DISTRIBUTED FRIEND-TO-FRIEND FRAMEWORK AND SERVICES USING SOCIAL NETWORKS

by

Matthew Jared Probst

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

School of Computing

The University of Utah

December 2012

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Matthew Jared Probst**
has been approved by the following supervisory committee members:

Sneha Kumar Kasera	, Chair	**7-November-2012** Date Approved
Alan Davis	, Member	**7-November-2012** Date Approved
John Regehr	, Member	**7-November-2012** Date Approved
Matthew Might	, Member	**7-November-2012** Date Approved
Evan L. Ivie	, Member	**7-November-2012** Date Approved

and by **Alan Davis**, Chair/Director of

the Department of **School of Computing**

and by Charles A. Wight, Dean of the Graduate School.

ABSTRACT

We develop a novel framework for friend-to-friend (f2f) distributed services (F3DS) by which applications can easily offer peer-to-peer (p2p) services among social peers with resource sharing governed by approximated levels of social altruism. Our framework differs significantly from typical p2p collaboration in that it provides a foundation for distributed applications to cooperate based on pre-existing trust and altruism among social peers. With the goal of facilitating the approximation of relative levels of altruism among social peers within F3DS, we introduce a new metric: SocialDistance. SocialDistance is a synthetic metric that combines direct levels of altruism between peers with an altruism decay for each hop to approximate indirect levels of altruism. The resulting multihop altruism levels are used by F3DS applications to proportion and prioritize the sharing of resources with other social peers. We use SocialDistance to implement a novel flash file/patch distribution method, SocialSwarm. SocialSwarm uses the SocialDistance metric as part of its resource allocation to overcome the necessity of (and inefficiency created by) resource bartering among friends participating in a BitTorrent swarm. We find that SocialSwarm achieves an average file download time reduction of 25% to 35% in comparison with standard BitTorrent under a variety of configurations and conditions, including file sizes, maximum SocialDistance, as well as leech and seed counts. The most socially connected peers yield up to a 47% decrease in download completion time in comparison with average nonsocial BitTorrent swarms.

We also use the F3DS framework to implement novel malware detection application—F3DS Antivirus (F3AV)—and evaluate it on the Amazon cloud. We show that with f2f sharing of resources, F3AV achieves a 65% increase in the detection rate of 0- to 1-day-old malware among social peers as compared to the average of individual scanners. Furthermore, we show that F3AV provides the greatest diversity of malware scanners (and thus malware protection) to social hubs—those nodes that are positioned to provide strategic defense against socially aware malware.

To Keri, my best friend and companion.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First, I am grateful for the support, guidance, patience, and constructive critique of my adviser, Sneha Kumar Kasera. Rather than pushing me toward specific topics that he was most interested in, he supported and helped me pursue my own research challenges and interests.

Second, I would like to thank my advisory committee for their support and guidance. While I was working on my undergraduate degree, Dr. Evan Ivie was my mentor. His love for computer systems and networks as well as his example of humility and selfless service were some of the key reasons why I chose to pursue a graduate degree.

Third, I am grateful for the example set by my father, Dr. Reed R. Probst, in completing a PhD, as well as for the substantial proof reading and editing assistance he provided on my publications and dissertation. I would also like to thank Sunny Stimmler for her editing help as well.

Finally, I would like to thank Dr. Jun Cheol Park for his collaboration, advice, and feedback during much of this research.

CHAPTER 1

INTRODUCTION

Over the past several years, there has been an explosion in the popularity and utility of online social networks (OSNs) [1] [2]. OSNs have become the dominant method for sharing photos [3], playing online games [4], finding employment [5], and finding and communicating with individuals having similar hobbies and interests [6]. Traditional out-of-band communication methods—such as phone calls, traditional mailed letters, and face-to-face conversations—are in many cases being replaced [7] [8] by messaging over online social networks. Even electronic messaging and content sharing that previously may have occurred directly in peer-to-peer (p2p) relationships also are increasingly being replaced [9] by messaging and content sharing over OSNs given the functionality these networks provide to interact and collaborate with a group of social peers.

Although typical OSNs provide useful methods for peers to collaborate, the centralized, proprietary, and commercial nature of those networks prevents the direct communication and sharing of resources that have been shown to be valuable in p2p applications. Ideally, applications would be able to exploit both direct p2p collaboration (for efficiency and scalability) and social awareness (for connections to real-world social relationships) concurrently. Developing p2p applications that are socially aware is, however, not trivial because of several challenges.

1.1 Challenges

In this section, we describe three key challenges in developing p2p applications that are socially aware.

1.1.1 Challenge 1: Extracting and Quantifying Trust

The complete metadata on relationships among OSN participants is typically held tightly [10] by the commercial entity running the network. Usually, there are no published APIs for easily extracting metadata from multiple OSNs to create independent applications. When openly published metadata on social relationships does exist, it is typically both limited (direct peer only) and inaccurate—relying only on simplistic user tagging of social relationships, which is known to poorly represent real social interaction and trust. Applications that wish to exploit existing trust among social peers must find and quantify better indicators of both direct and indirect (multihop) trust.

1.1.2 Challenge 2: Resource Allocation Strategy

One valuable use case for p2p applications is that of direct resource (such as bandwidth, CPU, memory, storage) sharing. The first generation of peer-to-peer (p2p) applications assumed full trust between even anonymous users. Such systems [11, 12] typically failed due to problems with free-riders (leeches) as well as the injection of malicious or inaccurate content. Second-generation p2p systems such as BitTorrent [13] were developed to protect resources from abuse by forcing resource trading. These resource protection mechanisms [14] create necessary inefficiencies when interfacing with completely untrusted peers. Thus, without effective peer selection and resource allocation, the probability that resources will be wasted increases (either lost to free-riding nodes or maliciously consumed, such as in a DOS attack). The strategy used for allocation of resources can also have utility within security related applications. Researchers studying containment of worms on cellular networks have shown [15] that ideal containment can be achieved when the most social nodes (the social-hubs) are patched first. Unlike cellular networks, however, true p2p networks do not have the benefit of either central knowledge of connectivity, centralized monitoring, or centralized management of protection mechanisms. Identifying methods to allocate protection resources to social-hubs in fully distributed systems—those nodes that are positioned to provide strategic defense against socially-aware malware—is a critical challenge. Decisions for the deployment of protection mechanisms in a fully distributed network are much more challenging than in networks that have centralized monitoring and management.

1.1.3 Challenge 3: Decentralized Messaging

The proprietary APIs of OSNs facilitate communication among peers only via the OSN's servers. OSNs commonly employ a traditional client-server (or client-cloud) architecture. The commercial entity behind an OSN handles all message passing, service arbitration, and request queuing among users and applications in that particular "ecosystem."

1.2 A Framework to Solve These Challenges

In this dissertation, we present a novel framework for friend-to-friend (f2f) distributed services (F3DS) to overcome these challenges for developers of new applications. We believe F3DS will enable a new genre of p2p applications that run independently, but exploit social relationship metadata from existing online social networks. These new applications can leverage direct communication and collaboration among participants. F3DS facilitates and maintains continuous direct f2f application collaboration using social relationships extracted from existing social networks.

We now provide an overview of how F3DS overcomes each of the three challenges presented in the previous section.

1.2.1 Extracting and Quantifying Trust

F3DS extracts social relationships from existing social networks to facilitate direct collaboration. With the goal of facilitating the approximation of relative levels of altruism among social peers, we introduce a new metric: SocialDistance. SocialDistance is a synthetic metric that combines direct levels of altruism between peers with an altruism decay for each hop to approximate indirect levels of altruism. The resulting multihop altruism levels are used by social peers to proportion and prioritize the sharing of resources with other social peers. F3DS provides modules for social relationship extraction and analysis—for SocialDistance approximation—that work with several different OSNs.

1.2.2 Resource Allocation Strategy

Using SocialDistance as a heuristic, F3DS empowers applications to intelligently select peers for collaboration and then prioritize resource sharing among those peers. In the particular context of applications where resource sharing provides a security protection benefit, we hypothesize that by prioritizing resource allocation based on SocialDistance, F3DS will ensure that the most social nodes will be the most protected.

1.2.3 Decentralizing Communication

F3DS simplifies the implementation and deployment of f2f applications by providing the following services to initialize and maintain f2f communication: identification and exchange of client IP addresses among social peers, generation and exchange of asymmetric keys for authentication and encryption of f2f messages, and message delivery and queuing services among social peers.

Our goal is as follows: *Create a cross application framework that simplifies collaboration and promotes intelligent resource sharing among social peers.* The resulting framework can be applied to a variety of f2f services. F2f services that are resource intensive (i.e., bandwidth, memory, CPU, storage capacity, storage I/O) are good candidates for F3DS. Such applications might include: f2f backup systems, f2f content caching and distribution, f2f intrusion detection/prevention systems (IDS/IPS), or f2f malware detection.

1.3 Evaluation

To evaluate the utility of SocialDistance for resource allocation within a distributed application, we design and implement a novel flash file/patch distribution method: SocialSwarm [16]. SocialSwarm leverages the SocialDistance metric to approximate the relative levels of altruism among peers in a BitTorrent swarm, so as to overcome the necessity of (and inefficiency created by) resource bartering among friends. In SocialSwarm, we leverage the existing BitTorrent protocol for communication among peers.

To evaluate the utility of the full F3DS framework (including SocialDistance analysis, peer prioritization, and direct f2f messaging) we implement a distributed malware detection application—F3DS Antivirus (F3AV)—on top of F3DS. F3AV leverages the SocialDistance metric to facilitate friend-to-friend virus scanning. We now provide an overview of these novel f2f applications.

1.3.1 SocialSwarm Overview

SocialSwarm facilitates file distribution among social peers as well as nonsocial peers (swarms that contain both socially enabled peers and nonsocially connected peers) by exploiting the well established BitTorrent incentives for collaboration between nonsocial peers. It also allows groups (teams) of peers with preestablished altruism between each other to use resources more effectively by reducing the requirement of games between members of the same team. Assistance to peers is prioritized proportional to social altruism. SocialSwarm can be described as a gather-then-share technique. Nodes first work as a team to interact with anonymous nonsocial peers, gathering socially rare chunks of the file being propagated. As the percentage of chunks held by members of the social group gradually increases, SocialSwarm-enabled group members turn inward and share the chunks altruistically among themselves.

Rather than giving social peers full preference for collaboration over nonsocial peers, SocialSwarm's gather-then-share approach with social peers exhibits levels of altruism that are inversely proportional to the overall rarity of the file chunks. Our approach gives faster file dissemination, because the social peers actively try to get file chunks that are rare in the group, thus ultimately benefiting the group in later stages when altruism comes into play. With these new ideas, our work is state-of-the-art for flash dissemination of large files in a p2p network modeled over a social network.

In our evaluation of SocialSwarm, we find that the most socially connected peers received the greatest benefit, in some cases providing up to a 47% decrease in average download completion time per peer in comparison with average nonsocial BitTorrent swarms.

1.3.2 F3AV Overview

To demonstrate the applicability and value of F3DS, we leverage it to design, implement, and evaluate F3AV, a novel N-version distributed malware detection system. F3AV provides collaborative malware detection among social peers with the greatest levels or protection being provided to security-critical social hubs [17]—the most social users on the network, which are the most likely to propagate large quantities of malware when infected by socially aware malware[1]. Using F3AV, we present and evaluate a novel method for varying the required diversity of virus scanners based on the age of the object being scanned so as to achieve a balance between high rates of malware detection and object scanning latency. Our evaluation using the Amazon Cloud shows that by concurrently leveraging diverse scanners across a social network, a user can achieve a 65% increase in the detection rate of 0- to 1-day-old malware as compared to the average detection rate of individual scanners.

We also show that F3AV provides the greatest scanner resources (including diversity of available scanners as well as overall quantity of results shared) to social hubs—the most social peers in the social network. By providing the greatest scanner resources to the social hubs, the protection of the entire social network is increased. Our implementation of F3AV on top of F3DS is publicly available on the F3DS website [18].

