3-10-2010

Scalable and Fault Tolerant Group Key Management

Nicholas A. Lupien

Scalable and Fault Tolerant Group Key Management

THESIS

Nicholas A. Lupien, Captain, USAF

AFIT/GCS/ENG/10-05

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GCS/ENG/10-05

Scalable and Fault Tolerant Group Key Management

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Nicholas A. Lupien, B.S.

Captain, USAF

March 2010

AFIT/GCS/ENG/10-05

Scalable and Fault Tolerant Group Key Management

Nicholas A. Lupien, B.S.
Captain, USAF

Approved:

Dr. K.M. Hopkinson (Chairman)	date
Maj J.M. Hemmes, PhD (Member)	date
Maj R.W. Thomas, PhD (Member)	date

Abstract

To address the group key management problem for modern networks this research proposes a lightweight group key management protocol with a gossip-based dissemination routine. Experiments show that by slightly increasing workload for the key update mechanism, this protocol is superior to currently available tree-based protocols with respect to reliability and fault tolerance, while remaining scalable to large groups. Java simulations show that the protocol efficiently distributes keys to large groups in the midst of up to 35 percent node failure rates. In addition, it eliminates the need for logical key hierarchy while preserving an overall reduction in rekey messages to rekey a group. The protocol provides a simple "pull" mechanism to ensure perfect rekeys in spite of the primary rekey mechanism's probabilistic guarantees, without burdening key distribution facilities. Parameters for overlay management and gossip are improved to minimize rekey message traffic while remaining tolerant to node failure.

Acknowledgements

I would like to thank my advisor, Dr. Ken Hopkinson, for his wisdom and guidance throughout this endeavor; my committee members, Maj Ryan Thomas and Maj Jeffrey Hemmes, for their valuable critiques along the way; and my family and friends for their support and encouragement.

Nicholas A. Lupien

Table of Contents

Page

List of Figures

List of Tables

Table Page

List of Abbreviations

Abbreviation		Page
IETF	Internet Engineering Task Force	4
RFC	request for comment	4
KD	key distributor	5
API	application programmer interface	6
PFS	perfect forward secrecy	6
WAN	wide area network	7
MBONE	multicast backbone	7
IP	Internet Protocol	8
IGMP	Internet Group Management Protocol	8
DoS	denial of service	8
GKMP	Group Key Management Protocol	9
GSAKMP	Group Secure Association Key Management Protocol	10
LKH	Logical Key Hierarchy	11

Scalable and Fault Tolerant Group Key Management

I. Introduction

Since the late 1990s, an increasing popularity in group-based messaging systems has stimulated research into secure group communication protocols. Today, secure group communication is utilized in a wide array of applications, from military command and control to power system monitoring and control and many others. The secure group communication problem is beset by several subproblems, namely the group key management problem, which asks: how can a group most efficiently share a common symmetric key secretly among communicating endpoints?

The group key management problem presents new challenges as modern network topologies evolve and practical limitations prevent the continued use of currently available algorithms. Specifically, with secure group communications spanning wide geographical regions and IP multicast becoming increasingly unsupported, group key management is forced to rely heavily on mechanism development above the network layer. Blindly applying dated multicast algorithms to overlay networks can result in undesirable consequences and exhibit low levels of fault tolerance; Chapter 2 discusses these drawbacks in detail.

1.1 Requirement for Scalability and Fault Tolerance in Future Networks

As the "Smart Grid" effort takes shape in North America, an initiative is underway to deploy and synchronize phasor measurement units (PMUs) across the continent. The resultant "synchrophasor" technology monitors the health of power systems at individual locations by measuring phase angles on transmission lines, and then comparing the waves from different locations against a common time source such as global positioning satellite (GPS) [15]. This type of measurement can be used effectively in power system state estimation and, when deployed continentally, can potentially predict national power system events [15].

The requirement for secure, scalable, wide-area communications follows directly from this initiative. Note that as a result of deregulation within the power industry, power system communications now rely almost exclusively on the Internet. As smaller companies responsible for transmission and distribution enter the picture with perhaps limited data capabilities, the need for fault tolerance becomes increasingly apparent. This challenge presents a considerable opportunity to create a fault tolerant algorithm for secure PMU communication in early stages of Smart Grid development.

1.2 Research Overview

To address the group key management problem for modern networks this research proposes a lightweight group key management protocol with a gossip-based dissemination routine. Experiments show that by slightly increasing workload for the key update mechanism, this protocol is superior to currently available tree-based protocols with respect to reliability and fault tolerance, while remaining scalable to large groups. In addition, it eliminates the need for logical key hierarchy while preserving an overall reduction in rekey messages to rekey a group. The protocol provides a simple "pull" mechanism to ensure perfect rekeys in spite of the primary rekey mechanism's probabilistic guarantees, without burdening key distribution facilities.

Benefits of this protocol are quantified versus tree-based dissemination in Java simulations on networks exhibiting various node failure rates.

1.3 Organization

This thesis is organized as follows:

- Chapter 1: Introduces the problem and research goals.
- Chapter 2: Discusses the history of group key management, gossip-based multicast algorithms, and the need for secure and fault tolerant protocols in future networks.

- Chapter 3: Discusses the specific problem domain, proposes a protocol to respond to the problem, and provides a methodology for evaluating the viability of the presented protocol over existing tree-based protocols.
- Chapter 4: Provides quantitative results for the experiments outlined in Chapter 3 and interprets the statistical significance of any cause and effect relationships.
- Chapter 5: Concludes with a summary of results and suggestions for future research.
- Appendix A: Includes additional figures from Chapter 4.

II. Literature Review

> "It is recognized that future networks will have requirements that will strain the capabilities of current key management architectures. One of these requirements will be the secure multicast requirement. The need for high bandwidth, very dynamic secure multicast communications is increasingly evident in a wide variety of commercial, government, and Internet communities. Specifically, the secure multicast requirement is the necessity for multiple users who share the same security attributes and communication requirements to securely communicate with every other member of the multicast group using a common multicast group net key."
>
> –RFC 2627 [19]

Early research into group security protocols began largely in response to several Internet Engineering Task Force (IETF) requests for comment (RFC), namely RFC 1949, "Scalable Multicast Key Distribution" in 1996 [1], and RFC 2627, "Key Management for Multicast: Issues and Architectures" in 1999 [19]. Since the publication of these RFCs, much research has been conducted to help solve the many challenges of secure group communication and the underlying problem of group key management. This chapter defines the group key management problem, outlines the specific challenges of group key management, and reviews the relevant work that has been accomplished until now. Section 2.1.5.1 summarizes current research into gossip-based multicast, and 1.1 discusses the emerging requirement for secure and fault tolerant multicast communications systems.

2.1 Group Key Management

2.1.1 Definition. Group communication enables a single node on a data network to send messages to many recipients. The group key management problem addresses the difficult task of distributing and controlling encryption keys to effectively secure group communication. Although point-to-point communications can be efficiently secured via public-key cryptography algorithms such as RSA [17] and Diffie-Hellman-Merkle [4], group communication is not efficiently encrypted with these algorithms. In order for messages with many destinations to be efficiently encrypted, members must agree upon and share a secret, symmetric key which they then use for

both encryption and decryption. Within the context of group communication, this is called the **group key**.

A **group key management protocol** addresses the distribution of group keys based upon the following primitives:

Centralization of key distribution and member management. A given protocol may stipulate that specific group members, called **key distributors** (KD), be responsible for generating, disseminating, and controlling group keys. Alternately, it may opt for a decentralized approach, leaving key distribution in the hands of ordinary group members.

Rekey frequency. One of the most defining aspects of a given protocol is the frequency at which KDs create and disseminate new keys. A protocol that opts to rekey at specific time intervals may be more reliable (nodes know when to ask for a new key if they miss an update), whereas a protocol that generates rekeys only after a group key has been compromised may be more secure [19].

Join policy. When a node requests to join a group and receive its dissemination key, the protocol specifies policy for authenticating the node, granting or denying its request, supplying it the current group key, and ensuring that it receives future key updates.

Leave policy. When a node requests to leave a group, the group key management protocol decides whether to rekey active nodes or continue using the existing group key. This depends on the particular group's security policy.

Removal policy. In the event that a node loses its group's trust and is selected for removal, the group key management protocol responds according to the given group's security policy. The consequences of this policy are closely related to those of rekey frequency. Note that a group key management protocol does not address the mechanisms that actually decide when to remove certain nodes; that is instead the responsibility of a group management or trust protocol.

Key dissemination mechanism. This describes the logical manner in which keys move from the KDs to the member nodes. A given protocol's chosen dissemination mechanism is developed based upon its specific goals and constraints. The following section outlines these goals and constraints, and Chapter 3 discusses them with respect to the specific problem domain.

2.1.2 Challenges. The fundamental challenge of group key management can be summarized in a question: how can group keys be shared most efficiently so that only the members of a group receive the key, and non-members cannot?

Like most problems in distributed systems and security, the answer is actually a second line of questions which collectively ask, "Which aspects of group key management are most important to the specific application?" The following list outlines the most common goals and constraints that arise in group key management, with respect to key initialization and group rekeying [5]:

- scalability
- platform portability and independence
- ease of use (end user and API)
- use of multicast (for the group key management system itself)
- performance (load, number of messages per rekey)
- key lifetime, escrow, perfect forward secrecy (PFS)
- cryptographic techniques (encrypting, signing)
- integration with security at other layers
- cost
- fault tolerance
- network topology (logical and physical)
- flexibility, user modifiability

The majority of research over the past decade focuses almost exclusively on optimizing some subset of the issues listed above. In particular, scalability, use of multicast, and performance have been high-value research areas and drive the majority of modern technique development. Platform independence has recently emerged as an area of interest for multicast over wide area networks (WAN), with the virtual abandonment of MBONE around the turn of the century. Development of WAN group communication protocols has been left to rely heavily on mechanism development above the network layer.

There is no single, unified solution that optimizes all of the subproblems listed above. Most solutions that exist today come with a system of trade-offs that value some aspects at the expense of others. The following subsections explore existing solutions to secure group communication and the group key management problem.

2.1.3 Early Solutions. The earliest and simplest group key management solutions, such as those found in RFC 1949 and RFC 2093, proposed frameworks to establish and disseminate session keys via flat topologies or trees. In this context, a flat topology is one in which a KD communicates directly with each member node to establish keys; as a group becomes very large, the KD will eventually become overburdened and cease functioning properly or efficiently. A tree topology utilizes a tree data structure to more efficiently delegate the key dissemination workload to interior tree nodes. See Figure 2.1 for the general concept of tree-based dissemination: the KD exists at the root of a tree whose remaining nodes represent other nodes on the network; when a new key must be disseminated, the key KD forwards the new key to its children only, and the children pass the key to their children, and so forth until every node in the tree has received the new key. These tree-based frameworks provide high levels of reliability and performance (i.e., few messages required for rekeys and removals, fast delivery from root to leaf) and are generally easy to implement. However, tree-based solutions come with the penalty of either low fault tolerance

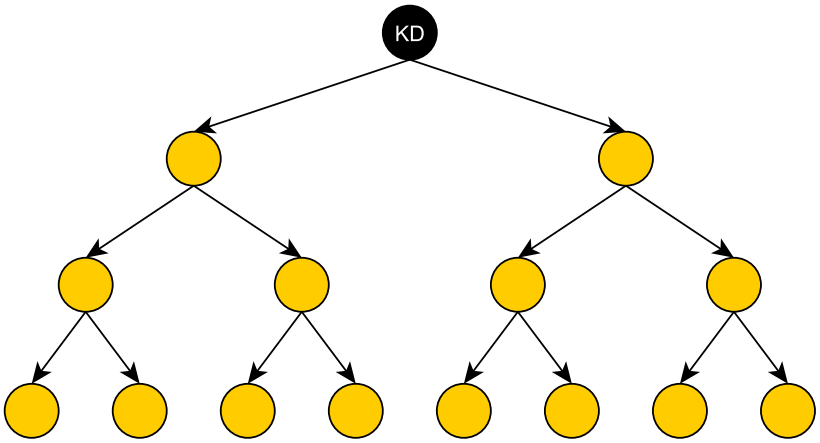

Figure 2.1: Tree-based dissemination via infrastructure or overlay

or lack of scalability. The following subsections explore the specific benefits and shortcomings of early implementations.

