12 I. Michel Lombera, L. E. Moser, P. M. Melliar-Smith, Y. T. Chuang, "A mobile peer-to-peer search and retrieval service for social networks," Proceedings of the IEEE 1st International Conference on Mobile Services, Honolulu, HI, June 2012, pp. 72-79.

WEBIST12 C. M. Badger, L. E. Moser, P. M. Melliar-Smith, I. Michel Lombera, Y. T. Chuang, "Declustering the iTrust search and retrieval network to increase trustworthiness," Proceedings of the 8th International Conference on Web Information Systems and Technologies, Porto, Portugal, April 2012, pp. 312-322.

ICOIN12 Y. T. Chuang, I. Michel Lombera, P. M. Melliar-Smith, L. E. Moser, "Detecting and defending against malicious attacks in the iTrust information retrieval network," Proceedings of the 26th International Conference on Information Networking, Bali, Indonesia, February 2012, pp. 263-268.

ICDCN12 P. M. Melliar-Smith, L. E. Moser, I. Michel Lombera, Y. T. Chuang, "iTrust: Trustworthy information publication, search and retrieval," Proceedings of the 13th International Conference on Distributed Computing and Networking, Hong Kong, China, January 2012, LNCS 7129, Springer, pp. 351-366.

MobiCASE11 I. Michel Lombera, Y. T. Chuang, L. E. Moser, P. M. Melliar-Smith, "Decentralized mobile search and retrieval using SMS and HTTP to support social change," Proceedings of the 3rd International Conference on Mobile Computing, Applications, and Services, Los Angeles, CA, October 2011, pp. 152-171.

ICOMP11 Y. T. Chuang, I. Michel Lombera, L. E. Moser, P. M. Melliar-Smith, "Trustworthy distributed search and retrieval over the Internet," Proceedings of the 2011 International Conference on Internet Computing, Las Vegas, NV, July 2011, pp. 169-175.

INTERNET11 I. Michel Lombera, Y. T. Chuang, P. M. Melliar-Smith, L. E. Moser, "Trustworthy distribution and retrieval of information over HTTP and the Internet," Proceedings of the Third International Conference on the Evolving Internet, Luxembourg City, Luxembourg, June 2011, pp. 7-13.

WH10 Y. T. Chuang, P. M. Melliar-Smith, L. E. Moser, "A patient-centered health monitoring system", Proceedings of the Wireless Health 2010, San Diego, CA, October 2010.

Teaching Certificates

2013	Certificate in College and University Teaching University of California, Santa Barbara
2010	National Teacher Certificate Education Ministry, Taiwan
1994	Yamaha Piano Teaching Certificate (Level 5) Kaohsiung, Taiwan

Languages

2013	Fluent in Chinese and English

Activities

2002-2008	Orchestra Member (violin) University of California, Santa Barbara
Apr 1999	Passed Entrance Exam National Taichung Second High School
Mar 1999	Passed Entrance Exam Music Department of National Tainan College of Arts
1998-1999	Special Scholarship Student University of South Dakota
1998-1999	Orchestra Concertmaster(Violin) University of South Dakota(Sept 1998-June 1999)

Honors and Awards

Spring 2013	PhD Dissertation Fellowship ECE Department, University of California, Santa Barbara
Mar 2013	Best Paper Award The IAENG International Conference on Computer Science
Feb 2012	Best Paper Award 26th International Conference on Information Networking
2002-2006	Dean's Honor List University of California, Santa Barbara
2002-2006	Engineering Honors Program University of California, Santa Barbara
May 2006	3rd Prize Winner in Piano Performing Arts Scholarship Foundation Competition
May 2006	Scholarship for $500 Prize in Piano Santa Barbara Music Club Competition
Oct 2006	Scholarship for $3000 Prize Rudi Schulte Family Foundation International Award
May 2001	Premio De Oro Awards in Mathematics Cerritos High School, California
May 2001	Premio De Oro Awards in History Cerritos High School, California
Jul 2001	4th Prize in Piano Victor Piano Competition, Adult Group
Dec 1999	3rd Prize in Piano Vienna Piano Competition, Young Person's Group
Dec 1999	2nd Prize in Piano Government Middle Taiwan Piano Competition
Feb 1999	Winner Award in Piano Sioux City Competition
Jul 1997	1st Prize in Piano Victor Piano Competition, Young Person's Group

Abstract

Trustworthy Distributed Search and Retrieval over the Internet

Yung-Ting Chuang

Currently, our trust in the accessibility of information over the Internet and the Web depends on benign and unbiased administration of centralized search engines. Unfortunately, a centralized search engine can be modified to filter, conceal, or censor information. We present a decentralized search and retrieval system, named iTrust, to address these problems.

First, we address the design and implementation of iTrust. Source nodes distribute metadata for information along with the address of the information to randomly chosen nodes. Similarly, requesting nodes distribute requests that contain keywords to randomly chosen nodes. Nodes that receive a request compare the keywords with the metadata they hold, and return the URL of the information to the requesting node if they have a match. For appropriate values of the parameters, the probability of a match is very high.

Next, we present statistical algorithms for detecting and defending against malicious attacks. Our statistical detection algorithm compares the empirical and analytical probabilities of a match to estimate the proportion of non-operational nodes in the membership. The defensive adaptation algorithm then increases the number of nodes to which the requests are distributed to

maintain the same probability of a match when some of the nodes are non-operational as when all of the nodes are operational.

Then, we consider an environment in which nodes join or leave the membership rapidly. Our adaptive membership protocol allows nodes to discover changes in the membership using the iTrust messages and responses that it normally receives. A node then dynamically adjusts its requesting rate, and sends more request messages to compensate for nodes from which it did not receive a response. The protocol exploits messages already being sent to distribute metadata or requests, thus reducing the need for additional messages.

Finally, we combine our adaptive membership protocol with our algorithms for detecting and defending against malicious nodes. The combined adaptation algorithm estimates changes in the membership to improve the behavior of iTrust. We demonstrate that the combined adaptive algorithm is effective and robust when the membership churn is high and when there are malicious nodes in the membership.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Our modern world relies heavily on the ability to publish, search for, and retrieve information over the Internet, which has created a highly distributed information society, distributed in both the sources of information and the uses of information. For reasons of efficiency and scalability, conventional search and retrieval over the Internet employs centralized search engines. In the Internet today, the administrators of those centralized facilities are benign.

Unfortunately, the experience of history, and even the common wisdom today, appear to indicate that we cannot depend on such administrators to remain benign and unbiased in the future. The incident of the Beacon advertising mechanism on Facebook [78] demonstrated how easily control over a centralized mechanism can be subverted. Perhaps the moment at which we are most dependent on our abilities to communicate over the Internet is also the moment at which centralized Internet search is most likely to be compromised.

One of the problems with a fully distributed system is that it is very difficult to infer information that cannot be measured directly. An example is

estimating the proportion of malicious nodes in the network. Another example is measuring the size of the network when the network has a lot of churn. The reason is that in a fully distributed system, nodes are joining and leaving the network; thus, no single node has a complete and up-to-date view of the network.

This Ph.D. Dissertation addresses the distribution and retrieval of information, without centralized search engines and without centralized search indices. It provides detection and defensive adaptation algorithms to protect against malicious nodes in the network, adaptive membership management protocols when the membership has a lot of churn, and statistical inference algorithms to manage the network when both the membership churn and the proportion of malicious nodes are high. The specific contributions of this Ph.D. Dissertation are presented in Section 1.2, and are described in detail in subsequent chapters.

1.1 Overview of iTrust

The iTrust system [2, 18, 19, 60, 61] is a decentralized and distributed information publication, search, and retrieval system that makes it difficult to censor or filter information accessed over the Internet. iTrust operates over an unstructured peer-to-peer network, and uses a probabilistic approach.

Some nodes (the source nodes) produce information, and make that information available to other nodes chosen at random. The source nodes also produce metadata that describes their information, and distribute that metadata to a subset of nodes chosen at random. The metadata are distinct from the information that they describe, and include a list of keywords and the URL of the source of the information.

Other nodes (the requesting nodes) request and retrieve information. Such nodes generate requests that contain keywords, and distribute the requests to a subset of the nodes chosen at random. The other nodes compare the keywords in the requests with the metadata that they hold. If such a node finds a match, which we call an *encounter*, the matching node returns the URL of the associated information to the requesting node. The requesting node then uses the URL to retrieve the information from the source node.

1.2 Contributions

In this section, we discuss the problems that we have addressed in our research and the main contributions of this Ph.D. Dissertation.

1.2.1 Trustworthy Distributed Search and Retrieval

Our trust in the accessibility of information over the Internet and the Web (hereafter referred to as the Internet) depends on centralized search engines

and centralized search indices to increase scalability and efficiency. In the Internet today, the administrators of those centralized facilities are benign. However, the experience of history shows that the centralized Internet search engines can be tampered with to bias the results and to conceal or censor information.

To prevent censorship and filtering of information accessed over the Internet, we present the iTrust system for a trustworthy publication, distribution and retrieval of information over the Internet, with no centralized mechanisms and no centralized control. iTrust is particularly valuable for individuals who fear that the conventional Internet search mechanisms might be subverted. Therefore, the very existence of iTrust helps to deter censorship attempts.

1.2.2 Protecting against Malicious Attacks in iTrust

Effective use of iTrust depends on information about the network that might be difficult to obtain, such as information about uncooperative or malicious nodes that might deliberately attempt to conceal information about sensitive topics by failing to report matches for requests related to those topics. The absence of centralized control makes it difficult to determine such information. Other peer-to-peer and dynamic networks [33, 56, 75, 80] face similar problems.

When a requesting node makes a request, it often results in multiple responses, which most people would consider to be a waste. Our observations

and contributions are that these multiple responses do not constitute waste but rather, they provide valuable information about the network that is difficult to obtain directly. In particular, the multiple responses allow a node to estimate the proportion of non-operational or subverted nodes.

Based on the responses to a request, we present novel statistical inference algorithms for detecting and defending against malicious attacks in the iTrust network. The detection algorithm detects changes in the proportion of operational nodes, and the defensive adaptation algorithm changes the number of nodes to which the requests are distributed, in order to maintain a high match probability. Experimental results substantiate the effectiveness of the detection and defensive adaptation algorithms for protecting against malicious attacks.

The novelty of our work is that we associate the absence of action with the intent to censor sensitive information, when absence of action is a permissible behavior if a node does not hold metadata for that information. We use a statistical method, together with an analytical model, to infer characteristics of the network that are unmeasurable directly. In iTrust, the nodes infer useful but unobservable information about the network by observing aspects of the behaviors of the nodes that are visible to them. More specifically, the nodes infer such information by monitoring the responses to their requests. As far as we are aware, iTrust is the first system that infers that nodes are not responding in order to conceal sensitive information.

1.2.3 Membership Management for iTrust

Joining and leaving the membership, or membership churn, is an inherent aspect of peer-to-peer networks and, is critical for designs and evaluations of the membership algorithms. Accurately estimating the joining and leaving of nodes is a challenging task, due to the large size and highly dynamic nature of peer-to-peer networks.

Prior work [12, 34, 51, 42] on membership has focused on an agreed accurate membership in the presence of unreliable processors and communication. Our membership algorithms are simpler and less costly than such prior work because they do not attempt to achieve an agreed accurate membership based on a consensus algorithm, which is known to be impossible [9]. Our membership algorithms for iTrust allow each member to have its own local view of the membership, but aim to keep that local view close to the actual membership.

In our membership algorithms, a requesting node discovers newly joining nodes and leaving (non-operational) nodes from the responses it receives to its requests. Thus, they exploit messages already required by the iTrust messaging protocol for search and retrieval, rather than additional messages used only for membership.

Lastly, each member estimates the churn in the membership by random sampling, and then adaptively adjusts its requesting rate to update its local view of the membership. Using a churn estimator, a node dynamically adjusts

its requesting rate or sends a few more request messages. Hence, our membership protocols work well when the membership has a lot of churn and when the membership is stable. Moreover, for appropriate values of the parameters, the membership accuracy, response time, and message cost are reasonable, and the match probability is high.

1.2.4 Statistical Inference and Dynamic Adaptation for iTrust

One of the problems with fully distributed system is the difficulty of inferring information that cannot be measured directly. An example of such information in iTrust is the proportion of malicious nodes in the membership. Another example is the size of the membership because in a fully distributed system in which many nodes are joining and leaving the membership, no single node has a complete and up-to-date view of the membership.

To address these problems, we provide statistical inference algorithms for estimating the proportion of malicious nodes and the size of the membership. These algorithms are sufficiently accurate and timely to allow them to be used to manage iTrust in the presence of both membership churn and malicious nodes. In particular, we present a Dynamic Adaptive Algorithm for iTrust that not only maintains a node's local view of the membership but also protects iTrust against malicious attacks. Our algorithm uses random sampling

and statistical inference to estimate metrics that are not directly observable. Experimental results demonstrate that the algorithm works well when the membership has a lot of churn, when the membership is stable, and when the membership has a lot of malicious nodes.

1.3 Organization of This Dissertation

The rest of this dissertation is organized as follows:

In Chapter 2, we discuss related work in terms of distributed search and information retrieval, malicious attacks, membership management, and membership churn.

In Chapter 3, we describe a novel information distribution and retrieval system, named iTrust, that operates over HTTP and the Internet and that provides trustworthy access to information [19, 60, 61]. First, we present an overview of the iTrust strategy, the implementation of the iTrust system, and the user interface. Then, we present a performance evaluation of the probability of a match based on both analysis and on emulation of iTrust.

In Chapter 4, we present novel statistical algorithms for detecting and defending against malicious attacks in the iTrust information retrieval network [13, 17, 18]. These algorithms employ the exponential weighted

moving average method to collect empirical data, and the chi-squared test to estimate the proportion of operational nodes.

In Chapter 5, we present adaptive and non-adaptive membership management protocols for the iTrust information retrieval network [14, 16]. We compare the performance of the non-adaptive and adaptive membership protocols, with respect to membership accuracy, match probability, response time, and message cost, for various values of the parameters when the membership churn is high and when the membership is stable.

In Chapter 6, we present statistical inference algorithms [15] to estimate the proportion of malicious nodes and the size of the membership, and demonstrate that these algorithms are sufficiently accurate and timely to allow them to be used to manage iTrust in the presence of membership churn and malicious nodes.

In Chapter 7, we conclude the dissertation with a summary of our contributions, and a discussion of directions for future research.

Chapter 2

Related Work

In this chapter, we discuss related work in the area of distributed search, malicious attacks, trustworthiness, membership management, and membership churn. First, in Section 2.1, we provide work related to distributed search in peer-to-peer networks. Distributed search strategies are categorized as either structured or unstructured, and we discuss related work to both categories. In addition, we provide related work on randomization and the square root functions.

Next, in Section 2.2, we present existing work related to trust. We begin the section by discussing social networks where the primary concern is trustworthiness. We then present work related to detection and defense against attacks to improve trustworthiness. Then, we discuss work related to the exponential weighted moving average and chi-squared statistic, which is widely used in intrusion and anomaly detection.

In Section 2.3, we discuss work related to membership management and membership churn. First, we introduce peer-to-peer membership protocols

based on an agreed accurate membership. We then present membership protocols based on gossiping and neighborhood selections biased toward beneficial nodes. Lastly, we discuss related work that considers membership churn, leaves and crashes. In addition, we present related work that focuses on estimating churn rate and adapting accordingly.

2.1 Peer-to-Peer Networks

Today, the centralized search engine strategy (such as that of Google [38]) is used commercially for Internet search, where metadata for the information are held in a centralized index [7]. Requests are submitted to the central site, where they are matched against the metadata. Matches are returned to the requesting node, which retrieves the information from the source node. The centralized search engine strategy is efficient and scalable, but it is vulnerable to manipulation by administrators.

In the publish/subscribe model [29], a query is submitted and stored, and the query subscriber is notified when the results become available. The centralized publish/subscribe approach uses a centralized index [11, 30, 46], where all of the information and the queries are published. The approach is neither efficient nor scalable for large networks, not to mention the issues of trust of the centralized site.

Bender *et al.* [4] recognize the need for decentralized peer-to-peer Web search because "existing Web search is more or less exclusively under the control of centralized search engines." Mischke and Stiller [64] provide a taxonomy of distributed search mechanisms in peer-to-peer networks. Another useful survey of such mechanisms is provided by Risson and Moors [71]. The distributed publish/subscribe approach can be categorized as either structure-based or gossip-based.

The structured approach [8, 44, 79, 81] requires the nodes to be organized in an overlay structure, based on Distributed Hash Tables (DHT) [70, 73, 74, 77, 95], skip lists [41, 69], rings [25, 59, 86] or trees [5]. The structured approach is more efficient than the gossip-based approach, but it involves additional overhead for constructing and maintaining the overlay network. Moreover, churn or malicious disruptions can break the structure.

The unstructured approach [20, 33, 49, 79, 96, 36, 37, 67, 80, 82, 88] is typically based on gossiping, uses randomization, and requires the subscriber nodes and the publisher nodes to find each other by exchanging messages over existing links. iTrust falls within the unstructured distributed search category.

Cohen and Shenker [22] have studied how replication can be used to improve search in unstructured peer-to-peer networks. They show that square root replication is theoretically optimal in terms of minimizing the overall search traffic. They replicate objects based on access frequencies (popularity-

ties), whereas iTrust uses uniform random replication of objects, so that popular nodes are not more vulnerable to attacks.

Gnutella [37] was developed as an information (music) distribution network, following the collapse of Napster [66]. Gnutella uses flooding of requests to find information, with each node redistributing requests up to a maximum depth or time-to-live. When a request reaches a node holding the information that matches a request, the node transmits the information to the requester. If the query rate is high, nodes quickly become overloaded and the system ceases to function satisfactorily. Extensions of Gnutella involve supernodes [89], which improve efficiency but reintroduce some of the trust risks of centralized strategies. An interesting variation [96] involves random walks for requests in which the number of nodes visited by a request is proportional to the square root of the request popularity of the information, as in [22]. Another interesting variation [24] exploits search trees whose node degrees approximate the square root of the size of the network. Like these researchers, we also exploit the square root function in iTrust.

Freenet [20, 67] is more sophisticated and efficient than Gnutella, because it learns from previous requests, both in directing subsequent requests to nodes where earlier similar requests were successful, and in pre-positioning metadata or information at nodes that have requested that information. Similar strategies can be found in [21, 28, 92, 94]. Freenet has two characteristics that make it unsuitable for trustworthy information distribution and retrieval

in large networks, namely: (1) Learning takes time and the first few requests for information are handled very inefficiently. (2) Nodes that successfully respond to requests receive more metadata and more requests. Thus, it is easy for a group of untrustworthy nodes to conspire together to gather most of the searches into their group, making Freenet vulnerable to subversion. The Adaptive Probabilistic Search (APS) method [83] also uses feedback from previous searches to direct future searches, instead of distributing the requests to randomly selected nodes, like iTrust does.

Ferreira *et al.* [1, 33] use a random-walk strategy in unstructured networks to replicate both queries and data to the square root of the number of nodes in the network. Their strategy uses random walks for replication, with biased probabilities to reduce the effects of heterogeneity. Unlike [22], in their system, replication of metadata and requests is independent of access frequency (popularity), as in iTrust. Like their system, iTrust exploits the square root function to replicate the metadata and the requests.

GIA [10] is an unstructured system that combines biased random walks with one-hop data replication. It uses heuristics to place most nodes within short reach of high capacity nodes. Lv *et al.* [56] start with uniform random replication of data, using square root replication to improve the performance, but then adaptively adjust the replication degree based on the query rate by creating and deleting replicas of the data. They also use random walks

to improve on the flooding of Gnutella. Like those systems, iTrust exploits randomization and replication.

Galanx [87] uses a local peer index to direct user queries to relevant nodes in the network. It is based on the Apache Web server and on the BerkeleyDB data store. iTrust likewise utilizes the Apache Web server, and maintains a local index of metadata and corresponding URLs for the data.

Pub-2-Sub [82] is a content-based publish/subscribe service for unstructured peer-to-peer networks of cooperative nodes. Instead of gossiping, Pub-2-Sub uses directed routing to distribute subscription and publication messages to the nodes. Their objective is efficiency, which is not the primary concern of iTrust.

Systems based on social networks [43, 58] exploit the trust that members have in other members, and route information and requests based on relationships between members. Existing social networks, like Facebook [31], are typically centrally administered and depend on benign administrators. Gummadi *et al.* [43] investigate the integration of social network search with Web search. They conclude that such integration can lead to more timely and efficient search experiences. Yang *et al.* [90] propose a search mechanism for unstructured peer-to-peer networks based on interest groups, formed by nodes with similar interests. iTrust likewise allows users interested in a particular topic or cause to form a social network, so that they can share information.

2.2 Trustworthiness

None of the above unstructured systems is particularly concerned with trust as is iTrust.

Two other systems that like iTrust, are concerned with trust are Quasar and OneSwarm. Quasar [88] is a probabilistic publish/subscribe system for social networks with many social groups. The key mechanism is a rendezvous-less event routing infrastructure that routes messages directly to nearby group members using local gradients of aggregated vectors or "gravity wells." The authors note that privacy and scalability concerns make centralized systems undesirable to the users of social networks. They state that "an unwarranted amount of trust is placed on these centralized systems to not reveal or take advantage of sensitive information." The iTrust distribution and retrieval network does not use a structured overlay, but it does use rendezvous nodes for matching metadata and requests, and it is concerned with trust. Thus, the trust objective of Quasar is quite different from that of iTrust.

OneSwarm [49] is a peer-to-peer data sharing system that allows data to be shared either publicly or anonymously, using a combination of trusted and untrusted peers. Content lookup and transfer are anonymous, congestion-aware, and multipath. OneSwarm provides anonymous publication like Freenet [20] and anonymous download like Tor [27]. Its initial goal is to protect the privacy of the users. The iTrust distribution and retrieval network does not aim

to conceal the users like OneSwarm does. It allows anyone to participate, and uses asymmetric encryption to prevent censoring by the Internet routers. OneSwarm aims to protect the users' privacy, whereas iTrust aims to support free flow of information and to prevent censorship and filtering of information.

Morselli *et al.* [65] describe an adaptive replication protocol with a feedback mechanism that adjusts the number of replicas according to the mean search length, in order to determine whether an object is replicated sufficiently. Our defensive adaptation algorithm for iTrust, described in Chapter 4, is quite different from the adaptive replication protocol that they describe.

Sarshar *et al.* [75] use random walks and bond percolation in power law networks with high-degree nodes. Such nodes are subject to overloading, and are vulnerable to malicious attacks. The authors note that "protocols for identifying or compensating for such attacks, or even recovering after such an attack has disrupted the network are yet to be designed." We address these needs in Chapter 4.

Leng *et al.* [53] present mechanisms for maintaining the desired degree of replication in BubbleStorm, when each object has a maintainer node. Our defensive adaptation algorithm for iTrust uses different techniques to maintain the desired degree of replication of requests.

Jesi *et al.* [50] identify malicious nodes in an overlay network based on gossiping, and place such nodes on a blacklist. They focus on hub attacks in which colluding malicious nodes partition the network by spreading false

rumors. iTrust does not use gossiping but rather, distributes the metadata and the requests to randomly selected nodes and, thus, is less subject to hub attacks.

Condie *et al.* [23] present a protocol for finding adaptive peer-to-peer topologies to protect against malicious peers that upload corrupt, inauthentic, or misnamed content. Peers improve the trustworthiness of the network by forming connections, based on local trust scores defined by past transactions. Effectively, their protocol disconnects malicious peers and moves them to the edge of the network. The adaptive algorithms of iTrust are quite different from those protocols.

Several network security researchers use the exponential weighted moving average and the chi-squared statistic, which we employ in Chapter 4 and 6. In particular, Goonatilake *et al.* [39] apply the chi-squared statistic to detect intrusions. Roberts [72] and Viinikka and Debar [84] use the exponential weighted moving average to discover anomalies in the network and issue alerts. Both groups of researchers discuss how to determine an appropriate λ (c in our notation), based on a fixed window size. For the iTrust membership protocols, we do not fix a window size but instead, consider all of the results from the time a node joins the membership to the time it leaves the membership. Moreover, we make c tunable, so that the user can set it for a particular network environment.

Ye and Chen [91] and Zhou *et al.* [97] present anomaly detection techniques for intrusion detection, based on the exponential weighted moving average and the chi-squared statistic. Both groups of researchers report that these techniques yield high detection rates, low false alarm rates, and early detection of intrusions. The iTrust system uses the EWMA method and the chi-squared test to determine the proportion of malicious nodes in a node's view of the membership.

Press *et al.* [68] use a modified chi-squared statistic to balance the weights of the buckets, in order to compare two datasets. Similarly, Belen [3] and Heckert [47] apply a modified chi-squared statistic to determine the similarity between attribute couples of a dataset and a projected subset. In Chapter 4, we demonstrate that the modified chi-squared statistic achieves better accuracy in estimating the proportion of operational nodes in the iTrust network.

2.3 Membership

Prior work on membership has focused on an agreed accurate membership in the presence of unreliable processors and unreliable communication. Chandra *et al.* [9] show that it is impossible to achieve an agreed accurate membership. Chockler *et al.* [12] provide a comprehensive survey of membership protocols and group communication systems, and of their formal specifications. Schiper and Toueg [76] provide an elegant formalization of the membership

problem that distinguishes between the problem of maintaining and agreeing on a set of members and the problem of determining which processes are working and should be members. Our membership protocols for iTrust are simpler and less costly than such prior work. They do not aim to achieve an agreed accurate membership based on a consensus algorithm. Rather, they allow each member to have its own local view of the membership, and aim to keep that local view close to the actual membership.

SCAMP [35] is a peer-to-peer membership service for gossip-based protocols that operates in a decentralized and self-configured manner, where no peer has global knowledge of the membership. A node that wishes to join (leave) the membership notifies some nodes in the network to add (remove) it to (from) their views. To prevent a node from becoming isolated, a node periodically tries to discover new nodes if it does not receive any messages for a given time period. Our membership protocols for iTrust place more emphasis on maintaining a node's view of the membership when the membership is subject to a lot of churn.

Zage *et al.* [93] present a network-aware and distributed membership protocol that improves the overall performance of a peer-to-peer network by biasing neighbor selections towards beneficial nodes, based on multiple system metrics and social network patterns. The authors demonstrate the effectiveness of their protocol through simulation, when the network has high churn rates. In the iTrust membership protocols, the nodes do not maintain their views of the

membership through biased neighbor selections. Rather, they discover newly joining and leaving nodes through the normal course of search and retrieval.

Liu *et al.* [55] describe a novel age-based membership protocol with a conservative neighbor maintenance scheme under churn, to retain desirable properties such as a low network diameter and clustering coefficient. Thus, a bootstrapping node recommends to a newly joining node only the nodes that have remained in its membership for a long period of time. However, with their protocol, a newly joining node might not discover other nodes in the membership very quickly. In the membership protocols for iTrust, a bootstrapping node sends its entire membership to a newly joining node.

Voulgaris *et al.* [85] present a membership management protocol, named CYCLON, for unstructured peer-to-peer networks, in which each node maintains a small and fixed-size neighbor list. The authors describe a shuffling protocol for large networks and provide an experimental analysis in which they examine the clustering coefficient and node degree distribution. The iTrust membership protocols differ from CYCLON in that each node tries to discover as many nodes as possible to include in its view of the membership. Their work is similar to the neighborhoods and de-clustering that we described previously for large iTrust networks [2], rather than the work described in Chapter 5.

BubbleStorm [53, 80] is a probabilistic search system for peer-to-peer networks built on random multi-graphs that performs exhaustive search. BubbleStorm replicates both queries and data, and combines random walks with

flooding. It also considers churn, leaves and crashes, like the iTrust membership protocol does. When a node joins the network, it finds an existing connection between two peers and interposes itself between them. When a node leaves the network, it re-connects those two peers before it leaves. If a node crashes without re-connecting its neighbors, the neighboring peer adds a connection to the other peer when it discovers the crashed node. Thus, BubbleStorm aims to maintain a fixed node degree at all of the nodes. The iTrust membership protocols do not restrict the nodes to a fixed node degree but rather, allow each node to maintain its local view of the membership.

PlanetP [26] maintains a local index that contains metadata for documents published locally by a peer, and a global index that describes all peers and their metadata in a Bloom filter, which it replicates throughout the network using gossiping. iTrust likewise maintains a local index of metadata and corresponding URLs for the data. However, iTrust does not use gossiping to distribute the membership but rather, allows each node to maintain its local view of the membership by detecting leaving (non-operational) nodes and discovering newly joining nodes through the normal course of operation of the messaging protocol.

Binzenhofer and Leibnitz [6] introduce a method for structured peer-to-peer networks that enables a peer to estimate the churn rate and to adapt accordingly. Information is piggybacked onto messages to avoid unnecessary overhead. In the iTrust membership protocols, a requesting node detects that

nodes are non-operational if they do not respond to its requests, and it discovers newly joining nodes from the responding nodes which piggyback newly joined nodes onto their responses, thus reducing the overhead.

RandPeer [54] randomly assigns members to bins, where the bins are organized into a tree structure. When the bins become full, they are split and the tree grows, resulting in a scalable structure with a logarithmic search cost. However, the tree structure is vulnerable to unavailability or malice of the root nodes. To render such a structure feasible for iTrust, it would be necessary to introduce cross links, possibly randomized, into the structure.

Heen *et al.* [48] describe a membership algorithm for structured peer-to-peer networks based on distributed hash tables. The algorithm requires an administrator, and is not completely distributed. However, it does provide communication between nodes that remains anonymous, exploiting cryptographic addressing to maintain anonymity. We plan to investigate whether such an anonymous addressing scheme is appropriate for iTrust.

Maki *et al.* [57] describe the use of cryptographic certificates to authenticate the members of an ad-hoc network. Such a strategy might be used by iTrust to ensure that malicious nodes do not attempt to impersonate other nodes when distributing misleading information. Their strategy is leader-based, but it permits multiple leaders, and might be applicable to more distributed membership protocols, such as those of iTrust.

2.4 Summary

In this chapter, first we presented work related to distributed search mechanisms in peer-to-peer networks. Next, we presented work related to trust. Finally, we discussed related work that focuses on membership management and membership under churn.

Chapter 3

Trustworthy Distributed Search and Retrieval

In this chapter, first we describe the design of iTrust, which is a novel information distribution and retrieval system with no centralized mechanisms and no centralized control. Next, we describe the implementation of the iTrust system and present the user interface. Finally, we present a performance evaluation of iTrust.

3.1 Design of iTrust

The iTrust information distribution and retrieval system involves no centralized mechanisms and no centralized control. We refer to the nodes that participate in an iTrust network as the participating nodes or the membership. An iTrust network might correspond to participants with specific interests, or it might correspond to a social network. Multiple iTrust networks within the

Internet may exist at any point in time, and a node may participate in several different iTrust networks at the same time.

In an iTrust network, some nodes (the source nodes) produce information, and make that information available to other participating nodes. The source nodes produce metadata that describes their information, and distribute that metadata to a subset of participating nodes that are chosen at random, as shown in Figure 3.1. The metadata are distinct from the information that they describe, and include a list of keywords and the URL of the source of the information.

Other nodes (the requesting nodes) request and retrieve information. Such nodes generate requests (also referred to as queries) that refer to the metadata, and distribute the requests to a subset of the participating nodes that are chosen at random, as shown in Figure 3.2.

The participating nodes compare the metadata in the requests (queries) they receive with the metadata that they hold. If such a node finds a match, which we call an encounter, the matching node returns the URL of the associated information to the requesting node. The requesting node then uses the URL to retrieve the information from the source node, as shown in Figure 3.3.

Distribution of metadata and requests to relatively few nodes suffices to achieve a high probability that a match occurs. As we have shown [60], in an iTrust membership with N nodes, distribution of the metadata to $M = 2\sqrt{N}$ nodes and distribution of the requests to $R = 2\sqrt{N}$ nodes results in a

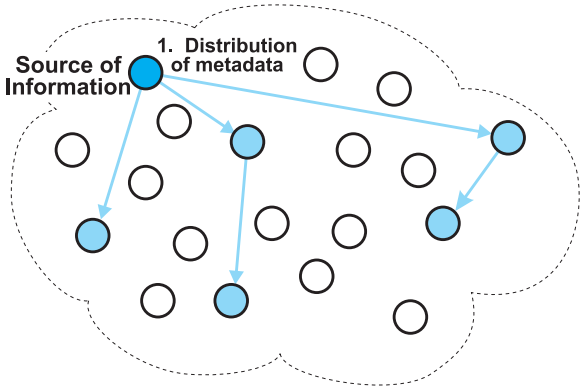

Figure 3.1: A node (a source node) distributes metadata, describing its information, to randomly selected nodes in the network.

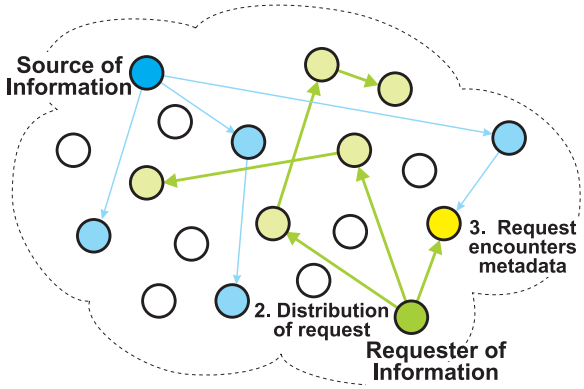

Figure 3.2: A node (a requesting node) distributes its request to randomly selected nodes in the network. One of the nodes has both the metadata and the request and, thus, an encounter occurs.