1.3.3 In-network Trust Maintenance

We recognize that maintenance of trust based only on a node's own direct observation of peers' behavior does not benefit from group knowledge—such as that provided by voting or reputation sharing systems. Individual formation of trust, however, is more resilient against bias of reputation and vote aggregators. For this reason, we also create a novel method for individual nodes in a distributed system to maintain trust with other nodes without relying on reputation sharing or voting methods. In our method, nodes share authoritative individual experiences with other nodes and combine collected experiences via a statistical model to maintain individualized trust levels with other system participants. Specifically, we created and evaluated a novel distributed algorithm [19] whereby nodes in a distributed system:

- Monitor their neighbor's behavior (individual "experiences").
- Store these experiences as "experience records."

[1]Malware that propagates itself over social networks

- Share "experience records" with other peers.
- Compute on-going statistical trust and a confidence interval around the trust mean based on direct and indirect experiences of peer behavior.

As part of future work, we envision enhancing F3DS to also leverage our novel statistical trust model for on going trust maintenance. By adding these capabilities to F3DS, nodes would be able to exploit statistical aggregation of both direct and third party experiences to maintain on going levels of peer trust.

1.4 Dissertation Statement

We can design and build a distributed systems framework that benefits from existing social relationships and altruism for newer, more effective p2p applications and services.

1.5 Dissertation Contributions

In this dissertation, we make the following contributions:

- Develop F3DS—a framework for friend-to-friend distributed services.
- Develop and evaluate SocialSwarm, distributed application for flash file distribution using the SocialDistance metric.
- Develop and evaluate F3AV—a novel social network–based distributed malware detection system—using the full F3DS framework.
- Develop and evaluate a distributed approach for statistical trust establishment within collaborative peer-to-peer systems.

1.6 Dissertation Organization

The rest of this dissertation is organized as follows. Our design of F3DS—a novel framework for friend-to-friend distributed services—is presented in detail in Chapter 2. Chapter 2 also presents our method for approximating SocialDistance among peers. In Chapter 3, we leverage the SocialDistance metric and show its utility for dynamic resource allocation among distributed system participants by implementing and evaluating a novel distributed application for flash file distribution—SocialSwarm.

We evaluate the full F3DS framework by using it to implement F3AV—a novel social network–based distributed malware detection system. Details of the F3AV design and the results of our evaluation are found in Chapter 4. In Chapter 5, we present our distributed algorithm for statistical establishment of trust based on peer behavior monitoring among distributed system participants and evaluate this method in the context of wireless sensor networks. As mentioned previously, we envision adding this distributed algorithm into F3DS as part of future work.

CHAPTER 2

EXPLOITING ALTRUISM IN SOCIAL NETWORKS WITH F3DS: A FRAMEWORK FOR FRIEND-TO-FRIEND DISTRIBUTED SERVICES

2.1 Introduction

In this chapter, we present a framework for friend-to-friend (f2f) distributed services (F3DS) to overcome these challenges for developers of new applications. We believe F3DS will enable a new genre of p2p applications that run independently, but exploit social relationship metadata from existing online social networks. These new applications can leverage direct communication and collaboration among participants. F3DS facilitates and maintains continuous direct f2f application collaboration using social relationships extracted from existing social networks.

This chapter is organized as follows. In Section 2.2, we provide the design details of F3DS. The implementation of F3DS is detailed in Section 2.3. Section 2.4 gives an overview of related work.

2.2 The F3DS Framework

Based on the goals outlined in Chapter 1, we design the F3DS framework using the following two basic components, as shown in Fig. 2.1:

1. Friend Service Bus (FSB): FSB is a service that runs on each user's client device and connects F3DS-enabled applications directly with devices used by social peers. The FSB identifies social relationships on existing social networks

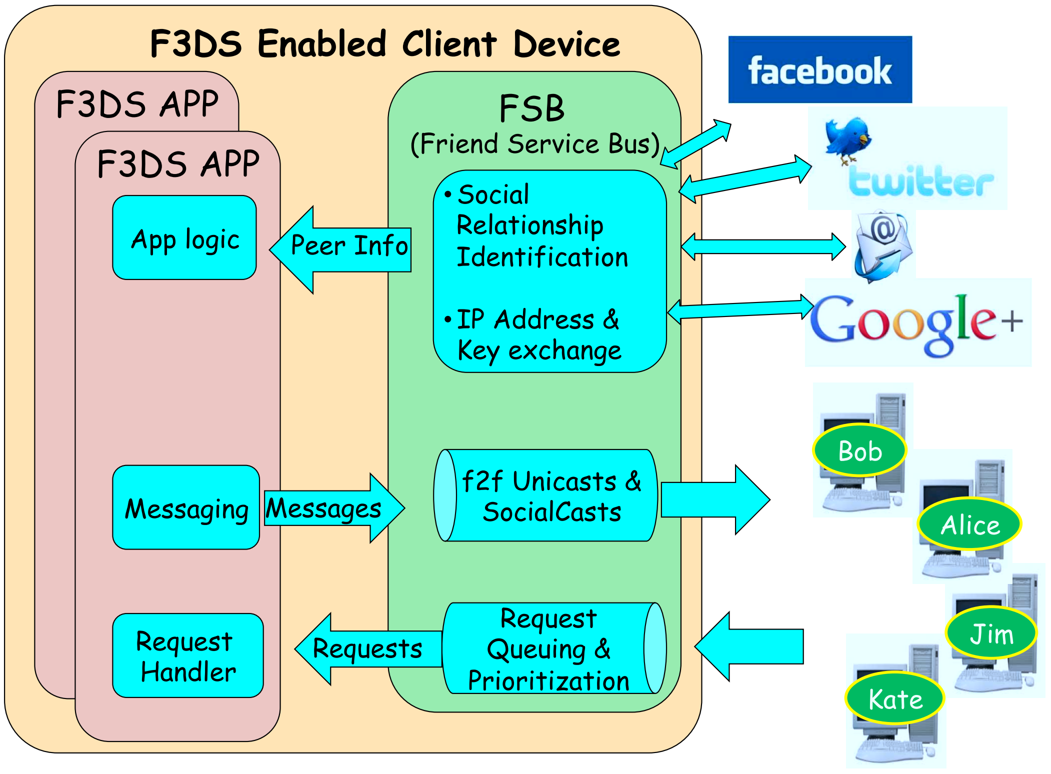

F3DS Enabled Client Device
F3DS APP
F3DS APP
App logic
Peer Info
Messaging
Messages
Request Handler
Requests
FSB
(Friend Service Bus)
• Social Relationship Identification
• IP Address & Key exchange
f2f Unicasts & SocialCasts
Request Queuing & Prioritization
facebook
twitter
Google+
Bob
Alice
Jim
Kate

Figure 2.1. F3DS Framework Overview

and provides each application with a list of social peers including each peer's current IP address, public key, and a metric to quantify the social relationship with that peer. The FSB provides f2f messaging services among social peers.

2. F3DS-enabled applications are those that are built on the F3DS framework by using one or more services from the FSB to collaborate with social peers. Each F3DS application can have its own purpose and objectives, ranging from entertainment to productivity to security. Multiple F3DS applications can run on each user's client device and share the same FSB.

The FSB running on each social peer provides four core services to F3DS-enabled applications: social relationship analysis, IP address and key exchange, f2f messaging, and queuing and prioritization of requests from social peers. We describe each of these four services in detail below.

2.2.1 Social Relationship Identification

Before nodes can collaborate directly with their peers, they must first identify true social peers. F3DS depends on one or more social networks so as to identify active social relationships among users (their relative levels of social connectivity). This can be any social network over which users interact: such as Twitter, Facebook, Weibo, Ren Ren, and Myspace. The traditional SMTP email social network can also be used. A significant body of research has been conducted in analyzing and quantifying relative levels of altruism—commonly referred to as distance—between social peers. The notion of distance between social peers has a long history in social science and was popularized with work originating in the 1920s by Emory Bogardus [20] on racial and ethnic inequality. We design F3DS to support any analysis module that will yield a quantifiable and relative level of altruism among social peers. For our particular implementation of F3DS, we approximate relative levels of altruism among peers within a social network using the "Social Distance" metric, which we now describe in detail.

2.2.2 Approximating SocialDistance

The notations used to approximate SocialDistance between social peers are shown in Table 2.1. We first approximate altruism between direct social peers followed by approximating SocialDistance between in-direct (multihop) peers.

2.2.2.1 Altruism between direct social peers. Altruism between two social peers should not be considered a dichotomy but rather a scale ranging from minimal to very high. Prompting system users to quantify levels of altruism for each of their social peers would be cumbersome and impractical. Instead, F3DS calculates a proportional and directed level of altruism between each given peer a and one of its peers b via equation (2.1).

$$A(a,b) = \frac{I(a,b)}{I(a,all)} \tag{2.1}$$

where $I(a,b)$ is the number of reciprocal interactions a has had within a given time window with b, and $I(a,all)$ is the number of reciprocal interactions a has had with all of its peers during the same window of time. Effectively, $A(a,b)$ represents the proportional willingness that a peer a has to share resources with each of its direct peers. It is important to note that $A(a,b)$ is from the perspective of a and is asymmetric.

2.2.2.2 Approximating SocialDistance between indirect (multihop) peers. The SocialDistance between two direct or indirect peers can be considered the inverse of the altruism between those peers: $D_s(a,b) = \frac{1}{A(a,b)}$. Although SocialDistance itself does not have any absolute meaning, it is a useful synthetic metric to assess relative depreciation of altruism across intermediate peers and to determine which indirect paths between peers yield the highest levels of altruism (the shortest SocialDistance

Table 2.1. Notations

Variable	Description
$A(a,b)$	Normalized level (0,1] of altruism a has towards a social peer b.
$A(s)$	The short form of $A(myself, s)$
$D_s(a,b)$	SocialDistance between a and b

paths). It is important to note that altruism between two peers, $A(a, b)$, is asymmetric and thus SocialDistance, $D_s(a, b)$, is also asymmetric.

Given the known levels of SocialDistance across two pairs of social peers $D_s(a, b)$ and $D_s(b, c)$, F3DS calculates a candidate multihop-directed SocialDistance from a to c with equation (2.2); $HopDecay$ is between 0 and 1 and is set by the evaluating peer a.

$$D_s(a, c)_{candidate} = \left(\frac{1}{\frac{1}{D_s(a,b)} \times \frac{1}{D_s(b,c)} \times HopDecay} \right) \tag{2.2}$$

The SocialDistance between any pair of peers (x, z) indirectly connected via a set of intermediary peers I is defined as the highest altruism level among all known paths between x and z. SocialDistance is calculated via equation (2.3).

$$D_s(x, z) = \forall i \in I : min \left(\frac{1}{\frac{1}{D_s(x,i)} \times \frac{1}{D_s(i,z)} \times HopDecay} \right) \quad where \left(\frac{1}{\frac{1}{D_s(x,i)} \times \frac{1}{D_s(i,z)} \times HopDecay} \right) \leq D_sMax \tag{2.3}$$

$HopDecay$ is thus applied for each additional hop in a given social network path. Note that F3DS narrows its search to a computationally reasonable search space by using a specified maximum useful SocialDistance (D_sMax) beyond which peers are not considered as social peers. These calculations are very similar to those of Dijkstra to find shortest paths; however, instead of adding path lengths, F3DS multiplies approximated levels of altruism.

For computing cumulative altruism values for a nonsocial peer who is multiple hops away, we take the product of the individual altruism values along the path. This conforms to the social network convention of having the decrease in altruism value proportional to the relative distance. The same rationale holds for having a linear $HopDecay$ applied at each hop on the multinode path between two peers.

2.2.2.3 Approximating altruism between indirect (multihop) peers. For all peers a and c for which there exists no direct social relationship, the Altruism between a and c is defined as the inverse of the SocialDistance between those peers: $A(a,c) = \frac{1}{D_s(a,c)}$

Details on the social relationship identification modules in our implementation are described in Section 2.3.1

2.2.3 IP Address and Key Exchange

To bootstrap direct f2f interaction after identifying true social peers, the FSB—running on each peer device—needs to learn the current IP address and TCP port numbers used by their social peers. The FSB on each peer also generates a simple public/private key pair for f2f message encryption and shares the public key with the user's social peers. In F3DS, peers use established messaging mechanisms on existing social networks to exchange IP address and public key information among peers. Exchange of IP addresses and public key information is one of the functions of the FSB.