2.1.3.1 Core Based Tree. Ballardie *et al.* proposed RFC 1949, which specifies a solution for Internet Protocol (IP) multicast. The protocol suggests integrating "Core Based Trees" into Internet Group Management Protocol IGMPv1 to provide a key management system in which "core" and non-core routers control the flow of multicast keys, potentially with worldwide application. However, since the core routers are defined statically, the resultant tree forms a rigid design that is susceptible to modern denial-of-service (DoS) attacks. If one interior node were to become unavailable, a potentially large section of the recipient population could no longer receive key updates and other multicast messages. A solution to this vulnerability is to maintain knowledge about larger sections of the tree at each node with the intent of bypassing faulty nodes; however, because of the added responsibility for non-core nodes, the CBT implementation provides no such solution. In addition to these shortcomings, CBT's implementation at the network layer makes it applicable only on networks where IP multicast is supported. Therefore this solution is limited

in scope by practical barriers; the requirement for specific router configuration makes it infeasible for WAN and ad hoc deployments.

2.1.3.2 GKMP. Harney and Muckenhirn presented Group Key Management Protocol (GKMP) in RFCs 2093 and 2094 [10] [9]. Unlike CBT, GKMP is an application-layer protocol that defines a role-based framework for key distribution instead of a network-level infrastructure. Although not required, GKMP may utilize and benefit from an underlying multicast messaging system. The framework specifies the designation of a central KD for each group that is responsible for creating and delivering new keys, but it does not specify a physical transport mechanism. However, with GKMP, the KD must verify every key update via a receipt message from each member node. Although this provides a high level of reliability, it does not scale well as groups become very large. In particular, KDs suffer a massive performance decrease during large group rekeys since each group member must respond directly to the distributor to confirm receipt.

2.1.4 Modern Solutions.

2.1.4.1 Iolus. Mittra [16] proposes Iolus, a framework for group key management that also allows secure multicasting. Iolus presents a tree-based dissemination model that places key management in the hands of a single group security controller (GSC) and several group security intermediaries (GSI), which function together as a distributed KD. Figure 2.2 shows a small-scale sample of a group utilizing the Iolus framework. The GSC creates the group key and sends it to only the GSIs that it knows about. The innermost GSIs then forward the key onto some group members and additional GSIs. This dissemination scheme reduces the overhead of a single rekey on any given node (including the GSC) to a constant, depending upon the tree formation policy. In addition, secure unicast links between each pair provide the ability to deliver key updates without the need for special "key encryption keys" used in some implementations.

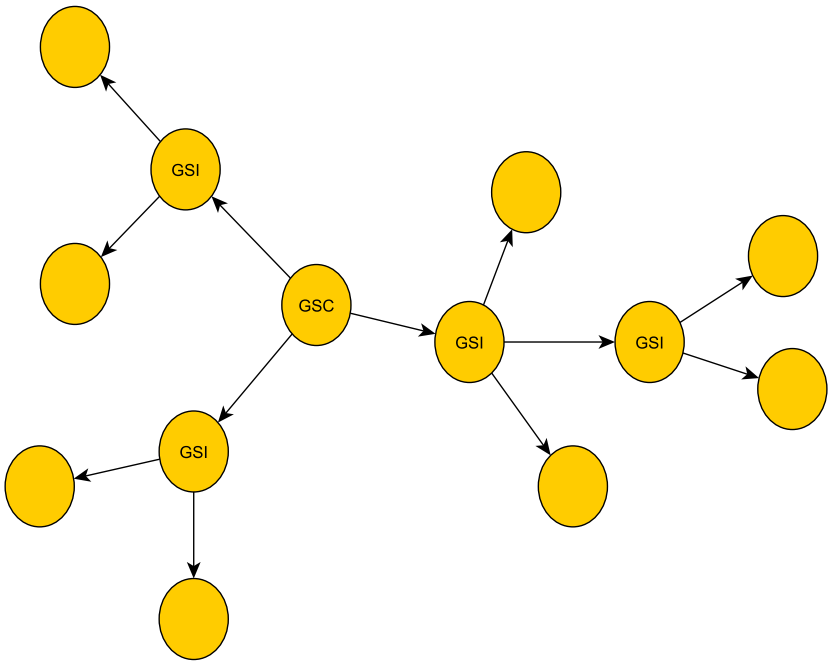

Figure 2.2: Tree-based key dissemination with Iolus utilizes a single GSC and multiple GSIs.

Although Iolus meets its goal of scalability, it lacks a necessary level of fault tolerance for large groups. Specifically, if a single GSI near the GSC becomes inoperable, the sizable portion of the group subordinate to that GSI will not receive key updates; this is a significant drawback in an asynchronous rekeying environment. Even with periodic rekeys, attempts to recover from a single faulty GSI will likely cascade into catastrophic failure when affected members between the faulty GSI and the edge bombard inner GSIs and the GSC with rekey requests.

However, the framework provides a valuable reference model upon which several other modern group key management protocols are based, including Hubenko's satellite key management protocol [12]; thus it is included in this document as a modern solution. We learn much from the delegation of key dissemination responsibilities and the use of secure unicast links for point-to-point propagation.

2.1.4.2 GSAKMP. RFC 4535 [8] defines the Group Secure Association Key Management Protocol GSAKMP, whose rekey policy, by default, relies

on Logical Key Hierarchy LKH. Developed in 1998, Logical Key Hierarchy (LKH) specifies the use of key graphs–trees that logically represent special "key encryption keys" (KEK) and users in a group [20]. Note the dissimilarity between LKH and tree-based dissemination protocols like Iolus; in LKH, interior tree nodes represent KEKs, and leaf nodes represent users. Each leaf has a single parent node that acts as the user's unique key. In tree-based dissemination protocols, interior nodes are physical endpoints.

LKH is useful for reducing bandwidth requirements when group controllers are solely responsible for the task of key dissemination. The number of rekey messages required to rekey an entire group after a single member is removed is $\alpha \log_\alpha N - 1$, where α is the degree of the tree and N is the total number of group members. However, this still tends to be a rather large burden on the group controller as N becomes very large. Also, any delegation of the key update mechanism via a tree-based dissemination routine encounters the same issues with fault tolerance as described in Section 2.1.4.1. Thus, even modern key management protocols like GSAKMP suffer many of the same issues as early protocols. Either central key managers accept the burden of rekeying large sections of groups, or the group relies on a fault-intolerant dissemination mechanism.

2.1.5 Gossip-based Security Solutions.

2.1.5.1 Gossip Definition.

Bimodal Multicast. Developed separately in 1987 by Demers and Birman [3] [2], "gossip" multicast (referred to by Demers as "epidemic") defined a framework for probabilistic message dissemination that opened a new area of research in distributed computing. This class of algorithms adapts the sociological phenomenon of the same name to network message handling; when a person becomes aware of some information that may be valuable to her community, she communicates that information to a small, random group of her friends; her friends follow suit and so

forth, until finally the message seems to have been communicated so much that there is no point in passing it any further. These are also called "epidemic" algorithms as they seem to model the spread of disease in living organisms [3].

Gossip Description. Originally designed for fully-connected networks, gossip defined a method for a single node on a network to deliver a message to all other endpoints on the same network without contacting each recipient node directly [2] [3].

As an example, consider a fully-connected network including nodes A through Z. If node A wants to deliver a message to nodes B through Z, it has several options. In a non-gossip implementation, node A contacts node B, then contacts node C, and so forth until it has contacted every other node. With gossip dissemination, node A randomly contacts a subset of other nodes (say, nodes D and F), and then nodes D and F randomly contact some other subset of nodes until the other nodes have likely received the message. Note the word "likely." The stochastic nature of gossip creates a small probability that some intended recipients will not receive a given message.

To reduce this probability (or at least account for it), gossip relies upon two basic parameters: a number of *rounds* that the message will be forwarded, and a percentage likelihood that a node will forward a message to another node. The latter is referred to in this text as *infectivity*.

Rounds can be defined several ways, but the premise is that, during one round, a node that has received a gossiped message will randomly forward the message to a neighboring node. This can be performed via a synchronized clock, where a round might correspond to one or more clock ticks. In this text, a round is defined as a non-synchronized countdown. That is, when the original message is constructed, it attaches a maximum round number. When the message is forwarded, the forwarding node decrements the rounds value; as the message traverses the network, it eventually reaches zero. When a message is received with a round number equal to zero, it is forwarded no more.

Infectivity is the likelihood that, during a round, a node will forward a gossiped message. When a node receives a gossiped message, it randomly selects recipient nodes from its neighbors and, based upon the infectivity-weighted random value, either forwards the message or simply waits that round.

These parameters impact the number of transmissions for a single message traversing a network and the likelihood that a node will miss a message. Increasing either parameter will result in more transmissions but fewer missed nodes. (In the remainder of this text, "messages," when used as a performance metric, refers to the actual number of transmissions.)

In addition to these parameters, this research investigates reducing the average k-connectivity of the network from a full mesh to increase reliability while reducing message traffic.

Benefits. Gossip algorithms provide highly probabilistic delivery guarantees while minimizing network resource consumption. In addition, this class of algorithms shows improved resilience to changes in network topology over existing multicast algorithms, particularly ones that are based upon routing trees. Of particular benefit to the problem domain, gossip-based protocols function well even when individual nodes have little or no advance knowledge about the surrounding network topology. Nodes need only know about their neighbors to successfully disseminate group messages [3] [2].

Modern Advancements. Modern implementations borrow the terms *infectivity* and *susceptibility* from epidemiology to describe the rate at which nodes seek to "push" data to their neighbors and the rate at which nodes desire data from neighbors. Hopkinson *et al.* [14] propose a "gravitational" gossip algorithm wherein member nodes propagate information based upon the product of these values per connected node pair. In [11], the same authors apply heuristics to gravitational gossip to dynamically modify gossip probabilities as network conditions change.