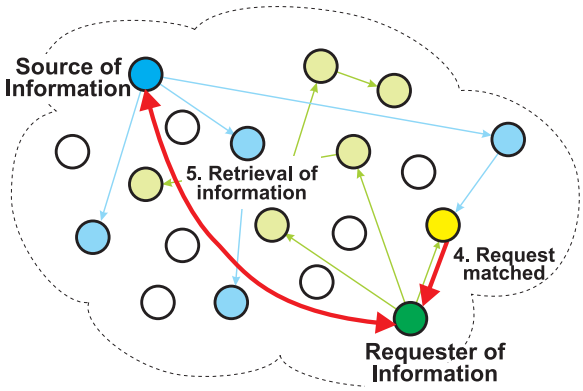

Figure 3.3: The node matches the metadata and the request and reports the match to the requester, which then retrieves the information from the source node.

probability of a match that exceeds $1 - e^{-4} \approx 0.9817$. Moreover, the strategy is robust. Even if some of the randomly chosen nodes are subverted or non-operational, the probability of a match is high, as shown in Section 3.4.3. Furthermore, it is not easy for a small group of nodes to subvert the iTrust mechanisms to control which information is delivered and which information is suppressed.

3.2 Implementation of the iTrust System

The iTrust system on a node consists of three distinct components that interact with each other to distribute metadata and requests and to retrieve information (resources). Figure 3.4 shows the three components: the Web server foundation, the application infrastructure, and the public interface. Arrows on connecting lines indicate the direction of information flow. The following subsections describe these three components and their interactions.

3.2.1 Web Server Foundation

The basis of the current implementation of iTrust is the Apache Web server compiled with several PHP standard modules and library extensions. The Web server foundation component contains no custom code; all software is used as is, which enables rapid node deployment. iTrust utilizes various standard modules, including the session and logging modules described below.

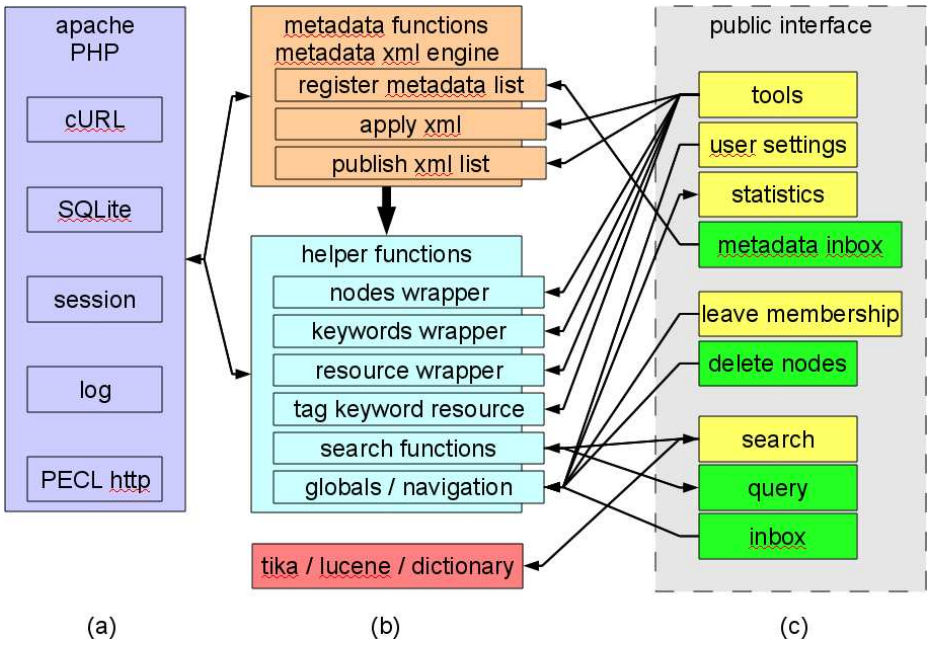

Figure 3.4: The iTrust system, which comprises (a) the Web server foundation, (b) the application infrastructure, and (c) the public interface.

The session module allows tracking of users on each node, so that multiple users can interact with the same node at the same time in a convenient manner (*i.e.*, without having to re-enter the same data on each Web page load). For example, session variables persist between multiple Web page fetches and between multiple resource retrievals. However, all session variables are purely for the convenience of the user, and a careful user may safely turn off session tracking (with only a minor inconvenience of re-entering certain data occasionally). In either case, all session data are deleted when the session (the Web browser window) is closed; there is no ability to identify a given user in subsequent sessions.

The logging module is enabled only for debugging and simulation, and can be disabled at any time by the node administrator. There is no direct relationship between the logging and session functions, *i.e.*, a user's actions cannot be tracked simply by viewing access logs (unless, of course, only one individual ever uses the node). The log file is written to disk but optionally, may be automatically emailed to the node administrator. In the case where there are multiple nodes on the same computer, all of the nodes share the same log file and prefix each log entry with a unique node identifier.

iTrust also utilizes compiled-in modules, including cURL, SQLite, and the PHP Extension Community Library (PECL) for HTTP, as described below.

The cURL functions are used primarily for inter-node communication and resource-specific actions. When a resource is added to a node, a call may be made to that resource's URL to scan for metadata automatically. cURL automatically follows HTTP redirects and resolves file dependencies (such as HTML frame sources and image sources). Both the fetched text and the fetched images are accessible to the Java jar files, as described below.

SQLite is used for all administrative information such as node, metadata and resource information. For example, the node membership is stored in a database table, and the relationships between the metadata and the resources are stored in a normalized table. SQL constraints enforce several fundamental iTrust features, such as non-duplicate node addresses in the membership and unique resource URLs. Use of SQLite as a PHP module, instead of MySQL or

PostgreSQL servers, aids with the rapid deployment of iTrust nodes. iTrust works on any reasonably modern Web host, because the file-as-a-database model of SQLite requires only minimal local write privileges.

The PHP Extension Community Library (PECL) for HTTP is an external compiled-in module used for inter-node search and metadata queries. A requesting node may use PECL HTTP to send a POST statement to a potential source node to search for the metadata that match the user's metadata query.

3.2.2 Application Infrastructure

The key iTrust methods reside in the application infrastructure; indeed, all of the node- and resource-related functions exist in this component. The infrastructure is divided into three parts: metadata-related functions, node- and resource-related functions, and Java jar files. All parts interact with the Web server foundation, whereas only some functions are exposed to the public interface component.

The creation and distribution of metadata, both internal and inter-node, are handled by the metadata-related functions. A node generates metadata from existing resources by invoking the metadata XML engine, which exhaustively scans all resources and creates an XML list describing the relationship between the metadata and the resource. Other metadata-related functions deal with the distribution of the XML list to other nodes, or with the receipt of XML lists distributed by other nodes. In the latter case, the received XML

lists are scanned, and the metadata are inserted into the current node. In this way, the metadata are replicated among participating nodes.

Node- and resource-related functions, also known as helper functions, deal with bookkeeping tasks. These functions include functions that insert nodes into the membership, insert keywords into the database, and upload or fetch resources. Resources can be tagged with metadata manually by the user, or they can be automatically scanned for metadata, depending on the user's preferences. Node querying and query relaying are also handled by the helper functions (mostly through the use of PECL HTTP). All user variables (per session) and global administrative variables are stored.

Java jar files are used to generate metadata quickly and easily, and to provide the user with many conveniences. Apache's Tika and Lucene packages are used to generate metadata from resources automatically and efficiently, in the case where the user chooses not to generate metadata manually. The WordNet dictionary is used to provide the user with functions, such as spell checking and synonym suggestions.

3.2.3 Public Interface

The public interface, through which the users and the system administrator interact with iTrust, is divided between human and computer interfaces. Computer interfaces (dark boxes on the right in Figure 3.4) handle all inter-node communication such as queries, resource distribution, and metadata list

distribution. All of the other interfaces (clear boxes on the right in Figure 3.4) are human-oriented and consist of PHP driven HTML Web pages; in fact, all human interaction with iTrust is through Web pages.

Administration is performed through the tools Web pages and other Web pages. Tools allow an administrator to add nodes or metadata keywords using simple HTML form text boxes. Adding resources requires uploading a file (form file input) or providing a URL (form text box input). User settings and statistics Web pages provide feedback to the administrator about the membership size, resource count, *etc.* An administrator may generate and distribute metadata XML lists or update the participating nodes' metadata lists. An administrator may also request that a node be removed from a node's membership. In this case, the request is activated through a human interface, and the request is distributed through the iTrust network using computer interfaces.

The most used feature of iTrust is the human interface for searching, where a user can enter a search query to request a resource. The query is sent from the current node to participating nodes using computer interfaces in a simple inbox-type fashion. Participating nodes read their inbox for queries, send back a response if there is a match, and independently decide whether to relay the query.

3.3 User Interface of iTrust

The iTrust user interface is a Web-based interface where the user can both administer and query the nodes through Web pages. Query results from multiple nodes are presented in a single Web page following a query. Node administration and user queries are separated into distinct Web pages to keep tasks distinct and easily manageable.

3.3.1 Node Administration

The user may change the membership, add source nodes, distribute metadata, and perform other administration tasks through the administration interface shown in Figure 3.5.

A node is added to the membership by entering the node IP address or URL on a comma-delimited list inside an HTML form text box. Double listing is not permitted; duplicates are removed from the list. However, multiple nodes are permitted as long as the Web site document root is distinct (*e.g.*, both www.example.com/foo and www.example.com/bar are allowed).

Figure 3.6 shows the resource insertion Web page. A resource is added to a node by means of an HTML form file control; this control permits the user to upload a file from his/her local machine. Alternately, a Web site URL can be specified, and the node then fetches the contents at that URL. The uploaded contents are post-processed, using the Apache Tika/Lucene package,

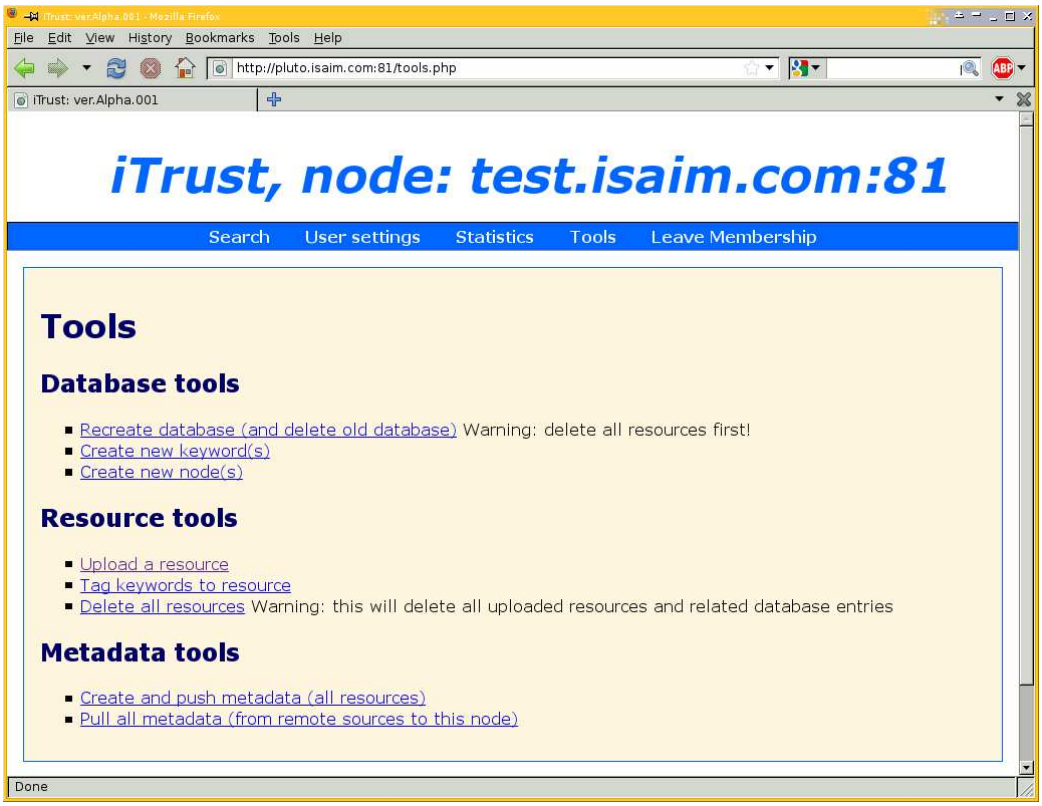

Figure 3.5: The Administration Interface.

to generate descriptive metadata (*i.e.*, keywords) automatically. The user may customize several parameters for metadata creation, including indexing by file raw content (literal text strings) or file meta content (file size, type, *etc.*). In addition to automatic metadata creation for an uploaded resource, the user may add new keywords or remove existing keywords. Finally, the user may assign an expiration date to the resource.

Administration tasks also include file administration functions to allow the user to setup, restore, or reset iTrust nodes easily. Clearing the membership, deleting all resources and metadata associations, and resetting a node to its initial setup state can all be done with a single button click. The task of

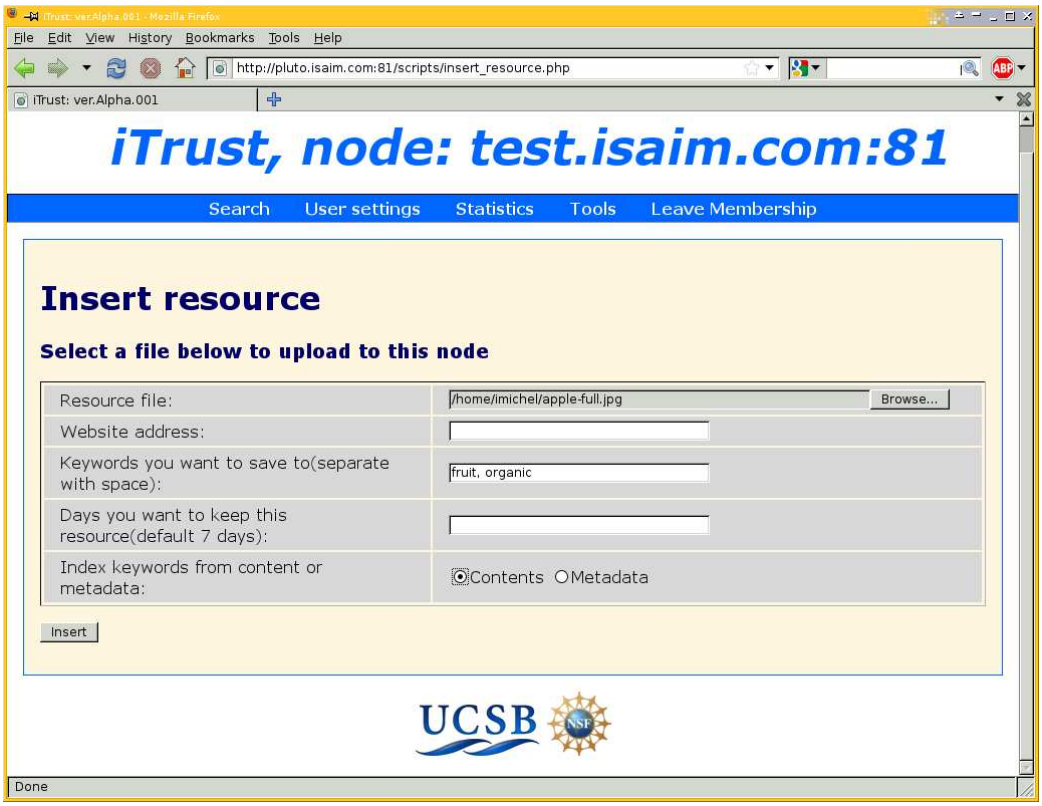

Figure 3.6: The Insert Resource Web page.

pushing all metadata changes to random nodes is also accomplished with a single button click.

3.3.2 User Queries

The user may perform queries, view the query results, and obtain resources through the user interface.

Querying is done through a single HTML form text box, whereupon the query is registered on the node and distributed throughout the iTrust network. The user is shown a status/wait Web page while the query is relayed among nodes; a result Web page is shown after a wait page timeout. The default

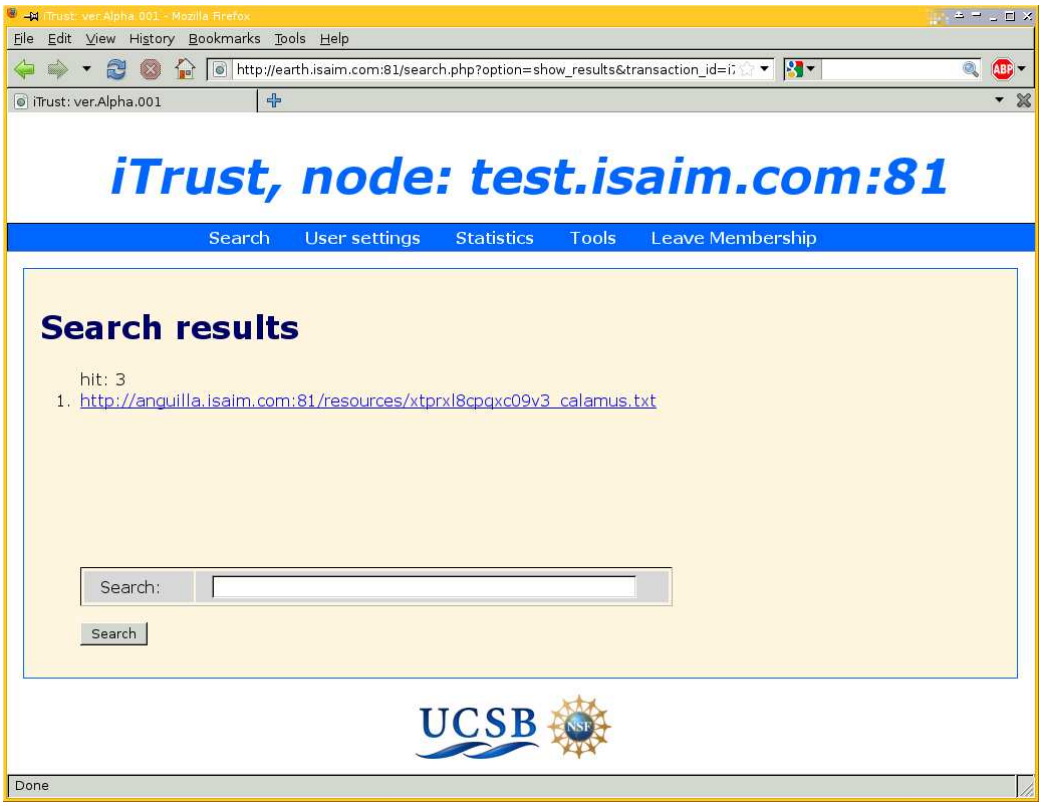

Figure 3.7: The Query Results Web page.

timeout is 3 seconds and, thus, a query incurs a 3 second latency between initialization of the query and display of the query results. However, the wait page timeout is also configurable by the node administrator.

Figure 3.7 shows the query results displayed on a new Web page (the wait page automatically redirects to the new page) in a simple HTML list. Each encounter is shown as a list item with the source address and resource handle encoded into a single URL.

The user may click the URL to retrieve the resource file; the format of the file is the originally uploaded format (there is no MIME-type modification).

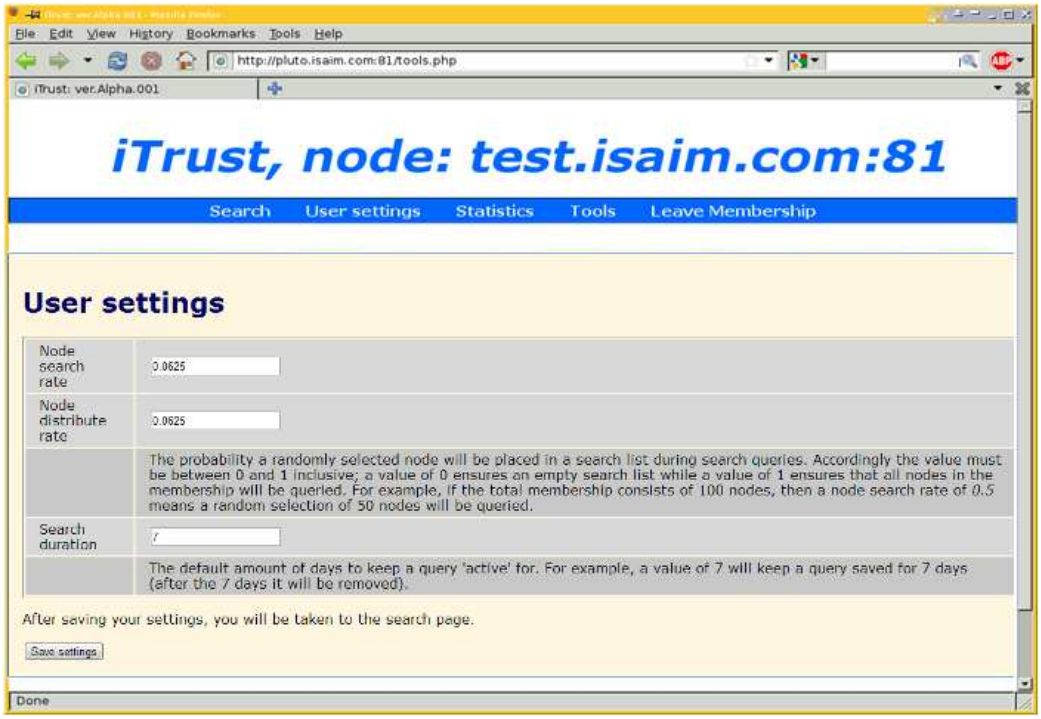

Figure 3.8: The user settings Web page.

If the Web browser recognizes the file type, it handles the data accordingly; otherwise, it calls the operating system to open the data file.

3.3.3 User Settings

Figure 3.8 shows the user settings Web page. For querying, the three primary user settings (which the user sets on the user settings page) are the number of nodes to which the metadata are distributed, the number of nodes to which the requests are distributed, and the search duration.

The number of nodes to which the metadata are distributed and the number of nodes to which the requests are distributed must, of course, be less than the number of participating nodes in the membership.

The search duration refers to the lifetime that a search query exists. The user may specify how many days a query will be stored in the database. When a user initiates a query, the system adds its creation time to the database. Later, when the user initiates a new query, the system checks and deletes expired queries from the database.

These user settings apply to the entire duration of a search session. The search session starts when a user accesses the search Web page and ends when the user exits the browser window or tab. The PHP session functions are used to automate this process.

3.4 Performance Evaluation of iTrust

3.4.1 Foundations

Assumptions and Notation

In the iTrust network, all of the requests are distributed and processed concurrently. We assume that the nodes have enough memory to store the source files, as well as the metadata that are generated and received, and that messages are delivered reliably (the iTrust implementation is based on HTTP and, thus, TCP).

We assume that all of the participating nodes in the iTrust network have the same membership set. However, some of the nodes in the iTrust network

might be unavailable, and some of the nodes might behave maliciously, *i.e.*, might not respond to a request when there is a match.

The primary parameters determining the performance of the iTrust network are:

- n: The number of participating nodes (*i.e.*, the size of the membership set)
- m: The number of participating nodes to which the metadata are distributed
- r: The number of participating nodes to which the requests are distributed
- x: The proportion of participating nodes that are operational (*i.e.*, $1-x$ is the proportion of participating nodes that are non-operational)
- K: The upper bound on the number k of responses to a request.

Analytical Model

The analytical model for the iTrust network is the hypergeometric distribution [32], which describes the number of successes in a sequence of random draws from a finite population without replacement. More specifically, in

findP(n, m, r, x, K)
1 for $k \leftarrow 1$ to K
2 $\quad P(k) \leftarrow \frac{(\frac{mx}{k} \ldots \frac{mx-k+1}{1})(\frac{n-mx}{r-k} \ldots \frac{n-mx-r+k+1}{1})}{(\frac{n}{r}\frac{n-1}{r-1} \ldots \frac{n-r+1}{1})}$
3 return P

Figure 3.9: Method for calculating the analytical probabilities $P(k)$ of specific numbers of matches.

iTrust, the analytical probability of k matches is given by:

$$\begin{aligned} P(k) &= \frac{\binom{mx}{k}\binom{n-mx}{r-k}}{\binom{n}{r}} \\ &= \frac{(\frac{mx}{k}\frac{mx-1}{k-1} \cdots \frac{mx-k+1}{1})(\frac{n-mx}{r-k}\frac{n-mx-1}{r-k-1} \cdots \frac{n-mx-r+k+1}{1})}{(\frac{n}{r}\frac{n-1}{r-1} \cdots \frac{n-r+1}{1})} \end{aligned} \tag{3.1}$$

for $mx + r \leq n$ and $k \leq \min\{mx, r\}$.

Pseudocode for calculating the analytical probabilities from Equation (3.1) is given in Figure 3.9.

From Equation (3.1), the probability $P(0)$ of *no* matches on any of the r trials is given by:

$$\begin{aligned} P(0) &= \frac{(\frac{n-mx}{r}\frac{n-mx-1}{r-1} \cdots \frac{n-mx-r+1}{1})}{(\frac{n}{r}\frac{n-1}{r-1} \cdots \frac{n-r+1}{1})} \\ &= \frac{n-mx}{n}\frac{n-mx-1}{n-1} \cdots \frac{n-mx+1-r}{n-r+1} \end{aligned} \tag{3.2}$$

where $mx + r \leq n$.

From Equation (3.2), the probability $P(k \geq 1)$ of *one or more* matches is given by:

$$\begin{aligned} P(k \geq 1) &= 1 - P(0) \\ &= 1 - \frac{n-mx}{n}\frac{n-mx-1}{n-1} \cdots \frac{n-mx+1-r}{n-r+1} \end{aligned} \tag{3.3}$$

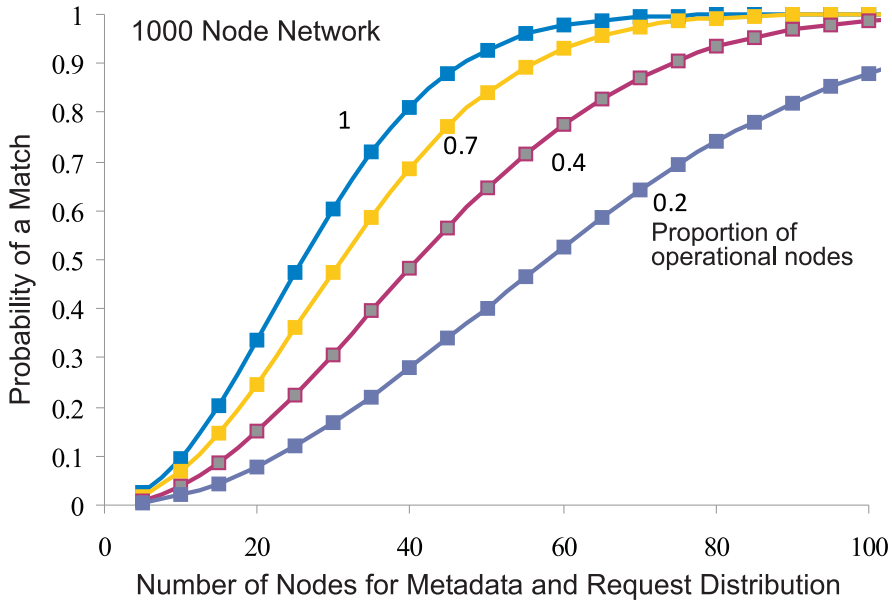

Figure 3.10: The probabilities of one or more matches as the number of nodes to which the metadata and the requests are distributed increases, for different proportions of operational nodes.

where $mx + r \leq n$.

The iTrust implementation handles requests with responses that return different URLs. However, Equation (3.3) applies to a single source node and a single requesting node. Consequently, the iTrust implementation and algorithms handle the matches for each URL separately.

Figure 3.10 shows the probabilities of a match for an iTrust membership with $n = 1000$ participating nodes, obtained from Equation (3.3), when a proportion of the nodes are non-operational. The figure shows the probability of a match as the number of nodes to which the metadata and the requests are distributed increases when a proportion $x = 1.0, 0.7, 0.4, 0.2$ of the participating nodes are operational. As the figure shows, iTrust retains significant

utility in circumstances in which a substantial proportion of the nodes are non-operational, which might be the circumstances in which the information is most needed.

3.4.2 Emulation of iTrust

Using our implementation of iTrust described in Section 3.2, we performed experiments to validate Equation (3.1). In our emulation of iTrust, we used libCURL (which is a free client-side URL transfer library for transferring data using various protocols) to collect the match probabilities.

Before we run our program, we provide the following input to the program: the number n of nodes in the membership, the number m of nodes for metadata distribution, the number r of nodes for request distribution, and the proportion x of operational nodes.

First, the program clears the data from the SQLite databases. Next, the program adds the nodes to the membership. Once all of the nodes are added to the membership, we call the source node to upload a file and the program then creates the corresponding metadata. Then, the program randomly selects nodes for metadata distribution, and distributes the metadata to those nodes. Next, the program randomly selects the nodes for request distribution, and distributes the requests to those nodes. Then, the program waits for 5 seconds. If one or more nodes has replied back to the program, it means that there is

a match and the program returns 1; otherwise, there is no match and the program returns 0.

We repeat the same process 100 times for the source nodes and correspondingly for the requesting nodes, and plot the mean results in our graphs. We collected data for 36, 72 and 144 participating nodes, when all of the nodes are operational. We also collected data for 144 participating nodes when 100%, 80% and 60% of the nodes are operational.

3.4.3 Performance Evaluation Results

First, we consider the analytical and emulation results for the probability of a match, as the number of participating nodes increases. Then, we consider the analytical and emulation results for the probability of a match, as the proportion of non-operational nodes increases.

Increasing the number of participating nodes

Figures 3.11, 3.12 and 3.13 show both the analytical results and the emulation results for 36, 72 and 144 participating nodes, all of which are operational. The analytical curves obtained from Equation (3.3) are shown in the background (light curves), and the emulation curves obtained from our iTrust implementation are shown in the foreground (dark curves). We see from these figures that the emulation results are very close to the analytical results.

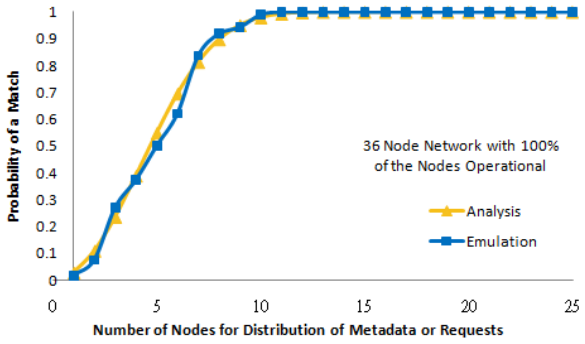

Figure 3.11: Match probability vs. number of nodes for distribution of metadata and requests with 36 participating nodes, all of which are operational.

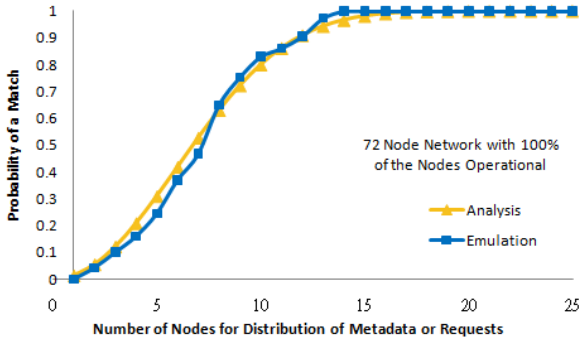

Figure 3.12: Match probability vs. number of nodes for distribution of metadata and requests with 72 participating nodes, all of which are operational.

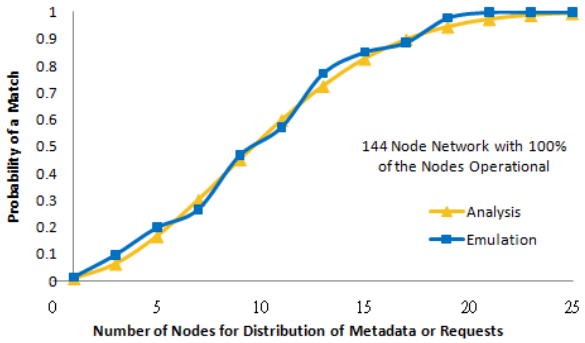

Figure 3.13: Match probability vs. number of nodes for distribution of metadata and requests with 144 participating nodes, all of which are operational.

Figure 3.11 shows the match probability versus the number of nodes for distribution of metadata and requests in a network when 100% of the 36 nodes are operational. From the figure, we see that the probability increases as the number of nodes to which the metadata and requests are distributed increases. The reason is that the more nodes to which the metadata and requests are distributed, the more matches there are.

When the membership contains more nodes, the match probability asymptotically approaches 1 more slowly than for a membership with fewer nodes. That is, if we distribute the metadata and the requests to the same number of nodes, but the membership contains more nodes, the probability of a match is less than that for a membership with fewer nodes.

When we increase the membership to 72 nodes in Figure 3.12, the curves approach 1 more slowly than do the curves in Figure 3.11 for a membership containing 36 nodes. In other words, as we increase the membership, we must distribute the metadata and the requests to more nodes to obtain a higher match probability. Similarly, in Figure 3.13, when we increase the membership to 144 nodes, we see that the curves grow even more slowly than do the curves in the 36 node and 72 node networks.

Suppose now, for example, that we want to achieve a 0.98 match probability, in these three cases, which involve 36, 72 and 144 nodes all of which are operational. In the 36 node network, we need to distribute the metadata and the requests to only 10 nodes to achieve a 0.98 match probability. However, in

the 72 node network, we need to distribute the metadata and the requests to 15 nodes to achieve a 0.98 match probability, whereas in the 144 node network, we need to distribute the metadata and the requests to 22 nodes to achieve a 0.98 match probability.

Thus, when we distribute the metadata and the requests to only a few nodes, the match probability is lower and the requester is unlikely to receive multiple responses from multiple matching nodes. When we distribute the metadata and the requests to more nodes, the match probability is higher and the requester will more likely receive multiple responses from multiple matching nodes. For a network with more participating nodes, the match probability grows more slowly than the match probability for a network with fewer participating nodes.