2.2.4 F2f Messaging

We define f2f messaging as sending secure messages from one peer's client device directly to a friend's client device and bypassing online social networks and message queues. Using such direct and secure messaging reduces the possibility of eavesdropping by politically, commercially, or maliciously motivated 3rd parties. Direct f2f messages also provide reduced network latency compared with traditional routing over social networks. The FSB provides applications with two types of f2f messaging. First, the FSB provides direct asynchronous messaging given a specific peer ID as a target. Second, the FSB provides a method whereby F3DS-enabled applications can send a single message to all of a user's social peers within a given proximity of SocialDistance.

2.2.5 Request Queuing and Prioritization

Given that clients have limited resources and the very common possibility that requests from peers—across one or more applications—would exceed a client's resource capacity, it becomes critical to be able to arbitrate between the resources available on a peer's client device and requests for those resources from social peers. The FSB accomplishes this arbitration via queuing and prioritization of requests.

The FSB receives incoming requests from peers into an ingress queue. The peer ID and application ID associated with each request is authenticated via public/private key request signing. Given that each peer has limited resources, the FSB prioritizes the ingress queue entries based on the SocialDistance of each peer that sends requests. Requesters who have a smaller SocialDistance to a peer offering services will have a higher probability of having their request being serviced. The FSB also throttles incoming requests by both limiting the rate of new requests as well as monitoring available resources on a node (CPU, Memory, disk I/O, network bandwidth). If sufficient spare resources are not available (tunable thresholds), then the process that dequeues requests—and passes them on to the specified application—sleeps for a configurable interval. F3DS-enabled applications can send response messages to previous service requests. Response messages enter the same ingress queue but always receive the highest priority (being put in the front of the queue). Each request includes a TTL (time-to-live) by the requester. This can be overridden (to a lower value) by the request recipient. When the TTL of a request expires before being serviced by an application, the request is dequeued and discarded. An application has no assurance that an F3DS service request sent to other peers will be serviced at all or handled within a desired time window. F3DS service requests are asynchronous and best-effort only.

2.3 F3DS Implementation

To provide some level of OS independence, we choose to implement F3DS as a set of python modules and services on top of an SQLite database. In this section, we first describe this F3DS implementation and then describe our F3AV implementation on top of F3DS. We publish our implementation on the F3DS web site [18].

Here, we provide a brief overview of our implementation of the F3DS social relationship analysis modules and the F3DS mechanisms for f2f messaging. We leave the majority of the design details to a separate extended version of this paper.

2.3.1 Online Social Network Modules

We implement F3DS to use proportional levels of reciprocal interactions between peers to approximate altruism and nominal SocialDistance [16] between those peers. We expand upon our earlier work by creating social network analysis modules as part of the FSB implementation not only for Facebook, but also for Twitter and standard email (SMTP + IMAP). For Twitter, reciprocal pairs of retweets as well as reciprocal pairs of directed tweets are both identified as forms of reciprocal communication. On Facebook, reciprocal pairs of wall postings are identified as reciprocal communication. With email, reciprocal pairs of email messages are identified as reciprocal communication. This conception of SocialDistance is a departure from Bogardus' original emphasis on its subjective, affective aspects. Nevertheless, we note that there is a generally positive relationship between the frequency of social interaction and the strength of a social tie (e.g., members of the same family will tend to have more social interaction than members of the same political party). Granovetter's classic distinction between weak and strong ties [21] is based on factors that are all positively associated with the frequency of social interaction. We use existing messaging mechanisms with Twitter, Facebook, and email to share user IP addresses as well as to distribute public encryption keys. We make the assumption that the existing messaging mechanisms over these forms of social media are reliable and secure enough for the purpose of IP address and key distribution. As part of this research, we do not consider threat models that attack distribution of IP addresses and public keys. For each peer that sends an IP address and public key, the FSB calculates the multihop nominal SocialDistance and communicates the IP, key, and nominal SocialDistance to each application (such as F3AV) via the shared database. The FSB also sets a perceived SocialDistance in the database to equal that of the nominal SocialDistance.

2.3.2 f2f Messaging

As part of the framework, each node runs a Lighttpd web server and provides both XMLRPC services—for messaging, as well as file serving. When messages from peers are received over XMLRPC into the FSB's incoming request queue, they are ordered based on the perceived SocialDistance of the peer that sent the messages. For our initial implementation of the FSB, we use a fixed request servicing (dequeue) rate. In future implementations, we plan to evaluate adapting request dequeing rates based on resource (CPU, memory, etc) availability. The FSB provides a python-based API to applications for enqueuing both f2f unicasts as well as f2f socialcasts. When sending unicasts, applications specify the peer ID to which they want to send the message. When sending socialcasts, applications specify maximum SocialDistance to which they are willing to send the message, and the FSB will send that message to each of the peers within that SocialDistance.

2.4 Related Work

The PeerSoN [22] system shares the F3DS goal of facilitating direct p2p interaction between system users. However, PeerSoN lacks application supporting facilities for social relationship extraction from existing OSNs. Cutillo et al. propose SafeBook [23] a system for protecting communication privacy of social networking users via direct p2p interaction. In contrast to F3DS, SafeBook relies on centralized administration of trust among users via a Trust Information Service (TIS). F3DS is fully distributed and does not have such requirements for trust in a single centralized service. Both PeerSoN and SafeBook lack experimental implementations for use with real-world applications.

Sprout [24] uses existing trust among social peers with the goal to provide reliable cross network message routing services. This is in contrast to F3DS which has the goal of providing a generic framework for any social p2p application service.

Yang et al. propose [25] a structure for information sharing between social network users via direct p2p collaboration. As opposed to F3DS which leverages existing trust among OSN users to provide p2p services, the goal of the system proposed by Yang is to help individuals find relevant content and knowledgeable collaborators who are willing to share their knowledge.

The notion of trust applied to social networks is a promising method to encourage sharing. SPROUT [24] models a social network-based routing scheme where the path selected has peers contributing to the highest computed gross trust value. We extend the multihop notion of trust to generate altruism values for spatially distant nonpeers by combining the intermediate altruism values and a linear decay factor proportional to spatial distance. Different approaches to trust computation over multiple hop distances have been tried. Walter et al. [26] calculate the trust of a node at a given distance to be the product of the trust of all nodes along the path. Decay-based multihop trust metrics have been studied extensively by Marti et al. [24]. Decay-based models are advantageous because they adhere to the social network trust model of having more confidence in nodes closer to the source. In calculating SocialDistance, we choose to implement a linear decay-based model, which is shown in the aforementioned work, to perform equally as well as the exponential decay-based model.

2.5 Conclusions

We have presented F3DS, a novel framework for friend-2-friend distributed services, with resource sharing governed by approximated levels of pre-existing social altruism among OSN participants.

In the next chapter, we present a novel flash file/patch distribution method, SocialSwarm, which leverages altruism between peers in an OSN to overcome the necessity of and the inefficiency created by negotiating for resources. SocialSwarm facilitates efficient file distribution among social peers. In SocialSwarm, peers leverage the SocialDistance metric to proportion and prioritize the sharing of resources with other social peers.

CHAPTER 3

SOCIALSWARM: EXPLOITING DISTANCE IN SOCIAL NETWORKS FOR COLLABORATIVE FLASH FILE DISTRIBUTION

Social networks can serve as an effective mechanism for distribution of vulnerability patches and other malware immunization code. We propose a novel approach—SocialSwarm—by which peers exploit distances to their social peers to approximate levels of altruism and to collaborate on flash distribution of large files. SocialSwarm supports heterogeneous BitTorrent swarms of mixed social and nonsocial peers. We implement SocialSwarm as an extension to the Rasterbar libtorrent library—widely used by BitTorrent clients—and evaluate it on a testbed of 500 independent clients with social distances extracted from Facebook. We show that SocialSwarm can significantly reduce the average file distribution time, not only among socially connected peers, but also among other swarm participants.

3.1 Introduction

Online social networks (OSNs) are perpetually increasing in popularity and utility. Unfortunately, most if not all OSNs have been heavily exploited for malicious purposes. One of the oldest and certainly the most pervasive of all OSNs, SMTP-based email, has been widely used to self-propagate malicious code, either automatically via client vulnerabilities or manually via social engineering, in messages sent to unsuspecting or inexperienced users. When new malicious code enters a social network, it commonly infects hub nodes—nodes with higher degrees of connectivity and malware exposure—more quickly than those users with relatively few social peers.

Brumley et al. [27] have found a technique for automatic generation of malware based on contents of a patch file. Computer users can therefore no longer expect a generous time window for patching their systems before malware is released to exploit the patch. It is becoming increasingly important to study and develop counter-malware techniques such as flash [28] patch distribution, which, like malware itself, exploits hub, cluster, and relative node distance properties to enhance security within OSNs. Unfortunately, existing methods for fast distribution of large files, including typical operating system and application patch files, suffer from two problems in comparison with OSN-based malware propagation. First, systems such as BitTorrent, and its existing derivatives, implement mechanisms to minimize free-riding; such mechanisms create inefficiencies. Second, existing peer-to-peer (p2p) file distribution systems do not ensure that social hubs receive the highest priority in receiving files; such prioritization of patch distribution is necessary to effectively counter OSN-based malware infection campaigns.

The mechanisms used to minimize free-riding are typically tit-for-tat and auction-based p2p incentives [29]. Although these incentives are valuable and necessary in fostering collaboration among purely self-interested individual peers, they come at the price of reduced efficiency. In order to barter for resources, BitTorrent reserves bandwidth in the form of unchoke slots [13]. This reserved bandwidth ensures that peers with which a node is attempting to barter are provided resources of reasonable value. Clients typically avoid increasing the number of unchoke slots because increased numbers reduce the value of each individual unchoke slot thereby increasing the difficulty of negotiating for higher levels of bandwidth. A client's offer of reserved bandwidth to one of its peers does not mean the bandwidth offered will actually be used. The recipient of the bandwidth offer may not have a sustained need for chunks held by the peer or a sustained capability of completely utilizing the offered bandwidth. This holds true even for systems that track and attempt to maintain peer reputation across different swarms [30]. For standard BitTorrent clients that do not track peer reputation across swarms, there is some probability, given optimistic unchoking, that peers will be offered and take bandwidth but will not reciprocate. Such peers are known as leeches. One of the goals of most p2p reputation-tracking systems is to handle leeches. Unfortunately, reputation-based systems are only valuable when kept fresh with a stream of evidence. For infrequent p2p users, reputation systems are commonly inaccurate and/or possibly punitive, given the lack of accumulated reputation evidence.

In this chapter, we propose a novel flash file/patch distribution method, SocialSwarm, which leverages altruism between peers in an OSN to overcome the necessity of and the inefficiency created by negotiating for resources. SocialSwarm facilitates efficient file distribution among social peers. Specifically, SocialSwarm enables groups or teams of social peers with pre-established altruism between each other to use resources more effectively by reducing the requirement of resource bartering between members of the same team. Assistance to peers is prioritized proportionally to social altruism. SocialSwarm also facilitates file distribution between social peers and nonsocial peers using the well established BitTorrent mechanism of resource bartering. SocialSwarm can be described as a gather-then-share technique. Nodes first work as a team to interact with anonymous nonsocial peers, gathering socially rare chunks of the file being propagated. As the percentage of chunks held by members of the social group gradually increases, SocialSwarm-enabled group members turn inward and share the chunks altruistically among themselves. In SocialSwarm, peers leverage the SocialDistance metric to proportion and prioritize the sharing of resources with other social peers.