Other Applications in Distributed Systems. Although originally applied to reliable database replication [3], gossip is applied successfully in other areas of distributed computing, for instance in mobile ad hoc routing and peer-to-peer overlay management. Haas *et al.* [7] apply gossip in mobile ad hoc routing and suggest parameters for gossip rates; however, their experiments are applied only to perfectly manicured grid overlays (requiring many links) and do not model node failure. Jelasity *et al.* [13] apply gossip for peer sampling in wide area overlay management. The decision to utilize gossip for overlays asserts a trend in fault tolerant computing; as the viability of tree-based and other hierarchical algorithms diminishes, fault-tolerant overlay management relies comfortably upon the high level of adaptability and low maintenance costs afforded by gossip.

2.1.5.2 Gossip with Virtual Synchrony. Yan *et al.* [21] propose gossip with virtual synchrony, a layered approach that is similar in spirit to the solution presented in this manuscript, but lacks critical implementation detail. Specifically, the authors' choice of virtual synchrony seems to indicate a preference of perfect rekey reliability at perhaps a large performance cost. Additionally, the supplied performance analysis models only the effect of network size on request overhead and throughput, against a protocol not specifically intended for group key dissemination. The existence of this protocol is noted primarily to credit the authors for establishing the legitimacy of a gossip-based dissemination policy.

2.1.5.3 GOSKEY. Graham [6] presents GOSKEY, a protocol for on-demand key distribution in which message senders provide a unique key for each message they send, acting as both a group member and KD. GOSKEY aims to reduce end-to-end delay of encrypted multicast messages and increase fault tolerance within a wireless ad-hoc network.

GOSKEY's functionality is summarized with its key dissemination routine, which is invoked when a group member wishes to encrypt a message to be sent to other group members [6]:

1. Member creates new group key K.
2. Member gossips K to its neighbors via pre-established secure unicast links.
3. Member encrypts message M with K to create M_K.
4. Member broadcasts M_K to its group via some separate pathway.

To support this routine, other nodes belonging to the same group implement the following routine upon receiving K:

1. Member receives K and immediately gossips K to its own neighbors.
2. Member stores K in a cache in expectation of receiving M_K.

Additionally, recipient nodes employ the following logic upon receiving M_K:

1. If Member has K in its cache, it decrypts M_K into M.
2. If Member does not have K, it requests K from the message's originator, then decrypts M_K into M.

Graham leaves components of these procedures abstracted, most importantly the gossip parameters used during experimentation and the method for creating and destroying the secure unicast links. Although it mentions the use of an access control list (ACL), it does not specifically utilize it in the context of any group membership action such as a join or a leave.

The solution provided in this manuscript borrows only GOSKEY's dissemination mechanism to achieve whole-group key agreement. It also leverages the mechanism to support the group key management primitives (e.g., join, remove, leave) and defines a peer linking method to create and maintain the underlying topology. This is described in further detail in Chapter 3.

2.1.6 Summary. Modern group key management protocols tend to suffer the same drawbacks as early protocols, namely with respect to fault intolerant tree-based key distribution policies. While some protocols have successfully reduced the

workload placed on KDs, group security controllers, and group controllers, they have only marginally advanced in delegating key dissemination responsibilities without sacrificing fault tolerance. Recent studies suggest the viability of gossip-based key dissemination but present them in fundamentally different contexts with different goals.

Chapter 3 proposes a gossip-based group key management protocol that efficiently supports secure, wide-area communications involving many nodes, while satisfying the additional requirement of fault tolerance.

III. Methodology

This chapter defines the group key management problem, provides an algorithm to solve it for the specified problem domain, and presents methods to test it empirically against tree-based dissemination.

3.1 Problem Definition

Group communication becomes increasingly complex with the addition of the security requirement. Intuitively, as specific security requirements are added, the complexity of the required algorithm increases accordingly. Most currently available algorithms for secure group communication value reliability above other requirements, often at the cost of scalability, and sometimes with an additional requirement of topology control. This chapter expresses a model for group key management that relaxes the reliability requirement of certain operations to a highly probabilistic level. As a result, this establishes greater fault tolerance, reduces the number of messages for rekey operations, and eliminates the need for advanced topology control. Most importantly, this design provides a greater level of fault tolerance over existing designs, such as tree-based algorithms discussed in Chapter 2. Specifically, a small number of failing nodes has a far less devastating effect on reliability at any given point.

3.1.1 Problem Domain. The selected group key management problem domain is defined by a) varying levels of infrastructural connectedness within the network under test, as well as b) varying levels of node failure. It should be made clear that this failure rate is different from the rate at which nodes deliberately request and rescind their group memberships. Although a given algorithm may treat the two situations the same, they are distinctly different actions with respect to the security model, described in Section 3.2.1.

3.1.2 Goals and Hypothesis. The proposed model for secure multicast communication aims to accomplish the following:

1. Maintain algorithmic complexity for exchanging encryption keys at a constant.

2. Reduce the algorithmic complexity for subgroup joins and leaves to a constant.
3. Minimize dependence upon central keying or certificate authorities for encryption and authentication.
4. Require nodes to have no knowledge of a global topology (i.e., each node knows only its neighbors).
5. Survive DoS and replay attacks.
6. Withstand faults by employing gossip-based "pull" routines and randomized topology control

The researcher hypothesizes that, given the relaxation of the reliability requirement, the proposed algorithm and model will exhibit higher levels of fault tolerance than currently available group key management protocols, with a very manageable impact on performance.

3.2 Approach

This section describes the security model and proposes an algorithm for group key management. It also addresses any assumptions, limitations, and vulnerabilities that may result. For a comprehensive explanation of security concepts, see [18].

3.2.1 Security Model. The goal of this research is to explore a new solution to the group key management problem–to provide member nodes on a network the ability to communicate securely to a group of other nodes on the network.

In order to classify this design as a legitimate key agreement protocol, the following description addresses the properties and actions of the proposed design.

The security model is defined by the following security primitives:

Authentication For a node to be able to receive or transmit any message on any part of the network, it must have the requisite credentials to do so. This protocol takes a hybrid approach to authentication by issuing signed certificates to

operate (CTOs) for validating a node's request to join a group. This is described in further detail in Section 3.2.3.1.

Forward secrecy A message that was encrypted with an expired group key cannot be decrypted with the current group key or any future group key.

Accordingly, the actions to support the security model are defined as follows:

Join The Join primitive provides a Requestor the ability to become a Member of a key management group. In order to join, a Requestor must have credentials that are recognizable by the group controller, and must not exist on the group controller's blacklist.

Leave A node may elect to leave the group and discontinue further receipt of (and obligation to retransmit) any key update messages. Since it has not been deliberately removed, this action does not require special security measures; the node notifies its neighbors of its intent to discontinue participation only for the proper functioning of the messaging protocol

Remove When a group has lost trust in a node, the security controller must take actions to preserve forward secrecy. Within the context of group key management, the Remove operation is responsible for ensuring that members selected for removal cannot receive the new key or any future key updates. It may also specify policy for performing out-of-band, or "asynchronous," Rekey operations to immediately close security holes.

Rekey The KD must generate and supply new group keys to members when directed by the group manager, or periodically by policy. The Rekey action dictates how the new keys are delivered to group member nodes so that only trusted members receive the new key and blacklisted members cannot.

3.2.2 Key Management Protocol Overview. With respect to the specific problem domain characteristics, this research strives to find the best algorithm for key exchange in networks marked by generally high levels of connectedness and vary-

ing levels of participation by specific nodes. As such, the protocol presented herein employs an algorithm that adapts to these constraints while accomplishing its goals.

The central component of the proposed design is gossip-based physical key distribution. This behavior is combined with random peer sampling to achieve an application-layer model that meets the goals outlined above. The following sections describe the proposed algorithm's key management logic.

3.2.3 Key Management Core Routines.

3.2.3.1 Join (Requestor).

A node wishing to join a secure group contact the KD to authenticate its connection to the group and begin receiving group key updates. The following pseudocode describes the join process concisely:

1. Requestor establishes secure public key infrastructure (PKI) session with KD via three-way handshake
2. Requestor requests certificate to operate (CTO) from KD
3. If Requestor is on blacklist, KD denies Requestor's request and ends secure session; else, KD generates CTO for Requestor with the following information:
   (a) List of peers with whom Member shall connect
   (b) Current group key
   (c) Timestamp
4. KD signs CTO with its private key and delivers signed CTO to Requestor
5. For each peer provided via CTO, Requestor:
   (a) Establishes secure PKI session with peer
   (b) Presents CTO to peer
   (c) Maintains secure session to propagate future key updates

The above pseudocode assumes in step 5b that the CTO being provided is current and valid. However, this may not be the case, as a blacklisted member may attempt to rejoin a group with an old or forged CTO. To prevent an unauthorized connection from being established, active members must validate these connection requests via the following auxiliary logic:

1. Member accepts Requestor's secure session request and labels session "Request"
2. Member receives Requestor's CTO
3. Member validates CTO integrity and authenticity with KD's public key; if CTO is corrupt or not authentic, Member ends session
4. Member compares CTO timestamp to current time; if CTO is too old, Member ends session
5. Member approves connection and relabels session "Key Management"

The above logic assumes the following is true:

1. Nodes know KD's destination address and its public key prior to submitting a request
2. A data path exists between each member and the KD (although they do not necessarily maintain a secure unicast link)
3. All members and KD have unique public and private key pairs
4. Nodes are synchronized against a common time source (e.g., global positioning satellite). The accuracy, or "stratum" level, of the time source should be considered when deciding rekey frequency and skew tolerance. Whereas Network Time Protocol (NTP) may be sufficient in applications with daily rekeying, GPS is more suitable to hourly rekeys or when asynchronous rekeys are common.

After the Join operation is successfully executed, a Requestor becomes a Member until either Leave or Remove operations are carried out against the Member. For efficiency, the secure session used for CTO verification is preserved and relabeled for group key updates.

3.2.3.2 Rekey (KD). Either at specific time intervals, or due to a member's removal, the KD will generate a new group key and disseminate the new key to its peers for propagation to the entire group. The KD executes the following logic to govern this procedure. The gossip-style behavior is achieved when the members connected to the KD receive the Rekey message; the gossip logic is described in the Rekey (Member) routine in Section 3.2.3.3.

1. KD generates random new group key
2. KD constructs Rekey message with the following information:
   (a) New group key
   (b) Timestamp
   (c) Recently blacklisted members
3. KD signs Rekey message with KD's private key
4. KD appends number of total requested gossip rounds (unsigned) and delivers to each of its peers, except for blacklisted members

Note that the gossip rounds field is left unsigned so that Members can decrement this value without affecting the integrity of the Rekey message itself. This presents a minor vulnerability since a "rogue" member may increase this value instead of decrementing it (see Section 3.2.3.3). This vulnerability is mitigated by the signed timestamp; Rekey propagation ceases after the globally-specified threshold has elapsed, and gossip round numbers are rendered moot.