Increasing the number of non-operational nodes

Figures 3.14, 3.15 and 3.16 show both the analytical results and the emulation results for 144 nodes, when 100%, 80% and 60% of the participating nodes are operational, *i.e.*, when 0%, 20% and 40% of the participating nodes are non-operational. The analytical curves obtained from Equation (3.3) are shown in the background (light curves), and the emulation curves obtained from our iTrust implementation are shown in the foreground (dark curves). Again, we see from these figures that the emulation results are very close to the analytical results.

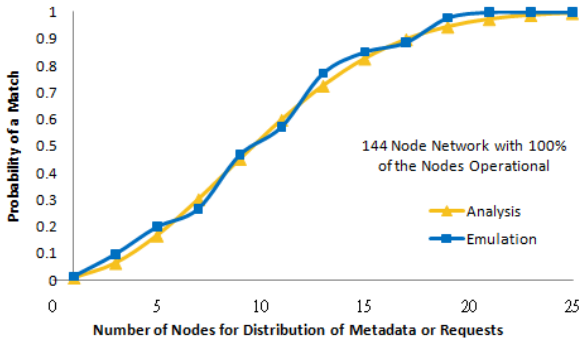

Figure 3.14: Match probability vs. number of nodes for distribution of metadata and requests with 144 participating nodes where 100% of the nodes are operational.

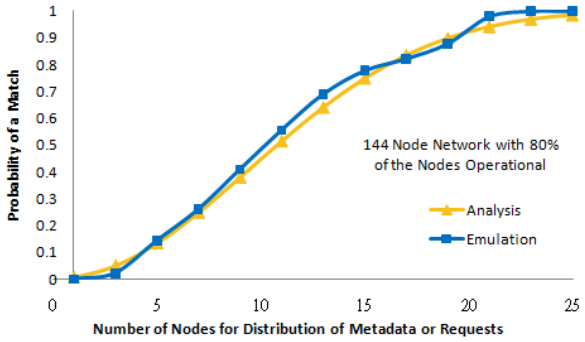

Figure 3.15: Match probability vs. number of nodes for distribution of metadata and requests with 144 participating nodes where 80% of the nodes are operational.

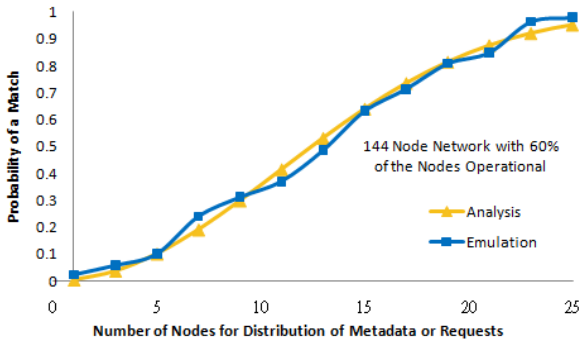

Figure 3.16: Match probability vs. number of nodes for distribution of metadata and requests with 144 participating nodes where 60% of the nodes are operational.

In Figures 3.14, 3.15 and 3.16, we see that the match probability curves increase as the number of nodes for distribution of metadata and requests increases. However, if we compare Figure 3.14 with Figure 3.15, we notice that the curves in Figure 3.14 asymptotically approach 1 faster than the curves in Figure 3.15. The reason is that in Figure 3.14 every node is operational, whereas in Figure 3.15 only 80% of the nodes are operational. Therefore, for distribution of metadata and requests to the same number of nodes, the probability of a match in Figure 3.14 is generally higher than it is Figure 3.15. Similarly, in Figure 3.16, where only 60% of the nodes are operational, the curves asymptotically approach 1 more slowly than the curves in Figures 3.14 and 3.15.

Suppose now, for example, that the metadata and the requests are distributed to 20 nodes in these three cases, all of which involve 144 nodes. If 100% of the 144 nodes are operational, the probability of a match is 0.96. But if 80% of the 144 nodes are operational, the probability of a match is 0.92, whereas if 60% of the 144 nodes are operational, the probability of a match is is 0.85, which is still quite good.

Thus, when all of the participating nodes are operational, the match probability is higher and the requester will likely receive multiple responses from multiple matching nodes. When there are fewer operational nodes, the match probability is lower and the requester is less likely to receive multiple responses from multiple matching nodes. Consequently, we must distribute the meta-

data and the requests to more nodes as the number of non-operational nodes increases, to obtain higher match probabilities. Nonetheless, iTrust retains significant utility even when not all of the nodes are operational, demonstrating that iTrust is quite robust.

3.5 Summary

We have described iTrust, a novel information publication, search and retrieval system with no centralized mechanisms and no centralized control. iTrust involves distribution of metadata and request (queries) containing keywords, matching of keywords and metadata, and retrieval of information corresponding to the metadata. We have shown that with iTrust, the probability of matching a query is high even if some of the participating nodes are subverted or non-operational. The iTrust system is particularly valuable for individuals who wish to share information, without having to worry about subversion or censorship of information.

Chapter 4

Protecting against Malicious Attacks in iTrust

The decentralized and distributed nature of iTrust makes it robust against malicious attacks that aim to prevent access to published information. One specific kind of malicious attack, addressed in this chapter, is to insert into the network a large number of nodes that behave normally except that they do not return responses, if they have a match. The appropriate reaction to such an attack is to increase the number of nodes to which the requests are distributed, thus restoring the probability of a match to the desired level.

Effective use of iTrust depends on information about the network that might not be easy to obtain, such as the size of the iTrust network. Even more difficult to obtain is information about uncooperative or malicious nodes that might deliberately attempt to conceal information about sensitive topics by failing to report matches for requests related to those topics. The absence of centralized control makes it difficult to determine such information. This

problem is not unique to iTrust; other peer-to-peer, ad-hoc, and dynamic networks face similar problems [33, 56, 75, 80].

In iTrust, we deal with non-operational nodes that have crashed or that have become disconnected. We also deal with malicious nodes that appear to behave normally, except that they do not match requests related to sensitive topics. Here, malice is represented not by explicit overt bad behavior but rather, by selective silence, where such silence would be correct behavior if the node did not hold metadata that matches the request.

Other researchers have developed techniques to cope with overt malicious behavior [52] and to cope with selfish nodes that do not respond in order to lighten their workloads [45]. In iTrust, we associate absence of action with the intent to censor sensitive information, when absence of action is a permissible behavior if the node does not hold metadata for that information. As far as we are aware, iTrust is the first system that infers that nodes are not responding in order to conceal sensitive information.

The novelty of this work is the use of a statistical method, together with an analytical model, to infer characteristics of the network that are not measurable directly. In iTrust, the nodes infer useful but unobservable information about the network by observing aspects of the behaviors of the nodes that are visible to them. More specifically, the nodes infer such information by monitoring the responses to their requests.

Often, a request results in multiple responses. The multiple responses do not constitute waste. Rather, they provide valuable information about the network, information that is difficult to obtain directly from the nodes. In particular, the multiple responses allow a node to estimate the proportion of non-operational or subverted nodes. The presence of non-operational or subverted nodes reduces the number of responses received for a request. Having detected a reduction in the number of responses, a node can then adaptively increase the number of nodes to which it distributes a request, to maintain a high probability of a match.

In this chapter, we describe this novel statistical inference technique for iTrust, which can also be applied to other networks such as [33, 56, 75, 80]. More specifically, we present novel statistical algorithms for detecting and defending against malicious attacks in the iTrust network, including:

- A detection algorithm that detects changes in the proportion of operational nodes, rejecting changes due to random fluctuations
- A defensive adaptation algorithm that changes the number of nodes to which the requests are distributed, thus compensating for changes in the proportion of operational nodes and maintaining a high probability of a match.

The detection and defensive adaptation algorithms utilize the exponential weighted moving average method to collect the observational data, from which

a requesting node determines the empirical probabilities of the numbers of responses received for its requests. The node then employs the chi-squared test to compare the empirical probabilities against the analytical (expected) probabilities, in order to estimate the proportion of operational nodes in the iTrust network. The node then increases or decreases the number of nodes to which it sends its requests, in order to maintain the same high probability of a match as when all of the nodes in the network are operational.

By design and by choice of appropriate parameters for the detection and defensive adaptation algorithms, we aim to achieve good performance, in particular:

- High probability of a match, despite changing network conditions
- High accuracy in estimating the proportion of operational nodes
- Low response time in detecting a change in the proportion of operational nodes.

4.1 Foundations

4.1.1 Assumptions and Notation

The primary parameters determining the performance of the iTrust network are:

- n: The number of participating nodes (*i.e.*, the size of the membership set)
- m: The number of participating nodes to which the metadata are distributed
- r: The number of participating nodes to which the requests are distributed
- x: The proportion of participating nodes that are operational (*i.e.*, $1-x$ is the proportion of participating nodes that are non-operational)
- K: The upper bound on the number k of responses to a request.

Our algorithm for detecting malicious attacks employs the method in Figure 3.9 for calculating the analytical probabilities, which it uses to estimate the proportion of nodes that are operational and, thus, the proportion of nodes that are non-operational or subverted.

Our algorithm for defending against malicious attacks uses Equation (3.3) to determine the amount by which to increase the number of nodes to which the requests are distributed, in order to compensate for an increase in the proportion of subverted or non-operational nodes. Simplistically, if only one-half of the nodes are non-operational, a requesting node needs to distribute its request to twice as many nodes. More precisely, these numbers are affected by the size of the membership set. In Sections 4.2, 4.3 and 4.4, we provide the more refined algorithms.

```
norm(P, K)
1  S ← 0
2  for i ← 1 to K
3     S ← S + P(i)
4  for k ← 1 to K
5     Q(k) ← P(k)/S
6  return Q
```

Figure 4.1: Method for normalizing the probabilities.

4.1.2 Normalization

In our experiments and in real-world deployments, we cannot use requests that return $k = 0$ responses, because zero responses can arise not only when the metadata and the requests are distributed to disjoint subsets of nodes, but also when there exists no metadata that matches a request. Thus, we exclude requests that return $k = 0$ responses. Moreover, for large values of k, the probability of k matches is negligibly small, so we also exclude those requests. That is, we determine a value K and exclude requests that return k responses for $k > K$. We then normalize the probabilities $P(k), 1 \leq k \leq K$.

To find the normalized probabilities $Q(k)$, $1 \leq k \leq K$, we perform the following calculation:

$$Q(k) = \frac{P(k)}{\sum_{i=1}^{K} P(i)} \tag{4.1}$$

Pseudocode for normalizing the probabilities is given in Figure 4.1.

We apply the normalization method given in Figure 4.1 to both the analytical probabilities and the observed probabilities before applying the chi-squared test, as described in Section 4.2.

4.1.3 Evaluation Metrics

For the evaluation of the detection and defensive adaptation algorithms in estimating the proportion x' of operational nodes, we employ the following metrics:

- *Match probability*: The probability that a request results in one or more matches
- *Accuracy*: The number of correct estimates of x' divided by the total number of estimates of x' within an interval of requests
- *Response time*: After a change in the actual proportion x of operational nodes, the number of requests before the algorithm changes the value of x'.

Ideally, the match probability and the accuracy are high and the response time is low. The defensive adaptation algorithm depends on a high accuracy value to maintain a high match probability. However, there are trade-offs between the accuracy and the response time, which we discuss in subsequent sections.

4.1.4 Ranges for the Evaluation

Initially, in our experiments, we consider two consecutive ranges: RangeA and RangeB. In RangeA $x = 1.0$, and in RangeB $x = 0.7$, as shown in Figure

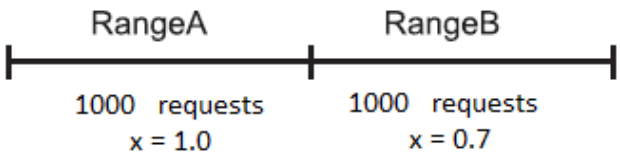

Figure 4.2: The two ranges, each containing 1000 requests that produce 1000 samples, for a total of 2000 samples.

4.2. We consider 1000 requests in RangeA, and 1000 requests in RangeB, for a total of 2000 samples in these experiments.

To calculate the accuracy, our program uses a counter to keep track of the number of correct estimates of x' in both RangeA and RangeB. The program ends when it reaches 2000 requests, and then reports the number of correct estimates. It then calculates the accuracy by dividing the number of correct estimates by the total number of samples (which is 2000 in these experiments). We repeat these steps 1000 times and then average the results to obtain the mean accuracy. In the experiments, we repeat the process but with RangeB set to $x = 0.4$, and then to $x = 0.2$.

To calculate the response time, our program calculates the number of requests from the start of RangeB to the point at which the program detects a change in the value of x'. We repeat these steps 1000 times and then average the results to obtain the mean response time in terms of the number R of requests. Then, we repeat the process but with RangeB set to $x = 0.4$, and then to $x = 0.2$.

The response time, in seconds, depends on the time interval between requests, *i.e.*, the time between issuing a request and obtaining the responses for that request plus the think time. If it takes S seconds between each request, then the response time in seconds is $S * R$ seconds. More frequent requests (*i.e.*, smaller values of S) yield a smaller response time. However, if the number of participating nodes is large, frequent requests might overload the iTrust network.

4.2 Detecting Malicious Attacks

The novel algorithm for detecting unobservable malicious attacks in the iTrust network estimates the proportion of operational nodes and, thus, the proportion of non-operational or subverted nodes.

For given values of n, m and r, the detection algorithm computes the analytical (expected) probabilities for the number k of matches for various values of x, such as $x = 1.0$, 0.7, 0.4, and 0.2, for the hypergeometric distribution given by Equation (3.1). These values of x enable the algorithm to discriminate the curves better and to estimate values of the proportion x' of operational nodes, that yield significant changes in the number r' of nodes to which the requests are distributed.

```
chiSquared(O_i, E_x, K)
1  χ² ← 0
2  for k ← 1 to K
3      χ² ← χ² + ((O_i(k) − E_x(k))²) / E_x(k)
4  return χ²
```

Figure 4.3: Chi-squared method.

4.2.1 Chi-Squared Test and Modified Chi-Squared Test

The detection algorithm uses Pearson's chi-squared goodness-of-fit test [40] to compare the normalized analytical probabilities and the normalized observed probabilities for different values of x, given by Equation (3.1) for the hypergeometric distribution. This test determines which of the analytical curves best matches the observed probabilities of a match. The chi-squared statistic is given by:

$$\chi^2 = \sum_{k=1}^{K} \frac{(o_k - e_k)^2}{e_k} \tag{4.2}$$

where:

- o_k: The actual number of observations that fall into the kth bucket
- e_k: The expected number of observations for the kth bucket
- K: The number of buckets into which the observations fall.

Pseudocode for the chi-squared method is given in Figure 4.3.

In addition to the chi-squared statistic, we also investigate the modified chi-squared statistic [68] defined by:

$$\chi^2 = \sum_{k=1}^{K} \frac{(K_1 o_k - K_2 e_k)^2}{o_k + e_k} \tag{4.3}$$

where:

- o_k: The actual number of observations that fall into the kth bucket
- e_k: The expected number of observations for the kth bucket
- K: The number of buckets into which the observations fall
- K_1 and K_2: The scaling constants defined by:

$$K_1 = \sqrt{\frac{\sum_{k=1}^{K} e_k}{\sum_{k=1}^{K} o_k}} \quad \text{and} \quad K_2 = \sqrt{\frac{\sum_{k=1}^{K} o_k}{\sum_{k=1}^{K} e_k}} \tag{4.4}$$

Because we normalize the probabilities (both those obtained from observation and those obtained from Equation (3.1)) before we apply the chi-squared statistic, both $\sum_{k=1}^{K} o_k$ and $\sum_{k=1}^{K} e_k$ equal 1, which means that $K_1 = 1$ and $K_2 = 1$. Thus, in our case, the modified chi-squared statistic is:

$$\chi^2 = \sum_{k=1}^{K} \frac{(o_k - e_k)^2}{o_k + e_k} \tag{4.5}$$

The only difference between the chi-squared statistic and the modified chi-squared statistic is the denominator. Note that from [68] (p. 616), the denominator in Equation (4.5) should not be the average of $o_k + e_k$ but the sum, *i.e.*, twice the average.

Pseudocode for the modified chi-squared method is given in Figure 4.4.

modifiedChiSquared(O_i, E_x, K)
1 $\chi^2 \leftarrow 0$
2 for $k \leftarrow 1$ to K
3 $\quad \chi^2 \leftarrow \chi^2 + \frac{(O_i(k)-E_x(k))^2}{O_i(k)+E_x(k)}$
4 return χ^2

Figure 4.4: Modified chi-squared method.

We apply the chi-squared method given in Figure 4.3 and the modified chi-squared method given in Figure 4.4 to the detection algorithm for estimating the proportion of operational nodes in the iTrust network.

4.2.2 Detection Algorithm

The detection algorithm collects statistical data on the number of responses that a requesting node receives for its requests, using the exponential weighted moving average method, described in Section 4.4.1. Then, it calculates the empirical probabilities from the statistical data. The algorithm excludes zero matches, because it cannot distinguish the case in which no node holds the metadata and the case in which no node holds both the metadata and the request.

Using the chi-squared test or the modified chi-squared test, the detection algorithm compares the normalized observed probabilities $O_i(k)$ and the normalized expected probabilities $E_x(k)$ for the number of matches obtained from the analytical model in Section 3.4.1, for $k = 1, 2, ..., K$ and $x = 1.0, 0.7, 0.4, 0.2$. It then chooses, as the observed proportion of operational nodes, the value of

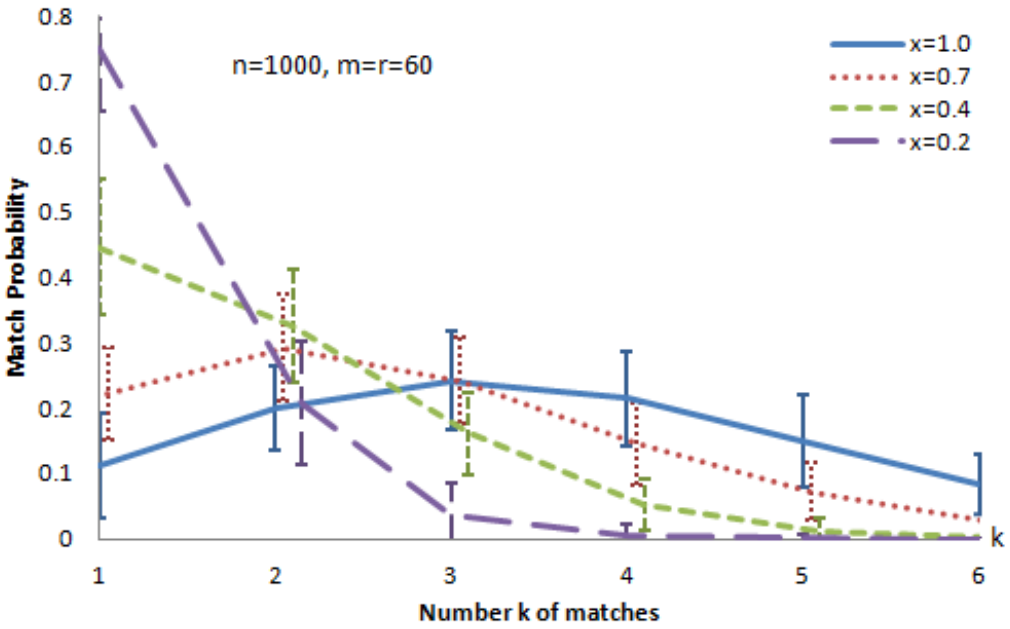

Figure 4.5: The normalized expected probabilities $E_{1.0}$, $E_{0.7}$, $E_{0.4}$, $E_{0.2}$ of the number k of matches for various proportions x of operational nodes. The error bars show the ranges within which the normalized observed probabilities occur.

x' ($x' = 1.0, 0.7, 0.4, 0.2$) for which the chi-squared value is the smallest. This value of x' is the algorithm's best estimate of the proportion of operational nodes, and corresponds to the curve with the best fit.

For $n = 1000$ and $m = r = 60$, Figure 4.5 shows the normalized expected probabilities $E_{1.0}$, $E_{0.7}$, $E_{0.4}$, $E_{0.2}$ of the number of matches for $x = 1.0, 0.7, 0.4, 0.2$ operational nodes, derived from the hypergeometric distribution given by Equation (3.1). As the figure shows, many of the requests result in multiple responses. The multiple responses are used to estimate the proportion of operational nodes in the iTrust network. The figure also shows error bars for the probabilities of the numbers of matches, derived from observations

of 500 batches with 40 requests in each batch. As the figure shows, there is considerable variability, and some overlap between the curves for the various values of x. Nonetheless, the chi-squared method provides good discrimination between the curves.

The parameters of the detection algorithm are:

- s: The number of requests for which samples are accumulated before the detection algorithm calculates χ^2 and estimates the value of x'
- t: The number of successive estimates by the chi-squared test that indicate the same change in the value of x' before the detection algorithm accepts that change and the defensive adaptation algorithm takes action to change the value of r'
- K: The upper bound on the number of buckets used for the responses to a request.

For the request accumulation s, if the algorithm evaluates χ^2 immediately after obtaining a new sample, the computational and memory costs are greater. However, if the algorithm waits too long to evaluate χ^2, the response time for detecting a change in x' is greater. Therefore, we seek to determine a value of s that balances these two extremes.

For the estimate confirmation t, a smaller value of t results in a "hair trigger" algorithm that responds more quickly to changes, but that makes

more mistakes. A larger value of t results in a more conservative algorithm that responds more slowly to change.

4.2.3 Determining Appropriate Values of s and t

For the chi-squared statistic, we consider three different values of s ($s = 1, 5, 10$), and three different values of t ($t = 1, 2, 3$).

For $s = 1$, the algorithm evaluates χ^2 each time it obtains a new sample. For $s = 5$, it evaluates χ^2 after the 5th new sample, rather than immediately. For $s = 10$, it evaluates χ^2 after the 10th new sample, rather than immediately.

For $t = 1$, the algorithm detects a change in the value of x' and takes action immediately. For $t = 2$, the algorithm takes action when it detects the same change in the value of x' in two consecutive estimates. For $t = 3$, the algorithm takes action when it detects the same change in the value of x' in three consecutive estimates.

We found the mean accuracy and the mean response time for the detection algorithm when action is taken immediately ($s = 1$ and $t = 1$) and when action is delayed ($s = 5$ or 10 and $t = 2$ or 3). The values of the other parameters for these experiments are $n = 1000$, $m = 60$, $r = 60$, $c = 0.97$, $d = 40$, and $K = 60$. Here, c is the weighting factor for the exponential weighted moving average method described in Section 4.4, where we determine that $c = 0.97$ is an appropriate value for the weighting factor. We performed each experiment for RangeA with $x = 1.0$ and RangeB with $x = 0.7$, 0.4, and 0.2.

s=1	1	2	3
Mean Accuracy			
x =1.0, 0.7	0.9240	0.9241	0.9246
x = 1.0, 0.4	0.9074	0.9081	0.9091
x = 1.0, 0.2	0.8789	0.8792	0.8794
Mean Response Time			
x =1.0, 0.7	32.1	34.5	40.8
x = 1.0, 0.4	69.5	72.4	74.0
x = 1.0, 0.2	148.7	149.7	150.7

s=5	1	2	3
Mean Accuracy			
x =1.0, 0.7	0.9216	0.9319	0.9380
x = 1.0, 0.4	0.9069	0.9120	0.9170
x = 1.0, 0.2	0.8784	0.8828	0.8855
Mean Response Time			
x =1.0, 0.7	36.0	47.0	56.5
x = 1.0, 0.4	75.0	90.0	100.0
x = 1.0, 0.2	150.5	156.5	161.5

s=10	1	2	3
Mean Accuracy			
x =1.0, 0.7	0.9216	0.9353	0.9426
x = 1.0, 0.4	0.8975	0.9077	0.9148
x = 1.0, 0.2	0.8779	0.8843	0.8887
Mean Response Time			
x =1.0, 0.7	45.0	67.0	80.0
x = 1.0, 0.4	83.0	99.0	122.0
x = 1.0, 0.2	153.0	163.0	173.0

Table 4.1: Mean accuracy and the mean response time for $s = 1, 5, 10$.

We performed each experiment 1000 times and then averaged the results to obtain the mean accuracy and the mean response time.

Table 4.1 shows the mean accuracy and the mean response time for $s = 1, 5$, and 10, respectively. The table shows a useful increase in the mean accuracy as s is increased from 1 to 5, but a smaller increase in the accuracy as s is further increased to 10. For the mean response time, increasing s from 1 to 5 causes a relatively small increase in the response time, whereas a further increase in s causes a larger response time. The improvements in the accuracy are more substantial when $t = 2$, than when $t = 1$, but the increase in the response time is still reasonable. Consequently, we chose $s = 5$ for our experiments, and investigate the effects of different values of t further in Section 4.2.4.

4.2.4 Determining an Appropriate Value of K

Next, we consider the number K of buckets used by the detection algorithm for the chi-squared statistic. In our previous experiments for $n = 1000$, $r = 60$, and $m = 60$, we set $K = 60$. However, for $k > 6$, the analytical probabilities $P(k)$ become small and, for $k > 15$, they become negligible. Thus, we investigate the two cases $K = 6$ and $K = 15$. In the detection algorithm, we only consider requests with k responses, $1 \leq k \leq K$.

The top of Figure 4.6 shows the mean accuracy, and the bottom of Figure 4.6 shows the mean response time for $K = 60$, 15, and 6 buckets. These

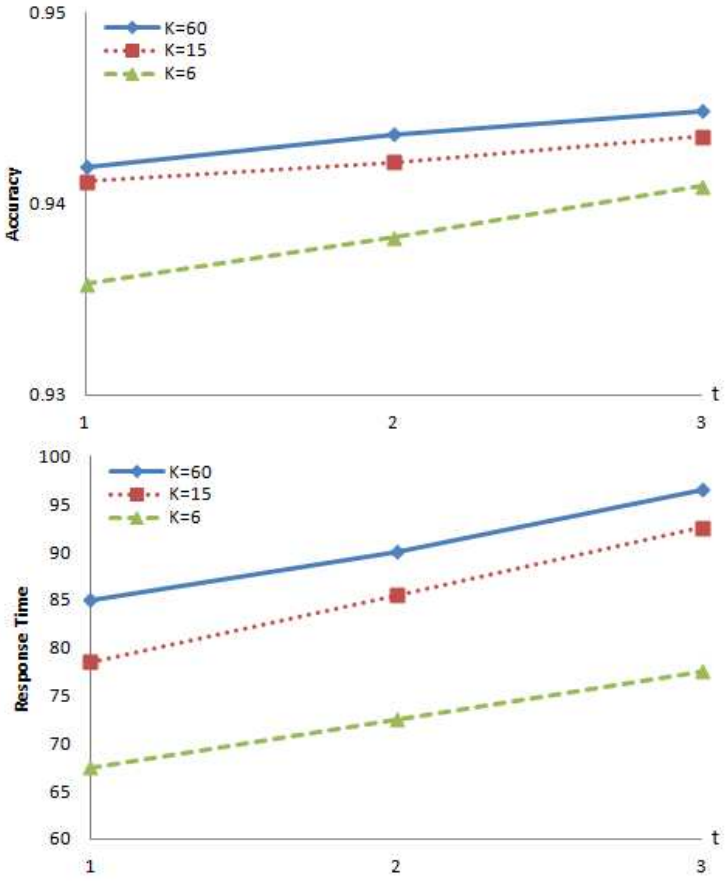

Figure 4.6: Mean accuracy and mean response time for K buckets where $K = 60$, 15, and 6 and $t = 1$, 2, and 3, where $n = 1000$, $m = r = 60$, $c = 0.97$, $d = 40$, $s = 5$, RangeA with $x = 1.0$ and RangeB with $x = 0.4$.

graphs were produced using $n = 1000$, $m = 60$, $r = 60$, $c = 0.97$, $d = 40$, $s = 5$, rangeA with $x = 1.0$, and rangeB with $x = 0.4$.

At the top of Figure 4.6, we see that the mean accuracy for $K = 6$ buckets is less than the mean accuracy for $K = 15$ and $K = 60$ buckets for all three values of t. Moreover, the accuracy values for $K = 15$ and $K = 60$ buckets are quite close to each other for all three values of t.

At the bottom of Figure 4.6, we see that the mean response time for $K = 6$ buckets is less than the mean response times for $K = 15$ and $K = 60$ buckets

for all three values of t. Moreover, the mean response time for $K = 15$ buckets is slightly less than the mean response time for $K = 60$ buckets for all three values of t.

Therefore, based on these results, we conclude that $K = 15$ buckets are enough buckets to achieve high accuracy and low response time.

4.2.5 Chi-Squared Test vs. Modified Chi-Squared Test

One of the drawbacks of the chi-squared statistic is that sometimes it places too much weight on some buckets, especially buckets with larger values of k in the case of $x = 0.2$ or $x = 0.4$. In particular, the mean accuracy for $x = 1.0$ and 0.2 and for $x = 1.0$ and 0.4 are not that high compared to $x = 1.0$ and 0.7. For example, for $x = 0.2$, the normalized expected probability for $k = 4$ matches is $E_{0.2}(4) = 0.007$. If the normalized observed probability is $O_i(4) = 0$, the small value 0.007 does not contribute much to the chi-squared statistic, because it is close to the expected value 0.007. But if $O_i(4) = 0.0933$, division by the small value 0.007 in the chi-squared statistic places too much weight on bucket 4, which affects the overall weights in the chi-squared statistic and, thus, results in an incorrect estimate of x'.

Therefore, we also investigated the modified chi-squared statistic given by (4.5. The values of the parameters for this experiment are $n = 1000$, $m = 60$, $r = 60$, $c = 0.97$, $d = 40$, $s = 5$, $t = 2$, and $K = 15$.

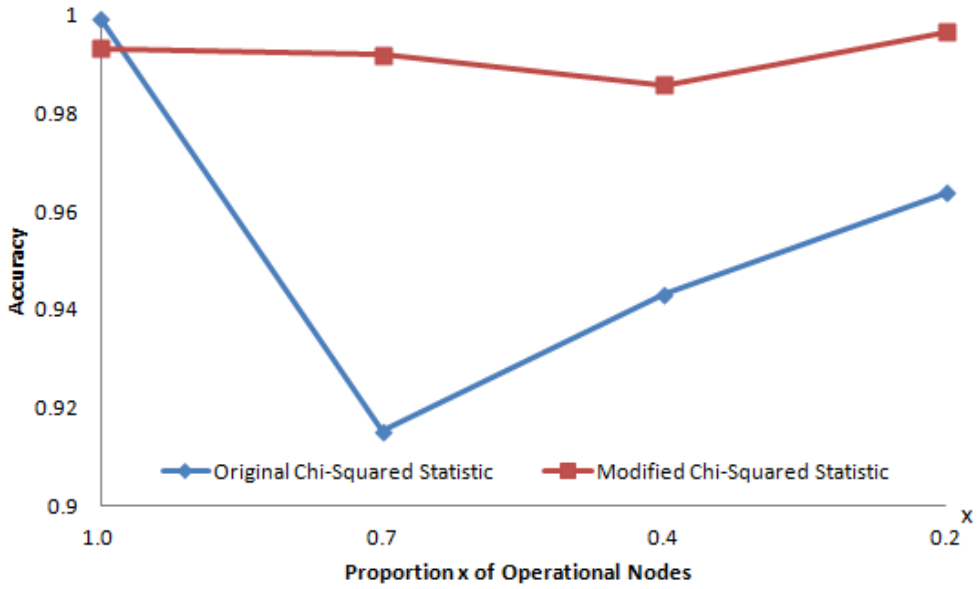

Figure 4.7: Chi-squared statistic vs. modified chi-squared statistic with $n = 1000$, $m = r = 60$, $c = 0.97$, $d = 40$, $s = 5$, $t = 2$, and $K = 15$.

Figure 4.7 shows the mean accuracy for the chi-squared statistic and the mean accuracy for the modified chi-squared statistic. As we see in the figure, the chi-squared statistic has slightly higher accuracy for $x = 1.0$, but the modified chi-squared statistic achieves higher accuracy for $x = 0.7$, $x = 0.4$, and $x = 0.2$. Therefore, we used the modified chi-squared statistic in our further experiments.

4.3 Defending against Malicious Attacks

Consider an iTrust network with $n = 1000$, $m = 60$, and $x = 1.0$ 0.7, 0.4, 0.2, as shown in Figure 4.8. The defensive adaptation algorithm first determines the value of y_0 for the point on the $x = 1.0$ curve corresponding to

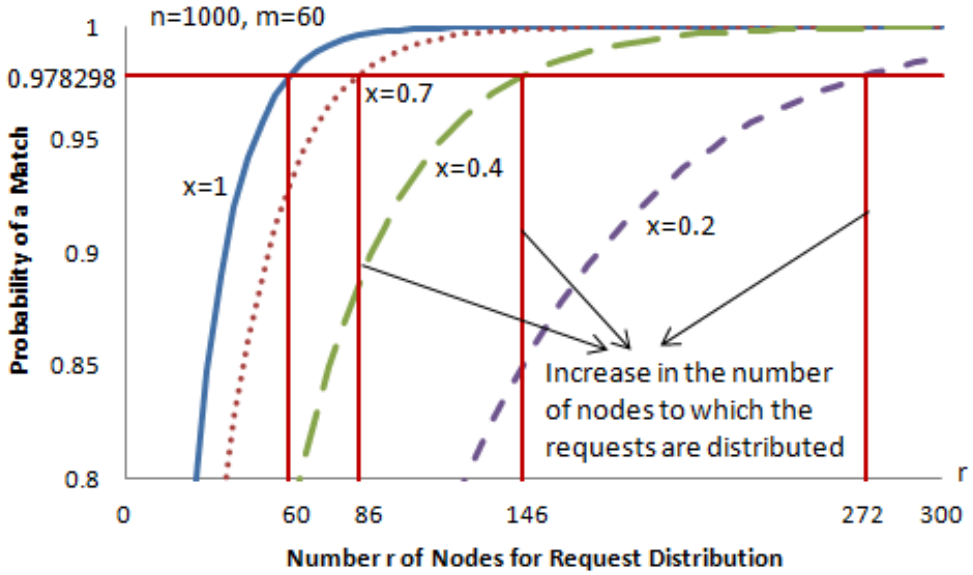

Figure 4.8: Based on the curves for $P(k \geq 1)$ for $x = 1.0, 0.7, 0.4, 0.2$, increase or decrease the number of nodes to which the requests are distributed to compensate for non-operational nodes to achieve the same probability of a match as when all of the nodes are operational.