We evaluate the effectiveness of SocialSwarm by implementing it as an extension to the Rasterbar libtorrent library [31] and deploy it on a testbed of 500 clients. Each client is assigned the identity of a real-world Facebook user and given connectivity characteristics of real-world networks. We find that SocialSwarm achieves an average file download time reduction of 25% to 35% in comparison with standard BitTorrent under a variety of configurations and conditions including file sizes, maximum SocialDistance, as well as leech and seed counts. The most socially connected peers yield up to a 47% decrease in download completion time in comparison with average nonsocial BitTorrent swarms.

The rest of this chapter is organized as follows. In Section 3.2, we present relevant related work. In Section 3.3, we introduce SocialDistance as an approximation of levels of altruism between peers in a social network and present the details of SocialSwarm which exploits the altruism approximations. Section 3.4 constitutes an overview of our implementation of SocialSwarm. In Section 3.5, we evaluate the performance of SocialSwarm. We conclude with a list of several areas for further investigation in Section 3.6.

3.2 Related Work

P2P networks, when compared to static single-source models, have exhibited faster transmission time and greater robustness in dissemination of large files [32]. These advantages have been repeatedly acknowledged in research on security patch propagation where the standard model for patch dissemination is p2p in nature [33, 34].

Gossip-based protocols have been successful in improving BitTorrent's file dissemination time. For example, CREW [35], a gossip-based protocol, clearly outperforms other p2p protocols including BitTorrent for small-sized (1MB) files. However, for bigger file sizes, CREW incurs a higher overhead than nongossip-based protocols. Lind et al. [36] show how small messages spread over social networks through gossip. Our work, however, focuses on the propagation of large files, which have very different propagation characteristics in comparison with small files and messages.

BitThief [37] highlights the free rider vulnerability of BitTorrent by demonstrating that entire files can be downloaded without reciprocation. Augmenting BitTorrent to handle the free riding problem has been the focus of numerous research exercises [38, 39]. The key idea of these solutions is to establish a trust value that a peer accumulates over the course of time. The trust metric thus penalizes peers that do not share and rewards those who are active sharers. Although the notion of trust is a step in the right direction, the peers that participate in a typical p2p swarm change constantly and the benefit of having a trust metric is often lost.

Zhu et al. [17] propose a novel method of worm containment in cellular networks by prioritizing the patching of mobile peers based on their social connectivity. Our technique of collaborative patch distribution—based on social trust on a high level—exploits the same connectedness property proposed by Zhu, but unlike his work, we do not assume any global or centralized knowledge of social connectivity between peers.

Friedman [40] describes the motivation of utilizing the social network as an excellent medium for patch distribution. One of the more convincing reasons presented is that OSNs like real life social networks tend to follow social norms. If computer security is treated as a given norm, a good peer would expeditiously forward a patch file to its social peers: first, to enhance the overall security of the social ecosystem, and, more importantly, to protect itself by having peer nodes that are malware-immune. Our work matches the efficiency of a p2p system with the complex dynamics of a social swarm to create a unique and robust file distribution system.

Different methodologies have been tried to incentivize sharing in social networks. 2Fast encourages sharing in a traditional p2p network by introducing the concept of "helper" peers, which assist "collector" peers—nodes interested in downloading a particular file [41]. The helpers use their idle bandwidth to collect chunks under the direction of the collectors with the ulterior motive that the collectors will help when the helpers need to download a file.

KARMA [42] proposes an incentive system with a more fluid "currency-like" mechanism, where a node can transfer some of its positive currency balance to bootstrap a lower placed node. Our work uses a multihop-based incentive metric that automatically confers the advantage of being associated with a higher placed node or even being part of a higher trust path.

The Tribler [43] system extends 2Fast to social networks by applying the concept of helper and collector peers to social cliques extracted from p2p networks by grouping peers of similar characteristics. Although the concept of a drone helper to retrieve content is useful for assisting a peer with the retrieval of some content, it does not assist a large group of social peers in identifying and retrieving socially rare chunks.

We share with Tribler the common goal of harnessing a social network for file distribution, but our work differs from Tribler's in several meaningful ways. The collaborative download in Tribler needs "helper" nodes that do not participate in the actual file being distributed. This is in contrast to our work, where all the nodes in the swarm actively collaborate in the file being shared. Another important operational aspect of Tribler is that the helper nodes need explicit approval from the collector node regarding the uniqueness and rarity of a file chunk before downloading it on behalf of the collector node. We develop the notion of social rarity of a chunk that gives a node sufficient confidence to download a chunk that is socially relevant to the clique to which it belongs. Tribler's incentive mechanism does not allow for a transitive relationship between a prospective collector node and a helper node. All incentives that are reclaimed correspond to the direct interactions in the past between the nodes in purview.

The standard BitTorrent protocol uses a fixed and small number (typically 5 to 8) of unchoke slots for playing tit-for-tat with swarm members. When a peer joins a swarm, it initially chooses a random chunk to download and then begins to offer this chunk to barter with other swarm members. During these initial stages of bootstrapping swarm entry and tit-for-tat negotiation, some portion of the peer's upload bandwidth is underutilized given the small and fixed number of unchoke slots. As has been shown in the SeCond [44] protocol, a more efficient use of p2p bandwidth is to freely share it with swarm peers regardless of reciprocity. Although free bandwidth sharing is more efficient, it is vulnerable to exploitation by any purely self-interested swarm member. Tit-for-tat thus serves as a required and effective enforcement mechanism for minimizing the level of free-riding possible in a swarm whose members are purely self interested.

Karame et al. [45] defend the analytical rationale behind decomposing p2p peers engaged in a collaborative download into "small coalitions" to give a near-optimal file distribution time. The key idea of this work is that the aggregation of locally optimal solutions obtained in the smaller teams form a globally optimal solution, which is often very expensive to compute if the problem is not decomposed. Our work builds upon these conclusions by engaging social peers to work together as teams.

Dynamic configuration of p2p peers of similar characteristics into teams has been shown to limit free riding in collaborative downloads [46]. A common approach tried in team-based collaborative downloads is to presume altruism with social peers while employing tit-for-tat policy with nonsocial peers. This model has been studied by Galuba et al. [47] where preference is always given to peers. In contrast, we employ a "gather-then-share" approach with social peers where the altruism exhibited with social peers is inversely proportional to the overall rarity of the file chunks. Our approach gives faster file dissemination because the social peers actively try to get file chunks that are rare in the group, which ultimately benefits the group in later stages when altruism comes into play.

3.3 SocialSwarm Design

SocialSwarm combines tit-for-tat/auction between nonsocial peers with an altruistic sharing of resources among social peers. A SocialSwarm client freely offers bandwidth to its social peers based on each of their SocialDistances. For nonsocial peers as well as those who are distant socially, a peer uses the standard BitTorrent method of engaging in tit-for-tat to negotiate bandwidth. In Section 3.3.1 we provide an overview of the design of SocialSwarm, which is followed, in Section 3.3.2, by details on the notations we use. Finally, Section 3.3.3 provides the core details of how these notations are combined and used within SocialSwarm.

3.3.1 Overview

With the goal of maximizing collaboration between social peers by reducing the inefficiencies of BitTorrent while still maintaining game-based techniques to encourage the cooperation of nonsocial peers, we have developed the following design characteristics for SocialSwarm.

- Full compatibility with the BitTorrent protocol: SocialSwarm is designed to leverage the existing benefits of the BitTorrent protocol, while enhancing the capabilities of BitTorrent clients to collaborate with social peers. SocialSwarm thus adapts to mixed swarms of socially connected and nonsocially connected peers.

- Social network independence: With its modular social network analyzer, SocialSwarm can exploit the history of connectivity and interaction among social peers within any capable social networking system, obviating reliance on any particular social protocol or messaging system.

- Coordination of chunk collection among social peers: Like standard BitTorrent, SocialSwarm begins by relying on peers seeking chunks of data that are rare system-wide. Over time, however, SocialSwarm increasingly seeks after chunks that are rare among social peers only—socially rare chunks. Thus, as a file download progresses, a peer transitions its focus from globally rare chunks towards socially rare chunks. This transition of chunk collection focus is the "gather" portion of the "gather-then-share" approach and is detailed in Section 3.3.3.2.

- Adaptive unchoke slot count and upload bandwidth allocation: Like standard BitTorrent, SocialSwarm begins by allocating a fixed number of unchoke slots for playing tit-for-tat/auction games with nonsocially connected peers. Over time as more chunks are acquired by social peers, SocialSwarm gradually decreases the number of unchoke slots allocated to tit-for-tat/auction and repurposes the bandwidth associated with those slots, offering unchoke slots freely to social peers who are granted bandwidth with a priority based on SocialDistance. This adaptive bandwidth allocation is the "share" portion of the "gather-then-share" approach and is detailed in Section 3.3.3.1.

- Targeted nonsocial peer selection for optimistic unchoke: Rather than a commonly used metric of time-since-last-unchoke for optimistic unchoke peer selection, SocialSwarm is designed to select peers probabilistically based on the social rarity of file pieces that they hold. Details of this peer selection method are given in Section 3.3.3.3.

Fig. 3.1 presents an example of how SocialSwarm running on Bob's client retrieves social peer interaction history from social networks on behalf of Bob. When a file distribution swarm starts, Bob's SocialSwarm client identifies Bob's social peers in the swarm, coordinates chunk collection with them, and altruistically shares bandwidth with them. The SocialSwarm clients interact with each other as well as with standard (nonsocial) BitTorrent clients. In this example, Jim's SocialSwarm client also has a connection to a leech. Although Jim initially uses resource bartering and preallocates resources to negotiate with the leech, over time, Jim reserves less bandwidth for

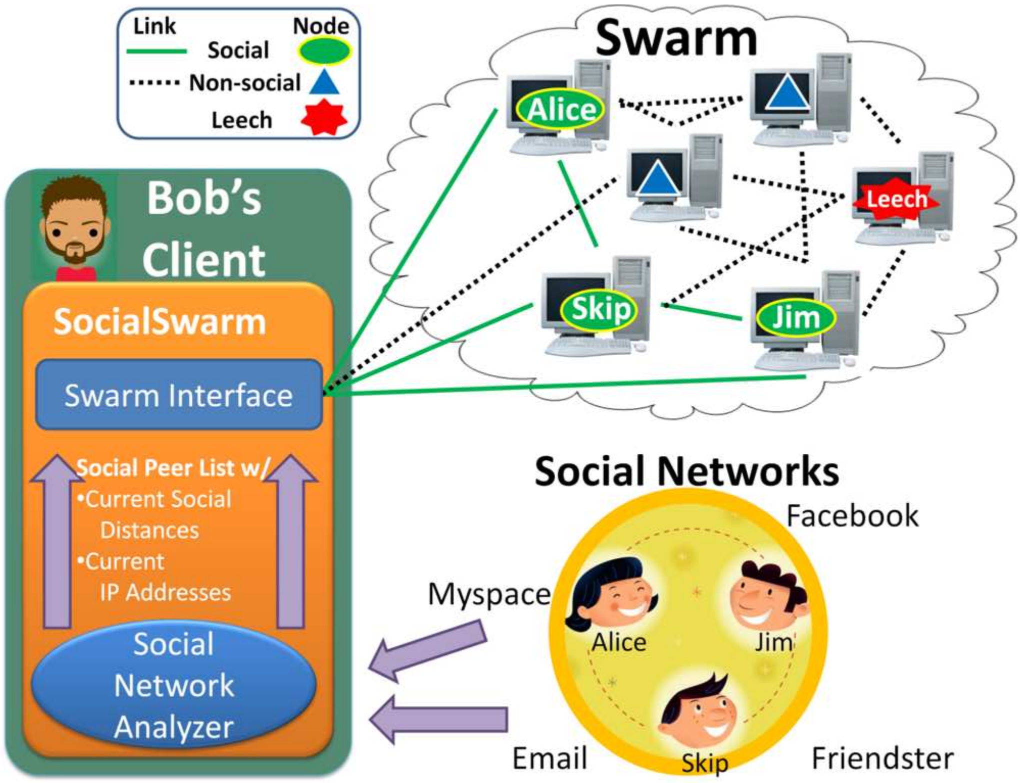


Figure 3.1. SocialSwarm Interaction Overview

resource bartering and openly shares its bandwidth with its social peers. By reserving fewer resources for specific peers, Jim's risk is diversified given that his reliance on any particular node is reduced. Thus, even though Jim is not explicitly aware of the leech, SocialSwarm's diversified collaboration allows it to be less affected by the leech in comparison with standard BitTorrent clients.