3.2.3.3 Rekey (Member). The heart of this protocol is each Member's responsibility to propagate Rekeys to other Members in the group. This stage introduces the gossip-based dissemination logic from which the protocol achieves its fault tolerant benefits. Members adhere to the logic below to propagate new group keys randomly to their neighbors:

1. Member receives Rekey message from peer (KD or other Member)

2. If Rekey message has already been received, Member does nothing; else,
3. Member authenticates Rekey message as follows:
   (a) Member verifies Rekey authenticity with KD's public key
   (b) Member checks Rekey timestamp; if timestamp predates current key's birthdate or Rekey threshold has elapsed, Member does nothing and exits; else:
4. Member applies new group key for later use in group communications
5. Member forwards the Rekey message to its peers
   (a) If gossip rounds value is 0, exit; else,
   (b) Decrement gossip rounds value by 1
   (c) Forward original Rekey message (with decremented round number) to random subset of peers (except blacklisted Members) based on current infectivity rate and round number

3.2.3.4 Remove (KD/Member). When a member is forcefully removed from the group, a rekey must be performed asynchronously so that the removed member cannot receive future group transmissions. This action is carried out jointly by the KD and the peers of the removed member. The KD executes the following:

1. KD receives direction to remove Member from group
2. KD directly notifies removed members' peers of upcoming removal via PKI
3. Each Member ends secure session with removed Member
4. KD performs asynchronous Rekey, adding removed members to blacklist

This process is remarkably simpler than existing solutions, which often require many "key encryption keys" to be generated, updated, and individually disseminated, such as in LKH [20]. Instead, this routine relies upon secure point-to-point sessions and a novel dissemination policy that prevents removed members from receiving the new group key.

3.2.3.5 Leave. A Member voluntarily departing a group may or may not have security implications. Therefore, it is within the group manager's discretion whether to perform a Rekey after a Member leaves. However, to correctly manage the key update overlay, the KD should be apprised of members leaving the group. Therefore, in implementations that do not require a Rekey, a departing Member timestamps, encrypts, signs, and transmits a single Leave message to the KD, and ends all peer sessions. If this is not done, future transmissions to the departed node are treated as missed messages.

3.2.4 Key Management Auxiliary Routines. Several of the above primitives depend upon auxiliary logic to ensure proper dissemination and reduce unnecessary communication between Members and the KD.

3.2.4.1 Await New Key / Neighbor Rekey Request (Member). The protocol employs periodic rekeys specifically to account for the probabilistic convergence of the gossip-based Rekey routine. That is, after each scheduled Rekey, a probabilistically small number of nodes will not receive the new key. To account for this discrepancy, each node enters an "active waiting" state at the scheduled Rekey intervals. If a new key does not arrive within a specified window following the scheduled Rekey, the node will request the new key from its peers, who probably received it. In the very rare instance that a node and none of its neighbors receive the current key, it can be estimated with high probability that a Rekey was in fact not performed. In this instance a node will wait a random period of time over a lengthy interval, then request a Rekey directly from the KD (and disseminate to its peers). The random wait prevents nodes from unwittingly overloading the KD, thereby exacerbating the impact of a missed Rekey. After receiving the new key, the left-out member executes the Rekey (Member) routine, omitting the propagation step.

For asynchronous rekeys, there is no way for missed nodes to know that they did not receive a new key without contacting peers. In these instances, a probabilistically small number of nodes may continue for some time (not exceeding the periodic Rekey

interval) without the current group key. However, providing a "pull" mechanism to account for this situation may cause more problems than it solves. Consider a single node that does not receive an asynchronous Rekey. If the left-out node receives a message encrypted with an unrecognized group key, it attempts to obtain the "new" group key from either its peers or the KD. If there was in fact an asynchronous Rekey, then the left-out node obtains the new group key with minimal impact to network traffic. However, it becomes too easy for a single malicious node to send out many messages encrypted with a false "new" key. The result is a saturated network, as all group members simultaneously rush to discover the unknown key. Therefore, if a message is received that has been encrypted with an unknown key, it is in the entire group's best interest for the left-out node to await the next periodic Rekey. Additionally, it may be in the KD's best interest to limit asynchronous Rekeys, at the cost of allowing blacklisted members to read encrypted group traffic until the next scheduled Rekey. This policy can be determined based on the group's specific security priorities.

3.2.4.2 Provide Missed Key (Member/KD). Following the above logic, missed synchronous Rekeys are followed by key requests. Each member must be able to respond to these requests by providing the signed Rekey message in its original state to the left-out member. To facilitate this feature, the KD and each member are responsible for keeping the last signed Rekey message it sent or received.

3.2.4.3 Island Detection (KD). After many joins and leaves, and during times of higher failure, it is possible that the peer network will form "islands," or subgroups that have no peer path to or from the KD. A simple method for rejoining these subgroups with the KD-reachable group is for the KD to deliver extra certificates to operate to nodes that request synchronous rekeys directly from the KD. Since after every synchronous rekey, the KD knows which nodes were reached and which were not, the KD can pair unreachable nodes with reachable nodes via the CTO. That is, the KD provides each unreachable node with the address of a node that was reached

during the last synchronous rekey. The KD is able to perform this operation with only an ACL and a single extra message, and need not retain any long-term special information about the topology to make this work.

3.2.4.4 Attack Prevention. This section summarizes the techniques employed by this protocol that prevent DoS and replay attacks.

Denial of Service. The protocol's time synchronization allows for certain actions to take place during predetermined time slots. For example, synchronous rekeys occur at specific times for only a short duration. During that period, a node that has not received the new key can request a retransmission from its neighboring nodes and the KD. However, such requests made outside this period will be ignored; this prevents a malicious node from permanently overloading the KD's queue and denying other members the KD's services.

Additionally, to prevent a bogus rekey message from being propagated throughout the network, the Rekey messages are authenticated with the KD's public key before being forwarded. Without this feature, a malicious node could create false rekey messages and "forward" them as though they were legitimate.

Replay Attacks. Time stamping and signing (in that order) all Rekey messages prevents replay attacks. Specifically, it prevents a malicious node from redelivering an old Rekey message and setting the group key to an old key that may have been compromised.

3.3 System Boundaries

The group key management problem aims to provide usable keys to authenticated users for the purpose of secure subgroup communications. For this reason, this system is limited in scope to the management and dissemination of keys, while the specific use of these keys is not studied. For instance, a system designer may opt to use a gossip-based group key management system but utilize a tree-based mul-

ticast algorithm or point-to-point messaging for the encrypted messages. Certainly, a key management subsystem lends itself to a wide variety of higher-level software implementations.

3.4 System Services

The actions outlined in Section 3.2.3 provide the major services of the proposed protocol. The Join, Leave, Remove, and Rekey actions offer users the necessary mechanisms to obtain and update group keys for the purpose of secure group communications.

3.5 Workload

For this study, the workload is a single Rekey message. The Rekey message originates at the KD and broadcasts outwards to the member nodes belonging to the same group. The message contains only the new key, and does not specify any blacklisted members.

3.6 Performance Metrics

1. Total messages sent. This is the sum of gossip "pushes," neighbor rekey requests, and requests sent to the KD that could not be answered via neighbor rekey requests.
2. KD-direct rekeys / Unreached healthy nodes. Since a specific goal is to minimize dependence upon central key authorities, the number of nodes that must be rekeyed by the KD are quantified separately. In an asynchronous rekeying environment, this number is equated to healthy nodes that are not reached by either the gossip push or pull routines.

See Section 3.8 for a further description of how these metrics are utilized for experimental design.

3.7 System Parameters

The following values are tested:

1. Peer links. This specifies the total number of link pairs used for key dissemination.
2. Infectivity. This is the percentage likelihood that a given node may spread a key update to a peer.
3. Gossip rounds. This is the total number of times that a given message instance will be passed along before it dies.
4. Topology. This research examines a randomized peer topology versus tree-based and "torus" topologies. A torus topology is a ring overlay that is engineered specifically for fault tolerance. See Figure 3.1 for a visual example.

3.8 Evaluation Technique

Java-based simulations are conducted for each experiment. Data for the averages of multiple runs are output to comma-delimited spreadsheets and analyzed with the JMP 8 statistical software package.

For the parameters listed in Section 3.7, the rekey mechanism is compared with the tree-based rekey mechanism favored by modern implementations.

3.9 Experiment Setup

Experiments are divided into five parts. Part 1 improves the parameters for synchronous rekeys with neighbor rekey requests enabled. Part 2 improves the parameters for asynchronous rekeys with island detection enabled. Part 3 compares improved asynchronous rekeys with the proposed protocol versus a tree-based dissemination routine that does not utilize any of the failure recovery methods described in this research. Part 4 tests whether the improved parameters for asynchronous rekeys can benefit from a managed overlay (specifically, a torus) over a random overlay. Part 5 tests scalability to large groups by comparing performance metrics of a

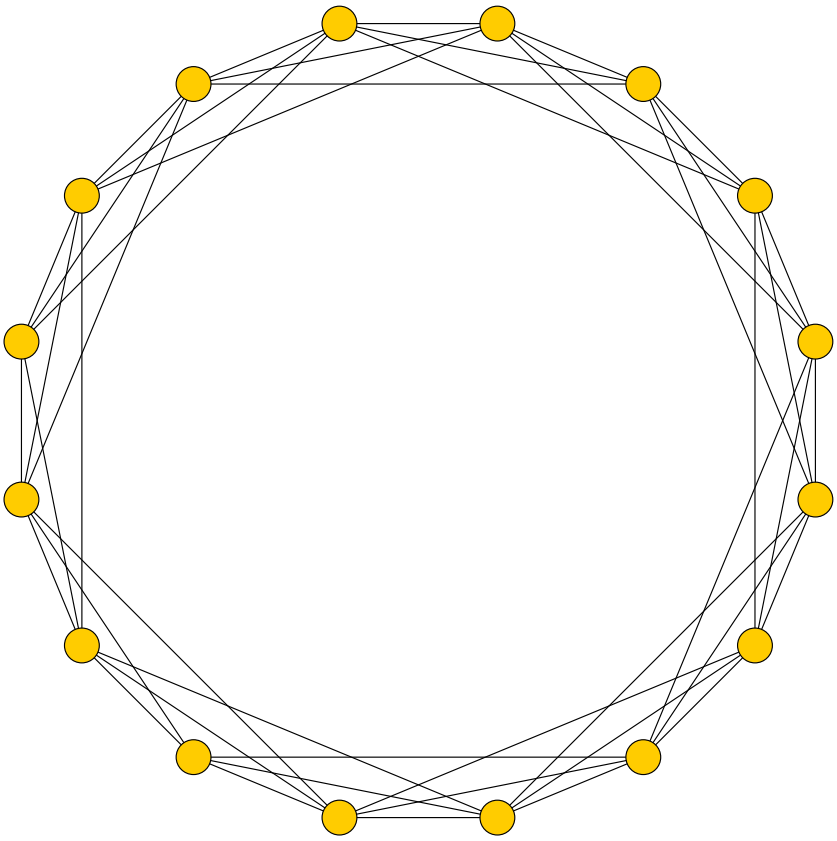

Figure 3.1: A torus overlay provides fault tolerance with many redundant links

1000-node group to those of a 100-node group. For each set of parameters, the simulation generates 2000 random graphs and simulates a single rekey event per graph, counting the messages as they are passed. After the rekey has completed, the simulation counts how many healthy nodes received the rekey message. For synchronous rekeys, the simulation counts how many nodes were forced to contact the KD directly after a round of neighbor rekey requests. In this way the simulation quickly inspects a "moment in time" on an arbitrary group that has already seen several members join and leave.