$n = 1000$ and $m = r = 60$. From Equation (3.3), it computes the probability of one or more matches to obtain $y_0 = 0.978298 = P(k \geq 1)$.

If the detection algorithm determines from empirical evidence that $x' = 0.7$ then, from the calculated value of y_0, the defensive adaptation algorithm determines the value of r' corresponding to y_0 on the $x = 0.7$ curve. That is, it iteratively solves the equation $y_0 = P'(k \geq 1)$ with $n = 1000$, $m = 60$, $x' = 0.7$, and r' to obtain $r' = 86$.

Similarly, if the detection algorithm determines from empirical evidence that $x' = 0.4$ or $x' = 0.2$, the defensive adaptation algorithm determines that $r' = 146$ or $r' = 272$, respectively. Figure 4.8 shows that for $x' = 0.7$, $x' = 0.4$, and $x' = 0.2$, the appropriate numbers of nodes to which to distribute

a request to compensate for non-operational or subverted nodes are $r' = 86$, $r' = 146$, and $r' = 272$, respectively. These increases maintain approximately constant the proportion of operational nodes that receive the node's request. For $x = 1.0$, the proportion is 0.063; for $x = 0.7$, the proportion is 0.0602; for $x = 0.4$, the proportion is 0.0582; and for $x = 0.2$, the proportion is 0.0544.

4.3.1 Defensive Adaptation Algorithm

Thus, when the detection algorithm estimates that the number x' of operational nodes is less than 1.0, the defensive adaptation algorithm increases the number r' of nodes to which the requests are distributed, to achieve the same probability of a match as when $x' = 1.0$.

Pseudocode for the defensive adaptation algorithm is given in Figure 4.9.

4.4 Combining the Algorithms

4.4.1 Exponential Weighted Moving Average Method

The detection and defensive adaptation algorithms, described in this paper, use the exponential weighted moving average (EWMA) method. The EWMA method is used to average the unnormalized observed probabilities $P_i(k)$ for k matches over a sequence of requests, in order to smooth those probabilities and reduce the noise inherent in the individual samples.

getR(n, m, r, prevX, x')
1 $y_0 \leftarrow 1 - \frac{(n-m)}{(n)}\frac{(n-m-1)}{(n-1)} \cdots \frac{(n-m-r+1)}{(n-r+1)}$
2 $r' \leftarrow r$
3 if prevX $> x'$ then
4 Repeat
5 $r' \leftarrow r' + 1$
6 $y' \leftarrow 1 - \frac{(n-mx')}{(n)}\frac{(n-mx'-1)}{(n-1)} \cdots \frac{(n-mx'-r'+1)}{(n-r'+1)}$
7 until $y' > y_0$
8 else if prevX $< x'$ then
9 Repeat
10 $r' \leftarrow r' - 1$
11 $y' \leftarrow 1 - \frac{(n-mx')}{(n)}\frac{(n-mx'-1)}{(n-1)} \cdots \frac{(n-mx'-r'+1)}{(n-r'+1)}$
12 until $y' < y_0$
13 return r'

Figure 4.9: Method for finding the value of r' that maintains the same probability of a match when some of the nodes are non-operational as does the value of r when all of the nodes are operational.

With the EWMA method, the requester issues requests, collects responses, and computes the unnormalized observed probabilities $P_i(k)$ dynamically using a weighting factor c, as follows:

$$
\begin{aligned}
P_0(k) &= 0 && \text{for} \quad 1 \le k \le K \\
P_i(k) &= 1 + cP_{i-1}(k) && \text{if } i > 0 \text{ and request } i \\
& && \text{results in } k \text{ responses} \\
P_i(j) &= cP_{i-1}(j) && \text{if } i > 0,\ j \neq k \text{ and } 1 \le j \le K
\end{aligned}
$$

For example, for $n = 1000$, $m = 60$, $r = 60$, $c = 0.97$, $K = 60$ buckets, and 7 requests, the requester might obtain $2, 3, 5, 4, 2, 4, 3$ responses for those 7 requests. Using the EWMA method, we obtain the values of $P_i(k)$, $1 \le k \le 60$, $0 \le i \le 7$, shown in Table 4.2.

$k =$	1	2	3	4	5	6	...	60
$i = 0$	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	0.97	1	0	0	0	0	0
3	0	0.9409	0.97	0	1	0	0	0
4	0	0.9127	0.9409	1	0.97	0	0	0
5	0	1.8853	0.9127	0.97	0.9409	0	0	0
6	0	1.8287	0.8853	1.9409	0.9127	0	0	0
7	0	1.7739	1.8587	1.8827	0.8853	0	0	0

Table 4.2: Unnormalized Observed $P_i(k)$, $1 \leq k \leq 60$, $0 \leq i \leq 7$.

Pseudocode for the exponential weighted moving average method is given in Figure 4.10.

For the exponential weighted moving average method, we investigate values of the following parameters:

- c: The weighting factor used in the EWMA method
- d: The number of requests in the initial transient for the EWMA method, *i.e.*, the detection algorithm starts to estimate x' after d samples.

Determining an Appropriate Value of c

For the exponential weighted moving average method, if the value of c is too high, the accuracy will be high but the response time will also be high. In contrast, if the value of c is too low, the response time will be low but the accuracy will also be low.

Figure 4.11 shows the mean accuracy and the mean response time for $c = 0.91$, $c = 0.93$, $c = 0.95$, $c = 0.97$, and $c = 0.99$. These results were

```
EWMA(numResp,P_i,K,c)
1  for k ← 1 to K
2    if numResp = k then
3        P_i(k) ← 1 + cP_{i−1}(k)
4    else
5        P_i(k) ← cP_{i−1}(k)
6  return P_i
```

Figure 4.10: Exponential weighted moving average method to compute the unnormalized observed probabilities $P_i(k)$.

obtained using $n = 1000$, $r = 60$, $m = 60$, $d = 40$, $s = 5$, $t = 2$, $K = 15$, RangeA with $x = 1.0$, and RangeB with $x = 0.7$, 0.4 and 0.2, using the modified chi-squared statistic.

In the figure, we see that both the accuracy and the response time increase as c increases to 0.97. The reason is that when c is larger, the program waits longer to ensure that there is indeed a change in the value of x. We note that when $c = 0.97$, the accuracy is relatively high for all three values of x and that the accuracy decreases for $c > 0.97$. Moreover, we note that when $c = 0.97$, the response time is relatively low for all three values of x and that the response time increases sharply for $c > 0.97$. Thus, we determine that $c = 0.97$ is an appropriate value of c for our experiments.

Although $c = 0.97$ might be an appropriate value of c for our experiments, iTrust offers the user the option to choose a value of c for the particular network environment. The experiments presented here are based on the HTTP implementation of iTrust over the Internet. However, iTrust is also implemented for SMS-capable mobile phones [62] and for Wi-Fi Direct-capable mo-

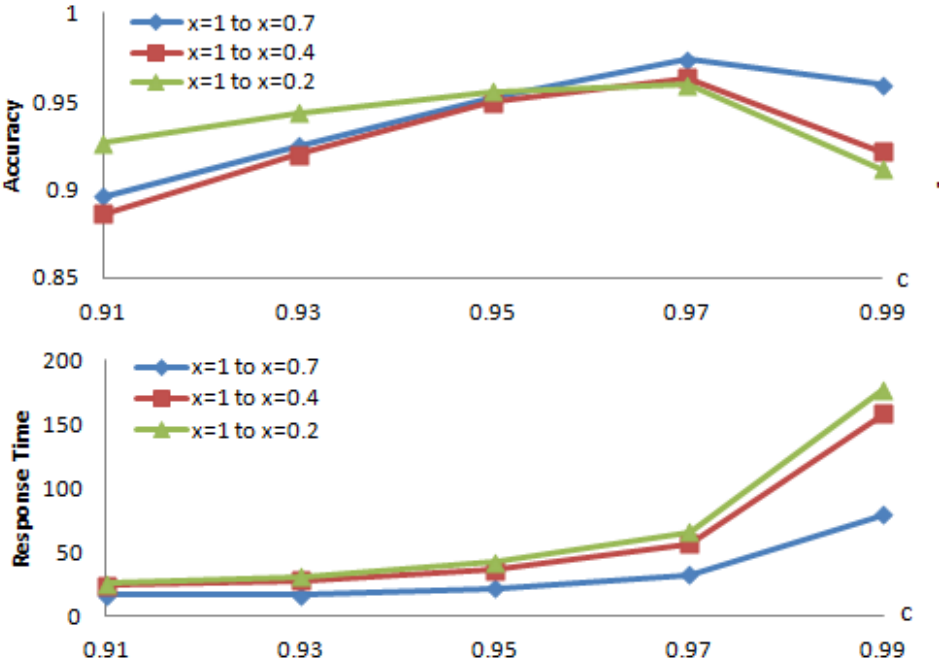

Figure 4.11: Mean accuracy and mean response time for various values of c with $n = 1000$, $m = r = 60$, $d = 40$, $s = 5$, $t = 2$, and $K = 15$.

bile phones [63]. In a mobile environment, the rate of change of the number of participating nodes might be greater, the cost of sending or receiving messages might be greater, and the rate of generating metadata or requests might be different than the corresponding rate over the Internet. Furthermore, the cost of smaller match probabilities might be viewed differently. Smaller values of c provide faster reactions to changes, and a higher risk of making mistakes. Consequently, different users, operating in different network environments with different objectives, might choose different values of c, and iTrust allows them to do so.

Determining an Appropriate Value of d

For the exponential weighted moving average method, we also determine an appropriate value of d, the number of requests in the initial transient before the program starts to estimate the value of x'. The program compares the value of x' against the value of x, and records the value of d when x' and x are the same. The program repeats this process 500 times, and then averages the values to obtain the mean value of d. We repeat this process for $x = 0.7$, $x = 0.4$, and $x = 0.2$. From these experiments, we decided to set $d = 40$ to make sure that the program waits long enough before it starts to estimate the value of x'.

4.4.2 Combined Algorithms

Pseudocode for the combined detection and defensive adaptation algorithms is given in Figure 4.12.

In line 1 to line 6, the variables are initialized. Line 7 starts an infinite loop where, in Line 8, the index i for the EWMA method is incremented.

Within the infinite loop, in line 9, the algorithm checks whether the value of x' has changed. If so, in line 10, the algorithm changes the value of r' based on the new value of x'. In line 11 to line 18, the algorithm calculates the normalized analytical (expected) probabilities $E_{1.0}, E_{0.7}, E_{0.4}, E_{0.2}$. In line 19, the algorithm sets $prevX$ to x'.

```
combinedAlgorithms(n, m, r, K, c, d)
1  r' ← r
2  x' ← 1.0
3  prevX ← x'
4  for k ← 0 to K
5    P_0(k) ← 0
6  i ← 0
7  while true do
8    i ← i + +
9    if prevX ≠ x' or i = 1 then
10       r' ← getR(n, m, r', prevX, x')
11       P ← findP(n, m, r', 1.0)
12       E_1.0 ← norm(P, K)
13       P ← findP(n, m, r', 0.7)
14       E_0.7 ← norm(P, K)
15       P ← findP(n, m, r', 0.4)
16       E_0.4 ← norm(P, K)
17       P ← findP(n, m, r', 0.2)
18       E_0.2 ← norm(P, K)
19       prevX ← x'
20   numResp ← getRequest(r')
21   P_i ← EWMA(numResp,P_{i-1},K,c)
22   if i > d then
23       O_i ← norm(P_i, K)
24       χ²_1.0 ← chiSquared(O_i, E_1.0, K)
25       χ²_0.7 ← chiSquared(O_i, E_0.7, K)
26       χ²_0.4 ← chiSquared(O_i, E_0.4, K)
27       χ²_0.2 ← chiSquared(O_i, E_0.2, K)
28       x' ← min(χ²_1.0, χ²_0.7, χ²_0.4, χ²_0.2)
```

Figure 4.12: Combined algorithms.

In line 20, the node issues a request to r' nodes selected at random. In line 21, the algorithm invokes the EWMA method to calculate the current unnormalized observed probability array P_i. In line 22, the algorithm checks whether i is greater than the initial transient d during which it does not make

estimates of x'. If so, in line 23, the algorithm normalizes the elements of the array P_i to obtain the normalized observed probability array O_i. In line 24 to line 27, the algorithm calculates the chi-squared value for the normalized observed probabilities O_i and the analytical (expected) probabilities $E_{1.0}$, $E_{0.7}$, $E_{0.4}$, $E_{0.2}$. In line 28, the algorithm estimates the value of x' based on the smallest value of χ^2 it obtained using the normalized observed probabilities O_i and the normalized expected probabilities $E_{1.0}, E_{0.7}, E_{0.4}, E_{0.2}$.

Then, the algorithm goes back to line 7. If it finds in line 9 that the value of x' has changed, it gets a new value of r' in line 10. These steps are repeated indefinitely.

4.5 Experimental Evaluation

To evaluate the iTrust detection and defensive adaptation algorithms, we used an emulation of iTrust based on the HTTP implementation of iTrust described in Chapter 3.2. In the emulation, we have multiple virtual hosts installed on a single Apache Web server, where each virtual host represents a node in the iTrust network. Each node has a separate SQLite database that resides on the Apache Web server, where it stores queries and resource information.

Before we start the emulation program, we initialize the values of n, m, r, and x. The program clears the node's resources and databases, and then

n	100	250	1000	2500	10000
$m = r \approx \lceil 2\sqrt{n} \rceil$	20	30	60	100	200

Table 4.3: Number n of nodes and number $m = r$ of nodes to which the metadata and the requests are distributed.

adds the nodes to the membership set. For each source node, iTrust creates the metadata for the document that the node wishes to share, and distributes the metadata to m randomly selected nodes in the membership set. Then, iTrust distributes requests to r randomly selected nodes in the membership set. After waiting for five seconds, the emulation program records a *sample*, *i.e.*, the number of nodes that found a match and returned a response.

4.5.1 Number of Nodes

In the previous sections, we described the results we obtained for $n = 1000$ nodes. Now, we investigate whether the accuracy or the response time change for different values of n. In addition, we investigate whether the value of c should be changed for different values of n. We use the modified chi-squared statistic in these experiments.

In particular, we investigate the following five values of n: $n = 100$, $n = 250$, $n = 1000$, $n = 2500$, and $n = 10000$. For each of these values of n, we set $m = r \approx \lceil 2\sqrt{n} \rceil$ as shown in Table 4.3.

Figure 4.13 shows the mean accuracy and the mean response time for $c = 0.91$, $c = 0.93$, $c = 0.95$, $c = 0.97$, and $c = 0.99$, as a function of the number n of nodes. In these experiments, we use $d = 40$, $s = 5$, $t = 2$, and

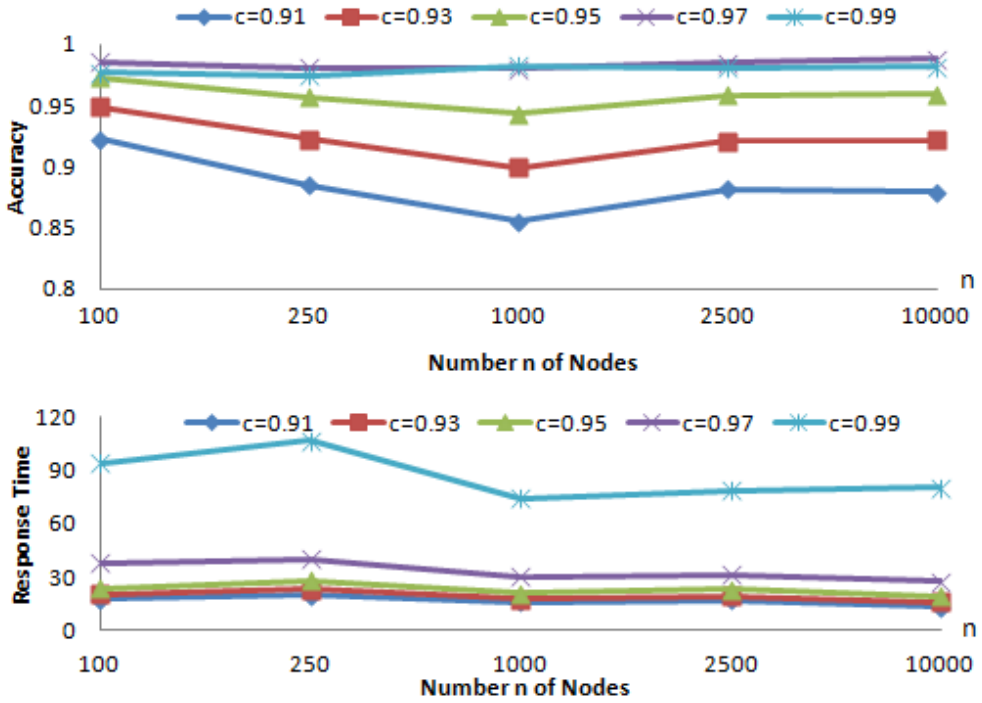

Figure 4.13: Mean accuracy and mean response time for various values of c with $d = 40, s = 5, t = 2, K = 15$, RangeA with $x = 1.0$, and RangeB with $x = 0.7$.

$K = 15$. We consider RangeA with $x = 1.0$ and RangeB with $x = 0.7$. We ran the program 1000 times and averaged the results to obtain the mean accuracy and the mean response time.

The results shown in the figure confirm the selection of $c = 0.97$ for all of these values of n. Smaller values of c result in lower accuracy, whereas $c = 0.99$ results in higher response time. We repeated the experiment with RangeB set to $x = 0.4$ and then $x = 0.2$, and the results were similar.

Figure 4.14 shows the mean accuracy and the mean response time for the five different values of n ($n = 100, 250, 1000, 2500, 10000$) and for RangeA with $x = 1.0$ and for RangeB with three different values of x ($x = 0.7, 0.4, 0.2$).

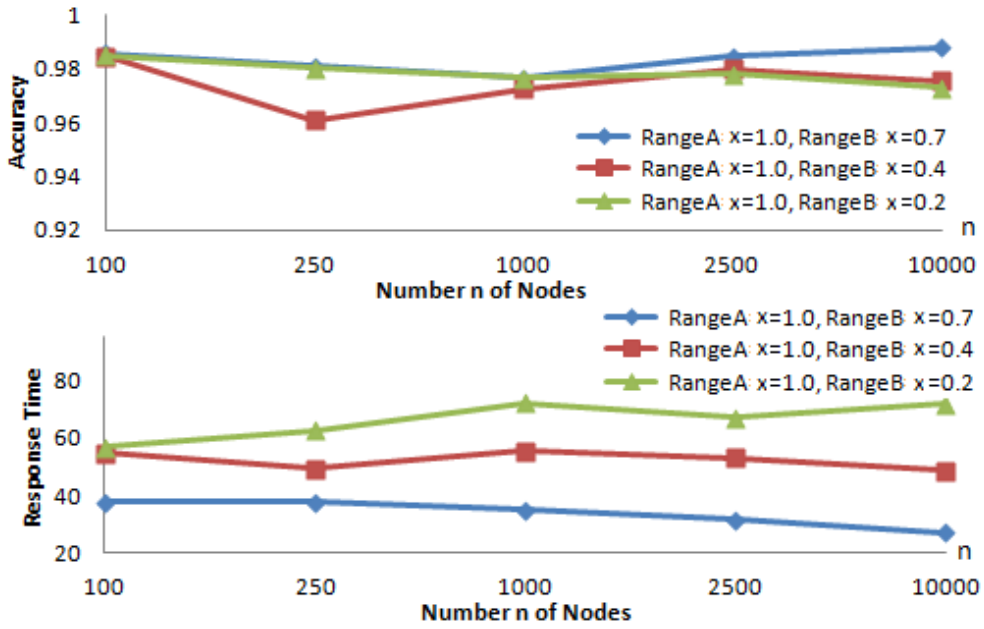

Figure 4.14: Mean accuracy and mean response time for various values of x with $c = 0.97$, $d = 40$, $s = 5$, $t = 2$, and $K = 15$.

From this figure, we see that the values of the accuracy for all of these values of n and for all of these values of x are quite high. In addition, for a given value of x, the response times are generally the same for all values of n. The response time for RangeA with $x = 1.0$ and RangeB with $x = 0.7$ is the smallest, followed by RangeB with $x = 0.4$, and then RangeB with $x = 0.2$. Overall, the response times in all cases are within an acceptable range.

Lastly, based on the results shown in Figure 4.14, we note that for all of these values of n, the accuracy and the response time are similar to the previous results for $n = 1000$.

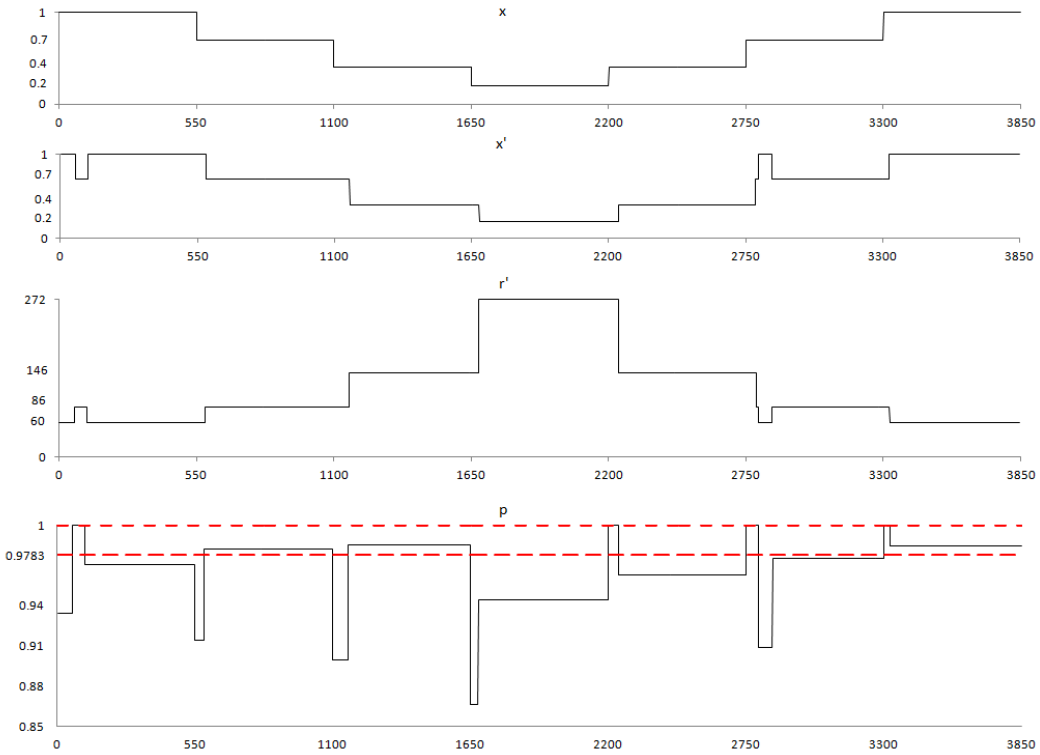

Figure 4.15: Four graphs (x, x', r', p) for the detection and defensive adaptation algorithms, where x is the actual value of x for the iTrust network. x' is the estimated value calculated using the detection algorithm. r' is the number of nodes to which the requests are distributed calculated using the defensive adaptation algorithm. p is the match probability for intervals between a change in the value of x or a change in the value of x'.

4.5.2 Extended Scenario

Now we consider an extended scenario in which $n = 1000$, $m = 60$, $c = 0.97$, $d = 40$, $s = 5$, $t = 2$, and $K = 15$ where, initially, $r = 60$ and $x = 1.0$. We use the modified chi-squared statistic in these experiments.

In Figure 4.15, the graph at the top represents the actual value of x in the iTrust network. The scenario starts with 550 requests where $x = 1.0$, and then decreases to $x = 0.7$, decreases to $x = 0.4$, decreases to $x = 0.2$, increases

to $x = 0.4$, increases to $x = 0.7$, and then finally increases to $x = 1.0$, with another 550 requests for each increase or decrease in the value of x.

The second graph represents the estimated value of x'. The third graph shows the change in the value of r', corresponding to a change in the value of x'. The fourth graph, at the bottom, shows the match probability p (*i.e.*, the probability of one or more matches) for intervals between a change in the value of x or a change in the value of x'.

First, we note that when the value of x decreases, there is a short interval during which this change is not yet reflected in the value of x'. Therefore, the match probabilities in this interval decrease, because the algorithm has not yet increased r'. Similarly, when the value of x increases, there is a short interval during which this change is not yet reflected in the value of x'. Therefore, the match probabilities in this interval increase slightly.

Next, we observe that when the value of x' is less than the value of x (as at request 70), the value of r' increases, resulting in the match probabilities being slightly larger during this short interval. On the other hand, when the value of x' is greater than the value of x (as at request 2800), the value of r' is less and the match probabilities are smaller during this short interval.

In addition, we note that the chance of incorrectly estimating the value of x' is quite small. Furthermore, we observe that the mean response time and the mean time to recover from an incorrect estimate of x' are quite small.

In particular, the mean response time is about 38.33, and the mean time to recover from an incorrect estimate of x' is about 40.

Lastly, the graph at the bottom of the figure shows that the match probabilities for the intervals are bounded by changes in the value of x or changes in the value of x'. We see that most of the match probabilities are close to the target value $0.978298 = P(k >= 1)$ calculated from Equation (3.3) with $n = 1000$, $m = 60$, $r = 60$, and $x = 1.0$. Note the short intervals between a change in the value of x and the subsequent change in the value of x', and the changes in the match probabilities in those intervals.

Overall, the experiments demonstrate that the detection algorithm is effective in detecting non-operational or subverted nodes, and that false alarms rarely occur. Moreover, the defensive adaptation algorithm is effective in defending against malicious attacks, because it adjusts the number of nodes to which the requests are distributed to maintain a high probability of a match. With appropriate choices of the parameters, the accuracy is high and the response time is low.

4.6 Summary

In this chapter, we have presented novel statistical algorithms for detecting and defending against malicious attacks in the iTrust information retrieval network. The algorithms employ the exponential weighted moving average

method and the chi-squared test to calculate the empirical probabilities for the numbers of responses to a request. The detection algorithm compares the normalized empirical probabilities against the normalized analytical probabilities for the numbers of matches, to estimate the proportion of nodes that are subverted or non-operational. The defensive adaptation algorithm increases or decreases the number of nodes to which the requests are distributed, to maintain a high probability of a match. Extensive experimental evaluations demonstrate the effectiveness of the algorithms for detecting malicious attacks and defending against them.

Chapter 5

Membership Management for iTrust

In this chapter, we present adaptive and non-adaptive membership management protocols for the iTrust information retrieval network. The iTrust *membership* consists of the nodes that participate in the iTrust network (also referred to as the *participating nodes*). Each node has a *local view* of the membership, which approximates the actual membership. Nodes can *join* the membership at any time; likewise, nodes can *leave* the membership at any time, either voluntarily or by crashing. A requesting node discovers a newly joined node when it receives a response from a node that included that newly joined node in its response. Likewise, a requesting node detects a non-operational node by not receiving a response from that node before a timeout, or by receiving an error code from TCP.

An extensive literature on membership exists, but most of that work is not relevant to iTrust. The membership protocols for iTrust are simpler and less

costly than prior membership protocols [12], because iTrust does not need to achieve an agreed accurate membership based on a consensus algorithm [9].

In this chapter, we compare the performance of the non-adaptive and adaptive membership protocols, with respect to membership accuracy, match probability, response time, and message cost, for various parameter values when the membership churn is high and when the membership is stable.

5.1 iTrust Membership Protocols

The membership protocols for iTrust do not aim to achieve an agreed accurate membership. Rather, they allow each member to have its own local view of the membership, but aim to keep that local view close to the actual membership. In this chapter, we present non-adaptive membership protocols for iTrust, and adaptive membership protocols for iTrust that use the exponential weighted moving average algorithm. We investigate the effectiveness of the membership protocols when the membership has a lot of churn and also when the membership is stable.

5.1.1 Joining the Membership

To join the membership, a node must first obtain the address of a bootstrapping node. To obtain the address of the bootstrapping node, the node

uses mechanisms outside the iTrust network, such as conventional Web search, e-mail, Twitter, printed publications, *etc.*

The steps involved when a node joins the membership are given below, and are illustrated in Figure 5.1.

1. A joining node contacts the bootstrapping node to obtain the bootstrapping node's current view of the membership.

2. The joining node then publishes its joining the membership to a subset of nodes randomly chosen from the current view of the membership it obtained from the bootstrapping node.

3. The randomly chosen nodes then add the new node to their local views of the membership.

Another node learns about the new node when it receives a response from a node that is aware of the new node.

5.1.2 Leaving the Membership

A node may leave the membership either voluntarily, or because it is faulty or disconnected. The steps involved in leaving the membership are simple:

1. To leave the membership, a node just leaves, without publishing that it is leaving.

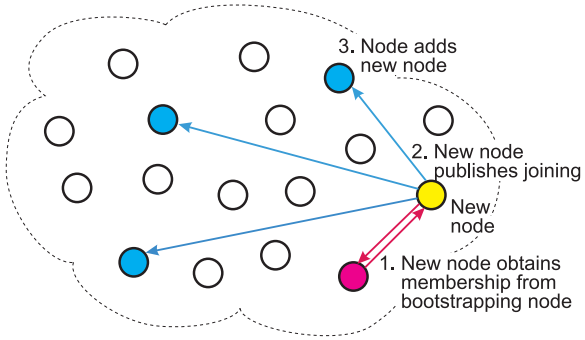

Figure 5.1: A node joins the membership by first contacting a bootstrapping node to obtain that node's current view of the membership, and then publishing its joining to randomly chosen nodes in that view.

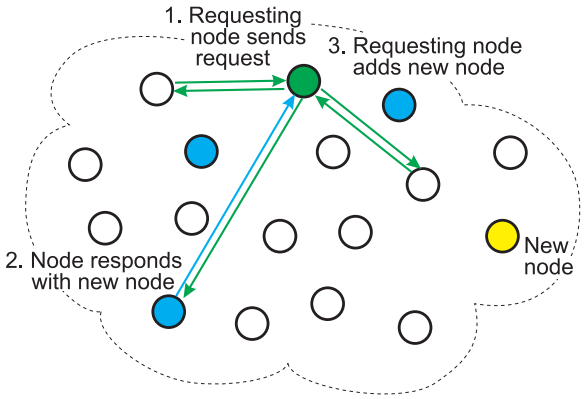

Figure 5.2: A requesting node distributes a request to nodes randomly chosen from its view of the membership. A node that receives the request returns the node(s) that it recently added to its view. A matching node also returns the URL of the document to the requesting node.

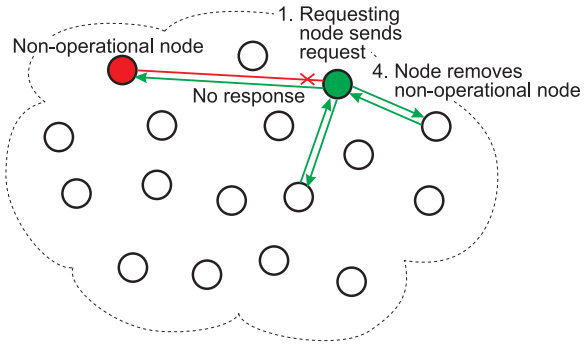

Figure 5.3: The requesting node does not receive a response to its request from a non-operational node. The requesting node sees a timeout expire or receives an error code from TCP, and then removes that node from its view.

Over time, each node individually discovers the departure of nodes when it sends requests to nodes that do not respond. It is not appropriate to allow a node to publish the departure of another node, because doing so might enable a malicious node to cause the removal of many nodes from the local views of other nodes.

5.1.3 Updating the Membership

In the messaging protocol described in Section 3.1, the requesting node expects to receive response messages only from the matching nodes. Other nodes that do not have a match are not required to send a response to the requesting node. We now modify that messaging protocol to enable a requesting node to detect non-operational (leaving) nodes and to discover newly joining nodes from the responses to its requests.

Now, the requesting node expects each node to which it sent a request to respond to its request with its recently joined member(s), regardless of whether or not that node has a match. Thus, a matching node sends in its response to the requesting node not only the URL of the document at the source node but also its recently joined member(s). If it does not have a match, the node simply responds to the requesting node with its recently joined member(s). Consequently, the requesting node discovers not only the URLs of the documents, but also newly joined nodes through the responses to its requests.

If the requesting node does not receive a response from a node within a timeout period or it receives an error code from TCP, then the non-responding node is considered to have left the membership voluntarily or to be faulty, and the requesting node removes that node from its view of the membership.

The steps involved in updating a requesting node's view of the membership are given below, and are illustrated in Figures 5.2 and 5.3.

1. A requesting node distributes its request to a subset of nodes randomly chosen from its local view of the membership.

2. A node that receives the request compares the keywords in the request with the metadata it holds. If it finds a match, the matching node responds to the requesting node with a message that contains the URL of the associated information and also its recently joined member(s). A node that does not find a match still responds to the requesting node with a message that contains its recently joined member(s).

3. When the requesting node receives the responses, it adds the new members obtained from the other nodes to its view of the membership.

4. If the requesting node does not receive a response from a node to which it sent a request before a timeout occurs, or if it receives an error code from TCP, then the non-responding node is considered to have left the membership voluntarily or to be faulty, and the requesting node removes that node from its view of the membership.

If the requesting node is also a source node then, after receiving the responses to its request, it distributes its metadata, together with the URL, for the information, to additional nodes according to the following steps:

1. The requesting (source) node calculates the number of nodes to which it needs to distribute its metadata, based on its current view of the membership.

2. Next, the requesting node subtracts the number of nodes to which it previously distributed metadata from the just calculated number.