3.3.2 Notations

Tables 3.1, 3.2, and 3.3 respectively provide an overview of the given, measured, and derived variables used by SocialSwarm. All variables are from the perspective of each node in the system. Each node independently receives, measures, and derives its own set of variables. Each measured variable is over a time period of t_i. Each derived variable is recalculated each t_i. The given and measured variables that could not be fully described in the table are described in detail in this section. The next section (3.3.3) describes the derived variables along with the SocialSwarm algorithm.

3.3.2.1 Overall rarity for each given chunk. Each peer first calculates $A(b)$ for all peers b in its social network using the methodolgy presented in Sec-

Table 3.1. Given Variables

Variable	Description
$A(a,b)$	Normalized level (0,1] of altruism a has towards a social peer b.
$A(s)$	The short form of $A(myself, s)$
$D_s(a,b)$	SocialDistance between a and b
D_sMax	Maximum SocialDistance whereby a client considers peers to be part of its social network.
C	Set of chunks in the file being downloaded. Knowledge of this set is provided by the .torrent file used to start the swarm.
t_i	The time interval i. The interval used here is the optimistic unchoke interval (commonly 30 seconds).
S	The set of social peers ($D_s << \infty$) that are using a SocialSwarm-enabled BitTorrent client. To form S, a list of all peers is retrieved from the torrent tracker; nonsocial peers are excluded.
N	The set of all other (nonsocial) peers participating in the swarm (where $SocialDistance = \infty$). To form S, a list of all peers is retrieved from the torrent tracker, and social peers are excluded.

Table 3.2. Measured Variables

Variable	Description
$V(t_i, c, n)$	0 or 1 indicating the availability of a particular chunk, c, at a particular peer, n, at time interval t_i. This information is shared between peers and the tracker as part of the BitTorrent protocol.
$B_s(t_i)$	The percentage of a client's upload bandwidth used at t_i for altruistic sharing with its social peers. (each client measures its own bandwidth)
$B_n(t_i)$	The percentage of a client's upload bandwidth used at t_i for bartering with its nonsocial peers. (each client measures its own bandwidth)
$R_o(t_i, c)$	The overall rarity of a chunk c across all peers, social and nonsocial, at t_i.
$R_s(t_i, c)$	The social rarity of a chunk c across its set S of social peers at t_i.
$R_n(t_i, c)$	The nonsocial rarity of a chunk c across its set N of nonsocial peers at t_i.

Table 3.3. Derived Variables

Variable	Description
$R_w(t_i, c)$	The combined weighted rarity at t_i.
$R_A(t_i)$	The average social rarity across all the chunks C at t_i.
$H(t_i, p)$	The weighted rarity of chunks held by a peer p at t_i.

tion 2.2.2. For each given chunk c, a peer calculates the overall rarity of the chunk across all peers in the swarm (peers in S and N) via equation (3.1).

$$R_o(t_i, c) = \begin{cases} 1 & \text{if } |S \cup N| = 0 \\ 1 - \frac{\sum_{n \in S \cup N} V(t_i, c, n)}{|S \cup N|} & \text{otherwise} \end{cases} \tag{3.1}$$

3.3.2.2 Social rarity for each given chunk. For each given chunk c, a peer calculates the rarity of the chunk across its set S of social peers using equation (3.2).

$$R_s(t_i, c) = \begin{cases} 1 & \text{if } \sum_{s \in S} A(s) = 0 \\ 1 - \frac{\sum_{s \in S} A(s) * V(t_i, c, s)}{\sum_{s \in S} A(s)} & \text{otherwise} \end{cases} \tag{3.2}$$

This equation allows each node to weigh the priority of each chunk proportionally with the altruism expressed towards each of the node's online and in-swarm social peers. The chunks that are rare to social peers connected by shorter SocialDistances, and thus higher levels of altruism, are assigned a higher priority than the chucks that are rare to peers connected by higher SocialDistances.

3.3.2.3 Nonsocial rarity for each given chunk. A peer calculates the rarity of a given chunk across its set N of nonsocial peers using equation (3.3).

$$R_n(t_i, c) = \begin{cases} 1 & \text{if } |N| = 0 \\ 1 - \frac{\sum_{n \in N} V(t_i, c, n)}{|N|} & \text{otherwise} \end{cases} \tag{3.3}$$

3.3.3 SocialSwarm Algorithm

SocialSwarm varies the behavior of standard BitTorrent in three basic ways. Table 3.4 lists these three changes with their respective input heuristics. Each of these actions and heuristics is described in detail below:

3.3.3.1 Adaptive bandwidth allocation. SocialSwarm leverages munificence between social peers by dynamically allocating a portion of available bandwidth toward free bandwidth sharing with social peers.

Karame et al. [45] show that combining locally optimal solutions of the smaller social teams would give a globally optimal solution for the entire social network. Hence, we introduce a concept of social rarity that is unique to different cliques in

Table 3.4. SocialSwarm in a Nutshell

	SocialSwarm Action	Input Heuristic
1	Vary % bandwidth offered to social vs nonsocial peers	% of game completed (from social group perspective)
2	Vary the set of targeted chunks based on the group (social or nonsocial) being collaborated with currently	Bandwidth % used by social & nonsocial peers
3	Probabilistically unchoke the nonsocial peers that hold the most desired chunks	Rank peers based on the rarity of the chunks that they hold

the social graph and is easy to compute. We also incorporate the overall rarity of a chunk to get a fair representation of actual rarity.

As the allocation level of bandwidth for social peers increases and is actively used, the number of unchoke slots available for bartering with nonsocial swarm peers decreases. To determine what portion of its available upload bandwidth a client should allocate to social peers, SocialSwarm uses the average rarity of chunks across social peers as a heuristic. Effectively the assessment of the rarity of all chunks across social peers is how SocialSwarm determines the stage of the game.

A SocialSwarm client estimates the average social rarity for all chunks at each t_i by normalizing the social rarity of all individual chunks by the chunk count using equation (3.4).

$$R_A(t_i) = \frac{\sum_{c \in C} R_s(t_i, c)}{|C|} \tag{3.4}$$

A SocialSwarm client then allocates a certain maximum percentage of its bandwidth (MaxSocialBandwidth=$(1 - R_A(t_i))$) for use with its social peers. Using levels of altruism towards social peers, a SocialSwarm client will put its social peers into an ordered list. Starting at the top of this list (those peers with the highest altruism), a peer will unchoke its social peers one by one until either the predetermined MaxSocialBandwidth percentage of upload capacity has fully been consumed by its social peers or a maximum limit on unchoked social peers is reached. This method ensures that the peers with the highest amount of aggregate social altruism—typically those peers who have a higher degree of connectivity—are allocated the greatest bandwidth and thus are potentially able to receive the file faster than peers with lower degrees of social connectivity.

All bandwidth not allocated or consumed by social peers is allocated to traditional BitTorrent unchoke slots (of reasonable size) and used for bandwidth bartering.

3.3.3.2 Chunk prioritization. When social clients initially join swarms and when social bandwidth available from social peers is scarce, they must barter for bandwidth and chunks with nonsocial peers. Initially, clients will target chunks that are rare across nonsocial peers. As social peers acquire an increasing percentage of chunks, the average rarity of chunks across social nodes decreases and more bandwidth is allocated toward social purposes. As social peers increase their usage of this bandwidth, a client will increasingly target chunks that are rare across social peers (as opposed to chunks that are rare across nonsocial peers).

SocialSwarm is thus analogous to a real-world tribe or clan whose members initially barter with nonclan members for goods not yet available in the clan. As more goods are obtained by clan members, they gradually decrease their external bartering and increase the amount of free sharing of goods within the clan. The amount of bartering with external, self-interested entities is thus determined by the availabilty of goods (chunks) within the clan. Here, availability is defined as both chunk possession and ability to share (bandwidth availability).

A SocialSwarm client accomplishes this collaboration by varying its calculation of chunk rarity based on the percentage of bandwidth actively being used by social peers, denoted by $B_s(t_i)$, and the percentage of bandwidth used by nonsocial peers, denoted by $B_n(t_i)$. Both $B_n(t_i)$ and $B_s(t_i)$ may be 0% concurrently if none of a node's bandwidth is being used by any of its peers.

The level of influence that social peers' chunk holdings exert over a node's concept of chunk rarity increases as the level of bandwidth sharing among social peers increases. When the majority of its bandwidth is used for bartering with nonsocial peers (when $B_n(t_i)$ is large), a SocialSwarm client will focus mostly on chunks that are rare across nonsocial nodes by making $R_n(t_i, c)$ dominant. Alternatively, when the majority of its bandwidth is used for collaboration with social peers (when $B_s(t_i)$ is large), a SocialSwarm client will focus mostly on chunks that are rare across social nodes by making $R_s(t_i, c)$ dominant. When little of its bandwidth is in use (when both $B_n(t_i)$ and $B_s(t_i)$ are small), a SocialSwarm client will use the traditional BitTorrent algorithm of focusing on chunks rare to the swarm overall by making $R_o(t_i, c)$ dominant.

Thus, using its current $B_s(t_i)$ and $B_n(t_i)$ bandwidth percentages as weights, a SocialSwarm client combines the social, nonsocial, and overall rarities to form a combined weighted rarity for each given chunk using equation (3.5).

$$\begin{aligned} R_w(t_i, c) = {} & R_n(t_i, c) * B_n(t_i) + \\ & R_s(t_i, c) * B_s(t_i) + \\ & R_o(t_i, c) * (1 - B_n(t_i) - B_s(t_i)) \end{aligned} \tag{3.5}$$

A SocialSwarm client prioritizes the download of chunks from its connected peers based on their combined weighted rarity, $R_w(t_i, c)$. This allows a client to coordinate its collection of socially rare chunks with its social peers.

3.3.3.3 Optimistic unchoke candidate selection. Typical BitTorrent implementations use either random selection or a longest-since-unchoke heuristic in deciding which peer should be optimistically unchoked for the next round. SocialSwarm instead probabilistically selects a peer out of a prioritized list ordered on availability of rare chunks at each peer. Thus, a peer will target a peer with the largest group of *rare chunks* at each time interval t_i by calculating the level of rare chunks held by each peer using equation (3.6).

$$\begin{aligned} & For\ p \in S \cup N, \\ & H(t_i, p) = \frac{\sum_{c \in C} V(t_i, c, p) \times R_w(t_i, c)}{|C|} \end{aligned} \tag{3.6}$$

Using its list of social peers ordered on $H(t_i, p)$, a peer will randomly choose the next peer for probabilistic unchoke using proportional selection based also on $H(t_i, p)$.

3.4 Implementation and Test Setup

In this section, we first present details of our SocialSwarm implementation. Second, we provide an overview of our test infrastructure. Next, we discuss the social network data set used to drive our tests, and finally, we analyze the performance of SocialSwarm in comparison with the standard BitTorrent protocol.

3.4.1 Implementation

We implement the SocialSwarm algorithm as an extension to the Rasterbar libtorrent library [31] version 0.13.1. Libtorrent is a library leveraged by a variety of different GUI- and text-based front ends to provide full BitTorrent functionality. Enhancing libtorrent with SocialSwarm as an extension allows SocialSwarm to be used with a variety of existing BitTorrent clients. To evaluate the SocialSwarm BitTorrent extension, we use an unmodified version of qBittorrent v1.1.0, a Qt-based libtorrent front end [48].