To improve the synchronous rekey parameters, a simple genetic algorithm is employed to minimize the following weighted sum fitness function:

$$fitness = averagetotalmessages + pmult * averageKDrequests \tag{3.1}$$

where *pmult* is a penalty multiplier, described further in the next paragraph. In this way, the algorithm rewards a low number of overall messages sent for a synchronous rekey and penalizes the expensive requests made directly to the KD. This achieves the goal of primarily maximizing efficiency and secondarily minimizing expensive fault situations for synchronous rekeys.

To improve the asynchronous rekey parameters, a similar genetic algorithm is employed to minimize the following weighted sum fitness function:

$$fitness = averagemissednodes + pmult * averagetotalmessages \tag{3.2}$$

where *averagemissednodes* corresponds to the number of nodes that are not reached during a single rekey push. This time, the algorithm rewards high levels of first-push convergence and penalizes situations that require many messages.

Note the importance of the penalty multiplier. Without it, the two fitness functions would be identical. In fact, over-penalizing KD-direct rekeys during a synchronous rekey would result in fitness values that resembled an asynchronous rekey. More concisely, *pmult* is set to indicate the level of importance of efficiency during synchronous rekeys, and the importance of reliability during asynchronous rekeys. This may vary between applications and administrator preference, but should not be too high in either case. A *pmult* of 3 is used for synchronous rekeys during experiments, indicating that a KD-direct rekey has the perceived negative effect on efficiency of sending 3 extra messages. For asynchronous rekeys, a *pmult* of 0.01 is used. In this way, the benefit of a small number of missed nodes can only be offset by a very large number of extra messages. Thus, the *pmult* value is the *weighted* aspect of each weighted sum function.

Each round of the genetic algorithm selects a set of parameters from the superior solution pool and sets between 1 and all of the parameters to a random new value in the mutating parameter's range. To maintain solution diversity, the probability that 1, 2, and 3 parameters will mutate in a given round are set to 60%, 25%, and 15%, respectively. Additionally, no unique set of parameters may to enter the superior solution set more than once.

For each scenario, $250,000$ trials are performed for failure rates ranging from 5% to 40% in 5% increments, maintaining a pool of 40 of the most superior parameter sets according to fitness value.

Once the improved parameters are achieved for each failure rate, the algorithm is then compared in simulations against unimproved tree-based solutions and improved torus-based solutions for the entire range of failure rates between 5% and 40% (36 failure rates in total). This exercise determines a) whether a random overlay topology

with improved parameters can perform with greater fault tolerance than existing tree-based routines (no improved parameters, no fault recovery heuristics) and b) whether heavily managing a fault tolerant overlay with improved parameters has any benefit.

To study scalability, improved parameters from the previous experiments are applied to a graph of 1000 nodes and performance is evaluated under 5–40% failure rates based upon the metrics described in section 3.6. The researcher hypothesizes that the total messages required to rekey the group will remain in direct proportion to the total number of nodes for a given failure rate; note that this increased burden is distributed among the member nodes rather than falling entirely on the KD as it would in other protocols.

3.9.1 Input Graphs. With the exception of the scalability experiment, the input graph for each trial is of size 100 nodes. The graph is generated randomly to model a "slice in time" that represents a given network that has encountered several member joins and leaves. By design, this may lead to disjoint graphs in situations where there are few links, thus the benefit of island detection and repair feature. For each parameter scenario, a new input graph is generated per run to ensure that performance metrics are measured as an average of many runs. The major benefit of this criteria is the practicality of measuring the protocol against a network that changes over time; the major disadvantage is the processing cost of generating many simulated networks.

3.10 Summary

A framework is provided for group key exchange that affords high levels of fault tolerance while keeping messaging load to a minimum. A method is provided to improve these parameters for varying levels of intermittent node failure among a given group, and experiments are presented to illustrate their value versus existing key management techniques. Experimental results are discussed in Chapter 4.

IV. Results

This chapter summarizes the results of the experiments outlined in Chapter 3.

4.1 Synchronous Rekey Parameter Improvement

After 250,000 generations, the optimal parameters for the synchronous rekey trials converge. For all failure rates, the algorithm favored the lowest possible number of links, high infectivity, and an average of 6.5 gossip rounds (+/- .2, 95% CI). Notably these parameters closely resemble parameters of a binary tree ($log_2 100 \approx 6.643$). That is, for synchronous rekeys, an algorithm that heavily favors efficiency will naturally suggest flooding via binary trees or a similar overlay. The analysis of fault tolerance follows: the expense of some member nodes having to contact the key distributor directly for a rekey is absorbed by the overwhelming efficiency of a tree-based algorithm. Note that for security concerns, this analysis applies only to synchronous rekeys, in which members have the ability to request rekey messages from other nodes and the KD.

4.2 Asynchronous Rekey Parameter Improvement

The improved parameters for asynchronous rekeys also converge, but on a much different set of values. In this scenario, the best solutions are marked by maximum infectivity levels ($95 - 100\%$), arbitrarily many gossip rounds, and a relatively low number of links that rose with respect to node failure rate. Indeed, a graph could be drawn to model the relationship between a given failure rate and the number of peer links required to minimize the fitness function.

To illuminate this relationship, extra simulations were built to measure the efficiency of each number of peer links over the entire parameter range while keeping infectivity set to 100% and gossip rounds set to infinity (i.e., a message flooding scenario). For each of the 36 failure rates from 5% to 40%, a regression is applied to the fitness values, and the number of links corresponding to the minimum response is computed with statistical software. The improved links values are then graphed

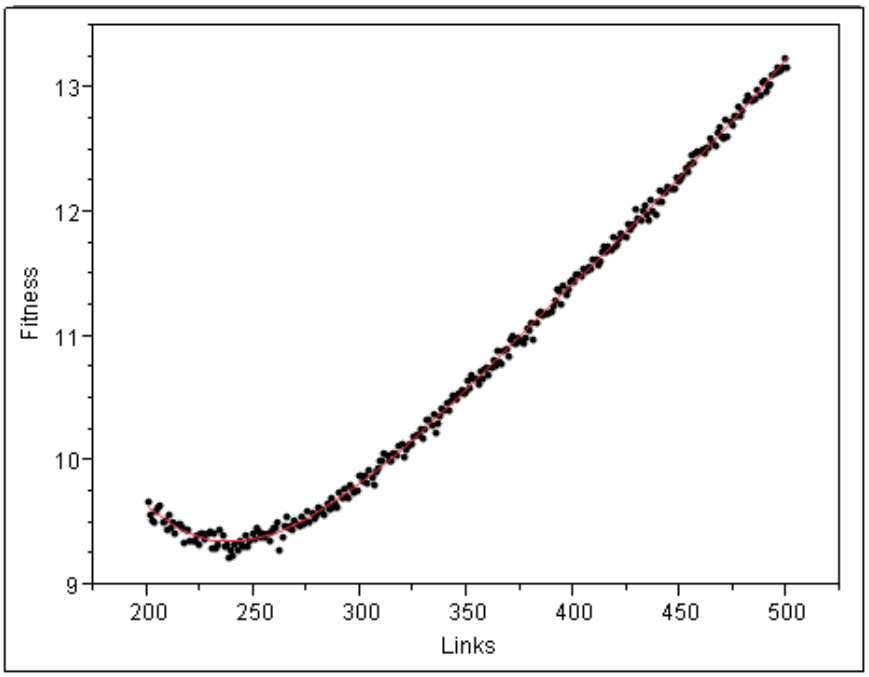

Figure 4.1: Fitness values by links, 5 percent failure rate

against their respective failure rates, and a final regression is applied to the improved links values. All regressions are found to be quartic with R-squared values not less than .96 and a 95% confidence interval. The graph in figure 4.3 is a useful tool for the KD's topology control function; for an observed node failure rate, it identifies the ideal number of links per node that a KD should assign to new nodes as they join. Regressions by failure rate are shown in figures 4.1 and 4.2. More regressions can be found at Appendix A.

4.3 Improved Asynchronous Rekeys with Heuristics Compared to Tree-based Dissemination (No Heuristics)

This experiment shows that a network that has performed simple island detection and correction will exhibit greater fault tolerance than a tree-based dissemination routine with no such heuristics during asynchronous rekeys, and for a reasonable cost.

Figure 4.4 shows the average number of nodes per healthy nodes that will miss an asynchronous rekey "push" for both a random overlay network and a tree-based

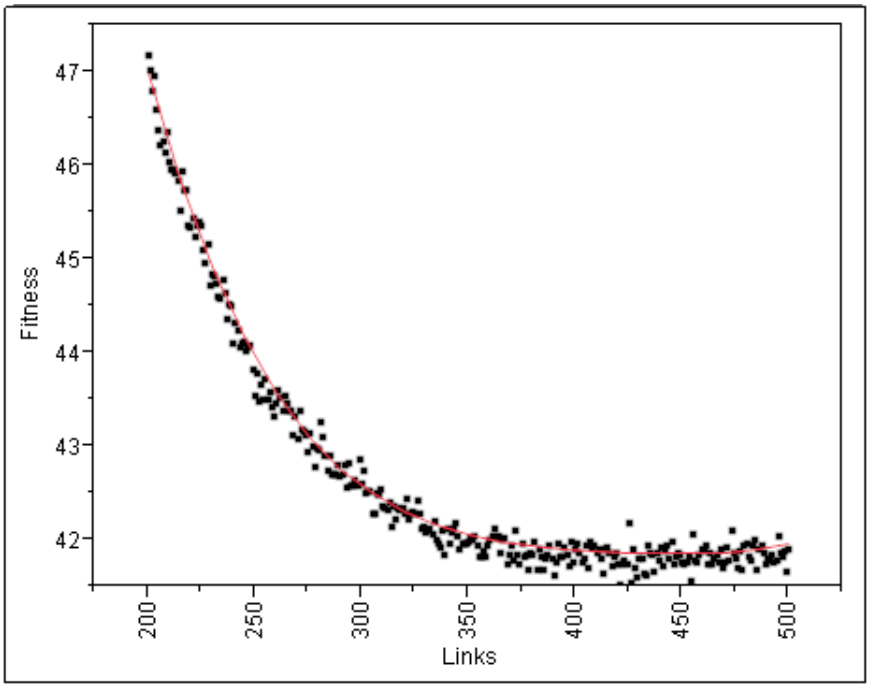

Figure 4.2: Fitness values by links, 40 percent failure rate

dissemination network. Even during periods of high failure rates, the proposed algorithm maintains a high level of convergence while the tree-based dissemination routine fails linearly with respect to node failure rate.

4.4 Random Overlay Compared to Managed "Torus" Overlay on Improved Asynchronous Rekeys

This experiment compares randomized overlay topology against a managed "torus" topology described in Chapter 3 with the goal of measuring the performance benefits of a managed overlay. Experimentation reveals that there are no tangible performance benefits to managing an overlay to such a degree.