3. Finally, the requesting node distributes its metadata to that many additional nodes, randomly chosen from its current view of the membership, but to which it had not sent the metadata previously.

For example, suppose that a requesting node currently has $N = 1024$ nodes in its local view of the membership. It distributes its request to $R = 2\sqrt{1024} = 64$ randomly chosen nodes. Suppose further that only 58 nodes reply to the requesting node. From the responses to its request, the requesting node detects that there are $64 - 58 = 6$ non-operational nodes. Suppose that as a result of receiving the responses from the 58 nodes, the requesting node adds 40 new nodes to its view of the membership. Consequently, the requesting node now has $N = 1024 - 6 + 40 = 1058$ nodes in its view of the membership. If the requesting node is also a source node, then it distributes its metadata to

$2\sqrt{1058} - 2\sqrt{1024} \sim 65 - 64 = 1$ more node randomly chosen from its new view of the membership.

5.2 Foundations

5.2.1 Environmental Variables

Membership churn refers to nodes joining and leaving the membership, and is represented by the following rates:

- JR: The Joining Rate, the number of nodes that join the membership per time unit. For example, $JR = 50$ means that 50 nodes join the membership per time unit.
- LR: The Leaving Rate, the number of nodes that leave the membership per time unit. For example, $LR = 50$ means that 50 nodes leave the membership per time unit.

When the membership has a lot of churn, both JR and LR are high. When the membership is stable, both JR and LR are low. These rates are an important consideration for the membership protocols. A node can not control or alter JR and LR, but it can adjust its requesting rate, as discussed in Section 5.5.

5.2.2 Parameters for the Membership Protocols

The parameters for the membership protocols are:

- N: The number of nodes in a node's local view of the membership.

- $LastJ$: The Last Joined members, the number of recently joined members that a node may report to the requesting node. For example, $LastJ = 2$ allows a node to report its two most recently joined members to the requesting node.

- Try: The number of times that a requesting node sends its request message, in an attempt to receive responses from $2\sqrt{N}$ nodes. Because some request messages might be sent to non-operational nodes, a requesting node might need to try several times before it receives responses from $2\sqrt{N}$ nodes.

- $TryMax$: The Maximum Try value, *i.e.*, the maximum number of times that a requesting node is allowed to try to send its request message.

- RR: The Requesting Rate, the number of times a node sends a request message to $R = 2\sqrt{N}$ nodes per time unit. For example, $RR = 10$ means that a node sends 10 distinct request messages per time unit, each of which is sent to $R = 2\sqrt{N}$ nodes.

- $RRMin$: The Minimum Requesting Rate, the minimum rate at which a node is allowed to make requests.

- $RRMax$: The Maximum Requesting Rate, the maximum rate at which a node is allowed to make requests.

- c: The weighting factor of the exponential weighted moving average method used by the membership protocols.

When a node joins the membership, it obtains the values of $LastJ$, $TryMax$, $RRMin$, $RRMax$, and c. These parameters are tunable, and their tunability is discussed in Section 5.3.

5.2.3 Performance Metrics

The performance metrics for the membership protocols are defined in terms of the following quantities:

- L: The number of leaving nodes that a requesting node has not detected.
- J: The number of joining nodes that a requesting node has not discovered.
- I: The number of nodes in the intersection of the requesting node's current view of the membership and the actual membership.

The requesting node's current view of the membership consists of $I + L$ nodes, whereas the actual membership consists of $I + J$ nodes. Figure 5.4 illustrates the quantities I, L and J.

The performance metrics for the membership protocols are:

- LND: The Leaves Not Detected, the proportion of leaving (non-operational) nodes in its current view that a requesting node has not detected at a

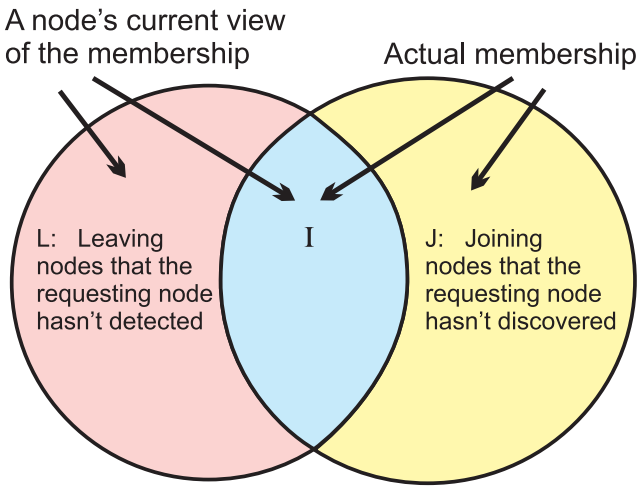

Figure 5.4: A node's current view of the membership vs. the actual membership.

particular time, defined by:

$$LND = \frac{L}{I + L} \tag{5.1}$$

- JND: The Joins Not Discovered, the proportion of newly joined nodes in the actual membership that a requesting node has not discovered at a particular time, defined by:

$$JND = \frac{J}{I + J} \tag{5.2}$$

- MA: The Membership Accuracy, the number of nodes in a node's current view that are in the actual membership divided by that number of nodes plus the number of leaving nodes not detected plus the number of joining nodes not discovered, defined by:

$$MA = \frac{I}{I + L + J} \tag{5.3}$$

Note that $MA = I/(I + L + J) = 1 - (L + J)/(I + L + J)$, where $L + J$ is the number of leaving nodes that the node has not detected plus the number of newly joining nodes that the node has not discovered and, therefore, $(L + J)/(I + L + J)$ represents the inaccuracy in the node's view of the membership.

- MP: The Match Probability of one or more responses for a request, averaged over all requesting nodes.

- RT: The Response Time required for a request, from the time that a node starts sending its request to the other nodes until it finishes receiving the responses to that request from those nodes, including responses for multiple tries, averaged over all requesting nodes.

- MC: The Message Cost per node per time unit, calculated as an average over all nodes over time.

5.2.4 Measured Values

Our membership protocols for iTrust use the following measured values:

- $Left$: The number of nodes that a requesting node detected are non-operational (have left the membership) since its last request.

- $Joined$: The number of nodes that the requesting node discovered have joined the membership since its last request.

- $NumNodes$: The number of nodes to which the requesting node sent its request, depending on the particular membership protocol.

Using these measured values, a node calculates a *Churn Estimator* for each request when it finishes receiving the responses to that request, defined as follows:

- CE: The Churn Estimator, an estimate of the churn in the membership (leaves and joins) obtained by random sampling, given by:

$$CE = \frac{Left + Joined}{NumNodes} \tag{5.4}$$

The Churn Estimator is used in the membership protocols to adapt the requesting rate RR, and is an input to the exponential weighted moving average algorithm.

5.2.5 Exponential Weighted Moving Average Algorithm

The adaptive membership protocols for iTrust uses the exponential weighted moving average (EWMA) algorithm to process a sequence of estimated values for the Churn Estimator CE, to smooth the estimated values and reduce the noise inherent in the individual data samples.

A requesting node issues requests (queries), collects responses, detects non-operational nodes, discovers newly joined nodes, and then computes the estimated value CE. To compute the estimated value CE, it applies the EWMA

```
EWMA(v, c, s)
1  if (t == 0) then s ← v
2  else s ← c × v + (1 − c) × s
3  return s
```

Figure 5.5: Pseudocode for the EWMA algorithm.

algorithm defined by:

$$\begin{aligned} s_1 &= v_1 \\ s_t &= c \times v_t + (1-c) \times s_{t-1} \quad \text{if } t > 1 \end{aligned} \tag{5.5}$$

where:

- c: The smoothing factor, $0 \leq c \leq 1$
- s_t: The output of the EWMA algorithm at time t.

For example, if $c = 0.7$ (which is the value of c we use in our experiments), the current value $v_t = 0.85$ and the previous value $s_{t-1} = 0.95$, then the new value s_t is:

$$\begin{aligned} s_t &= c \times v_t + (1-c) \times s_{t-1} \\ &= 0.7 \times 0.85 + (1-0.7) \times 0.95 \\ &= 0.88 \end{aligned}$$

The pseudocode for the EWMA algorithm is given in Figure 5.5.

Although $c = 0.7$ is an appropriate value of c for our experiments, iTrust offers the user the option to choose a value of c for the particular network environment. Different users, operating in different network environments with

different objectives, might choose different values of c, and iTrust allows them to do so.

5.2.6 Experimental Methodology

To evaluate the membership protocols, we performed experiments using a simulation of iTrust. In the simulation, we can control the leaving rate LR and the joining rate JR, which a real-world deployment would not allow us to do. Moreover, in the simulation, we can compare a node's current view of the membership against the actual membership.

Before we start the simulation program, we initialize the value of N, the number of nodes in the actual membership. The program adds all of the nodes to each node's view of the membership, so that each node has the complete initial membership. Then, at each time step, nodes might join the membership, leave the membership, and send requests. Different nodes might have different views of the membership, and different nodes might make requests at different rates. At each time step, the program computes the performance metrics.

5.3 Non-Adaptive Membership Protocol

The Non-Adaptive Membership Protocol implements the membership protocol for iTrust, described in Section 5.1, which involves requesting nodes updating their local views of the membership, as other nodes join and leave

```
NonAdaptive(N, RR)
1 while true do
2   nextRequestTime ← time + (timeunit/RR)
3   wait until (time == nextRequestTime)
4   R ← 2 × sqrt(N)
5   responses ← makeRequests(view, R)
6   responded ← 0
7   for (j ← 0 to R) do
8     if (responses[j].noResponse) then
9       removeNode(view, responses[j].node)
10      N ← N − 1
11    else
12      responded ← responded + 1
13      if (responses[j].recent) then
14        isNew = addNode(view, responses[j].recentNode)
15        if (isNew) then
16          N ← N + 1
```

Figure 5.6: Non-Adaptive Membership Protocol.

the membership. The pseudocode for the Non-Adaptive Membership Protocol is given in Figure 5.6. The inputs for the Non-Adaptive Membership Protocol are N and RR. Here N is the number of nodes in the node's local view of the membership and RR is the node's requesting rate.

Line 1 starts an infinite loop where, in line 2, $nextRequestTime$ is set to the current *time* plus $timeunit/RR$, which is the time when the node sends its next request. As time passes, *time* is automatically incremented (not shown in the pseudocode); moreover, *timeunit* is the length of the time unit. In line 3, the algorithm waits until the current *time* reaches the $nextRequestTime$.

In line 4, the algorithm sets the number R of nodes to which the node sends its request message to $2\sqrt{N}$, where N is the number of nodes in the node's

current view of the membership. In line 5, the node sends its request message to R nodes, and waits for responses from those nodes. In line 6, the algorithm sets *responded* to 0.

In line 7, the algorithm iterates through the *responses* array. In line 8, the algorithm checks whether the node received a response from node j. If not, in line 9, it removes the non-responsive node from the node's view and, in line 10, it decrements the number N of nodes in that view. Otherwise, in line 12, the algorithm increments *responded*. In line 13, the algorithm checks whether node j's response contains a recent node. If so, in line 14, it invokes $addNode()$ to add the recent node to its view. The procedure $addNode()$ returns a boolean $isNew$ to indicate whether the recent node was already present in the node's view. In line 15, the algorithm checks whether the recent node is indeed new and, if so, in line 16, it increments the number N of nodes in the node's view. Control then returns to continue the iteration through the *responses* array in line 7.

When the algorithm finishes iterating through the *responses* array, it goes back to line 1, and repeats these steps indefinitely.

5.3.1 Investigation of $LastJ$

For the Non-Adaptive Membership Protocol, we investigate $LastJ$, the number of newly joined nodes that a responding node may report to a requesting node, and its effect on LND (the proportion of leaving nodes that

the requesting node has not detected) and JND (the proportion of newly joined nodes that the requesting node has not discovered).

In the Non-Adaptive Membership Protocol, a requesting node distributes its request to $R = 2\sqrt{N}$ nodes chosen at random from its local view of the membership. Initially, we required those nodes to return their entire views to the requesting node, and the requesting node to update its view accordingly. The problem is that the requesting node obtains some non-operational nodes from other nodes that have not yet detected that those nodes are non-operational. Thus, the requesting node adds back into its view too many non-operational nodes, including nodes that it recently removed. The requesting node can not distinguish between such non-operational nodes that left the membership, and nodes that left the membership and recently re-joined.

Several possible solutions to this problem exist. One solution is that once a requesting node has obtained the views from the other nodes, it sends a "verify" message to confirm whether those nodes are indeed operational. Such a solution consumes a lot of network bandwidth. Another less costly solution is to require the $2\sqrt{N}$ nodes to return, to the requesting node, their "most recently joined members," rather than their entire views. We adopt the latter solution and investigate how $LastJ$ affects LND and JND.

We consider a scenario where $N = 1024$ nodes with a high leaving rate ($LR = 300$), a high joining rate ($JR = 300$), and a low requesting rate ($RR = 10$). Figure 5.7, 5.8, and 5.9 show the graphs for LND and JND over time for

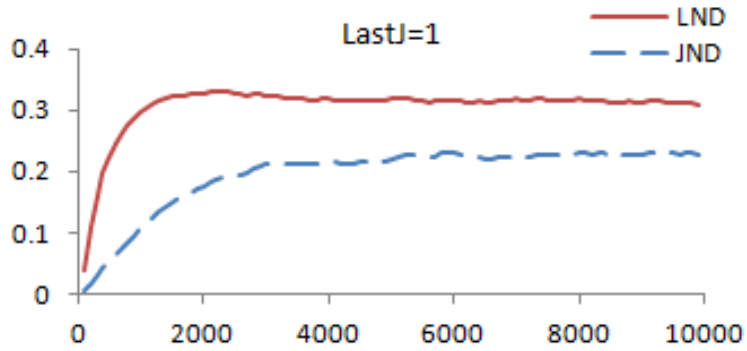

Figure 5.7: Graphs for the Non-Adaptive Membership Protocol, showing LND and JND over time for $LastJ = 1$, where $N = 1024$ initially, $LR = 300$, $JR = 300$, and $RR = 10$.

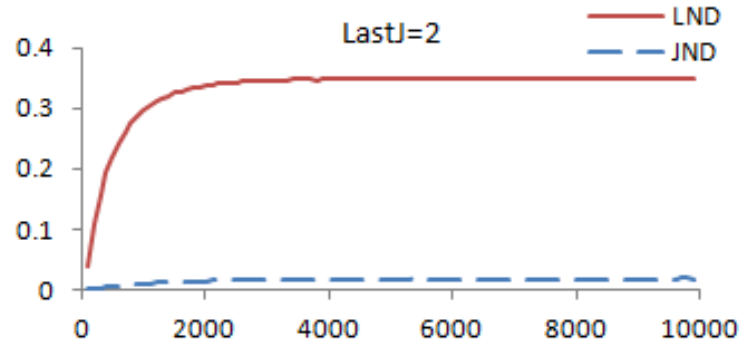

Figure 5.8: Graphs for the Non-Adaptive Membership Protocol, showing LND and JND over time for $LastJ = 2$, where $N = 1024$ initially, $LR = 300$, $JR = 300$, and $RR = 10$.

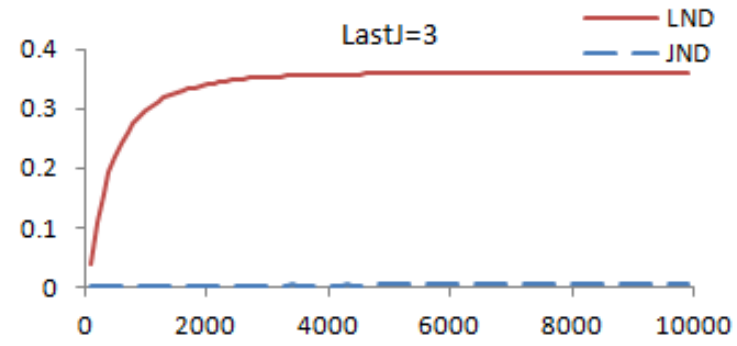

Figure 5.9: Graphs for the Non-Adaptive Membership Protocol, showing LND and JND over time for $LastJ = 3$, where $N = 1024$ initially, $LR = 300$, $JR = 300$, and $RR = 10$.

this scenario. Increasing $LastJ$ from $LastJ = 1$ in the left graph to $LastJ = 2$ in the middle graph results in a decrease in JND but an increase in LND. Increasing $LastJ$ from $LastJ = 2$ in the middle graph to $LastJ = 3$ in the right graph results in a slight decrease in JND and little change in LND.

Thus, increasing $LastJ$ definitely helps the requesting node to discover more joining nodes as it issues more requests. However, increasing $LastJ$ also causes the requesting node to add back into its local view too many non-operational (leaving) nodes. Setting $LastJ = 2$ increases LND with worse results than setting $LastJ = 1$. Similarly, setting $LastJ = 3$ does not help to decrease LND. We conclude that $LastJ = 1$ is an appropriate value to use for our further experiments.

5.4 Retry R Membership Protocol

As we have just seen, increasing $LastJ$ does not help to detect leaving (non-operational) nodes. Thus, we investigate other methods to reduce LND, the proportion of leaving nodes that a requesting node has not detected.

When a node distributes a request message to $2\sqrt{N}$ nodes, it might not receive $2\sqrt{N}$ responses for its request, because non-operational (leaving) nodes do not respond. Thus, first, we investigate a retry method that allows the requesting node to distribute its request message to more than $2\sqrt{N}$ nodes until it receives $2\sqrt{N}$ responses. We call this protocol the Retry R Membership

```
RetryR(N, TryMax, RR)
1 while true do
2   nextRequestTime ← time + (timeunit/RR)
3   wait until (time == nextRequestTime)
4   R ← 2 × sqrt(N)
5   resRec ← 0
6   Try ← 1
7   while ((Try <= TryMax) and (resRec < R)) do
8     responses ← makeRequests(view, R − resRec)
9     responded ← 0
10    for (j ← 0 to (R − resRec)) do
11      if (responses[j].noResponse) then
12        removeNode(view, responses[j].node)
13        N ← N − 1
14      else
15        responded ← responded + 1
16        if (responses[j].recent) then
17          isNew = addNode(view, responses[j].recentNode)
18          if (isNew) then
19            N ← N + 1
20    resRec ← resRec + responded
21    Try ← Try + 1
```

Figure 5.10: Retry R Membership Protocol.

Protocol. The pseudocode for the Retry R Membership Protocol is given in Figure 5.10. The inputs for the Retry R Membership Protocol are N, $TryMax$, and RR. Here, N is the number of nodes in the node's local view of the membership, $TryMax$ is the number of times a node is allowed to try to obtain $2\sqrt{N}$ responses, and RR is the node's requesting rate.

Line 1 starts an infinite loop where, in line 2, $nextRequestTime$ is set to the current $time$ plus the time $timeunit/RR$ until the next request. In line 3, the algorithm waits until the current $time$ reaches $nextRequestTime$.

In line 4, the algorithm sets the number R of nodes to which the node send its request message to $R = 2\sqrt{N}$, where N is the number of nodes in the node's current view of the membership. In line 5 and line 6, the algorithm sets the number $resRec$ of responses received to zero and the number Try of tries to one, and then starts the while loop in line 7. Then, in line 8, the node sends its request message to $R - resRec$ nodes, and waits for responses from those nodes. In line 9, the algorithm sets *responded* to 0.

In line 10, the algorithm iterates through the *responses* array. In line 11, the algorithm checks whether the node received a response from node j. If not, in line 12, it removes the non-responsive node from the node's view and, in line 13, it decrements the number N of nodes in that view. Otherwise, in line 15, it increments *responded*. In line 16, the algorithm checks whether j's response contains a recent node. If so, in line 17, it invokes $addNode()$ to add the recent node to the node's view. The procedure $addNode()$ returns a boolean $isNew$ to indicate whether the recent node was already present in the node's view. In line 18, the algorithm checks whether the recent node is indeed new and, if so, in line 19, it increments the number N of nodes in the node's view. Control then returns to continue the iteration through the *responses* array in line 10.

After it has finished iterating through the *responses* array, in line 20, the algorithm increases $resRec$ by *responded*, the number of responses in the *responses* array and, in line 21, the algorithm increments Try. Control then

returns to the while loop in line 7 to determine whether Try is less than or equal to $TryMax$ and $resRec$ is less than R. If both of those conditions are satisfied, the algorithm goes through the while loop again.

Otherwise, the algorithm goes back to line 1, and repeats these steps indefinitely.

5.4.1 Investigation of Try

We now investigate values of the variable Try, the number of times that a requesting node tries to send its request message, in order to receive $2\sqrt{N}$ responses. For example, $Try = 1$ means that a requesting node sends its request message to $R = 2\sqrt{N}$ nodes regardless of the number of responses that it receives. $Try = 2$ means that a requesting node tries a second time and sends its request message to $Left$ nodes, where $Left$ nodes did not respond on its first Try and, similarly, for $Try = 3$. $Try = \infty$ means that a requesting node sends its request message repeatedly until it receives responses from $2\sqrt{N}$ nodes for that particular request message.

Table 5.1 shows the membership accuracy MA, match probability MP, response time RT, and message cost MC for $Try = 1, 2, 3, \infty$, where $N = 1024$ initially, $LastJ = 1$, $LR = 300$, $JR = 300$, and $RR = 10$ for the Retry R Membership Protocol.

We see that as Try is increased from $Try = 1$ to $Try = 2$, both MA and MP are greatly increased but, when Try is further increased to $Try = 3$, there

Try	1	2	3	∞
MA	0.5966	0.6821	0.6986	0.7048
MP	0.9345	0.9817	0.9865	0.9864
RT	6.0	11.9274	17.6573	24.1682
MC	3.8552	5.1538	5.4527	5.5598

Table 5.1: Retry R with $Try = 1, 2, 3, \infty$.

is not much increase in either MA or MP. We also see that both the response time RT and the message cost MC increase as Try is increased. To obtain a substantial increase in the membership accuracy and the match probability with a reasonable increase in the response time and the message cost, we use $Try = 2$ in our further experiments.

5.5 Adaptive RR Membership Protocol

The next membership protocol we consider adjusts the requesting rate RR, based on the value of CE, the Churn Estimator. We call this protocol the Adaptive RR Membership Protocol. The pseudocode for the Adaptive RR Membership Protocol is given in Figure 5.11. The inputs for the Adaptive RR Membership Protocol are N, RR, $RRMin$, $RRMax$, and c. Here N is the number of nodes in the node's local view of the membership, and RR is the node's initial requesting rate.

In line 1, the variable for the Churn Estimator CE (which controls the Adaptive RR Membership Protocol and whose values are averaged using the EWMA algorithm) is initialized to 0. Line 2 starts the infinite loop where, in

```
AdaptiveRR(N, RR, RRMin, RRMax, c)
1 CE ← 0
2 while true do
3   nextRequestTime ← time + (timeunit/RR)
4   wait until (time == nextRequestTime)
5   Left ← 0
6   Joined ← 0
7   R ← 2 × sqrt(N)
8   responses ← makeRequests(view, R)
9   for (j ← 0 to R) do
10    if (responses[j].noResponse) then
11      removeNode(view, responses[j].node)
12      N ← N − 1
13      Left ← Left + 1
14    else
15      if (responses[j].recent) then
16        isNew = addNode(view, responses[j].recentNode)
17        if (isNew) then
18          N ← N + 1
19          Joined ← Joined + 1
20  currentCE ← (Left + Joined)/R
21  CE ← EWMA(CE, currentCE, c)
22  if CE > RRMin/RRMax then
23    RR ← RRMax × CE
24  else
25    RR ← RRMin
```

Figure 5.11: Adaptive RR Membership Protocol.

Line 3, $nextRequestTime$ is set to the current $time$ plus the time $timeunit/RR$ until the next request. In line 4, the algorithm waits until the current $time$ reaches $nextRequestTime$.

In line 5 and line 6, the variables $Left$ and $Joined$ (which count the changes in the node's view of the membership) are initialized to zero. In line 7, the algorithm sets the number R of nodes to which the node sends its request

message to $R = 2\sqrt{N}$, where N is the number of nodes in its current view of the membership. Then, in line 8, the node sends its request message to R nodes, and waits for their responses.

In line 9, the algorithm iterates through the *responses* array. In line 10, the algorithm checks whether the node received a response from node j. If not, in line 11, it removes the non-responsive node from the node's view and, in line 12 and line 13, it decrements the number N of nodes in that view and increments the number $Left$ of nodes that have left. Otherwise, in line 15, the algorithm checks whether j's response contains a recent node. If so, in line 16, the algorithm invokes $addNode()$ to add the recent node to the node's view. The procedure $addNode()$ returns a boolean $isNew$ to indicate whether the recent node was already present in the node's view. In line 17, the algorithm checks whether the recent node is indeed new and, if so, in line 18 and line 19, it increments the number N of nodes in the node's view and increments the number $Joined$ of nodes that have recently joined. Control then returns to continue the iteration through the *responses* array in line 9.

After it has processed the *responses* array, in line 20 the algorithm calculates $currentCE$ and then, in line 21, applies the EWMA algorithm to obtain the smoothed value of CE.

In line 22 to line 25, the algorithm calculates the value of the requesting rate RR for the next time unit, corresponding to the smoothed value of CE. Then, the algorithm goes back to line 2 and repeats these steps indefinitely.

	Non-Adaptive	Retry R	Adaptive RR
MA	0.5966	0.6821	0.8581
MP	0.9345	0.9817	0.9728
RT	6.0	11.9274	6.0
MC	3.8552	5.1538	12.5690

Table 5.2: Non-Adaptive vs. Retry R with $Try = 2$ vs. Adaptive RR with $Try = 1$ and $RRMax = 100$.

5.5.1 Non-Adaptive vs. Retry R vs. Adaptive RR

Figures 5.12, 5.13, and 5.14 show the graphs for LND and JND over time for the Non-Adaptive Membership Protocol (top graph), the Retry R Membership Protocol (middle graph), and the Adaptive RR Membership Protocol (bottom graph). Here $N = 1024$ initially, $LastJ = 1$, $LR = 300$, $JR = 300$, and $c = 0.7$. For the Non-Adaptive and Retry R Membership Protocols, $RR = 10$. For the Adaptive RR Membership Protocol, $RRMin = 1$, $RRMax = 100$, and $RR = 10$ initially.

In Figure 5.13, we see that for the Retry R Membership Protocol, LND decreases to about 0.26 and JND decreases to about 0.12. In Figure 5.14, for the Adaptive RR Membership Protocol, LND greatly decreases to about 0.14 and JND decreases to almost zero. Thus, from these graphs, we see that increasing RR is more effective than increasing the number of tries, in decreasing both LND and JND.

Table 5.2 presents the membership accuracy MA, match probability MP, response time RT, and message cost MC for the Non-Adaptive, Retry R, and Adaptive RR Membership Protocols.

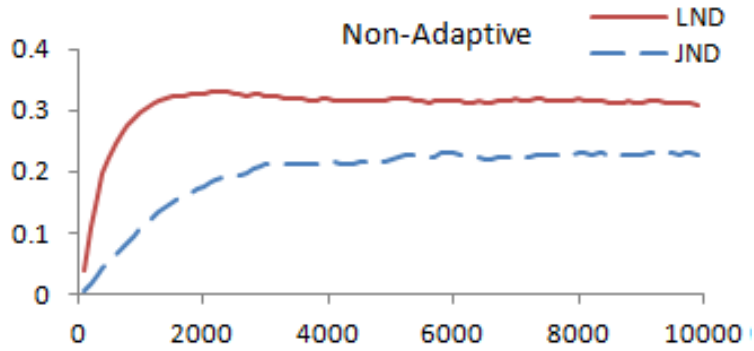

Figure 5.12: Graphs showing LND and JND over time for the Non-Adaptive Membership Protocol, where $N = 1024$ initially, $LastJ = 1$, $LR = 300$ and $JR = 300$.

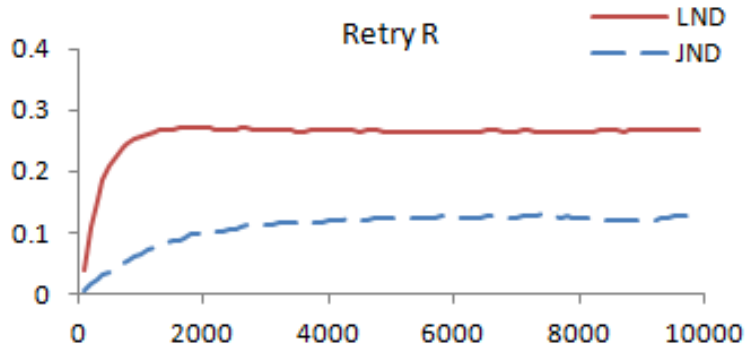

Figure 5.13: Graphs showing LND and JND over time for the Retry R Membership Protocol, where $N = 1024$ initially, $LastJ = 1$, $LR = 300$ and $JR = 300$.

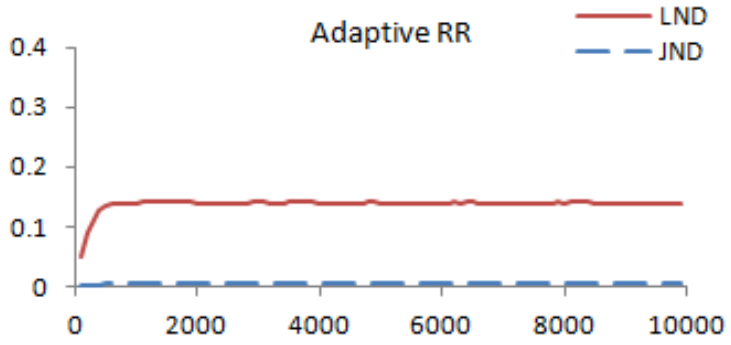

Figure 5.14: Graphs showing LND and JND over time for the Adaptive RR Membership Protocol, where $N = 1024$ initially, $LastJ = 1$, $LR = 300$ and $JR = 300$.

For the membership accuracy MA, we see that the Adaptive RR Membership Protocol has the highest membership accuracy, whereas the Non-Adaptive and Retry R Membership Protocols have a much lower membership accuracy. For the match probability MP, the Retry R Membership Protocol achieves the highest match probability, whereas the Non-Adaptive Membership Protocol has the lowest match probability. For the response time RT, both the Non-Adaptive and Adaptive RR Membership Protocols have a low response time, whereas the Retry R Membership Protocol has a much higher response time. For the message cost MC, the Adaptive RR Membership Protocol has a message cost that is about 3 times that of the Non-Adaptive Membership Protocol and about 2.5 times that of the Retry R Membership Protocol.

Thus, for the Adaptive Membership Protocol, when the requesting rate is high, the membership accuracy is also high, but the message cost is much too high. Similarly, even though the Retry R Membership Protocol achieves the highest match probability, its response time is too high and its membership accuracy is too low.

Therefore, we continue our investigations into an adaptive membership protocol that is intermediate between the Retry R Membership Protocol and the Adaptive RR Membership Protocol.

5.6 Combined Adaptive Membership Protocol

Now, we consider a membership protocol that not only adapts a node's requesting rate RR based on the Churn Estimator CE but also tries a second time ($Try = 2$) to obtain $2\sqrt{N}$ responses to a node's request. We call this protocol the Combined Adaptive Membership Protocol. The pseudocode for the Combined Adaptive Membership Protocol is given in Figure 5.15. The inputs for the Combined Adaptive Membership Protocol are N, RR, $RRMin$, $RRMax$, and c. Again, N is the number of nodes in the node's local view of the membership, and RR is the node's initial requesting rate.

In line 1, the variable for the Churn Estimator CE (which controls the Combined Adaptive RR Membership Protocol and whose values are averaged using the EWMA algorithm) is initialized to 0. Line 2 starts the infinite loop where, in Line 3, $nextRequestTime$ is set to the current $time$ plus the time $timeunit/RR$ until the next request. In line 4, the algorithm waits until the current $time$ reaches $nextRequestTime$.

In line 5 and line 6, the variables $Left$ and $Joined$ are initialized to zero. In line 7, the algorithm sets the number R of nodes to which to send its request message to $R = 2\sqrt{N}$, where N is the number of nodes in its current view of the membership and, in line 8, it initializes to zero the variable $resRec$, which counts the number of nodes from which it received responses.

```
CombinedAdaptive(N, RR, RRMin, RRMax, c)
1 CE ← 0
2 while true do
3   nextRequestTime ← time + (timeunit/RR)
4   wait until (time == nextRequestTime)
5   Left ← 0
6   Joined ← 0
7   R ← 2 × sqrt(N)
8   resRec ← 0
9   Try ← 1
10  while ((Try <= 2) and (resRec < R)) do
11    responses ← makeRequests(view, R − resRec)
12    responded ← 0
13    for (j ← 0 to (R − resRec)) do
14      if (responses[j].noResponse) then
15        removeNode(view, responses[j].node)
16        N ← N − 1
17        Left ← Left + 1
18      else
19        responded ← responded + 1
20        if (responses[j].recent) then
21          isNew = addNode(view, responses[j].recentNode)
22          if (isNew) then
23            N ← N + 1
24            Joined ← Joined + 1
25    resRec ← resRec + Responded
26    Try ← Try + 1
27  currentCE ← (Left + Joined)/(R + R − resRec)
28  CE ← EWMA(CE, currentCE, c)
29  if CE > RRMin/RRMax then
30    RR ← RRMax × CE
31  else
32    RR ← RRMin
```

Figure 5.15: Pseudocode for the Combined Adaptive Membership Protocol.

In line 9, the variable Try is initialized to 1. The loop commencing with line 10 is potentially executed twice; it is executed once if the number of

responses received ($resRec$) in the first try is equal to the number R of nodes to which the node sent its request message. In line 11, the node sends its request message to $R - resRec$ nodes, and waits for the responses from those nodes. In line 12, the algorithm sets the variable $responded$ to 0.