SocialSwarm-enabled libtorrent receives a list of known social peers, including relative SocialDistances for each peer and the peer's most recent known global IP address. SocialSwarm compares its list of known social peer IP addresses with the IP addresses of each of the peers in BitTorrent swarms as received from the BitTorrent tracker to find social peers who are participating in each swarm. Once a social peer is identified, SocialSwarm-enabled libtorrent uses a new flag on the BitTorrent extended peer handshake to determine if the social peer is SocialSwarm-enabled. If a social peer is identified, but does not support the SocialSwarm protocol, then SocialSwarm libtorrent will treat the peer as a nonsocial peer. Apart from matching IP addresses and checking its SocialSwarm flag, SocialSwarm currently does no other social peer verification. A social network analyzer is developed to take a set of user interactions within the social network—in this case Facebook wall postings—and first calculate proportional levels of direct altruism between the Facebook users in the data set and then calculate levels of indirect altruism. More details are found in the next section.

3.4.2 Social Network Data Set

To evaluate SocialSwarm, we use an anonymized data set from interactions—wall postings—of 500 Facebook users [49]. For each social network member, we analyze the number of reciprocal wall postings within a given time period. Each pair of reciprocal postings is considered a single interaction. These interactions are used to determine the single-hop/direct levels of altruism between Facebook users. We use the inverse of this altruism as our single-hop/direct SocialDistance between peers. Using these single-hop SocialDistances, we calculate the multihop altruism and SocialDistances using a $HopDecay$ of 0.95 with the method described in Section 3.3. After constructing the social tree for all users in the Facebook data set, we choose a single peer and do a breadth first walk over the tree until a total of 500 social peers is selected (traversed). We then assign each of these 500 social peers to a virtual SocialSwarm client for our evaluation. Each node has knowledge only of its own social peers (rather than global knowledge) and considers all other nodes outside of its maximum SocialDistance as “nonsocial.”

Each client is assigned a unique virtual machine with a unique IP address. We make the assumption that the social network analyzer has a method of determining a public IP address for known social peers. This IP address determination would occur either via extraction from existing social network interactions or some extension to those interactions that enables extraction of peer IP addresses for direct and indirect social peers. IP address identification of users is dependent on the social network being used. Email, for example, commonly includes the IP address of the original sender as one of the headers. This is also true for certain webmail systems, such as Hotmail and Yahoo mail. There are methods of obtaining users’ IP addresses on MySpace [50, 51] and until recently, IP addresses of Facebook users could be directly extracted from the email notifications sent on activity between social peers, such as wall postings [52]. This information, when coupled with the peer interaction data set, could uniquely give the IP addresses of the users helping to bootstrap our social network. To incorporate dynamically changing IP addresses, Koolean [53] proposes a solution whereby each user is associated with a permanent identifier and coupled with a challenge-response mechanism with the social peers, the user is verified, and the current IP address of the user is deemed authentic. The updated connection details of the newly joined peer are distributed across the social network. Because the main focus in our work is to demonstrate faster file distribution, we have not yet incorporated automatic identification of social peer IP addresses into the social network analyzer.

The 500 users in our experiments were extracted from a much larger data set of Facebook wall postings which contained over 43,000 nodes in total. We thus utilized less than 1.2% of the total social network data set. Unlike our evaluation nodes, real-world users of SocialSwarm would not be restricted to searching for social peers among an isolated group of 500 nodes, but rather could search for and utilize any available peers on their full social network within their chosen max SocialDistance. For example, by increasing the number of social network nodes that we included in our analysis to 5000 and given no variation in maximum SocialDistance, we found a 55% average increase in the number of peers a node would consider as its peers. Fig. 3.2

shows the number of single-hop/direct and multihop peers each node considers as part of its social network given a maximum SocialDistance. Fig. 3.3 shows the number of peers each node considers as part of its social network when the subset of the data is expanded to include 5000 nodes total. Although our experiments constrained each node to search only for social peers within a very small subset of the social network, and given the differences between Fig. 3.2 and Fig. 3.3, it is clear that in real-world deployments of SocialSwarm nodes would have many additional peers from which to choose.

A peer continues a shortest-path-first search of its social network adding new social peers until some maximum allowable SocialDistance is reached. Based on this Facebook data set, Fig. 3.4 shows the average number of social peers each user has in relation to the maximum allowable SocialDistance. Although the number of additional peers continues to increase at a reasonable rate with greater SocialDistances, the relative altruism expressed toward those additional peers significantly degrades. Fig. 3.5 shows the average altruism expressed by each node to its social peers as the

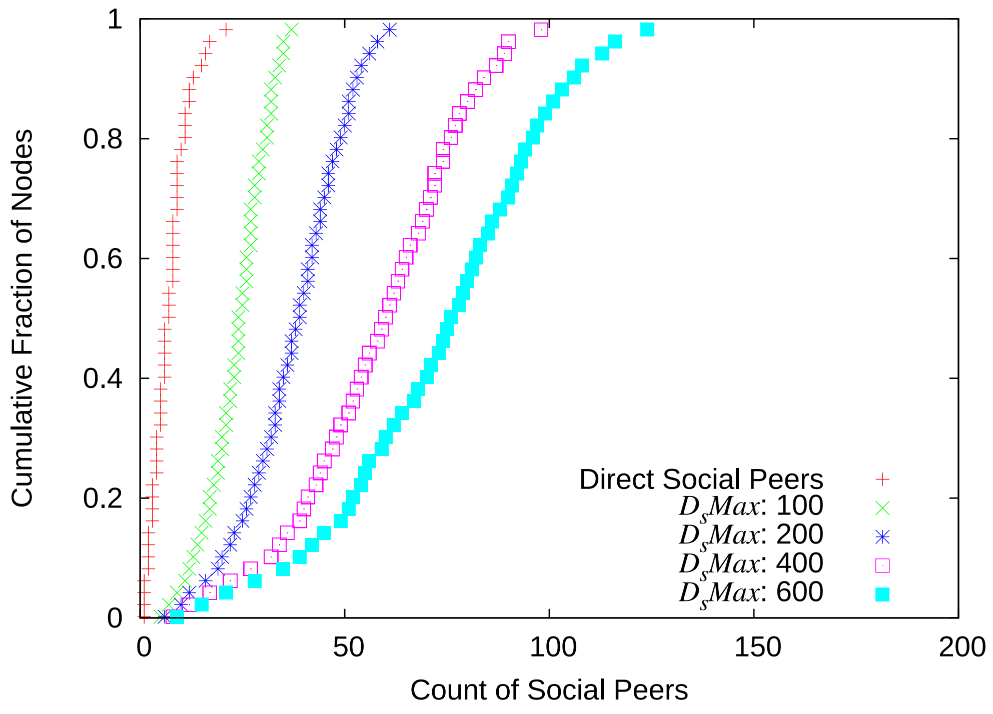


Figure 3.2. CDF of Social Peer Count for 500 Nodes

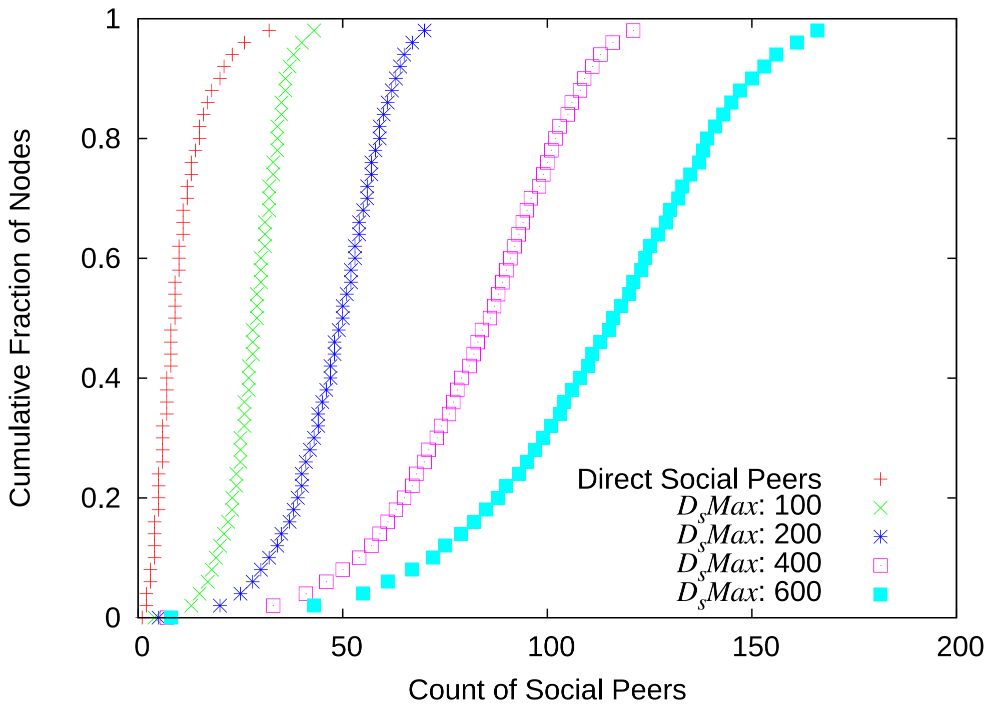


Figure 3.3. CDF of Social Peer Count for 5000 Nodes

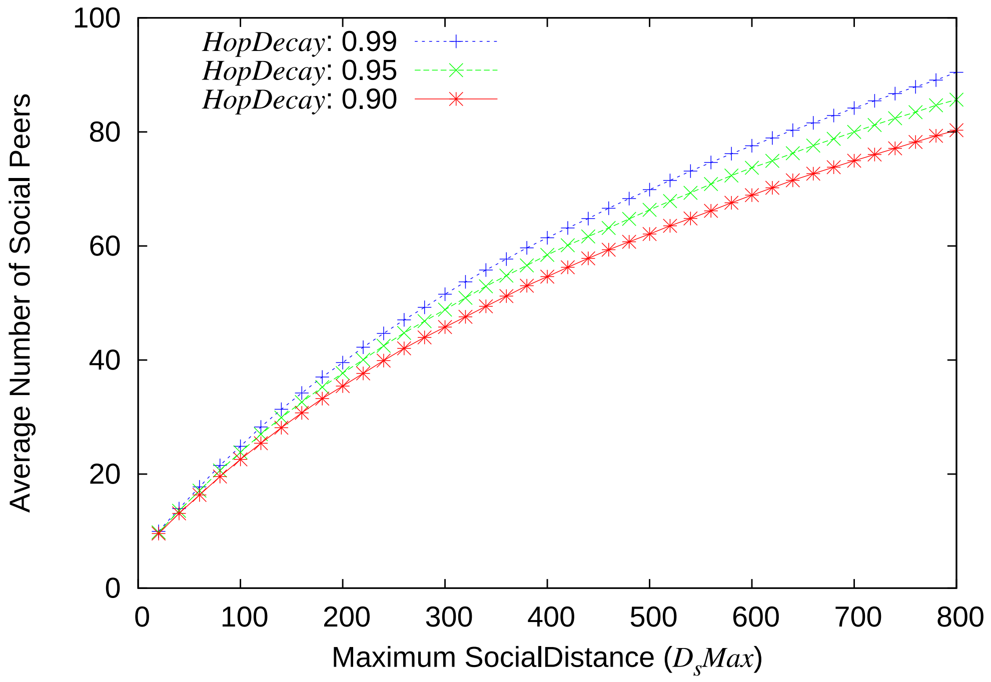


Figure 3.4. Number of Social Peers in Network Based on Max SocialDistance

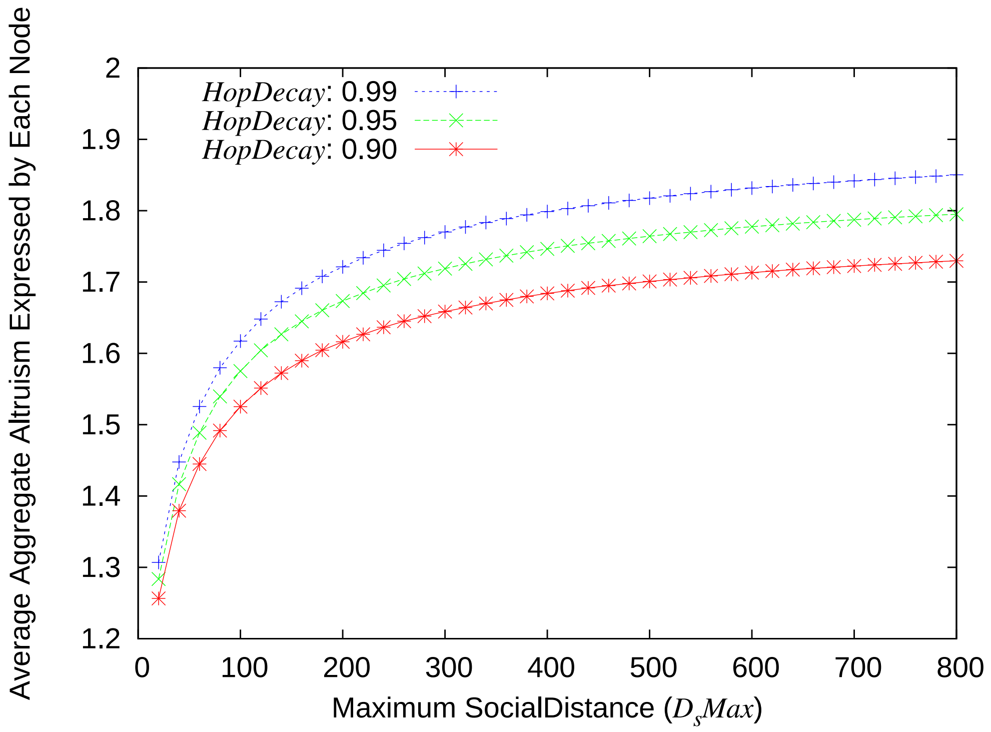


Figure 3.5. Average Aggregate Altruism from Each Peer Based on Maximum SocialDistance

maximum SocialDistance is varied. This figure shows a drop off in altruism increase beyond a maximum SocialDistance of 150 to 200.