Each topology is tested with the improved parameters from Section 4.2 for multiple failure rates and produce the graph at figure 4.5. For the parameters specified in Section 4.2, there is no significant performance increase for utilizing a managed overlay instead of a random overlay. In fact, the extra CTO messages required to

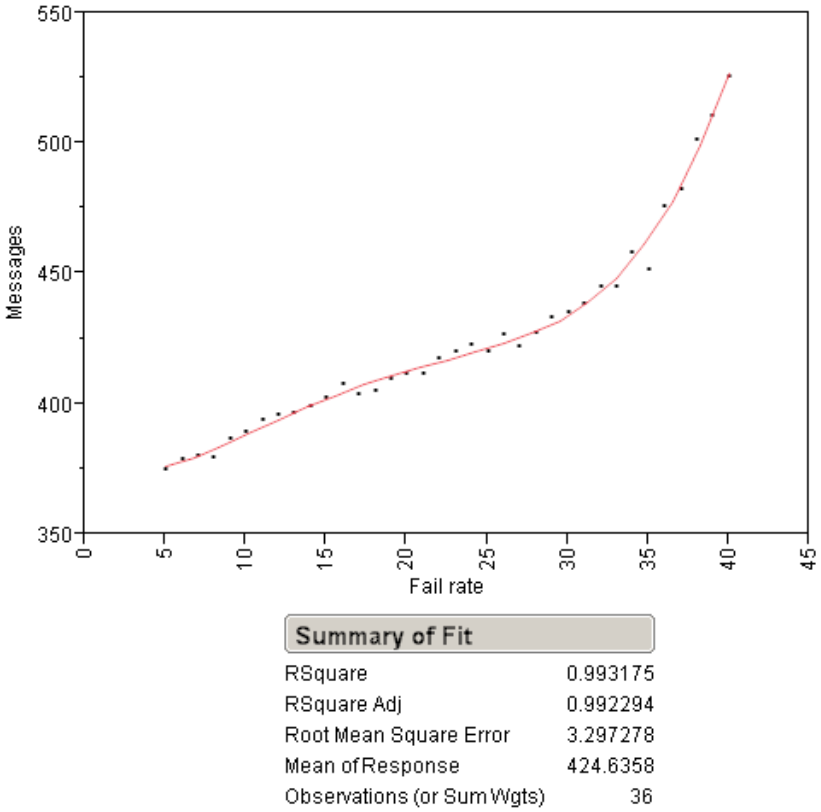

Figure 4.3: Improved links parameter by node failure rate

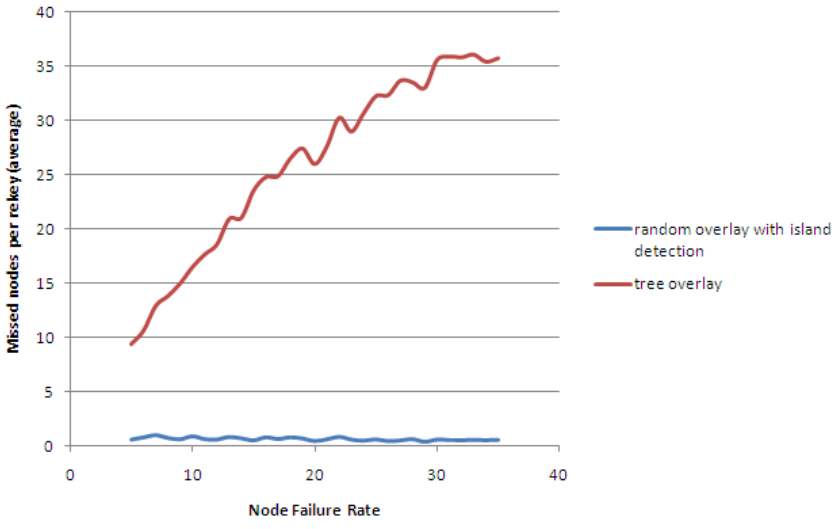

Figure 4.4: Random overlay with island detection versus tree-based dissemination without island detection

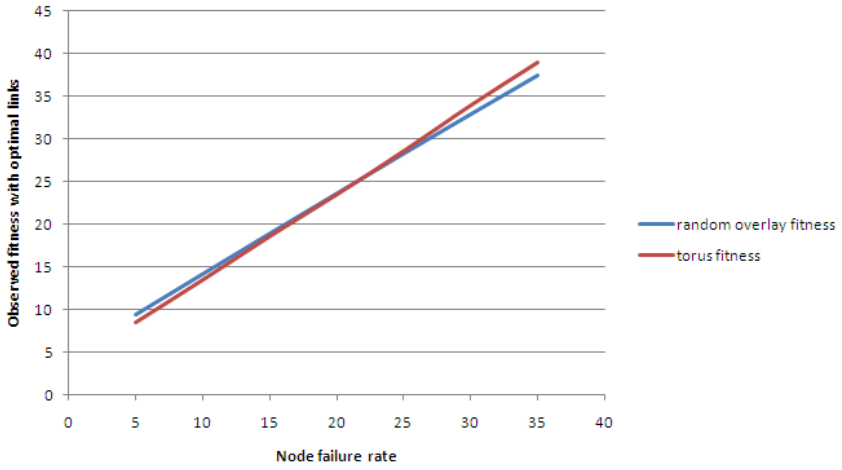

Figure 4.5: Improved parameters, random overlay versus torus overlay

Table 4.1: Improved links value for 100-node and 1000-node groups, by failure rate

	100	1000
5	237.86	2479.59
20	287.71	2765.91
35	363.19	3243.86

Table 4.2: Messages for 100-node and 1000-node groups with improved parameters, by failure rate

	100	1000
5	375.15	3842.7
20	411.52	3863.96
35	451.41	3947.86

bring a degraded torus topology back to a managed state make the torus topology less efficient than randomization.

4.5 Scalability to 1000-node Groups

The results of the trials against the 1000-node input graphs indicate that the proposed protocol is indeed scalable to very large groups. Quartic regressions are shown in Figures 4.6, 4.7, and 4.8 that correspond to 5%, 20%, and 35% failure rates, respectively. Minimizing the fitness value for these regressions, ideal links and messages values are calculated and shown in Tables 4.1 and 4.2. It is interesting to note that as the failure rate increases, the total messages per node is actually lower for the 1000-node group.