In line 13, the algorithm iterates through the $responses$ vector. In line 14, the algorithm checks whether the node received a response from node j. If not, in line 15, the node removes the non-responsive node from its view and, in line 16 and line 17, it decrements the number N of nodes in its view and increments the number $Left$ of nodes that have left. Otherwise, in line 19, the algorithm increments $responded$. Then, in line 20, the algorithm checks whether j's response contains a recent node. If so, in line 21, the algorithm invokes $addNode()$ to add the recent node to the node's view. The procedure $addNode()$ returns a boolean $isNew$ to indicate whether the new node was already present in the node's view. In line 22, the algorithm checks whether the recent node is indeed new and, if so, in line 23 and line 24, it increments the number N of nodes in the node's view and increments the number $Joined$ of newly joined nodes in that view. In line 25, the algorithm increases $resRec$ by $responded$, the number of nodes that responded in this try. In line 26, the algorithm increments Try, and control then returns to the while loop in line 10 to determine whether Try is less than or equal to 2 and $resRec$ is less than R. If both of those conditions are satisfied, the algorithm goes through the while loop again.

After the algorithm has completed the while loop in line 10, it calculates $currentCE$ in line 27 using the values of $Left$, $Joined$, and $resRec$ it obtained in the loop. Then, in line 28, the algorithm invokes the EWMA algorithm to calculate the smoothed value of CE.

In line 29 to line 32, the algorithm calculates the value of the requesting rate RR for the next time unit, corresponding to the smoothed value of CE. Then, the algorithm goes back to line 2 and repeats these steps indefinitely.

With $Try = 2$, a requesting node sends its request to more nodes to try to obtain $2\sqrt{N}$ responses to its request and, thus, the Combined Adaptive Membership Protocol does not need to increase the requesting rate RR as much as does the Adaptive RR Membership Protocol. Thus, we realize some savings in the message cost MC, compared to the Adaptive RR Membership Protocol.

5.6.1 Investigation of $RRMax$

Now we investigate different values of $RRMax$ for the Combined Adaptive Membership Protocol, in particular, $RRMax = 100, 50, 30$.

Table 5.3 shows the values of the membership accuracy MA, match probability MP, response time RT, and message cost MC for the Combined Adaptive Membership Protocol in these three cases.

We see that as $RRMax$ is decreased from $RRMax = 100$ to $RRMax = 50$, MA decreases and MP slightly decreases. When $RRMax$ is further decreased

RRMax	100	50	30
MA	0.8663	0.8198	0.7579
MP	0.9843	0.9836	0.9822
RT	11.9874	11.9883	11.9885
MC	13.4939	9.3156	6.9104

Table 5.3: Combined Adaptive with $RRMax = 100, 50, 30$.

to $RRMax = 30$, again MA decreases and MP slightly decreases. We also see that as $RRMax$ is decreased, the response time RT remains about the same and the message cost MC decreases substantially, such that the message cost for $RRMax = 100$ is nearly twice that for $RRMax = 30$. Thus, to keep the message cost relatively low while obtaining reasonable membership accuracy, we choose $RRMax = 50$ in our further experiments.

5.6.2 Retry R vs. Adaptive RR vs. Combined Adaptive

Table 5.4 shows the membership accuracy MA, the match probability P, the response time RT, and the message cost MC for the Retry R, Adaptive RR, and Combined Adaptive Membership Protocols. Here, $N = 1024$ initially, $LastJ = 1$, $LR = 300$, $JR = 300$ and $c = 0.7$. For the Retry R Membership Protocol, $Try = 2$ and $RR = 10$. For the Adaptive RR Membership Protocol, $Try = 1$, $RRMin = 1$, $RRMax = 100$, and $RR = 10$ initially. For the Combined Adaptive Membership Protocol, $Try = 2$, $RRMin = 1$, $RRMax = 50$, and $RR = 10$ initially.

From Table 5.4, we see that the membership accuracy MA for the Combined Adaptive Membership Protocol is 0.8198, which is much better than that

	Retry R	Adaptive RR	Combined Adaptive
MA	0.6821	0.8581	0.8198
MP	0.9817	0.9728	0.9836
RT	11.9274	6.0	11.9883
MC	5.1538	12.5690	9.3156

Table 5.4: Retry R with $Try = 2$ vs. Adaptive RR with $Try = 1$ and $RRMax = 100$ vs. Combined Adaptive with $Try = 2$ and $RRMax = 50$.

for the Retry R Membership Protocol, but worse than that for the Adaptive RR Membership Protocol. We also see that the match probability MP for the Combined Adaptive Membership Protocol is 0.9836, which is better than the match probability for the Adaptive RR Membership Protocol and slightly better than the match probability for the Retry R Membership Protocol. We see further that the response time RT for the Combined Adaptive Membership Protocol is about the same as that for the Retry R Membership Protocol, which is about double that of the Adaptive RR Membership Protocol. We also see that the message cost MC of the Combined Adaptive Membership Protocol lies between that of the Retry R Membership Protocol and that of the Adaptive RR Membership Protocol, and is about three-fourths that of the Adaptive RR Membership Protocol.

Figure 5.16 shows the Pareto Optimal curve for the membership accuracy and the message cost with various Membership Protocols. Our goal is to have the membership protocol to achieve the highest membership accuracy with the lowest message cost. From this figure, we can see the Combined Adaptive Membership Protocol balances the message cost MC against the membership

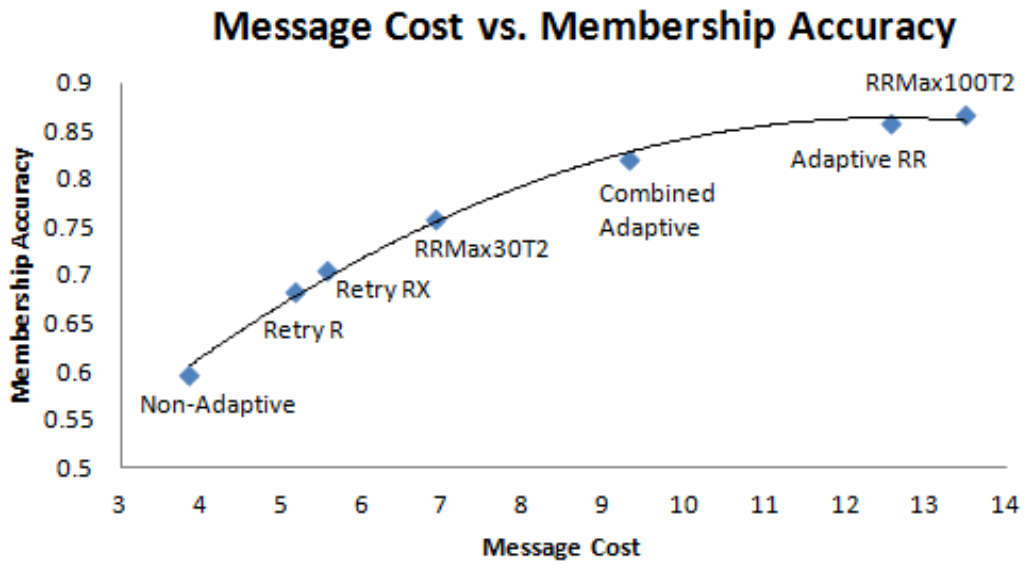

Figure 5.16: Pareto optimal curve for the Membership Accuracy and Message Cost for various Membership Protocols

accuracy MA. The message cost MC of the Combined Adaptive Membership Protocol is less than that of the Adaptive RR Membership Protocol and, correspondingly, the membership accuracy MA of the Combined Adaptive Membership Protocol is greater than that of both the Retry R and the Non-Adaptive Membership Protocols.

5.7 Extended Scenario

Now, we investigate the effectiveness of the Combined Adaptive Membership Protocol to see how well it handles the following extreme cases:

- Case 1: High leaving rate, high joining rate
- Case 2: High joining rate, low leaving rate

- Case 3: Low joining rate, high leaving rate.

In particular, we consider an extended scenario that comprises the following five scenarios:

- Scenario 1: $LR = 10$, $JR = 10$ for time 0 to 3000
- Scenario 2: $LR = 300$, $JR = 300$ for time 3000 to 6000
- Scenario 3: $LR = 0$, $JR = 300$ for time 6000 to 9000
- Scenario 4: $LR = 300$, $JR = 0$ for time 9000 to 12000
- Scenario 5: $LR = 0$, $JR = 0$ for time 12000 to 15000.

For all five scenarios, we set $LastJ = 1$ and $c = 0.7$. Initially, there are $N = 1024$ nodes in the membership, and each node's view of the membership is the entire membership. As time progresses, each member changes its view of the membership. The number M of nodes to which the metadata are distributed and the number R of nodes to which the requests are distributed are both set to $2\sqrt{N}$, where N is the number of nodes in the node's current view of the membership at a given time step.

We compare the effectiveness of the Combined Adaptive Membership Protocol and the Non-Adaptive Membership Protocol by considering the extended scenario that comprises these five scenarios.

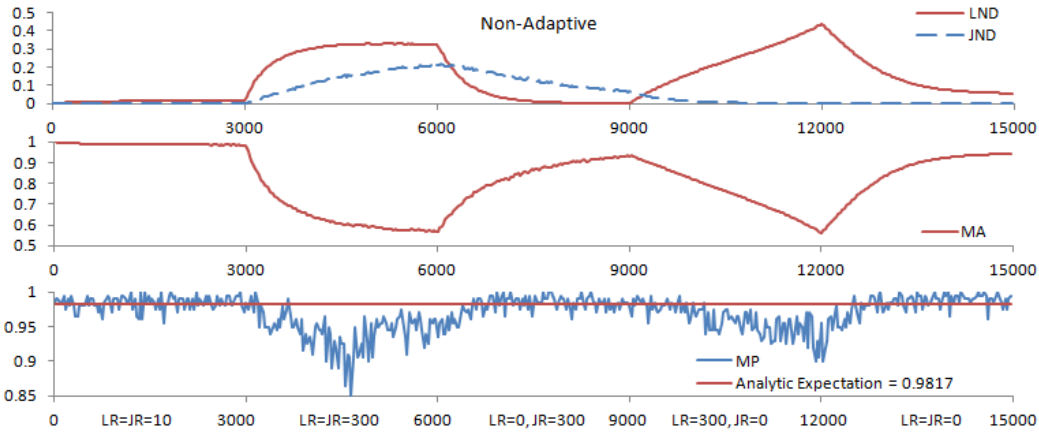

Figure 5.17: Graphs of LND, JND, MA and MP for the Non-Adaptive Membership Protocol where $LastJ = 1$ and $c = 0.7$.

5.7.1 Non-Adaptive

Figure 5.17 shows the graphs of LND, JND, MA, and MP for the Non-Adaptive Membership Protocol. Here, $LastJ = 1$ and RR is fixed at $RR = 10$ for all five scenarios.

In the first scenario, the leaving, joining, and requesting rates are low and the same ($LR = JR = RR = 10$). The values of LND and JND remain low, because a node detects non-operational (leaving) nodes and discovers newly joining nodes within a short time interval. The membership accuracy MA remains high throughout the first scenario ($MA = 0.9873$). The match probability MP is generally higher than the value obtained from the analytic formula (0.9817) given in [60].

In the second scenario, the values of LR and JR are much higher than the value of RR ($LR = JR = 300$ and $RR = 10$). The values of LND and JND increase, because a node can not detect enough non-operational

(leaving) nodes and can not discover enough newly joined nodes within a short time interval. The membership accuracy MA dramatically decreases to about 0.5852. Moreover, the match probability MP is quite variable, decreasing to about 0.85 and then increasing to about 0.9350.

In the third scenario, the value of LR is low, the value of JR is high, and the value of RR is low ($LR = 0$, $JR = 300$, and $RR = 10$). The values of LND decrease because LR drops to $LR = 0$. The values of JND remain high because JR remains high. The membership accuracy MA is higher than that in the second scenario (because $LR = 0$), and slowly increases to about 0.9331 at the end of the third scenario. Similarly, the match probability MP slowly increases from about 0.9350 to about 0.9850.

In the fourth scenario, the value of LR is high, the value of JR is low, and the value of RR is low ($LR = 300$, $JR = 0$, and $RR = 10$). The values of LND increase to about 0.4384. In addition, the membership accuracy MA steadily decreases to about 0.5610. The reason is that most of the nodes have not yet discovered all of the newly joined nodes from the third scenario, but now more nodes are leaving the membership. The match probability MP fluctuates considerably, decreasing to about 0.9.

Lastly, in the fifth scenario, the values of both LR and JR are low and the value of RR is also low ($LR = JR = 0$ and $RR = 10$). Thus, the membership accuracy MA slowly increases to about 0.9426. In addition, the match

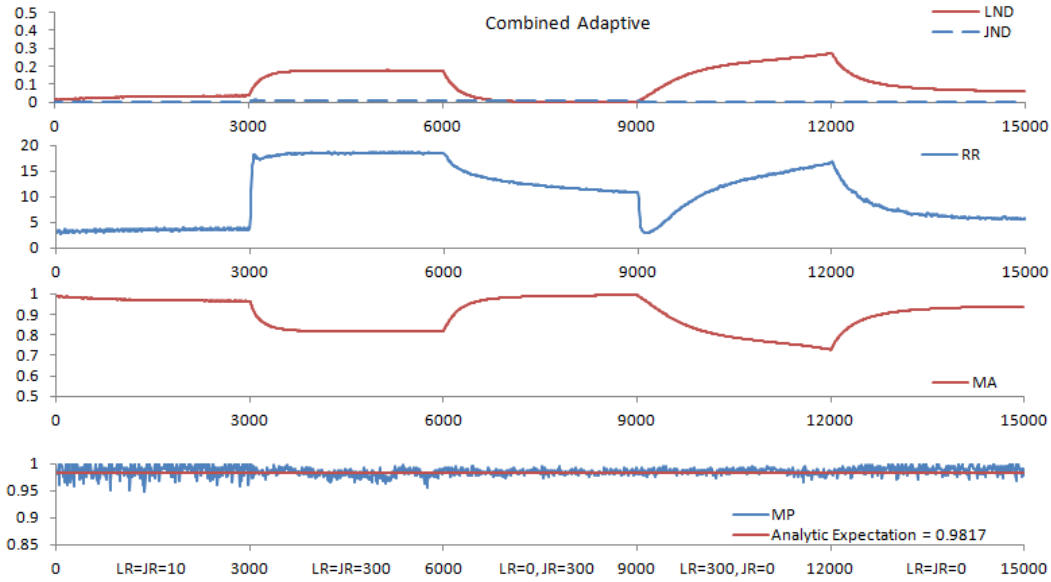

Figure 5.18: Graphs of LND, JND, RR, MA, and MP for the Combined Adaptive Membership Protocol, where $RR = 10$, $LastJ = 1$, $c = 0.7$, $RRMax = 50$, and $Try = 2$.

probability MP increases and remains high, hovering around the analytic expectation (0.9817).

5.7.2 Combined Adaptive

Figure 5.18 shows LND, JND, RR, MA, and MP for the Combined Adaptive Membership Protocol. Here, $LastJ = 1$, $c = 0.7$, $Try = 2$, $RRMax = 50$, and $RR = 10$ initially.

In the first scenario, the values of the leaving rate and the joining rate are low ($LR = JR = 10$). Thus, the values of both LND and JND are low, because there are not many non-operational (leaving) nodes or newly joined nodes. The value of RR quickly decreases to 3.6764, in order to reduce the

message cost MC. The membership accuracy MA remains high throughout the first scenario, and the match probability MP hovers around the analytic expectation (0.9817).

In the second scenario, the values of LR and JR are high ($LR = JR = 300$), and are much higher than the values of RR ($RRMax = 50$). Note that the values of JND and LND shown in Figure 5.18 are much less than the corresponding values of JND and LND for the Non-Adaptive Membership Protocol shown in Figure 5.17. For the Combined Adaptive Membership Protocol, the value of RR is increased to about $RR = 18.5161$, which results in a membership accuracy MA of about 0.8251 compared to about 0.5852 for the Non-Adaptive Membership Protocol. Lastly, the match probability MP remains high throughout the second scenario, and hovers around the analytic expectation (0.9817), whereas in the Non-Adaptive Membership Protocol it decreases to about 0.85.

In the third scenario, the value of LR is low and the value of JR is high ($LR = 0$, $JR = 300$). Because LR is low, the values of LND remain close to zero. The values of JND also remain close to zero. The Combined Adaptive Membership Protocol adjusts the value of RR to about $RR = 12.6922$. Joining nodes are discovered relatively quickly, and there are no new leaving nodes to detect because $LR = 0$. Thus, the membership accuracy MA increases from about 0.8251 to about 0.9739. The match probability MP still remains high, and hovers around the analytic expectation (0.9817).

In the fourth scenario, the value of LR is high and the value of JR is low ($LR = 300$, $JR = 0$). Because the value of LR is high, the values of LND increase to about 0.2669. The values of JND remain low, because $JR = 0$. The Combined Adaptive Membership Protocol increases RR to about $RR = 16.5080$, in order to detect leaving nodes more quickly. However, leaving nodes are still not detected quickly enough, so LND increases and MA decreases. The membership accuracy MA decreases to about 0.7307 at the end of the fourth scenario. Finally, the match probability MP remains high, and hovers around the analytic expectation (0.9817), in contrast to the Non-Adaptive Membership Protocol where the match probability fluctuates and decreases to about 0.9.

Lastly, the fifth scenario has a low value of LR and a low value of JR ($LR = JR = 0$). The Combined Adaptive Membership Protocol decreases the value of RR to about $RR = 5.6349$, in order to reduce the message cost MC. The membership accuracy MA increases to, and remains at, about 0.9375 during most of the fifth scenario. Moreover, the match probability MP remains high, and hovers around the analytic expectation (0.9817).

Finally, we average the membership accuracy MA, the match probability MP, the response time RT, and the message cost MC over all five scenarios. Table 5.5 shows the overall values of these metrics for the Non-Adaptive, Retry R, Adaptive RR, and Combined Adaptive Membership Protocols, averaged over all five scenarios. As we see from the table, the Combined Adaptive

	Non-Adaptive	Retry R	Adaptive RR	Combined Adaptive
MA	0.8149	0.8542	0.9217	0.8982
MP	0.9704	0.9841	0.9801	0.9858
RT	6.0	10.8262	6.0	11.0339
MC	5.0490	5.7335	8.8284	6.3305

Table 5.5: Non-Adaptive vs. Retry R with $Try = 2$ vs. Adaptive RR with $Try = 1$ and $RRMax = 100$ vs. Combined Adaptive with $Try = 2$ and $RRMax = 50$.

Membership Protocol achieves a membership accuracy MA of 0.8982, which is quite good. Moreover, the Combined Adaptive Membership Protocol achieves the best match probability ($MP = 0.9858$) of all four protocols. The response time RT of the Combined Adaptive Membership Protocol is slightly more than that for the Retry R Membership Protocol, and is much more than that for the Non-Adaptive and Adaptive RR Membership Protocols. The message cost MC of the Combined Adaptive Membership Protocol is less than that for the Adaptive RR Membership, and is slightly more than that for the Non-Adaptive and Retry R Membership Protocols, as shown in Table 5.5.

Figure 5.19 shows the Pareto Optimal curve for the membership accuracy and the message cost with various Membership Protocols. From this figure, we can see the message cost MC of the Combined Adaptive Membership Protocol is less than that for the Adaptive RR Membership, and its Membership Accuracy MA are higher than that for the Non-Adaptive and Retry R Membership Protocols. This figure shows us that our Combined Adaptive Membership Protocol can balance the message cost MC against the member-

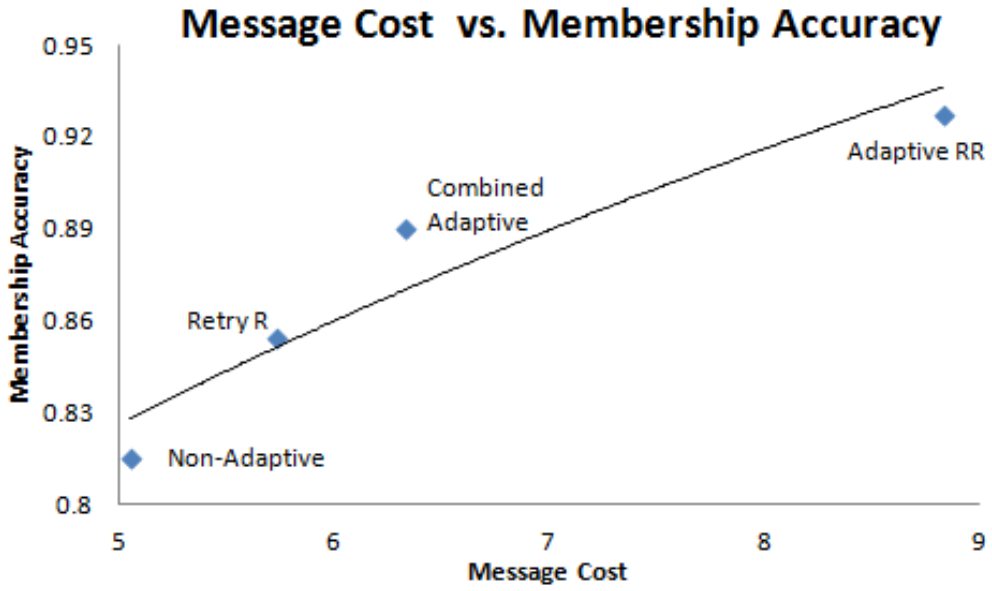

Figure 5.19: Pareto optimal curve for the Membership Accuracy and Message Cost for various Membership Protocols

ship accuracy MA quite well, as it does not require too much message cost MC but still can maintain a high membership accuracy MA throughout all five scenarios.

Overall, these experiments demonstrate that the Combined Adaptive Membership Protocol is effective in detecting non-operational (leaving) nodes and in discovering newly joining nodes. When the leaving rate and the joining rate are high, the Combined Adaptive Membership Protocol quickly increases RR to obtain a high membership accuracy. Moreover, when the leaving rate and the joining rate are low, the Combined Adaptive Membership Protocol decreases RR, in order to maintain a reasonable response time and a reasonable message cost, while still maintaining a reasonable membership accuracy and

a high match probability. As a result, the Combined Adaptive Membership Protocol works well not only when the membership has a lot of churn, but also when the membership is stable.

5.8 Summary

In this chapter, we have presented adaptive and non-adaptive membership management protocols for the iTrust information retrieval network. The membership protocols for iTrust allow each member to maintain its own local view of the membership, and aim to keep that local view close to the actual membership. A node that receives a request sends to the requesting node a response that contains newly joined member(s) in its local view of the membership. The node also sends in its response the URL of the document, if the keywords in the query match metadata that it holds.

A requesting node discovers newly joining nodes from the responses it receives to its requests. Likewise, a requesting node detects leaving (non-operational) nodes when it does not receive responses from those nodes before a timeout, or when it receives an error code from TCP. Thus, the iTrust membership protocols exploit messages already required by the iTrust messaging protocol for search and retrieval.

As our experiments demonstrate, for appropriate values of the parameters, particularly for the Combined Adaptive Membership Protocol, the member-

ship accuracy, response time, and message cost are reasonable, and the match probability is high. The Combined Adaptive Membership Protocol works well not only when the membership has a lot of churn but also when the membership is stable.

Chapter 6

Statistical Inference and Dynamic Adaptation for iTrust

In previous chapters, we have described a fully distributed membership algorithm for iTrust [16], as well as algorithms for protecting iTrust against malicious nodes [17]. Such malicious nodes do not return responses indicating that they have a match between the metadata they hold and the keywords in the requests they receive, effectively censoring that information.

In this chapter, we combine the dynamic adaptive membership algorithm for increasing the requesting rate and the detection and defensive adaptation algorithm for protecting against malicious nodes. Both of these algorithms depend on information that cannot be measured directly, in particular, the current size of the membership and the current proportion of malicious nodes in the membership. Statistical inference algorithms provide estimates for both of these metrics. These algorithms are sufficiently accurate and timely to allow them to be used to manage the iTrust system in the presence of membership churn and malicious nodes.

6.1 Design of iTrust

The iTrust information publication, search and retrieval system is fully distributed, and involves no centralized mechanisms and no centralized control. In this chapter, we have refined our design of iTrust based on the strategy presented in Chapter 3.

The nodes that participate in an iTrust network constitute the *membership* of the network. The membership changes dynamically as nodes join and leave the membership. Nodes that leave the membership do so voluntarily, or because they have failed or have become disconnected. Moreover, some of the nodes in the membership might be malicious. Each node has its own *local view* of the membership. In iTrust, we do not aim to maintain an agreed accurate membership; rather, we aim to keep a nodes's local view of the membership close to the actual membership.

6.1.1 Joining the Membership

To join the membership, a node first obtains the address of a bootstrapping node, as shown in Figure 6.1. To obtain the address of the bootstrapping node, the node uses mechanisms outside the iTrust network, such as conventional Web search, e-mail, Twitter, printed publications, *etc.*

The steps involved when a node joins the membership are as follows:

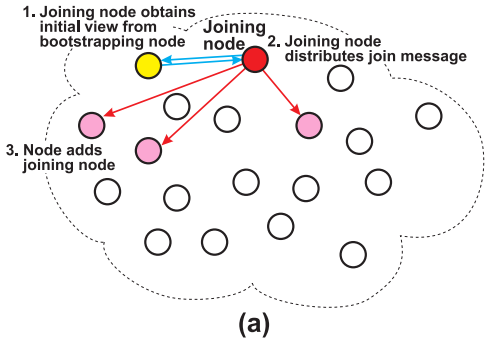

Figure 6.1: Joining the membership. A joining node obtains its initial view of the membership from a bootstrapping node, and distributes a join message to randomly chosen nodes in that view. Those nodes add the newly joined node to their views.

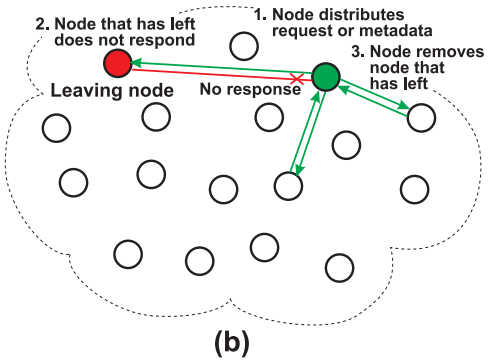

Figure 6.2: Leaving the membership. A leaving node simply leaves. When it does not respond to a request, other nodes detect that it has left and remove it from their views.

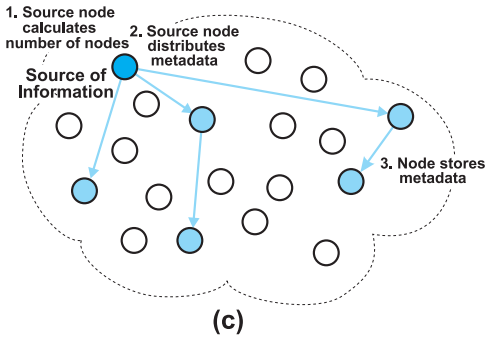

Figure 6.3: Distributing metadata. A source node distributes metadata to randomly chosen nodes in its view. Those nodes store the metadata and the URL locally, and respond with their newly joined nodes. The source node updates its view accordingly.

1. A joining node contacts a bootstrapping node to obtain the bootstrapping node's current view as its initial view.

2. The joining node then distributes a join message to randomly chosen nodes in its initial view.

3. Each randomly chosen node then adds the newly joined node to its local view, and sends a response message to the newly joined node.

Over time, a node learns about a newly joined node when it receives a response message from a node that is aware of the newly joined node.

6.1.2 Leaving the Membership

A node may leave the membership, as shown in Figure 6.2, voluntarily or because it has failed or has become disconnected. The steps involved when a node leaves the membership are as follows:

1. To leave the membership, a node simply leaves, without publishing its leaving.

Over time, a node discovers the departure of nodes when it sends a message to those nodes and they do not respond. It is not appropriate to allow a node to publish the departure of another node, because doing so might enable a malicious node to cause the removal of many nodes from other nodes' views.

6.1.3 Distributing Metadata

A *source node* produces metadata that describes its information, and distributes that metadata to randomly chosen nodes in its local view, as shown in Figure 6.3. The metadata are distinct from the information they describe, and include a list of keywords and the URL of the information at the source node.

The steps involved when a source node distributes its metadata are as follows:

1. A source node calculates the number of nodes to which it needs to distribute its metadata, based on its current view.

2. Next, the source node distributes its metadata together with the URL of the information to randomly chosen nodes in its current view.

3. Each randomly chosen node stores the metadata and the URL locally, and sends a response message to the source node.

When the source node detects the departure of nodes from the membership, and those nodes hold metadata that it distributed, it re-calculates and re-distributes the metadata to additional nodes.

6.1.4 Distributing Requests

A *requesting node* generates requests (queries) that contain keywords, and distributes its requests to randomly chosen nodes in its local view. The re-

questing node obtains the URLs of the documents from the matching nodes, which it uses to retrieve the documents from the source nodes.

Some of the matching nodes might be malicious and respond to the requesting node, but not with the URL of the document corresponding to metadata they hold, thus effectively hiding or censoring that information.

The steps involved when a requesting node distributes requests and updates its view are as follows:

1. A requesting node distributes its request to randomly chosen nodes in its local view, as shown in Figure 6.4.

2. A node that receives the request compares the keywords in the request with the metadata it holds. If it finds a match, the node responds to the requesting node with a message that contains the URL of the document. A node that does not find a match still responds to the requesting node.

3. The requesting node can then use the URL, provided by the matching node, to retrieve the information from the source node.

6.1.5 Updating the View

A source node or a requesting node expects each node to which it sent a message to respond to its message with its recently joined member(s) (and, in the case of a request, the URL of the document if it has a match). Thus, a

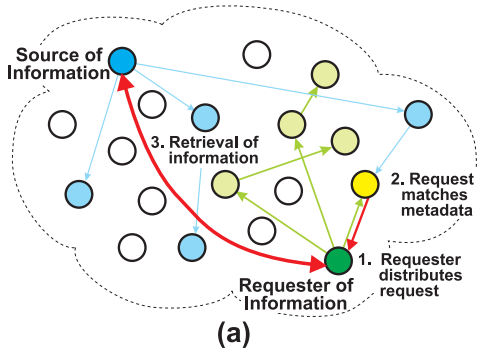

Figure 6.4: A requesting node distributes its request to randomly chosen nodes in its view of the membership. If a node matches the keywords in the request against metadata it holds, it responds with the URL for the information, which the requesting node uses to retrieve the information from the source node.

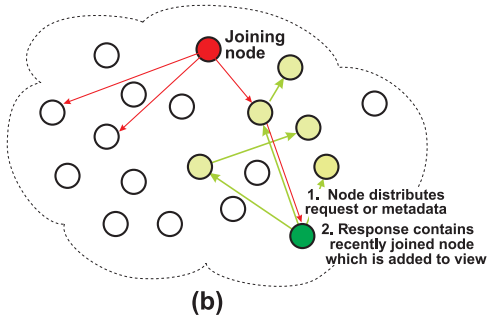

Figure 6.5: A node that receives a request or metadata responds with the most recently joined node in its view. The distributing node adds that newly joined node to its view.

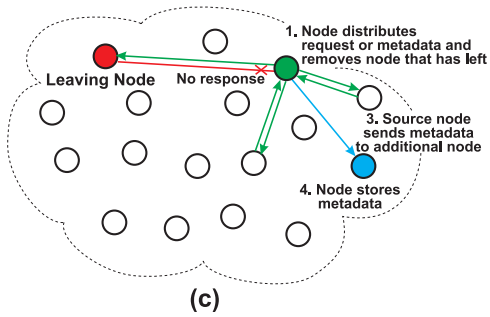

Figure 6.6: If the node sends a request or metadata to a node that has left, it receives no response and removes that node from its view. If it had sent metadata to the leaving node, it sends that metadata to an additional node.

source node or a requesting node discovers newly joined nodes and adds them to its view.

If a source node or a requesting node does not receive a response from a node within a timeout period or it receives an error code from TCP, then it considers the non-responding node to have left the membership and removes that node from its view.

The steps involved in updating a node's view are as follows:

1. A source node or a requesting node sends metadata or a request, respectively, to randomly chosen nodes.

2. When the source node or the requesting node receives the responses, it adds the newly joined members, obtained from the other nodes, to its view, as shown in Figure 6.5.

3. If the source node or the requesting node does not receive a response from a node to which it sent a request before a timeout occurs, or if it receives an error code from TCP, then it considers the non-responding node to have left the membership, and removes that node from its view, as shown in Figure 6.6.

If a node was, at a prior time, a source node then, after receiving the responses to its message, in addition to the above steps, it distributes its previously distributed metadata along with the URL, to additional nodes, as shown in Figure 6.6, according to the following steps:

1. The node calculates the number of nodes to which it needs to distribute its metadata, based on its current view of the membership.

2. Next, the node subtracts from that number the number of nodes to which it previously distributed metadata, and then it adds the number of nodes that had the metadata previously and that left the membership.

3. Then, the node distributes the metadata to that many additional nodes, randomly chosen from its view, but to which it had not previously sent the metadata.

4. Each randomly chosen node stores the metadata and the URL, and sends a response to the source node.

6.2 Model for iTrust

6.2.1 System and Fault Model

The iTrust system operates over the Internet and is implemented using HTTP, which operates over TCP. As such, communication is "reliable." However, nodes may crash, or may go offline and become disconnected. That is, nodes in the membership are assumed to follow the iTrust protocols and algorithms, except that:

- A node may leave the membership voluntarily.

- A node may crash or become disconnected, in which case it leaves the membership involuntarily.
- A malicious node responds to a request but, if it has a match between the keywords in a request and the metadata it holds, it fails to report that match.

Note that a malicious node responds to requests because it knows that if it does not, it will be immediately removed from the requesting node's view of the membership.

Actual Membership and View of the Membership

The actual membership and a node's view of the membership are represented in terms of the following quantities:

- I: The number of nodes in the intersection of the node's view of the membership and the actual membership.
- J: The number of joining nodes that a node has not discovered.
- L: The number of leaving nodes that a node has not detected.