It is important to note that SocialSwarm is not dependent on any specific social peers being online or available; SocialDistance is used for prioritizing bandwidth offered. Any unused bandwidth due to offline peers will be offered to other social peers who are online, even if they have a higher SocialDistance. If no social peers are online or if those that are online are not consuming bandwidth, then SocialSwarm will revert to interacting with nonsocial peers exclusively.

3.4.3 Test Infrastructure

Our testbed consists of 20 high-performance servers. Each server has 24GB of RAM and 8 Intel-based Xeon CPU cores (two quad core Xeon L5420 processors per system). All servers are fully connected through a gigabit switch with a fully connected 68Gbps back-plane. A torrent tracker is run on a separate machine that also has full gigabit connectivity to the same network switch. On each server, we create 25 virtual clients for a total of 500 virtual clients. Each client runs Debian Linux version 5.0.3 inside of an OpenVZ virtual container. The storage for each virtual container is located on a set of high performance SAN arrays with 15K rpm drives to ensure that the probability of I/O contention is minimized. Network tuning and shaping is put in place so that each virtual client can be tuned independently with each of the following metrics: incoming maximum throughput, outgoing maximum throughput, incoming packet latency, and outgoing packet latency.

CPU, network, memory, and disk I/O are monitored on all servers to ensure that resource contention between the virtual environments did not occur.

3.5 Evaluation

We evaluate SocialSwarm using the infrastructure described in Section 3.4.3. We now provide an overview of our testing methodology, followed by the results of our evaluation.

3.5.1 Evaluation Methodology and Criteria

In order to evaluate the flash-file distribution speeds of SocialSwarm in comparison with standard BitTorrent, we preload a single peer in the system with the file contents, making it the sole seed for the system. We then start all clients within 10 seconds of each other. We assume that in real-world use, external mechanisms for communicating torrent availability and automatic triggering of swarm participation exist. Likely such a mechanism would use messaging capabilities of the social networks themselves. In all experiments, we use the parameters, shown in Table 3.5, as input to each of the tests, unless otherwise specified.

The network configuration (including latency between peers) was made independent of each node's social connectivity. A vast majority (489 peers) of the 500 peers are assigned a maximum upload bandwidth of 256Kbit (32KB) per second and a maximum download bandwidth of 1Mbit (128KB) per second. The remaining 11 peers are assigned a maximum upload bandwidth of 2.5MB/second and a maximum download bandwidth of 5MB/second. One of these 11 high-speed peers is chosen as the seed. These bandwidth capabilities attempt to simulate a mix of home users with slower Internet connections combined with a few corporate/educational/FTTH (fiber-to-the-home) users (including the seed) with much faster Internet connections [54]. Each data point provided in this section represents an average across 10 runs, with each run using an identical configuration of nodes including seeds.

Table 3.5. Baseline Test Parameters

Parameter	Value
File Size	25 MB
RTT Interpeer Latency	48 ms
Altruism $HopDecay$	0.95
Maximum SocialDistance (D_sMax)	400
Maximum Number of Concurrently Unchoked Social Peers	30
Leeches (Noncontributing Peers)	0
Seed Bandwidth	2.5MB/sec

3.5.2 Comparison of Basic Download Time

In this section, we evaluate the average download time of SocialSwarm compared to that of standard BitTorrent. One of our first tests is to compare the average time required for a single file to be dispersed to all participating peers.

Fig. 3.6 provides a cumulative density function (CDF) of the 500 peer file distribution time for a fully socially enabled run as well as a nonsocially enabled run.

As shown in Table 3.6, the average download time of SocialSwarm for the 499 peers is reduced by 25.7% compared to BitTorrent. The performance gain (33.5%) for the most socially connected peers (top 1%) is greater than the one (15.7%) of the least socially connected peers (bottom 1%).

Fig. 3.7 shows the average download rate per peer over time. The first minute or so of the experiment shows a significant spike and fluctuation in the download rate for all peers. This is due to the fact that all peers are initiating connections with the tracker as well as with each other. All peers are sharing chunk availability maps with every other peer with which they initiate connections.

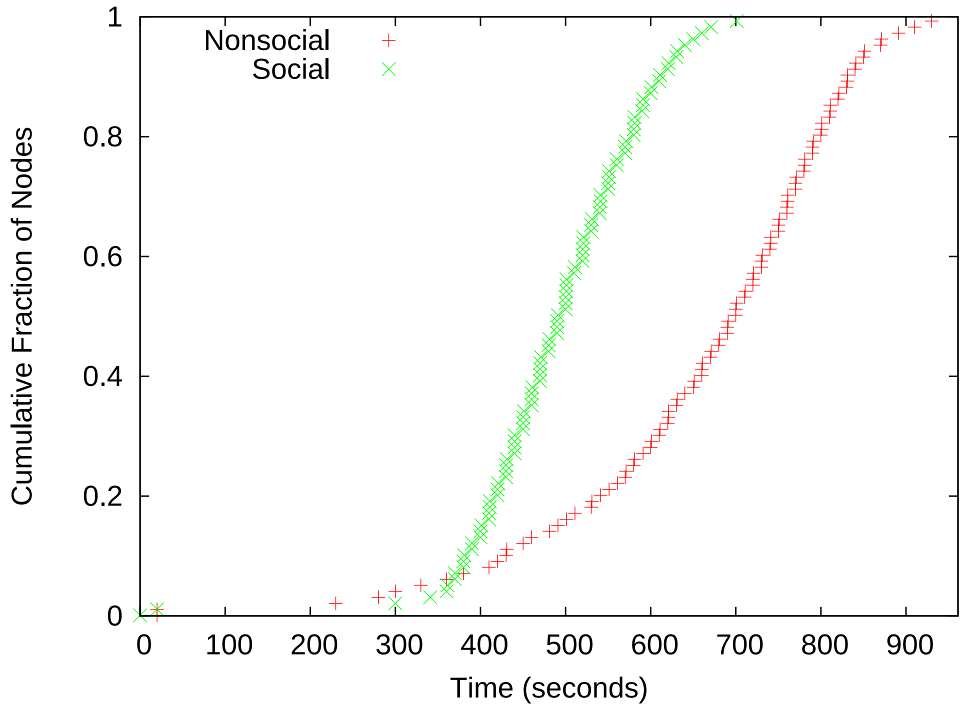


Figure 3.6. Social vs Nonsocial CDF of 25MB file and 0 Leeches

Table 3.6. Average Download Time and Percent Improvement with a 99% confidence interval

	Nonsocial	Social	Top 1% Social	Bottom 1% Social
Download Time (sec)	654±11	486±3	435±17	551±22
Improvement (%)	base	25.7	33.5	15.7

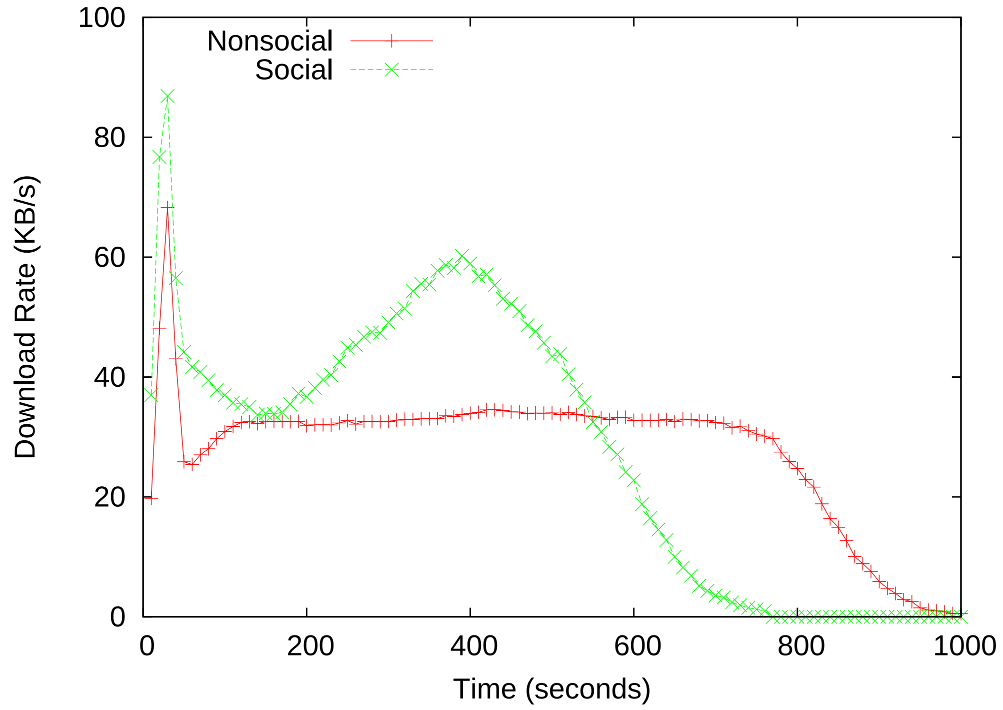


Figure 3.7. Client Download Rate Comparison

After about 180 seconds, the nonsocial peers level out in their sustained bandwidth usage. The social peers, however, slowly allocate more bandwidth to social peers as the average social rarity of chunks decreases; this is shown in Fig. 3.8. It is this bandwidth surge—the peak of which is around 400 seconds into the test—that allows social peers to complete earlier and turn into seeds earlier.

The results of the first 60 seconds of average social rarity are inaccurate due to lack of social peer and chunk availability information during system start-up and initialization. Social peers must find and establish connections with other social peers, then receive piece availability bitmaps from those social peers before declaring that chunks are truly socially rare.