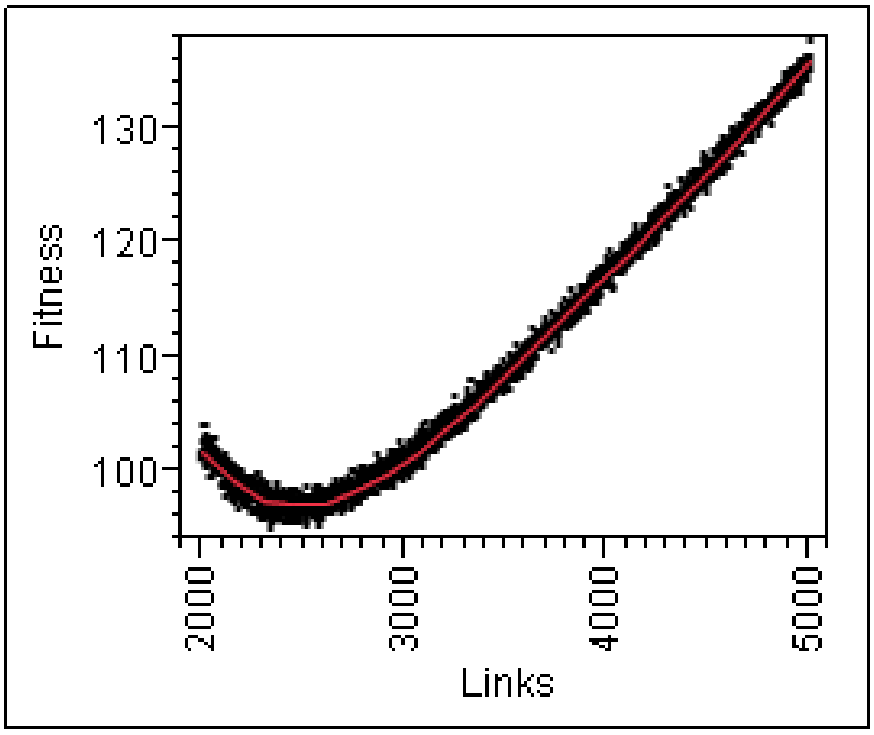

Figure 4.6: Fitness values by links, 1000 nodes, 5 percent failure rate

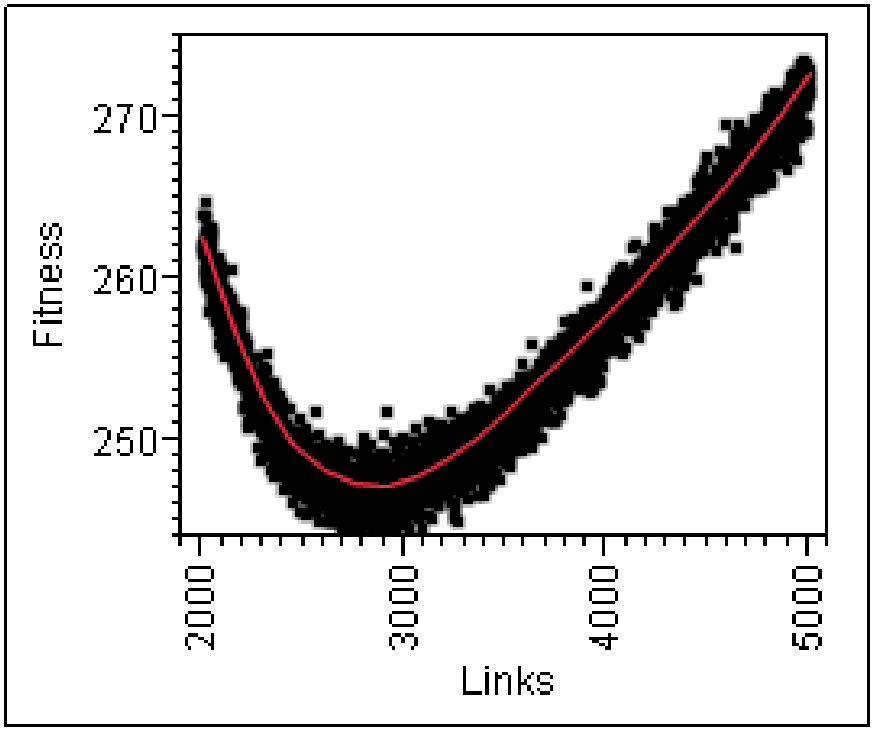

Figure 4.7: Fitness values by links, 1000 nodes, 20 percent failure rate

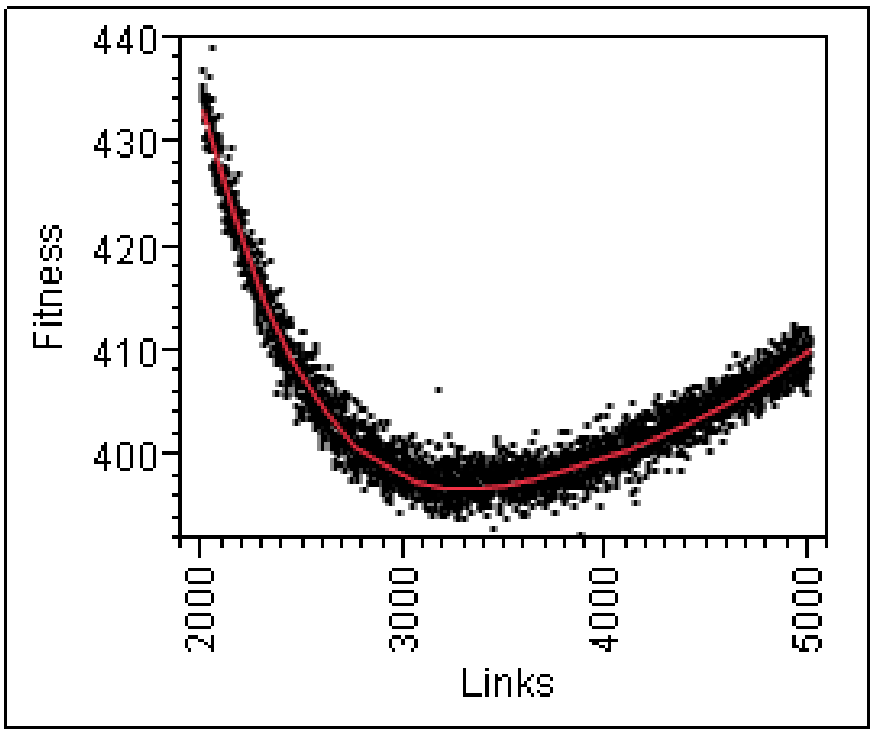

Figure 4.8: Fitness values by links, 1000 nodes, 35 percent failure rate

V. Conclusions

5.1 Significance and Interpretation of Analysis

The results of the five experiments indicate that the proposed protocol succeeds in providing a lightweight, scalable, and fault-tolerant solution to the group key management problem. Parameter improvement provides a simple way of calculating more efficient parameters for a given group based upon 1) the number of nodes in the group and 2) the expected or calculated node failure rate in the group. Utilizing these parameters and simple heuristics to detect disjoint subgroups, the protocol achieves a level of fault tolerance not attainable in current tree-based solutions. Additionally, experiments demonstrate that random peer overlays suffer no noticeable performance disadvantage when compared to highly managed overlays.

The results of the analysis in Chapter 4 indicate the different objectives for synchronous and asynchronous rekeys produce very different topologies with different parameters. It is found that with the neighbor rekey request feature, synchronous rekeys favor tree-like topologies. Conversely, asynchronous rekeys aim to deliver keys with more links relative to the node failure rate on the network. The resultant topology is a sparse mesh whose degree grows as the network's failure rate increases over time.

Unfortunately a group cannot utilize both topologies at once, and the administrator must select one that is appropriate to the security requirements of the network. For instance, if a group can tolerate a small number of nodes not having the group key for the period of time between synchronous rekeys, he may opt to establish a tree-based dissemination with neighbor rekey request capabilities to account for link failures (versus node failures). On the other hand, if it is crucial that asynchronous rekeys reach as many nodes as possible, the administrator might opt for the topology with the number of links recommended via the solution in section 4.2. In either case, the combination of heuristics provides an administrator with several advantages over current physical dissemination schemes for a low overhead cost.

5.2 Open Research

This research effort has practical application in any many wide-area distributed computing environments, namely electrical power systems monitoring and control and mobile ad hoc networks. The proposed key management protocol provides drop-in security for distributed systems that lack defined roles and hierarchy or whose topology changes over time.

Current efforts to monitor and control the electrical power grid such as NASPI will benefit greatly from power system simulations that incorporate security mechanisms. Incorporating the proposed key management protocol into an electrical power and network simulator (e.g., EPOCHS) can provide valuable information to Smart Grid development teams. As the size of the NASPI network grows to many thousands of sensors, the need for a simple, fault-tolerant security solution will be absolutely critical.

Indeed, mobile ad hoc platforms are in greater need of a distributed security solution than ever before. Unmanned aerial vehicles, which provide highly sensitive, often classified video feeds to military commanders and intelligence personnel, have been the subject of scrutiny since the discovery that these feeds are sent almost exclusively unencrypted. As these networks grow in size and become more autonomous, the need becomes imminent for an encryption solution. The proposed key management protocol can provide an expedient solution for this circumstance, but there is more research to be done before it can be deployed to a MANET environment. Further study must be done in the area of peer selection and the viability of direct communication with a key distributor. Specifically, a sub-protocol for peer selection must consider physical and spatial limitations when building the network, as it cannot reasonably rely upon random generation. Additionally, direct communications with a trusted key distributor may be very costly or impractical due to transmission capabilities and land features. The plethora of prior research into physical MANET modeling and peer selection suggest that an immediate investment in this field is highly feasible.