As illustrated in Figure 6.7, the actual membership consists of $I + J$ nodes, whereas a node's view of the membership consists of $I + L$ nodes. Of the I nodes that are in the node's view of the membership and also in the actual membership, some of those nodes might be malicious. Likewise, some of the L nodes and some of the J nodes might be malicious.

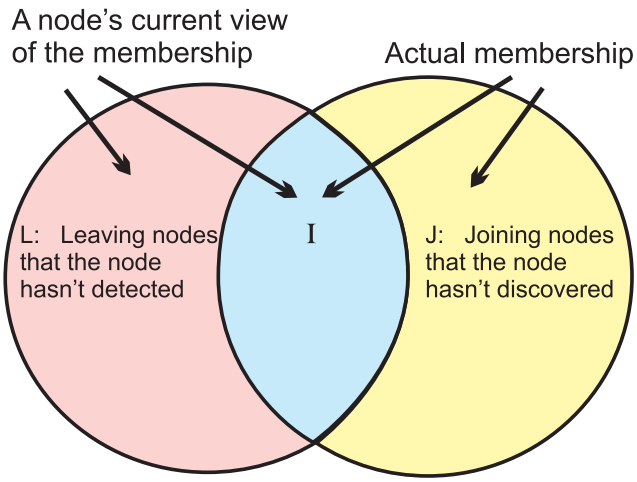

Figure 6.7: A node's current view of the membership vs. the actual membership.

The quantities I, J and L, are used in the definition of the variables for the Dynamic Adaptive Algorithm in Section 6.4 and in the definition of the performance metrics for the evaluation in Section 6.5.

Membership Churn

Membership churn refers to nodes joining and leaving the membership, and is represented by the following rates:

- JR: The Joining Rate, the number of nodes that join the membership per time unit.
- LR: The Leaving Rate, the number of nodes that leave the membership per time unit.

When the membership has a lot of churn, both JR and LR are high. Conversely, when the membership is stable, both JR and LR are low. A node

can not directly know or control JR and LR, but it can adjust its rate of distributing requests or metadata in order to update its view of the membership more quickly, in particular to detect leaving nodes more quickly.

Malicious Nodes

Malicious nodes can disrupt the behavior of the iTrust protocols and algorithms by hiding or censoring information. Thus, they must adapt to circumvent such malicious nodes. To do so, they make use of the following quantity:

- X: The proportion of non-malicious nodes in the actual membership at a particular point in time, *i.e.*, $1-X$ is the proportion of malicious nodes in the actual membership at that point in time.

When the membership has a lot of malicious nodes, $1 - X$ is high and X is low. Conversely, when the membership has only a few malicious nodes, $1 - X$ is low and X is high. A node can not know or control X, but it can adjust the number of nodes to which it distributes its request message in order to obtain more responses to its request message.

6.3 Protecting Against Malicious Nodes

The algorithm for detecting malicious nodes estimates the proportion of non-malicious nodes and, thus, the proportion of malicious or subverted nodes.

For given values of n, m and r, the detection algorithm computes the analytical (expected) probabilities for the number k of matches for various values of X, such as $X = 1.0$, 0.8, 0.6, 0.4 and 0.2 using Equation (3.1). These values of X enable the algorithm to discriminate the curves better and enable it to find the estimated proportion x of non-malicious nodes (and, thus, the estimated proportion $1 - x$ of malicious nodes) that yield significant changes in the number r of nodes to which to distribute requests.

6.3.1 Detecting Malicious Nodes

The detection algorithm collects data on the number of responses that a requesting node receives for its request, using the EWMA() method, shown Figure 4.10 from Chapter 4. Then, it calculates the empirical probabilities from the data. The algorithm excludes zero matches, because it can not distinguish the case in which no metadata exists from the case in which no node holds both the metadata and the request.

Using the modified chi-squared test, shown in Figure 4.4, the detection algorithm compares the normalized observed probabilities $O(k)$ and the normalized analytical probabilities $P[X](k)$, for $k = 1, 2, ..., kMax$ and $X = 1.0, 0.8, 0.6, 0.4, 0.2$. It then chooses, as the observed proportion of non-malicious nodes, the value of x ($x = 1.0, 0.8, 0.6, 0.4, 0.2$) for which the chi-squared value is the smallest. This value of x is the algorithm's best estimate of the pro-

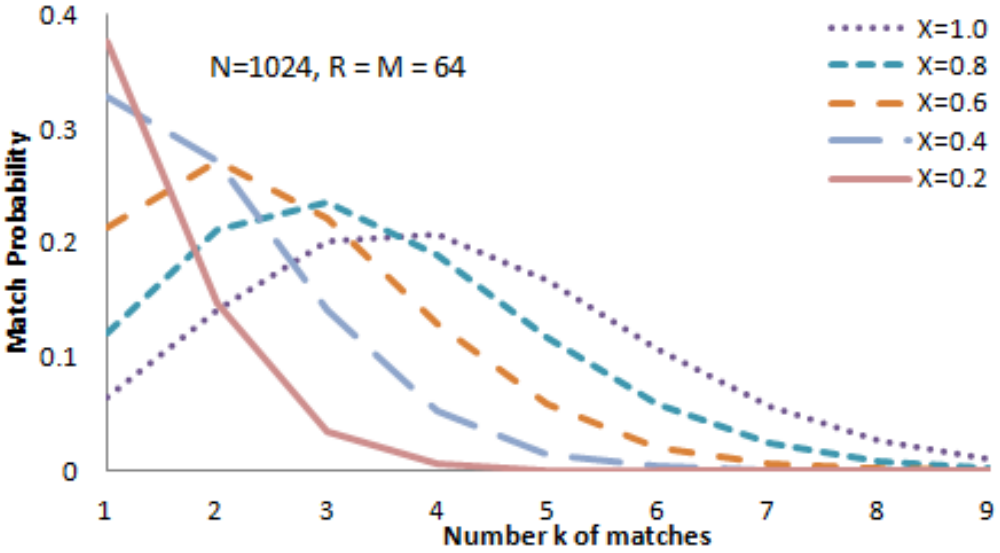

Figure 6.8: The normalized analytical probabilities $P[X]$ of the number k of matches for different proportions X of non-malicious nodes. The detection algorithm tries to match the curve for the normalized observed probabilities against the curves for the normalized analytical probabilities shown in the figure, in order to estimate the proportion x of non-malicious nodes in a node's view of the membership.

portion of non-malicious nodes, and corresponds to the curve with the best fit.

For $N = 1024$ and $M = R = 64$, Figure 6.8 shows the graphs of the normalized analytical probabilities $P[X]$ for $X = 1.0, 0.8, 0.6, 0.4, 0.2$, obtained from Equation (3.1). As the figure shows, many of the requests result in multiple responses. These multiple responses are not waste, as they are used to estimate the proportion x of non-malicious nodes in a node's view of the membership. As the figure shows, there is some overlap between the curves for the different values of X. Nonetheless, the chi-squared test provides good discrimination between the curves.

The parameters and variables for the detection method $xGet()$ that determines the estimated proportion x of non-malicious nodes are as follows:

- O_i: The array of unnormalized observed probabilities at time i.
- n: The number of nodes in a node's view of the membership.
- m: The number of nodes in a source node's view to which it distributes metadata.
- r: The number of nodes in a requesting node's view to which it distributes requests.
- $kMax$: The upper bound on the number k of responses to a request.
- $P[X]$: The array of analytical probabilities for a particular value of X.
- x: The estimated proportion of non-malicious nodes in a node's view, *i.e.*, $1 - x$ is the estimated proportion of malicious nodes in its view.

Pseudocode for the detection algorithm is given in Figure 6.9.

6.3.2 Defending against Malicious Nodes

When the detection algorithm estimates that the proportion x of non-malicious nodes is less than 1.0, the defensive adaptation algorithm increases the number r of nodes to which the requests are distributed, to achieve the same probability of a match as when $x = 1.0$.

```
xGet(O, n, m, r, kMax)
1  O ← norm(O, kMax)
2  X = 0.2
3  for (j ← 1 to 5) do
4    X ← 0.2 × j
5    P[X] ← findP(n, m, r, x, kMax)
6    P[X] ← norm(P[X], kMax)
7    chiSq[X] ← modChiSq(O, P[X], kMax)
8  x ← min(chiSq[0.2],chiSq[0.4],chiSq[0.6],chiSq[0.8],chiSq[1.0])
9  return x
```

Figure 6.9: Detection method for estimating the value of x.

For example, consider an iTrust network with $N = 1024$, $M = 64$, and $X = 1.0, 0.8, 0.6, 0.4, 0.2$. The defensive adaptation algorithm first determines the value of y_0 for the point on the $X = 1.0$ curve corresponding to $N = 1024$ and $M = R = 64$. Using Equation (3.3), it computes the probability of one or more matches to obtain $y_0 = 0.9817 = P(k \geq 1)$, as shown in Figure 6.10.

If the detection algorithm determines from empirical evidence that $x = 0.8$ then, from the calculated value of y_0, the defensive adaptation algorithm determines the value of r corresponding to y_0 on the $x = 0.8$ curve. That is, it solves the equation $y_0 = P(k \geq 1)$ with $N = 1024$, $M = 64$, $x = 0.8$ and r to obtain $r = 80$. Similarly, if the detection algorithm determines from empirical evidence that $x = 0.6$, $x = 0.4$ or $x = 0.2$, the defensive adaptation algorithm determines that $r = 106$, $r = 156$ or $r = 289$, respectively.

The parameters and variables for the defensive adaptation method $rGet()$ that determines the number r of nodes to which a requesting node distributes its request, are as follows:

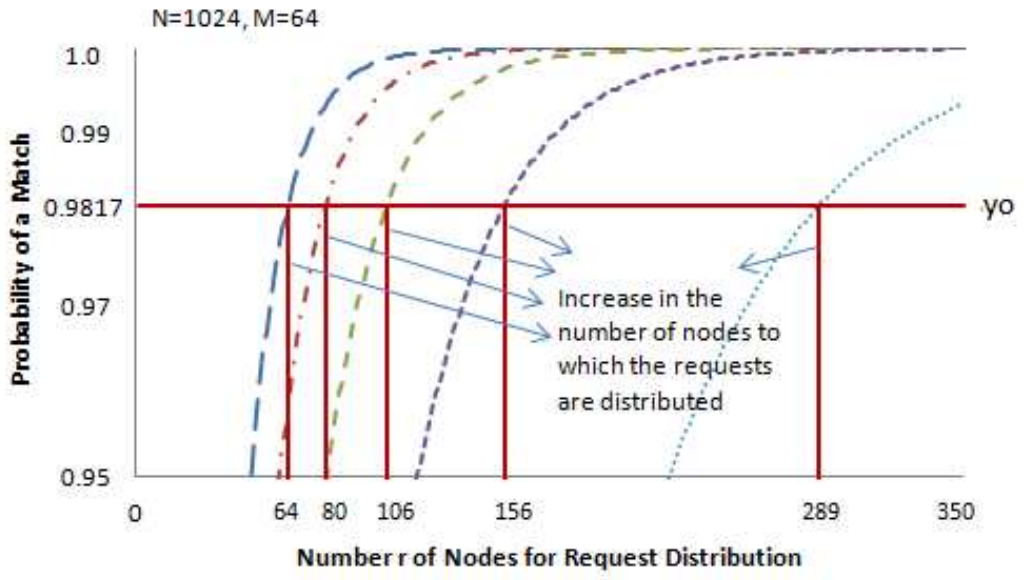

Figure 6.10: Based on the curves for $P(k \geq 1)$ for $X = 1.0, 0.8, 0.6, 0.4, 0.2$, the defensive adaptation algorithm increases or decreases the number of nodes to which the requests are distributed to compensate for malicious nodes, in order to achieve the same probability of a match as when none of the nodes is malicious.

- n: The number of nodes in a node's view of the membership.
- m: The number of nodes in a source node's view to which it distributes metadata.
- y_0: The probability $y_0 = P(k \geq 1) = 0.9817$ of one or more matches when $R = M = 2\sqrt{N}$ and $X = 1.0$.
- r: The number of nodes in a requesting node's view to which it distributes its request, in order to maintain the same probability of a match when some of the nodes are malicious as when none of the nodes is malicious.

rGet(n, m, y_0, x)
1 $r = 0$
2 Repeat
3 $r \leftarrow r + 1$
4 $y \leftarrow 1 - \frac{(n-mx)}{(n)} \frac{(n-mx-1)}{(n-1)} \cdots \frac{(n-mx-r+1)}{(n-r+1)}$
5 until $(y > y_0)$
6 return r

Figure 6.11: Method for finding the value of r that maintains the same probability of a match when some of the nodes are malicious as when none of the nodes is malicious.

- x: The estimated value of x returned by the detection method $xGet()$.

Pseudocode for the defensive adaptation algorithm is given in Figure 6.11.

6.4 Dynamic Adaptation Algorithm

Now we present the Dynamic Adaptive Algorithm that allows a node to maintain its local view of the membership, and that also detects and defends against malicious nodes. The Dynamic Adaptive Algorithm uses the *Distribute*() method, described in Section 6.4.3, to distribute join messages, request messages, and metadata messages.

For both a requesting node and a source node, the Dynamic Adaptive Algorithm finds the intersection estimator IE, and then calculates n and m. If the node is a requesting node, the algorithm then estimates x, the proportion of non-malicious nodes in its view. It then adjusts the number r of nodes to which a request is distributed. Next, the algorithm calculates the churn

estimator CE, and finally it adjusts the rate rmr of distributing metadata and request messages.

6.4.1 Parameters and Variables

The parameters and variables for the Dynamic Adaptive Algorithm are defined as follows:

- $rmrMax$: The maximum rate at which a node is allowed to distribute requests or metadata.
- $kMax$: The upper bound on the number k of matches.
- c: The weighting factor of the EWMA() method used for the unnormalized observed probabilities.
- d: The number of requests in the initial transient for the EWMA() method.
- $view$: A node's current view of the membership.
- n: The size of the node's current view.
- m: The number of nodes in a source node's view to which it distributes metadata, *i.e.*, $m = 2\sqrt{n}$ nodes.
- r: The number of nodes in a requesting node's view to which it distributes requests, *i.e.*, $r = 2\sqrt{n}$ initially and subsequently is calculated

using $rGet()$ in order to achieve the same match probability as when $x = 1.0$.

- rmr: The node's rate of distributing requests (metadata), *i.e.*, the number of times a node sends a request (metadata) message to r (m) nodes per time unit.

- x: The estimated proportion of non-malicious nodes in a node's view, where $x = 1.0$ initially and subsequently is calculated using $xGet()$, *i.e.*, $1 - x$ is the estimated proportion of malicious nodes in the node's view.

- $xPrev$: The previous value of x.

- O: The unnormalized observed probability array used in the EWMA() method.

- $prevO$: The previous unnormalized observed probability array used in the EWMA() method.

- $numNodes$: The number of nodes to which a node sent its message, including first and possibly second tries.

- $numMatches$: The number of nodes that report matches to a requesting node.

- $joined$: The number of nodes that a node discovered have joined the membership since its last message.

- $left$: The number of nodes that a node detected have left the membership since its last message.

- IE: The intersection (I) estimator obtained by random sampling and defined by:

$$IE = \frac{numNodes - left}{numNodes} \tag{6.1}$$

  The IE estimator is used to estimate n, m and r in I.

- nIE: The estimate of the number of nodes in the intersection I of the node's view of the membership and the actual membership.

- mIE: The estimate of the number m of nodes in the intersection I of the node's view of the membership and the actual membership.

- rIE: The estimate of the number r of nodes in the intersection I of the node's view of the membership and the actual membership.

- CE: The CE estimator, an estimate of the membership churn (leaves and joins), obtained by random sampling and defined by:

$$CE = \frac{left + joined}{numNodes} \tag{6.2}$$

  The churn estimator is used to adjust the rate rmr of sending request and metadata messages.

A node's view of the membership consists of $I + L$ nodes, but the actual membership consists of $I + J$ nodes, as shown in Figure 6.7. If a node sends

its message to one of the L (malicious or non-malicious) nodes, then it detects and removes that node immediately. If the node sends a request to one of the I nodes and that node is non-malicious and has a match, then the matching node reports that match. If the node sends a request to one of the I nodes and that node does not have a match, then that node responds with its newly joined nodes and no match.

The intersection estimator IE provides an estimate of the intersection I, *i.e.*, the proportion of nodes from which it receives responses. It is obtained, using random sampling, by dividing the number of nodes from which the node receives responses (*i.e.*, $numNodes - left$) by $numNodes$. If the algorithm were to use the hypergeometric distribution in Equation (3.1) using the value $I + L$, it might obtain results that are different from those obtained using the value $I + J$. Thus, as an approximation, the algorithm uses I as the size of the node's current view of the membership. However, a node does not actually know the value of I. Therefore, the algorithm first estimates the values nIE, mIE and rIE, using the IE estimator before it applies Equation (3.1) to find the estimated proportion x of non-malicious nodes.

A node does not choose the newly joined nodes in J to which to send its message, because it has not yet discovered those nodes. However, other nodes might be aware of those nodes and might have sent metadata to them. Thus, again, the results obtained from the hypergeometric distribution in Equation

(3.1) might be affected. However, a node discovers newly joined nodes quite quickly, so the effect is small even with high joining rates and we ignore it.

The churn estimator CE provides an estimate of the membership churn (leaves and joins). It is obtained, using random sampling, by adding $left$ and $joined$, and then dividing by $numNodes$, the number of nodes to which the node sent its message. The rate rmr of sending metadata and requests is increased or decreased, depending on the churn estimator CE.

6.4.2 Pseudocode for the Dynamic Adaptation Algorithm

The pseudocode for the Dynamic Adaptive Algorithm is given in Figure 6.12.

In line 1, a newly joining node gets its initial view of the membership of size n from a bootstrapping node. In line 2 r, x, $xPrev$ and i are initialized. In line 3, the algorithm sets the entries of the unnormalized observed probability array O to zero and, in line 4, it sets $numNodes$, $numMatches$, $joined$ and $left$ to 0. In line 5, the newly joining node distributes a join message to randomly chosen nodes in its initial view. As a result, it might learn about newly joined or left nodes and obtain a new view; in line 6, the algorithm determines the number n of nodes in this new view and, in line 7, it calculates r, m and rmr.

```
DynamicAdaptive(rmrMax, kMax, c, d)
1 view = getBootView(n)
2 r ← 2√n; x ← 1.0; xPrev ← 1.0; i ← 0
3 for (k ← 0 to kMax) do O(k) ← 0
4 numNodes ← 0; numMatches ← 0; joined ← 0; left ← 0
5 distribute(view, r, numNodes, numMatches, joined, left)
6 n ← n + joined − left
7 r ← 2√n; m ← 2√n; rmr ← 10
8 while (true) do
9   nextTime ← time + (timeunit/rmr)
10  wait until (time == nextTime)
11   if (isSourceNode) then
12     distribute(view, m, numNodes, numMatches, joined, left)
13   if (isRequestingNode) then
14     distribute(view, r, numNodes, numMatches, joined, left)
15     prevO ← O
16     O ← EWMA(numMatches, prevO, kMax, c)
17     i ← i + 1
18  IE ← (numNodes − left)/numNodes
19  nIE ← IE × n; mIE ← IE × m; rIE ← IE × r
20  n ← n + joined − left
21  m ← 2√n
22  if ((isRequestingNode) and (i >= d)) then
23     x ← xGet(O, nIE, mIE, rIE, kMax)
24  if (xPrev == x) then
25     r ← rGet(n, m, 0.9817, x)
26  else xPrev ← x
27  if (isSourceNode) then
28     sendMoreMetadata(view, m, mIE)
29  CE ← (left + joined)/numNodes
30  if (CE > 1/rmrMax) then
31     rmr ← rmrMax × CE
32  else rmr ← 1
33  numNodes ← 0; numMatches ← 0; joined ← 0; left ← 0
```

Figure 6.12: Pseudocode for the Dynamic Adaptive Algorithm.

Line 8 starts an infinite loop where, in line 9, $nextTime$ is set to the current $time$ plus $timeunit/rmr$, which is the time when the node sends its next request and/or metadata message. As time passes, $time$ is automatically incremented (not shown in the pseudocode); moreover, $timeunit$ is the length of the time unit. In line 10, the algorithm waits until the current $time$ reaches the $nextTime$.

In line 11, the algorithm checks whether the node is a source node. If so, in line 12, it calls $Distribute()$ to distribute the node's metadata message to m randomly chosen nodes in the node's view. The $Distribute()$ method returns $numNodes$, $numMatches$, $joined$, $left$, and the (possibly new) $view$ of the membership.

In line 13, the algorithm checks whether the node is a requesting node. If so, in line 14, it calls $Distribute()$ to distribute the node's request message to r randomly chosen nodes in the node's view. In line 15, the algorithm sets the previous unnormalized observed probability array $prevO$ to O and then, in line 16, calls the EWMA() method to calculate the current unnormalized observed probability array O. In line 17, it increments i.

In line 18, the algorithm calculates the intersection estimator IE for the proportion of nodes in the node's current view of the membership and the actual membership, using random sampling. Then, in line 19, it calculates the estimates nIE, mIE and rIE of n, m and r in the intersection I. Then, in

line 20, the algorithm updates n based on the current values of n, $joined$ and $left$ and, in line 21, it sets m to $2\sqrt{n}$.

In line 22, the algorithm checks whether the node is a requesting node and whether i is greater than or equal to d, the initial transient after which it starts to make estimates of x. If so, in line 23, the algorithm calls $xGet()$ to estimate the value of x.

In line 24, the algorithm checks whether $xPrev$ is equal to the current x (*i.e.*, whether it has encountered two identical estimates of x before it changes the value of r). If so, in line 25, the algorithm calls $rGet()$ to update the value of r. Otherwise, in line 26, it sets $xPrev$ to the current x.

In line 27, the algorithm checks whether the node is a source node. If so, in line 28, it checks how many of the nodes to which it previously sent metadata have now left, and distributes its previously distributed metadata to that many more nodes, to increase the number of nodes holding the metadata up to mIE nodes.

In line 29, the algorithm calculates the churn estimator CE using the values of $left$, $joined$ and $numNodes$ it obtained in the loop. Then, in line 30 to line 32, the algorithm calculates the value of rmr for the next time step, using the value of CE. In line 33, the algorithm sets $numNodes$, $numMatches$, $joined$ and $left$ to 0 before it goes back to line 7 to repeat these steps indefinitely.

6.4.3 Distribute Method

The *Distribute*() method is used to distribute join messages, request messages, and metadata messages to *rm* nodes. This method makes two attempts to distribute a message to randomly chosen nodes in a node's view, in order to receive *rm* responses. From those responses, or the lack thereof, the node learns about newly joined and left nodes, and updates its view accordingly.

The variables for the *Distribute*() method are defined as follows:

- *try*: The number of times that a node sends its message, in an attempt to receive responses from *rm* nodes.
- *responses*: An array that stores the responses to a message that a node distributed to randomly chosen nodes in its first or second try.
- *responded*: The number of nodes that responded to a node's message in its first and possibly second try.
- *isNew*: A boolean that indicates whether the discovered node is actually a new node or is a node that was already present in the node's view of the membership.

The pseudocode for the *Distribute*() method is shown in Figure 6.13.

In line 1, *try* and *responded* are initialized. The loop starting with line 2 is executed once if, on the first try, *responded* is equal to *rm*; otherwise, it is executed twice.

```
Distribute(view, rm, numNodes, numMatches, joined, left)
1 try ← 1; responded ← 0
2 while ((try <= 2) and (responded < rm)) do
3   responses ← sendMessage(view, rm − responded)
4   numNodes ← numNodes + (rm − responded)
5   if (try == 1) then first ← 0; last ← rm − 1
6   else first ← rm; last ← rm + (rm − responded) − 1
7   for (j ← first to last) do
8   if (responses[j].noResponse) then
9     removeNode(view, responses[j].node)
10    left ← left + 1
11  else
12    responded ← responded + 1
13    if (responses[j].newJoin) then
14      isNew = addNode(view, responses[j].newNode)
15      if (isNew) then
16        joined ← joined + 1
17    if (responses[j].hasMatch) then
18      numMatches ← numMatches + 1
19  try ← try + 1
```

Figure 6.13: Pseudocode for the *Distribute*() method.

In line 3, the node sends its message to $rm - responded$ nodes, and waits for the responses from those nodes. In line 4, the algorithm sets $numNodes$ to $numNodes + (rm - responded)$. In line 5 and line 6, the algorithm sets $first$ and $last$ for $try = 1$ and $try = 2$. When $try = 1$, the message is sent to rm nodes. When $try = 2$, the message is sent to the number of nodes that failed to respond when $try = 1$. Then, in line 7, the algorithm iterates through the *responses* vector. In line 8, the algorithm checks whether the node received a response from node j. If not, in line 9, it removes the non-responsive node from the node's current view and, in line 10 it increments $left$. Otherwise

(line 11), in line 12, it increments *responded*. Then, in line 13, the algorithm checks whether j's response contains a newly joined node. If so, in line 14, it calls $addNodes()$ to add the newly joined node to the node's view. The method $addNode()$ returns a boolean $isNew$ to indicate that the newly joined node is indeed new and not already present in the node's view. In line 15, the algorithm checks whether $isNew$ is true and, if so, in line 16, it increments *joined*. In line 17, the algorithm checks whether j's response contains a match and, if so, in line 18, it increments $numMatches$.

In line 19, the algorithm increments try, and control then returns to the while loop in line 2 to determine whether try is less than or equal to 2 and *responded* is less than rm. If so, the algorithm goes through the while loop again.

6.5 Performance Evaluation

6.5.1 Experimental Methodology

To evaluate the Dynamic Adaptive Algorithm, we performed experiments using a simulation of iTrust. In the simulation, we can control the rates at which nodes join and leave the membership, which a real-world deployment would not allow us to do. Moreover, in the simulation, we can compare a node's current view of the membership against the actual membership, and

we can compare the estimated proportion of malicious nodes against the actual proportion of malicious nodes.

In the simulation program, first the number N of nodes in the actual membership is initialized to $N = 1024$. The program adds all of the nodes in the actual membership to each node's view, so that each node has the complete initial membership and $n = 1024$. Then, at each time step, some nodes join the membership, leave the membership, distribute metadata messages, and distribute request messages. Moreover, some nodes might behave maliciously, by not responding to a request with a match when they do have a match.

When a node joins the membership, it obtains the values of n, $rmrMax$, $kMax$, c and d. In the simulation, we set $rmrMax = 30$, $kMax = 15$, $c = 0.97$ and $d = 40$, which we have determined by experimentation.

We investigate the effectiveness of the Dynamic Adaptive Algorithm by considering two extended scenarios, each consisting of five individual scenarios. In the first extended scenario, we consider different values of $JR = LR$ with a fixed value of X, namely, $X = 0.6$, *i.e.*, with a relatively high proportion of malicious nodes, namely, $1 - X = 0.4$. In the second extended scenario, we consider different values of X with fixed values of JR and LR, namely, $JR = LR = 300$, *i.e.*, with high joining and leaving rates.

In the simulation, a time unit comprises 300 time steps. If $JR = LR = 300$, for example, one node joins and one node leaves at each time step. In this

example, if $rmr = 10$, a node distributes a metadata message or a request message (or both) every 30 time steps.

6.5.2 Performance Metrics

The performance metrics for the Dynamic Adaptive Algorithm are as follows:

- JND: The joins not discovered, *i.e.*, the proportion of newly joined nodes in the actual membership that a node has not discovered have joined the membership at a particular time, defined by:

$$JND = \frac{J}{I + J} \tag{6.3}$$

- LND: The leaves not detected, *i.e.*, the proportion of leaving nodes in a node's view that the node has not detected have left the membership at a particular time, defined by:

$$LND = \frac{L}{I + L} \tag{6.4}$$

- MA: The membership accuracy, *i.e.*, the number of nodes in a node's view that are also in the actual membership divided by that number of nodes plus the number of leaving nodes not detected plus the number of joining nodes not discovered, defined by:

$$MA = \frac{I}{I + L + J} \tag{6.5}$$

- MP: The match probability of one or more responses for a request, averaged over all requesting nodes.

- RT: The response time for a request, from the time that a requesting node starts distributing its request message to other nodes until it finishes receiving their responses, including responses for the first and second tries, averaged over all requesting nodes.

- MC: The message cost per node per time unit, calculated as an average over all nodes over time.

6.5.3 Varying JR and LR

First, we consider the extended scenario in which JR and LR vary and $X = 0.6$. This extended scenario comprises the following five scenarios:

- Scenario 1: $JR = LR = 0$ for time 0 to 20,000

- Scenario 2: $JR = LR = 10$ for time 20,000 to 40,000

- Scenario 3: $JR = LR = 50$ for time 40,000 to 60,000

- Scenario 4: $JR = LR = 100$ for time 60,000 to 80,000

- Scenario 5: $JR = LR = 300$ for time 80,000 to 100,000.

Figure 6.14 shows the graphs of JND, LND, x, r, rmr, MA and MP for these five scenarios.

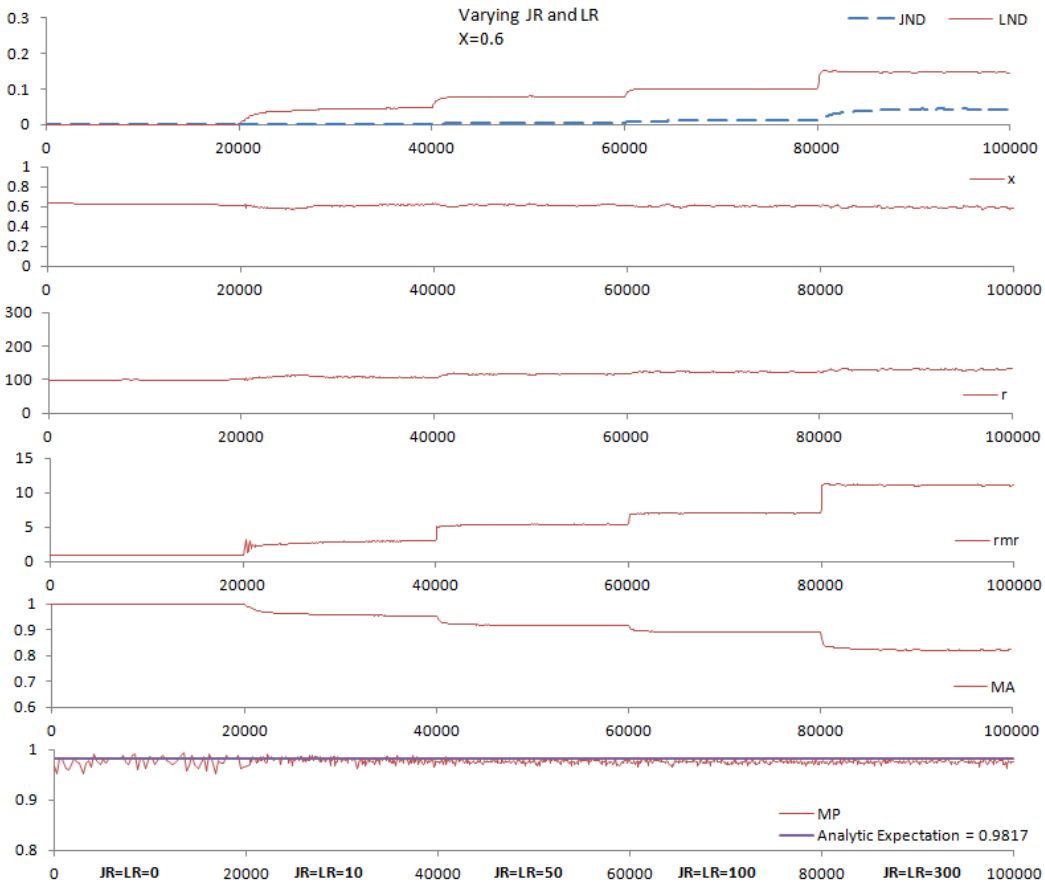

Figure 6.14: Graphs of JND, LND, x, r, rmr, MA and MP for the Dynamic Adaptive Algorithm, where JR and LR vary and $X = 0.6$.

In the extended scenario, JR and LR increase from 0 to 10 to 50 to 100 and then to the high rate of 300. The values of JND remain quite low, but the values of LND increase steadily, starting at 0 when $LR = 0$ and reaching 0.1472 when $LR = 300$, because the node cannot detect leaving nodes sufficiently rapidly.

As JR and LR (and hence JND and LND) increase, the chi-squared method's estimate of the proportion x of non-malicious remains close to the actual value $X = 0.6$. The defensive adaptation algorithm increases the num-

$JR = LR$	0	10	50	100	300	Mean
MA	1.0000	0.9590	0.9190	0.8917	0.8234	0.9059
MP	0.9758	0.9791	0.9761	0.9752	0.9753	0.9764
RT	6.0169	11.2556	11.3719	11.3451	11.3984	10.9300
MC	0.6541	1.8480	4.0174	5.6518	9.4487	4.3240

Table 6.1: Varying JR and LR with $X = 0.6$. The columns labeled 0, 10, 50, 100, 300 contain the mean values of MA, MP, RT and MC for the given scenarios and the last column contains the mean values over all five scenarios.

ber r of nodes to which the requesting node distributes its request from 99.1289 in scenario 1 to 129.6163 in scenario 5.

As JR and LR increase, the churn estimator CE increases, which causes the algorithm to increase rmr, reaching 11.0787 in scenario 5, so that the node discovers joining nodes and detects leaving nodes more rapidly.

Table 6.1 shows the mean values of MA, MP, RT and MC for each of the five scenarios and finally over all five scenarios. The mean membership accuracy MA decreases as JR and LR increase, because a node can not detect leaving nodes rapidly enough when LR is much higher than the rate rmr of distributing request and metadata messages. Nevertheless, the mean value of MA over all five scenarios is 0.9059, which is quite good. Likewise, the overall mean value of MP for all five scenarios is high (0.9764), which is close to the analytical expectation (0.9817).

The response time RT for the first scenario is much less than that of the other scenarios, because no nodes are joining or leaving the membership and, thus, a requesting node obtains responses from all of the nodes on its first try. In the other four scenarios, the requesting node sends its request message to

additional nodes to compensate for nodes detected to have left. The response times for these four scenarios are quite similar, and the mean value of RT for all five scenarios is 10.9300.

Finally, as JR and LR increase, more nodes are discovered to have joined and detected to have left the membership, increasing CE and thus rmr, which results in an increase in the message cost MC. However, the overall mean value of MC for all five scenarios is 4.3240, which we consider to be reasonable.

6.5.4 Varying X

Next, we consider the extended scenario in which X varies and $JR = LR = 300$. The extended scenario comprises the following five scenarios:

- Scenario 1: $X = 1.0$ for time 0 to 20,000
- Scenario 2: $X = 0.8$ for time 20,000 to 40,000
- Scenario 3: $X = 0.6$ for time 40,000 to 60,000
- Scenario 4: $X = 0.4$ for time 60,000 to 80,000
- Scenario 5: $X = 0.2$ for time 80,000 to 100,000.

Figure 6.15 shows the graphs of JND, LND, x, r, rmr, MA and MP for these five scenarios.

In the extended scenario, X decreases from 1.0 to 0.8 to 0.6 to 0.4 to 0.2, *i.e.*, the proportion $1 - X$ of malicious nodes increases from 0.0 to 0.2 to 0.4

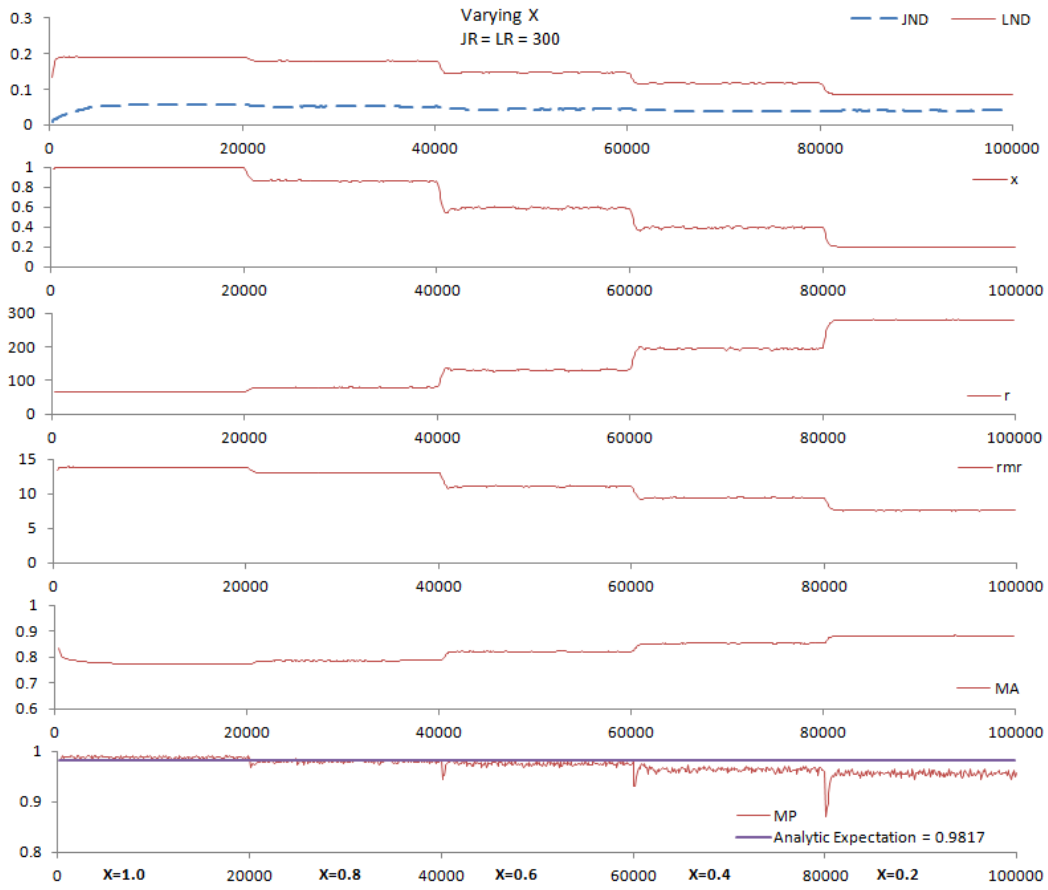

Figure 6.15: Graphs of JND, LND, x, r, rmr, MA and MP for the Dynamic Adaptive Algorithm, where X varies and $JR = LR = 300$.

to 0.6 to the high proportion of 0.8. The values of JND remain relatively low, and the values of LND decrease from 0.1887 in scenario 1 to 0.1790, 0.1476, 0.1174 and 0.0863 in scenario 5, as a result of increasing r.

As the number of malicious nodes increases, the requesting node receives fewer matches, and the chi-squared estimate x decreases from 0.9969 in scenario 1 to 0.8642, 0.5925, 0.3955 and 0.2039 in scenario 5. These reductions in x cause the defensive adaptation algorithm to increase r from 66.2033 in scenario 1 to 78.9022 to 129.7150 to 193.5547 to 278.5190 in scenario 5, in or-

der to maintain a high match probability. As r is increased, a requesting node discovers newly joining nodes and detects leaving nodes and, thus, updates its view of the membership, which causes LND to decrease and MA to increase.

As r increases, CE decreases, which causes the algorithm to reduce rmr from 13.8222 in scenario 1 to 13.0271 to 11.0930 to 9.4167 to 7.6008 in scenario 5, in order to reduce the message cost. Contrast the behavior here, with increasing X and constant $JR = LR = 300$, increasing r and decreasing rmr, with the behavior in Figure 6.14, with increasing JR and LR and constant $X = 0.6$, resulting in nearly constant x and increasing rmr.

The match probability MP remains high for the first three scenarios but tends to fluctuate more in scenarios 4 and 5. When X is low, there are few nodes that report a match, resulting in a low match probability. Because rmr is much lower than LR, a source node cannot detect leaving nodes rapidly enough. Thus, there is a period when some of the nodes that hold the metadata leave the membership but the source node has not yet updated its view and re-distributed the metadata to more nodes. Thus, a node that makes requests in this period has a lower match probability. Later, after the source node updates its view, it re-distributes its metadata to more nodes, in order to increase the match probability. Also interesting is the brief period in which MP is significantly reduced following the transition from $X = 0.4$ to $X = 0.2$. This reduction occurs because the defensive adaptation algorithm has not yet

$X =$	1.0	0.8	0.6	0.4	0.2	Mean
MA	0.7767	0.7852	0.8197	0.8519	0.8802	0.8228
MP	0.9880	0.9795	0.9750	0.9632	0.9546	0.9720
RT	11.4259	11.5419	11.3971	11.2291	11.0286	11.3246
MC	6.4595	7.1495	9.4751	11.6760	13.6993	9.6919

Table 6.2: Varying X with $LR = JR = 300$. The columns labeled 1.0, 0.8, 0.6, 0.4, 0.2 contain the mean values of MA, MP, RT and MC for the given scenarios and the last column contains the mean values over all five scenarios.

detected the change and increased r. Similar but smaller reductions also occur at the transitions from $X = 0.8$ to $X = 0.6$ and from $X = 0.6$ to $X = 0.4$.

Note also that in scenario 5, the graph of the estimate x is very flat. The reason is that the normalized analytical probability curve for $X = 0.2$ is sufficiently different from the curves for $X = 0.4, 0.6, 0.8, 1.0$, as shown in Figure 6.8. Thus, in the case $X = 0.2$, the algorithm seldom makes a mistake when it uses the chi-squared method.

Table 6.2 shows the mean values of MA, MP, RT and MC for each of the five scenarios and finally over all five scenarios. The mean membership accuracy MA increases as X decreases, because the algorithm increases r as X decreases. When r is high, a requesting node detects leaving nodes and discovers newly joined nodes and, thus, updates its view of the membership. The overall mean value of MA for all five scenarios is 0.8228, which is quite good. On the other hand, the mean match probability MP slightly decreases as X decreases. However, the overall mean value of MP for all five scenarios is still quite high (0.9720).

The response time RT is about the same for all five scenarios, with a overall mean value of 11.3246 for all five scenarios. Finally, the message cost MC increases as X decreases and, thus, r increases. However, the overall mean value of MC for all five scenarios is 9.6919, which we consider to be reasonable.

6.5.5 Effectiveness of the Dynamic Adaptive Algorithm

Overall, these experiments demonstrate that the Dynamic Adaptive Algorithm is effective in detecting leaving nodes, discovering newly joining nodes, and estimating the proportion of non-malicious (and, thus, malicious) nodes.

When the actual proportion X of non-malicious nodes increases or decreases, the Dynamic Adaptive Algorithm finds the estimated proportion x of non-malicious nodes. The estimate x is close to the actual value of X, even when JR and LR increase to 300 and are thus much higher than the rate rmr of distributing request and metadata messages. Using x, the algorithm then adjusts the number r of nodes to which a requesting node distributes its request, which enables the algorithm to maintain a high match probability even when some of the nodes are malicious.

Moreover, when JR and LR are high, the Dynamic Adaptive Algorithm increases the rate rmr of distributing request and metadata messages so that a node discovers joining nodes and detects leaving nodes more rapidly, to obtain a high membership accuracy and a high match probability. On the other hand,

when JR and LR are low, the Dynamic Adaptive Algorithm decreases rmr, to maintain a reasonable message cost.

Thus, the Dynamic Adaptive Algorithm adjusts r and rmr appropriately to obtain a reasonable membership accuracy and a high match probability, while still maintaining a reasonable message cost.

6.6 Summary

In this chapter, we have described a Dynamic Adaptive Algorithm for iTrust, that maintains a node's local view of the membership and that protects iTrust against malicious nodes. The algorithm uses random sampling and statistical inference to estimate metrics that are not directly observable in the distributed system. Experimental results demonstrate that the algorithm works well when the membership has a lot of churn, when the membership is stable, and when the membership has a lot of malicious nodes.

Chapter 7

Conclusions and Future Work

This Ph.D. Dissertation addresses a distributed information publication, search and retrieval system, named iTrust, with no centralized mechanisms and no centralized control. It presents statistical algorithms for detecting and defending against malicious attacks in iTrust, and adaptive membership management algorithms for iTrust in the presence of membership churn. Finally, it combines these statistical inference and dynamic adaptation algorithms to manage both membership churn and malicious nodes in iTrust. In this chapter, we summarize the specific contributions and present possible future work.

7.1 Trustworthy Distributed Search and Retrieval

In Chapter 3, we presented the iTrust system, a novel information publication, search and retrieval system with no centralized mechanisms and no centralized control. iTrust involves distribution of metadata and requests,

matching of requests and metadata, and retrieval of information corresponding to the metadata. The iTrust system is particularly valuable for individuals who wish to share information, without having to worry about subversion or censorship of information. The very existence of iTrust can help deter attempts to subvert conventional Internet search mechanisms, and can provide assurances to individuals that the information they seek will be available to them.

Our contributions include:

- Implementation of the iTrust distributed publication, search and retrieval system for the Internet with no centralized mechanisms and no centralized control.

- Evaluation of iTrust which shows that for appropriate choices of the parameters, the probability of matching a query is high, even if some of the participating nodes are subverted or non-operational.

In future work, we plan to investigate the efficiency, scalability, and reliability of the iTrust system in Emulab. We also plan to investigate different classes of nodes, effects of geographical locations, and network and processing loads. In addition, we plan to investigate the scalability of iTrust to thousands of nodes and to extrapolate those results to millions of nodes. Furthermore, we plan to evaluate the ease of installation and use of iTrust with various user populations.

In the future, we also plan to apply the ideas of iTrust presented in this Ph.D. Dissertation to social networks, so that a user has the freedom to share his/her files, such as photos, directly with others without going to a centralized administrator. Similarly, we plan to apply the ideas of iTrust to a music library search engine, so that a musician or performer can share his/her work in a music network and make his/her performances more dynamic and more diverse.

7.2 Protecting against Malicious Attacks in iTrust

In Chapter 4, we presented novel statistical algorithms for detecting and defending against malicious attacks in the iTrust information retrieval network. These algorithms employ the exponential weighted moving average method and the chi-squared test to calculate the empirical probabilities for the numbers of responses to a request. The detection algorithm compares the normalized empirical probabilities against the normalized analytical probabilities for the numbers of matches, to estimate the proportion of nodes that are subverted or non-operational. The defensive adaptation algorithm increases or decreases the number of nodes to which the requests are distributed, to maintain a high probability of a match. Extensive experimental evaluations demonstrate the

effectiveness of the algorithms for detecting malicious attacks and defending against them.

Our contributions include:

- Recognition that multiple responses to a request provide valuable information about the network that is difficult to obtain directly from the nodes, and use of the multiple responses to estimate the proportion of non-operational or subverted nodes.

- Demonstration that in iTrust, individual nodes can detect a potential malicious attack, and can adapt to an attack to maintain trustworthy distributed search and retrieval even when the network is under attack.

- Statistical inference techniques such that the nodes can infer the characteristics of the network that are not measurable directly. In other words, the nodes infer useful information about the network by local observations of the behaviors of the nodes.

In future work, we plan to investigate other kinds of malicious attacks on iTrust and to develop other detection and defensive adaptation algorithms to handle such attacks. We also plan to scale the number of nodes in the iTrust network to millions of nodes. Such scaling requires that each node is aware of only a small neighborhood of the nodes in the network and that the metadata and the requests are forwarded from one neighborhood to another.

7.3 Membership Management for iTrust

In Chapter 5, we presented membership algorithms for the iTrust information retrieval network. The membership algorithms for iTrust allow each member to maintain its own local view of the membership, and to keep that local view close to the actual membership. A node that receives a request sends to the requesting node a response that contains newly joined member(s) in its local view of the membership. The node also sends in its response the URL of the document, if the keywords in the query match metadata that it holds.

A requesting node discovers newly joining nodes from the responses it receives to its requests. Likewise, a requesting node detects leaving (non-operational) nodes when it does not receive responses from those nodes before a timeout occurs, or when it receives an error code from TCP.

As our simulations demonstrate, for appropriate values of the parameters, particularly for the Combined Adaptive Membership Protocol, the membership accuracy, response time, and message cost are reasonable, and the match probability is high. The Combined Adaptive Membership Protocol works well not only when the membership has a lot of churn but also when the membership is stable.

Our contributions include:

- Demonstration that each member can maintain its own local view close to the actual membership, without the use of a consensus algorithm to achieve an agreed membership.

- Exploitation of messages already required by the iTrust messaging protocol for search and retrieval, rather than requiring extra messages for membership.

- Introduction of a churn estimator which is used to adjust a node's requesting rate dynamically.

- Demonstration that the dynamic adaptive membership algorithms are effective in estimating the churn in several scenarios based on the responses a node receives to its requests.

In future work, we plan to refine our adaptive membership algorithms to handle millions of nodes. Such scaling requires that each node is aware of only a small neighborhood of the nodes in the network. In addition, we plan to continue our investigation of the performance of the iTrust membership protocols in other scenarios.

7.4 Statistical Inference and Dynamic Adaptation for iTrust

In Chapter 6, we described a dynamic adaptive algorithm for iTrust based on statistical inference, that maintains a node's local view of the membership and that protects iTrust against malicious attacks. Experimental results demonstrate that the algorithm works well when the membership has a lot of churn, when the membership is stable, and when the membership has a lot of malicious nodes.

Our contributions include:

- Use of random sampling and statistical inference to infer information that is not easy to obtain or that is expensive to collect, such as detecting the proportion of malicious nodes or the size of the membership when the membership churn is high.

- Dynamic adjustment of the number of nodes to which a requesting node distributes its requests and dynamic adjustment of the rate of distributing request and metadata messages to obtain a reasonable membership accuracy and a high match probability, while maintaining a reasonable message cost.

- Demonstration of the effectiveness of the dynamic adaptive algorithm in detecting leaving nodes, discovering newly joining nodes, and estimating

the proportion of non-malicious and, thus, malicious nodes in the iTrust membership.

In the future, we plan to apply these statistical inference and adaptation techniques to other algorithms and applications, such as scheduling algorithms for many core architectures. We also plan to create other dynamic adaptation algorithms using random sampling and statistical inference for distributed systems and computer networks, more generally.

Bibliography

[1] A. Awan, R. A. Ferreira, S. Jagannathan, and A. Grama. Distributed uniform sampling in unstructured peer-to-peer networks. In *Proceedings of the 39th International Conference on System Sciences*, pages 223–232, Kauai, HI, January 2006.

[2] C. M. Badger, L. E. Moser, P. M. Melliar-Smith, I. Michel Lombera, and Y. T. Chuang. Declustering the iTrust search and retrieval network to increase trustworthiness. In *Proceedings of the 8th International Conference on Web Information Systems and Technologies*, pages 312–322, Porto, Portugal, April 2012.

[3] R. Belen. *Detecting Disguised Missing Data*. PhD thesis, Middle East Technical University, 2009.

[4] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. P2P content search: Give the Web back to the people. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems*, Santa Barbara, CA, February 2006.

[5] S. Bianchi, P. Felber, and M. Gradinariu. Content-based publish/subscribe using distributed r-trees. In *Proceedings of Euro-Par*, pages 537–548, Rennes, France, August 2007.

[6] A. Binzenhofer and K. Leibnitz. Estimating churn in structured P2P networks. In *Proceedings of the 20th International Teletraffic Conference on Managing Traffic Performance in Converged Networks*, pages 630–641, Ottawa, Canada, June 2007.

[7] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the 7th International Conference on the World Wide Web*, pages 107–117, Brisbane, Australia, April 1998.

[8] M. Castro, P. Druschel, A. M. Kermarrec, and A. I. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489–1499, 2002.

[9] T. D. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost. On the impossibility of group membership. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 322–330, Philadelphia, PA, May 1996.

[10] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P systems scalable. In *Proceedings of the ACM Special Interest Group on Data Communication Conference*, pages 407–418, Karlsruhe, Germany, August 2003.

[11] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for Internet databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 379–390, Dallas, TX, June 2000.

[12] G. V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: A comprehensive study. *ACM Computing Surveys*, 33(4):427–469, 2001.

[13] Y. T. Chuang, P. M. Melliar-Smith, L. E. Moser, and I. Michel Lombera. Detection and defensive adaptation algorithms for protecting against malicious attacks in the iTrust information retrieval network. In preparation.

[14] Y. T. Chuang, P. M. Melliar-Smith, L. E. Moser, and I. Michel Lombera. Membership management for the iTrust information retrieval network. Submitted.

[15] Y. T. Chuang, P. M. Melliar-Smith, L. E. Moser, and I. Michel Lombera. Statistical inference and dynamic adaptation for the iTrust search and retrieval system. Submitted.

[16] Y. T. Chuang, P. M. Melliar-Smith, L. E. Moser, and I. Michel Lombera. Discovering joining nodes and detecting leaving nodes in the iTrust membership protocol. In *Proceedings of the International Multi-Conference on Engineers and Computer Scientists*, pages 189–194, Hong Kong, China, March 2013.

[17] Y. T. Chuang, P. M. Melliar-Smith, L. E. Moser, and I. Michel Lombera. Protecting the iTrust information retrieval network against malicious attacks. *Journal of Computing Science and Engineering*, 6(3):179–192, September 2012.

[18] Y. T. Chuang, I. Michel Lombera, P. M. Melliar-Smith, and L. E. Moser. Detecting and defending against malicious attacks in the iTrust information retrieval network. In *Proceedings of the International Conference on Information Networking*, pages 263–268, Bali, Indonesia, February 2012.

[19] Y. T. Chuang, I. Michel Lombera, L. E. Moser, and P. M. Melliar-Smith. Trustworthy distributed search and retrieval over the Internet. In *Proceedings of the International Conference on Internet Computing*, pages 169–175, Las Vegas, NV, July 2011.

[20] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, Berkeley, CA, July 2001. Springer.

[21] E. Cohen, A. Fiat, and H. Kaplan. Associative search in peer to peer networks: Harnessing latent semantics. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 51(8):1861–1881, 2007.

[22] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proceedings of the ACM Special Interest Group on Data Communications Conference*, pages 177–190, Pittsburgh, PA, August 2002.

[23] T. Condie, S. D. Kamvar, and H. Garcia-Molina. Adaptive peer-to-peer topologies. In *Proceedings of the 4th IEEE International Conference on Peer-to-Peer Computing*, pages 53–62, Zurich, Switzerland, August 2004.

[24] B. F. Cooper. Quickly routing searches without having to move content. In *Proceedings of the International Workshop on Peer-To-Peer Systems*, pages 163–172, Ithaca, NY, February 2005.

[25] A. Crainiceanu, P. Linga, A. Machanavajjhala, J. Gehrke, and J. Shanmugasundaram. P-Ring: An index structure for peer-to-peer systems. Technical report, Cornell University, 2004.

[26] F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. PlanetP: Using gossiping to build content addressable peer-to-peer information sharing communities. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, pages 236–246, Seattle, WA, June 2003.

[27] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 12–21, San Diego, CA, August 2004.

[28] C. Doulkeridis, K. Norvag, and M. Vazirgiannis. Desent: Decentralized and distributed semantic overlay generation in P2P networks. *IEEE Journal on Selected Areas in Communications*, 25(1):25–34, 2007.

[29] P. T. Eugster, P. A. Felber, R. Guerraoui, and A. M. Kermerrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.

[30] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *Proceedings of the ACM Special Interest Group on Data Communication Conference*, pages 115–126, Santa Barbara, CA, May 2001.

[31] Facebook. http://www.facebook.com.

[32] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume I. John Wiley & Sons, 1968.

[33] R. A. Ferreira, M. K. Ramanathan, A. Awan, A. Grama, and S. Jagannathan. Search with probabilistic guarantees in unstructured peer-to-peer networks. In *Proceedings of 5th IEEE International Conference on Peer-to-Peer Computing*, pages 165–172, Konstanz, Germany, August 2005.

[34] A. Galleni and D. Powell. Consensus and membership in synchronous and asynchronous distributed systems. Technical Report 96104, Laboratory for Analysis and Architecture of Systems, Toulouse, France, April 1996.

[35] A. Ganesh, A. M. Kermarrec, and L. Massoulié. SCAMP: Peer-to-Peer lightweight membership service for large-scale group communication. In *Proceedings of the 3rd International Workshop on Networked Group Communication*, pages 44–55, London, United Kingdom, November 2001.

[36] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks: algorithms and evaluation. *Performance Evaluation*, 63(3):241–263, 2006.

[37] Gnutella. http://en.wikipedia.org/wiki/Gnutella.

[38] Google. http://www.google.com.

[39] R. Goonatilake, A. Herath, S. Herath, and J. Herath. Intrusion detection using the chi-square goodness-of-fit test for information assurance, network, forensics and software security. *Journal of Computing Sciences in Colleges*, 23(1):255–263, October 2007.

[40] P. E. Greenwood and M. S. Nikulin. *A Guide to Chi-Squared Testing*, volume 280. Wiley-Interscience, March 1996.

[41] R. Guerraoui, S. B. Handurukande, A. M. Kermarrec, F. Le Fessant, K. Huguenin, and E. Riviere. Gosskip, an efficient, fault-tolerant and self organizing overlay using gossip-based construction and skip-lists principles. In *Proceedings of the 6th IEEE International Conference on Peer-to-Peer Computing*, pages 12–22, Cambridge, United Kingdom, September 2006.

[42] R. Guerraoui and A. Schiper. The generic consensus service. *IEEE Transactions on Software Engineering*, 27(1):29–41, 2001.

[43] K. P. Gummadi, A. Mislove, and P. Druschel. Exploiting social networks for Internet search. In *Proceedings of the 5th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 79–84, Irvine, CA, November 2006.

[44] A. Gupta, O. D. Sahin, D. Agrawal, and A. El Abbadi. Meghdoot: Content-based publish/subscribe over P2P networks. In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, pages 254–273, Toronto, Canada, October 2004.

[45] D. Hales. From selfish nodes to cooperative networks - Emergent link-based incentives in peer-to-peer networks. In *Proceedings of the Fourth International Conference on Peer-to-Peer Computing*, pages 151–158, Bologna, Italy, August 2004.

[46] E. N. Hanson, C. Carnes, L. Huang, M. Konyala, L. Noronha, S. Parthasarathy, J. B. Park, and A. Vernon. Scalable trigger processing. In *Proceedings of the 15th International Conference on Data Engineering*, pages 266–275, Sydney, Australia, March 1999.

[47] A. Heckert. Chi-square two sample tests, September 2006. http://www.itl.nist.gov/div898/software/dataplot/refman1/auxillar/chi2samp.htm.

[48] O. Heen, E. L. Merrer, C. Neumann, and S. Onno. Distributed and private group management. In *Proceedings of the 31st International Symposium on Reliable Distributed Systems*, pages 191–200, Irvine, CA, October 2012.

[49] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Privacy preserving P2P data sharing with OneSwarm. In *Proceedings of the ACM SIGCOMM Conference*, pages 111–122, New Delhi, India, September 2010.

[50] G. P. Jesi, D. Hales, and M. Van Steen. Identifying malicious peers before it's too late: A decentralized secure peer sampling service. In *Proceedings of the 1st International Conference on Self-Adaptive and Self-Organizing Systems*, pages 237–246, Boston, MA, July 2007.

[51] R. I. Khazan. Group membership: A novel approach and the first single-round algorithm. In *Proceedings of the 23rd ACM Symposium on Principles of Distributed Computing*, pages 347–356, Newfoundland, Canada, July 2004.

[52] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the Third SIAM International Conference on Data Mining*, pages 25–36, San Francisco, CA, May 2003.

[53] C. Leng, W. W. Terpstra, B. Kemme, W. Stannat, and A. P. Buchmann. Maintaining replicas in unstructured P2P systems. In *Proceedings of the 4th International Conference on Emerging Networking Experiments and Technologies*, pages 19–30, Madrid, Spain, December 2008.

[54] J. Liang and K. Nahrstedt. RandPeer: Membership management for QoS sensitive peer to peer applications. In *Proceedings of the 25th International Conference on Computer Communications*, pages 1–10, Barcelona, Spain, April 2006.

[55] H. Liu, X. Liu, W. Song, and W. Wen. An age-based membership protocol against strong churn in unstructured P2P networks. In *Proceedings of the 2011 International Conference on Network Computing and Information Security*, pages 195–200, Guilin, China, May 2011.

[56] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th International Conference on Supercomputing*, pages 84–95, Baltimore, MD, June 2002.

[57] S. Maki, T. Aura, and M. Hietalahti. Robust membership management for ad-hoc groups. In *Proceedings of the 5th Nordic Workshop on Secure IT Systems*, Reykjavik, Iceland, October 2000.

[58] S. Marti, P. Ganesan, and H. Garcia-Molina. Sprout: P2P routing with social networks. In *Proceedings of the 1st International Workshop on Peer-to-Peer Computing and Database*, pages 425–435, Crete, Greece, March 2004.

[59] P. M. Melliar-Smith and L. E. Moser. O-Ring: A fault tolerance and load balancing architecture for peer-to-peer systems. In *Proceedings of the 28th International Conference of the Chilean Computer Science Society*, pages 25–33, Santiago, Chile, November 2009.

[60] P. M. Melliar-Smith, L. E. Moser, I. Michel Lombera, and Y. T. Chuang. iTrust: Trustworthy information publication, search and retrieval. In *Proceedings of the 13th International Conference on Distributed Computing and Networking*, pages 351–366, Hong Kong, China, January 2012.

[61] I. Michel Lombera, Y. T. Chuang, P. M. Melliar-Smith, and L. E. Moser. Trustworthy distribution and retrieval of information over HTTP and the Internet. In *Proceedings of the 3rd International Conference on the Evolving Internet*, pages 7–13, Luxembourg City, Luxembourg, June 2011.

[62] I. Michel Lombera, L. E. Moser, P. M. Melliar-Smith, and Y. T. Chuang. Mobile decentralized search and retrieval using SMS and HTTP. *ACM Mobile Networks and Applications Journal*, 18(1):22–41, 2013.

[63] I. Michel Lombera, L. E. Moser, P. M. Melliar-Smith, and Y. T. Chuang. Mobile ad-hoc search and retrieval in the iTrust over Wi-Fi Direct network. In *Proceedings of the 9th International Conference on Wireless and Mobile Communications*, Nice, France, July 2013, to appear.

[64] J. Mischke and B. Stiller. A methodology for the design of distributed search in P2P middleware. *IEEE Network*, 18(1):30–37, 2004.

[65] R. Morselli, B. Bhattacharjee, A. Srinivasan, and M. A. Marsh. Efficient lookup on unstructured topologies. In *Proceedings of the 24th ACM Symposium on Principles of Distributed Computing*, pages 77–86, Las Vegas, NV, July 2005.

[66] Napster. http://www.napster.com/.

[67] A. Oram. *Peer-to-Peer: harnessing the benefits of a disruptive technology*. O'Reilly Media, Inc., 2001.

[68] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in Fortran: The Art of Scientific Computing*. Cambridge University Press, Cambridge, United Kingdom, 2007.

[69] W. Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.

[70] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM Conference*, pages 161–172, San Diego, CA, August 2001.

[71] J. Risson and T. Moors. Survey of research towards robust peer-to-peer networks: Search methods. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 50(17):3485–3521, 2006.

[72] S. W. Roberts. Control chart tests based on geometric moving averages. *Technometrics*, 1(3):239–250, August 1959.

[73] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the 2001 IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–350, Heidelberg, Germany, November 2001.

[74] O. D. Sahin, A. Gupta, D. Agrawal, and A. El Abbadi. A peer-to-peer framework for caching range queries. In *Proceedings. 20th International Conference on Data Engineering*, pages 165–176, Boston, MA, 2004.

[75] N. Sarshar, P. O. Boykin, and V. P. Roychowdhury. Percolation search in power law networks: Making unstructured peer-to-peer networks scalable. In *Proceedings of the 4th International Conference on Peer-to-Peer Computing*, pages 2–9, Zurich, Switzerland, August 2004.

[76] A. Schiper and S. Toueg. From set membership to group membership: A separation of concerns. *IEEE Transactions on Dependable and Secure Computing*, 3(1):2–12, 2006.

[77] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures and Protocols for Computer Communications*, pages 149–160, San Diego, CA, August 2001.

[78] L. Story and B. Stone. Facebook retreats on online tracking. *The New York Times*, 30, 2007.

[79] W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *Proceedings of the 2nd International Workshop on Distributed Event-Based Systems*, pages 1–8, San Diego, CA, June 2003.

[80] W. W. Terpstra, J. Kangasharju, C. Leng, and A. P. Buchmann. Bubblestorm: Resilient, probabilistic, and exhaustive peer-to-peer search. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures and Protocols for Computer Communications*, pages 49–60, Kyoto, Japan, August 2007.

[81] D. A. Tran and T. Nguyen. Publish/subscribe service in can-based P2P networks: Dimension mismatch and the random projection approach. In *Proceedings of 17th International Conference on Computer Communications and Networks*, pages 1–8, St. Thomas, Virgin Islands, August 2008.

[82] D. A. Tran and C. Pham. Enabling content-based publish/subscribe services in cooperative P2P networks. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 52(11):1739–1749, August 2010.

[83] D. Tsoumakos and N. Roussopoulos. Adaptive probabilistic search for peer-to-peer networks. In *Proceedings of the IEEE 3rd International Conference on Peer-to-Peer Computing*, pages 102–109, Linkoping, Sweden, September 2003.

[84] J. Viinikka and H. Debar. Monitoring IDS background noise using EWMA control charts and alert information. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection*, pages 166–187, French Riviera, France, September 2004.

[85] S. Voulgaris, D. Gavidia, and M. Van Steen. CYCLON: Inexpensive membership management for unstructured P2P overlays. *Journal of Network and Systems Management*, 13(2):197–217, June 2005.

[86] S. Voulgaris, E. Riviere, A. M. Kermarrec, and M. Van Steen. Sub-2-Sub: Self-organizing content-based publish and subscribe for dynamic and large scale collaborative networks. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems*, Santa Barbara, CA, February 2006.

[87] Y. Wang, L. Galanis, and D. J. DeWitt. Galanx: An efficient peer-to-peer search engine system. In *Technical Report*. University of Wisconsin, Madison, 2003.

[88] B. Wong and S. Guha. Quasar: A probabilistic publish-subscribe system for social networks. In *Proceedings of the 7th International Workshop on Peer-to-Peer Systems*, Tampa Bay, FL, February 2008.

[89] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems*, pages 5–14, Vienna, Austria, July 2002.

[90] J. Yang, Y. Zhong, and S. Zhang. An efficient interest-group-based search mechanism in unstructured peer-to-peer networks. In *Proceedings of the 2003 International Conference on Computer Networks and Mobile Computing*, pages 247–252, Shanghai, China, October 2003.

[91] N. Ye and Q. Chen. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. *Quality and Reliability Engineering International Journal*, 17(2):105–112, March 2001.

[92] L. Yin and G. Cao. Dup: Dynamic-tree based update propagation in peer-to-peer networks. In *Proceedings of the 21st International Conference on Data Engineering*, pages 258–259, Tokyo, Japan, April 2005.

[93] D. Zage, C. Livadas, and E. M. Schooler. A network-aware distributed membership protocol for collaborative defense. In *Proceedings of the International Conference on Computational Science and Engineering*, pages 1123–1130, Vancouver, Canada, August 2009.

[94] D. Zeinalipour-Yazti. *Information retrieval in peer-to-peer systems*. PhD thesis, University of California, Riverside, 2003.

[95] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004.

[96] M. Zhong and K. Shen. Popularity-biased random walks for peer-to-peer search under the square-root principle. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems*, Santa Barbara, CA, February 2006.

[97] B. Zhou, Q. Shi, and M. Merabti. Intrusion detection in pervasive networks based on a chi-square statistic test. In *Proceedings of the 30th International Computer Software and Applications Conference*, pages 203–208, Chicago, IL, September 2006.