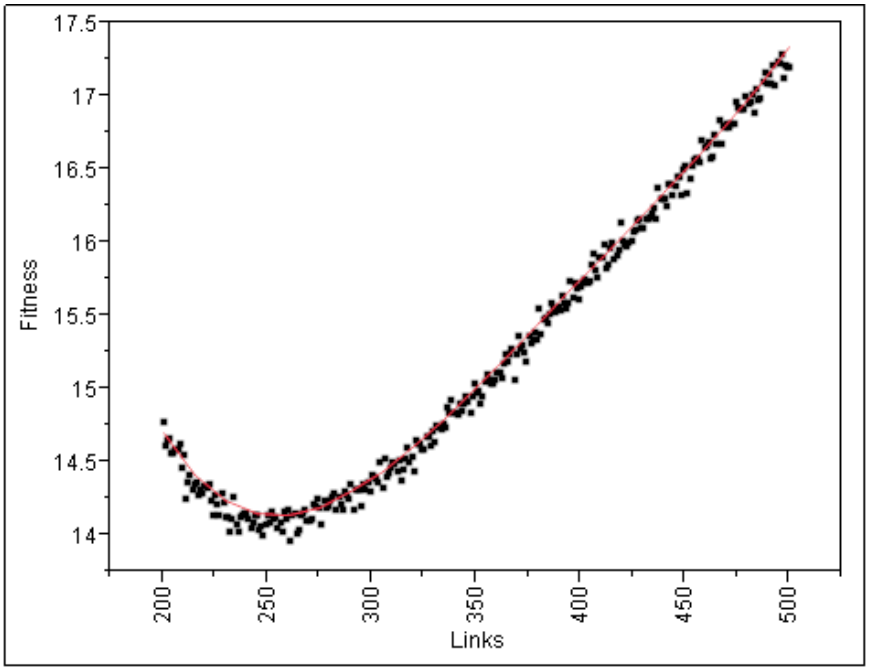

Figure A.1: Fitness values by links, 10 percent failure rate

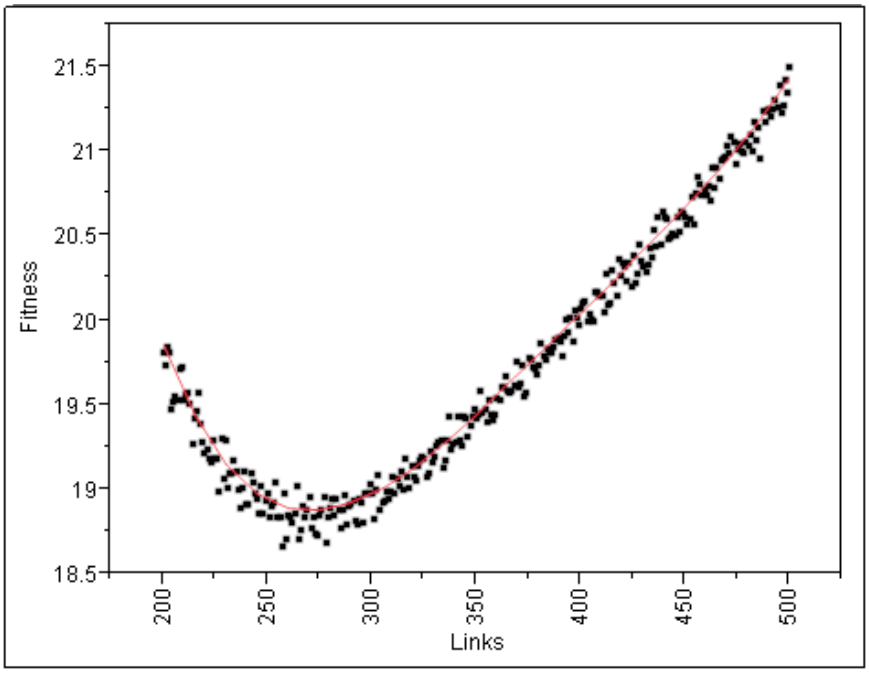

Figure A.2: Fitness values by links, 15 percent failure rate

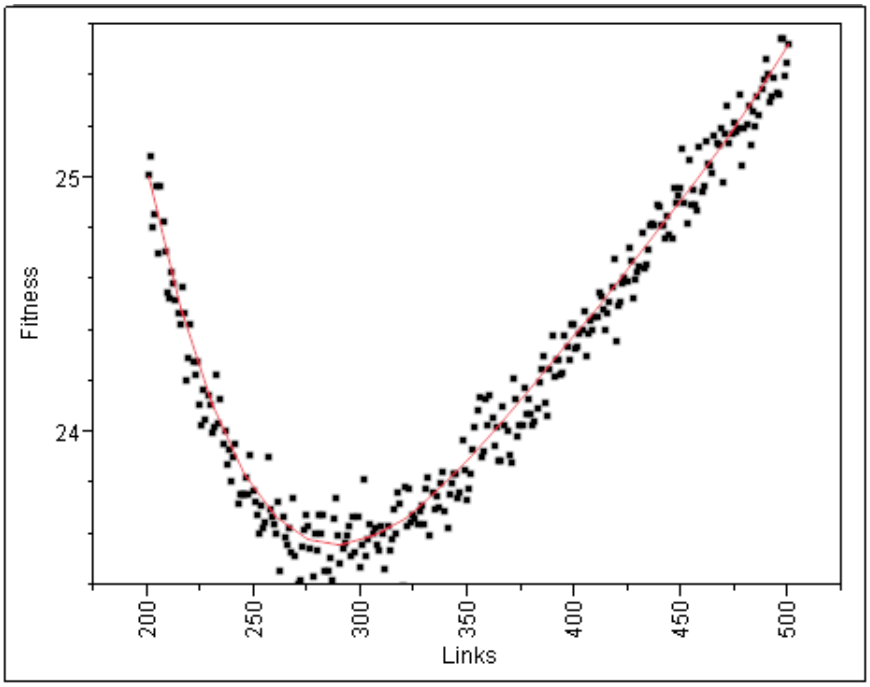

Figure A.3: Fitness values by links, 20 percent failure rate

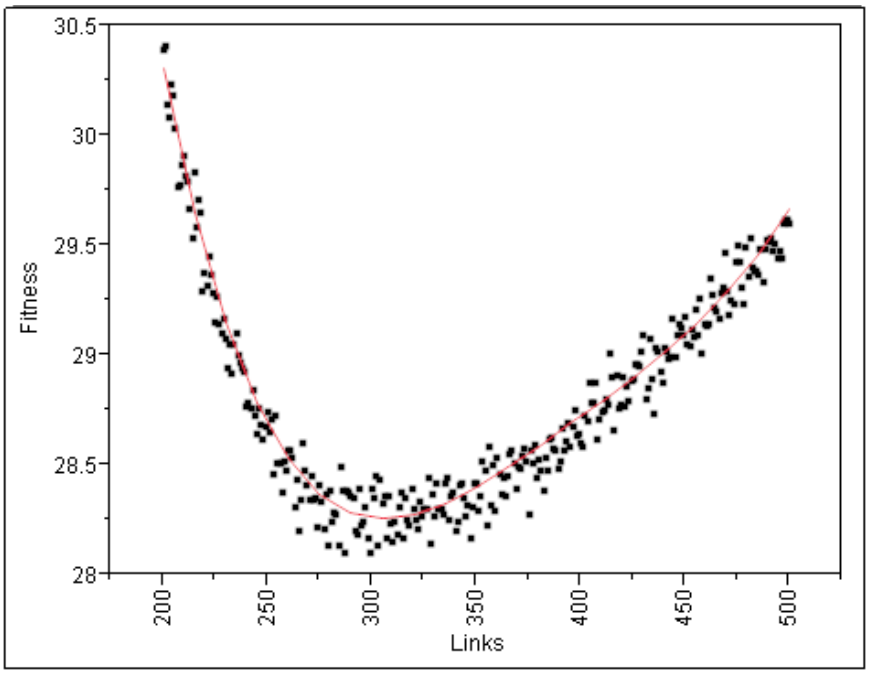

Figure A.4: Fitness values by links, 25 percent failure rate

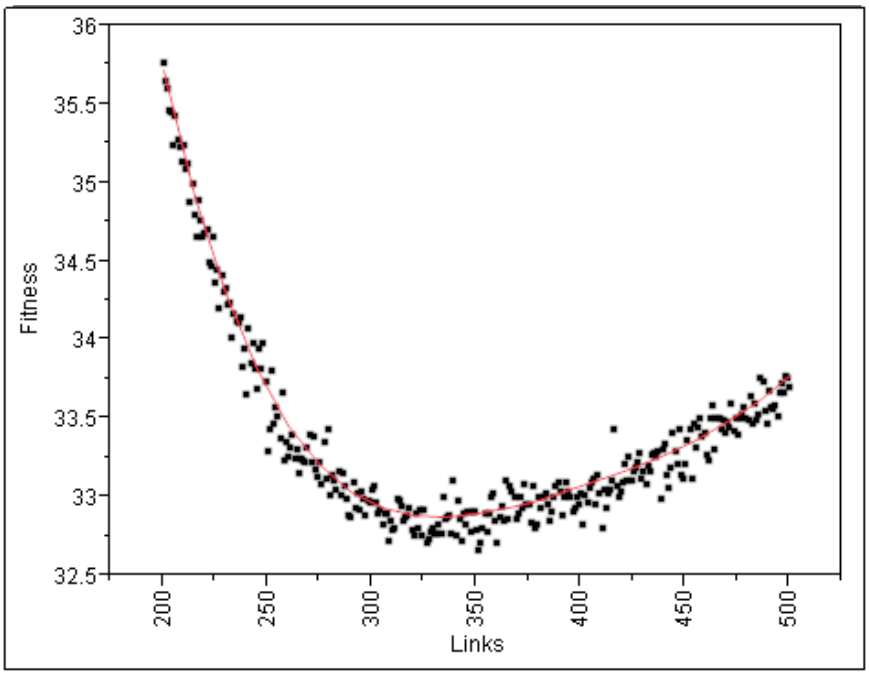

Figure A.5: Fitness values by links, 30 percent failure rate

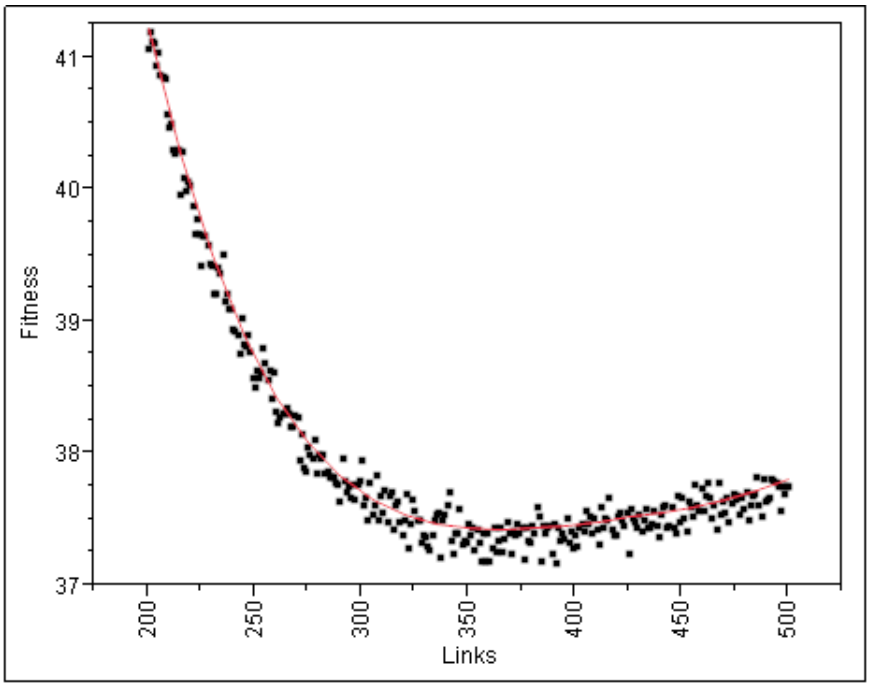

Figure A.6: Fitness values by links, 35 percent failure rate

Bibliography

1. Ballardie, A. "Scalable Multicast Key Distribution". RFC 1949 (Experimental), May 1996. URL `http://www.ietf.org/rfc/rfc1949.txt`.

2. Birman, Kenneth P., Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. "Bimodal multicast". *ACM Trans. Comput. Syst.*, 17(2):41–88, 1999. ISSN 0734-2071.

3. Demers, Alan, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. "Epidemic algorithms for replicated database maintenance". *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, 1–12. ACM, New York, NY, USA, 1987. ISBN 0-89791-239-4.

4. Diffie, Whitfield and Martin E. Hellman. "New Directions in Cryptography". *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976. URL `citeseer.ist.psu.edu/diffie76new.html`.

5. Dunigan, Thomas H. "Group Key Management". Oak Ridge National Laboratories, June 2001. URL `http://www.csm.ornl.gov/~dunigan/gkm.html`.

6. Graham, Daniel F. *On-demand Key Distribution for Mobile Ad-hoc Networks*. Master's thesis, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, March 2007. AFIT/GCS/ENG/07-12.

7. Haas, Zygmunt J., Joseph Y. Halpern, and Li Li. "Gossip-based ad hoc routing". *IEEE/ACM Trans. Netw.*, 14(3):479–491, 2006. ISSN 1063-6692.

8. Harney, H., U. Meth, A. Colegrove, and G. Gross. "GSAKMP: Group Secure Association Key Management Protocol". RFC 4535 (Proposed Standard), June 2006. URL `http://www.ietf.org/rfc/rfc4535.txt`.

9. Harney, H. and C. Muckenhirn. "Group Key Management Protocol (GKMP) Architecture". RFC 2094 (Experimental), July 1997. URL `http://www.ietf.org/rfc/rfc2094.txt`.

10. Harney, H. and C. Muckenhirn. "Group Key Management Protocol (GKMP) Specification". RFC 2093 (Experimental), July 1997. URL `http://www.ietf.org/rfc/rfc2093.txt`.

11. Hopkinson, Kenneth, Kate Jenkins, Kenneth Birman, James Thorp, Gregory Toussaint, and Manu Parashar. "Adaptive Gravitational Gossip: A Gossip-Based Communication Protocol with User-Selectable Rates". *IEEE Transactions on Parallel and Distributed Systems*, 20(12):1830–1843, 2009. ISSN 1045-9219.

12. Hubenko, V.P., R.A. Raines, R.O. Baldwin, B.E. Mullins, R.F. Mills, and M.R. Grimaila. "Improving Satellite Multicast Security Scalability by Reducing Rekeying Requirements". *Network, IEEE*, 21(4):51–56, July-August 2007.

13. Jelasity, Márk, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. "Gossip-based peer sampling". *ACM Trans. Comput. Syst.*, 25(3):8, 2007. ISSN 0734-2071.

14. Jenkins, Kate, Ken Hopkinson, and Ken Birman. "A Gossip Protocol for Subgroup Multicast". *Distributed Computing Systems Workshops, International Conference on*, 0:0025, 2001.

15. Liu, Yilu, Lamine Mili, Jaime De La Ree, Reynaldo Francisco Nuqui, and Reynaldo Francisco Nuqui. "State Estimation and Voltage Security Monitoring Using Synchronized Phasor Measurements", 2001.

16. Mittra, Suvo. "Iolus: A Framework for Scalable Secure Multicasting". *SIGCOMM*, 277–288. 1997. URL `citeseer.ist.psu.edu/mittra97iolus.html`.

17. Rivest, R. L., A. Shamir, and L. Adleman. "A method for obtaining digital signatures and public-key cryptosystems". *Commun. ACM*, 21(2):120–126, 1978. ISSN 0001-0782.

18. Stamp, Mark. *Information Security: Principles and Practice*. John Wiley & Sons, 2005. ISBN 0471738484.

19. Wallner, D., E. Harder, and R. Agee. "Key Management for Multicast: Issues and Architectures". RFC 2627 (Informational), June 1999. URL `http://www.ietf.org/rfc/rfc2627.txt`.

20. Wong, Chung Kei, Mohamed Gouda, Wong Mohamed, Gouda Simon, and Simon S. Lam. *Secure Group Communications Using Key Graphs*. Technical report, 1998.

21. Yao, Yan, Weiming Fu, and Shiyong Zhang. "Gossip-based Multicast Loss Recovery Mechanisms in Group Key Distribution". *Computer and Information Technology, 2005. CIT 2005. The Fifth International Conference on*, 429–434. Sept. 2005.

Index

The index is conceptual and does not designate every occurrence of a keyword. Page numbers in bold represent concept definition or introduction.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
03–15–2010	Master's Thesis	Sept 2008 — Mar 2010

4. TITLE AND SUBTITLE

Scalable and Fault Tolerant Group Key Management

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Lupien, Nicholas A., Capt, USAF

5d. PROJECT NUMBER

10-173

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/10-05

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
Attn: Dr. John Matyjas
525 Electronics Parkway
Rome, NY 13441
John.Matyjas@rl.af.mil; DSN 587-4255

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RIGE

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

To address the group key management problem for modern networks this research proposes a lightweight group key management protocol with a gossip-based dissemination routine. Experiments show that by slightly increasing workload for the key update mechanism, this protocol is superior to currently available tree-based protocols with respect to reliability and fault tolerance, while remaining scalable to large groups. In addition, it eliminates the need for logical key hierarchy while preserving an overall reduction in rekey messages to rekey a group. The protocol provides a simple "pull" mechanism to ensure perfect rekeys in spite of the primary rekey mechanism's probabilistic guarantees, without burdening key distribution facilities. Benefits of this protocol are quantified versus tree-based dissemination in Java simulations on networks exhibiting various node failure rates.

15. SUBJECT TERMS

group key management, secure group communication, epidemic algorithms

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Kenneth M. Hopkinson
U	U	U	UU	60	19b. TELEPHONE NUMBER *(include area code)* (937) 255–3636, ext 4579; kenneth.hopkinson afit.edu