These figures show that for both social and nonsocial swarms, rarity of chunks nears zero around 200 seconds before the download rate nears zero. This is because rarity is averaged across both downloading clients and seeds. Swarm participants that become seeds earlier do not necessarily decrease the average bandwidth used per node. The unchoke slots vacated by these newly formed seeds are quickly reoccupied

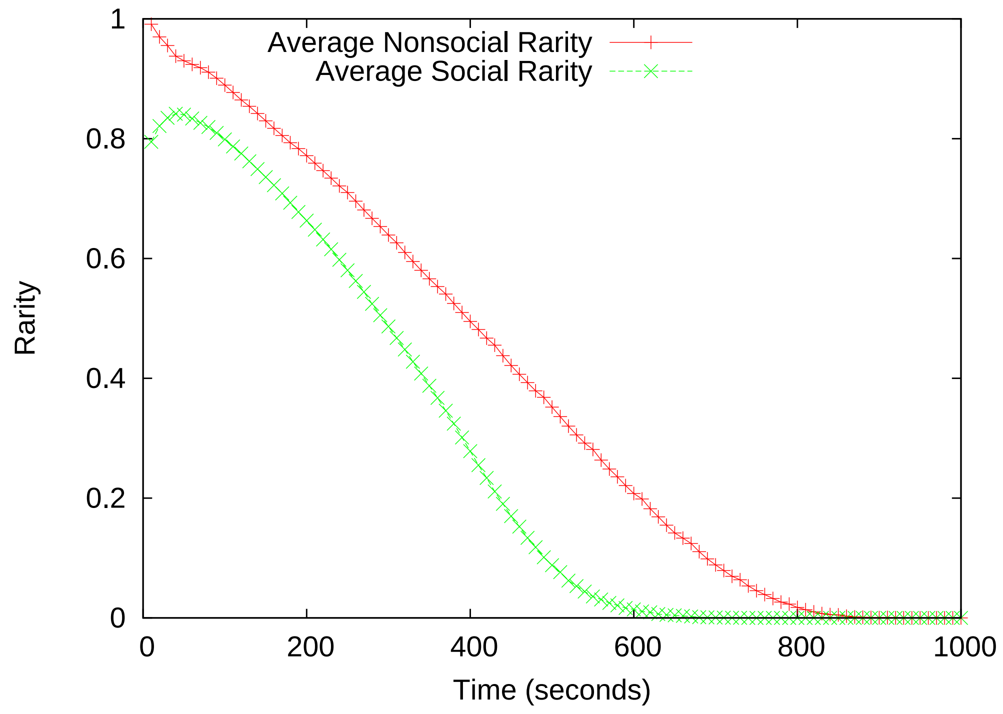


Figure 3.8. Chunk Rarity Reduction Comparison

by other peers, and the new seeds reduce the average rarity of chunks.

3.5.3 Effect of File Size

In order to see the impact of file size on the performance of flash file distribution, we use four different file sizes from 25M to 100M, increased by 25M, as shown in Fig. 3.9. The x-axis represents the file size and the y-axis shows the average peer throughput (KB/s). With an increase of file size from 25MB to 100MB, the performance of standard BitTorrent increased by 4% on average and the performance between social nodes increased by 9% on average.

It is observed that the greatest increase in bandwidth is realized by the most socially connected peers. The 1% of peers in the system with the highest degree of social connectivity realized a 16% increase in performance between a 25MB file and a 100MB file.

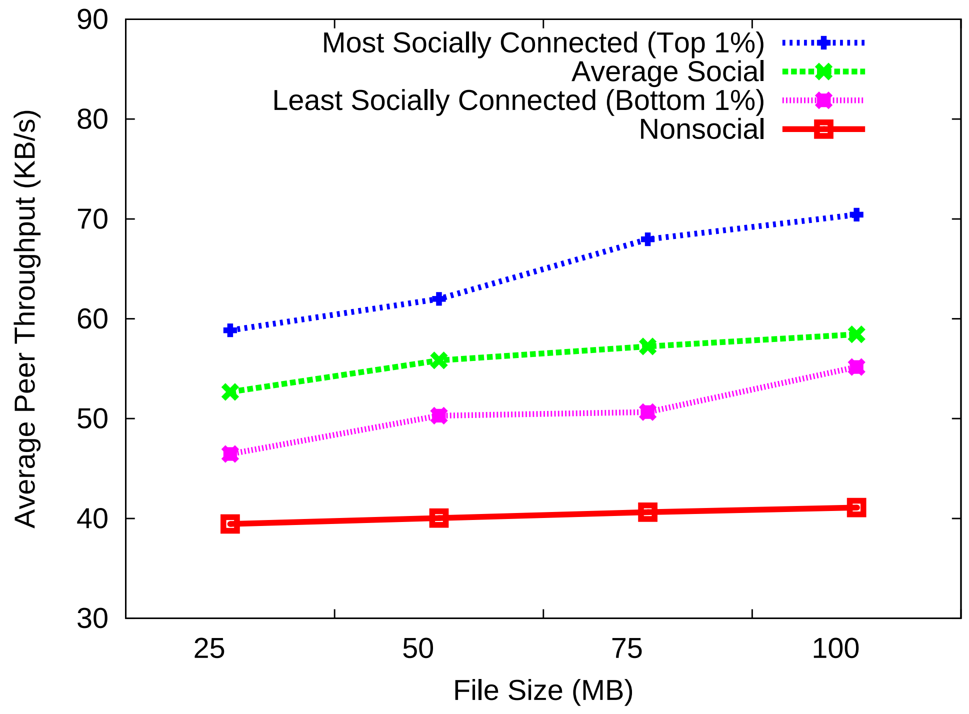


Figure 3.9. Effect of File Size on Peer Throughput

3.5.4 Maximum SocialDistance

Maximum SocialDistance ($D_s Max$) is one of the important parameters in SocialSwarm. By way of review, this is the maximal SocialDistance whereby a peer would consider a peer to be part of its social network. Maximum SocialDistance can thus be considered as a radius from a peer to the perimeter of its social network.

Fig. 3.10 shows the average per peer throughput as maximum SocialDistance is increased. A maximum SocialDistance of 0 is effectively the same as disabling the SocialSwarm protocol. It can be seen that even low maximums of SocialDistance—such as 25—yield considerable improvements in per-client throughput compared with nonsocial clients. As bandwidth utilization improves while increasing the maximum SocialDistance, the percentage of improvement decreases at each step.

3.5.5 Effect of Additional Seed Capacity

Table 3.7 shows the negligible effect of adding a second high bandwidth seed into the system. This reinforces the fact that BitTorrent's performance is much more

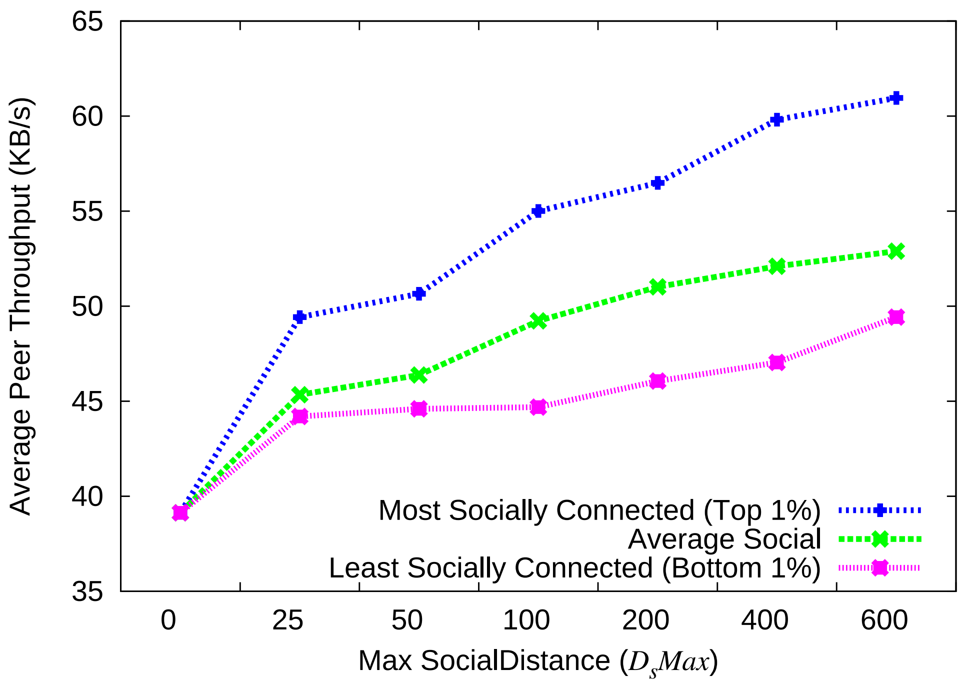


Figure 3.10. Effect of Maximum SocialDistance on Peer Throughput

Table 3.7. Average Download Time and Improvement for Two Seeds

	Nonsocial	Social
1 Seed (sec)	654	491
2 Seeds (sec)	649	486
Improvement (%)	0.6	1.1

dependent on p2p bandwidth and unchoke slot availability than on seed bandwidth.

3.5.6 Effect of Leeches

We conduct several experiments to identify how SocialSwarm compares with standard BitTorrent when faced with varied numbers of nonsocial leeches (additional peers each consuming bandwidth from the system but not contributing reciprocally). We make the assumption that unless they are infected with malware, SocialSwarm-enabled peers will typically behave properly and share their bandwidth resources altruistically with their social peers (and not leech bandwidth from social peers).

As shown in Fig. 3.11, the CDF between social and nonsocial torrent download times follows the same pattern as the baseline 25MB tests (Fig. 3.6). Table 3.8 compares the average download time between the base run of 0 leeches with the download time when 100 and 200 leeches are present, respectively.

Fig. 3.12 shows the relative throughput degradation as number of leeches is increased (from 0 to 100 and from 0 to 200). The throughput percentages in this graph are relative to runs without leeches in the swarm. Thus, although nonsocial swarms have a lower throughput than social swarms, the 0 leech mark in this graph is shown at 100%, representing no performance degradation in comparison with each swarm type's base case. Leeches are intentionally configured with a very small level of upload bandwidth capacity. Leeches are added to the swarm before the 500 peers are started. This is done with the goal of intentionally establishing connections to and consuming bandwidth from the seed before other nodes start. In the case of socially enabled swarms, we assume that although leeches may have social relationships with each other, they have no social relationships to other peers within the swarm. It is clear that nonsocially enabled peers have the greatest performance degradation

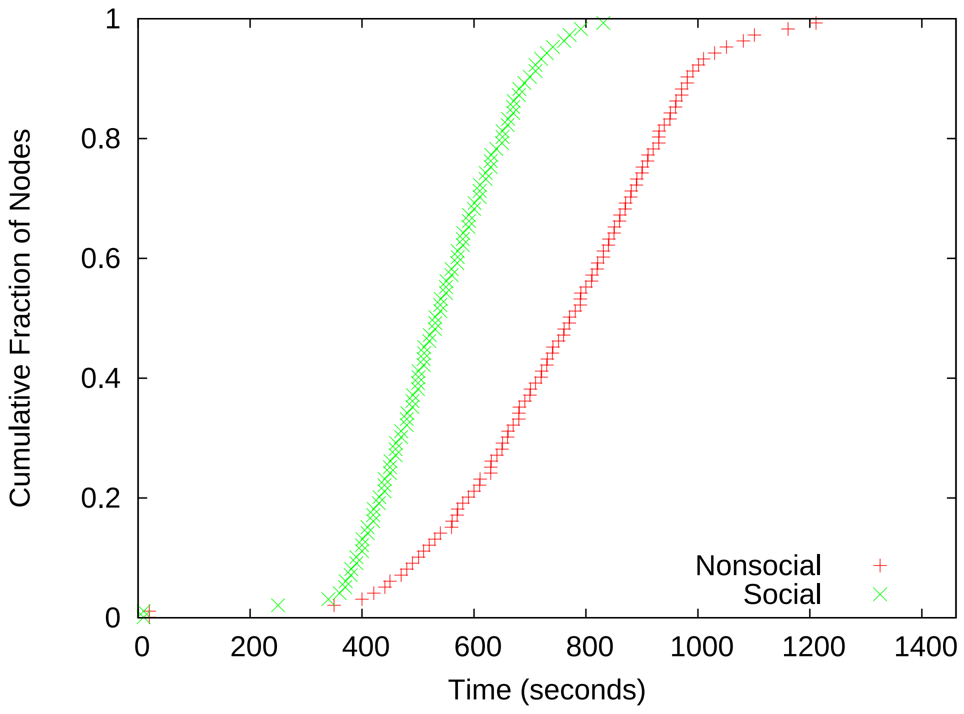


Figure 3.11. Social vs Nonsocial CDF of 25MB File and 100 Leeches

Table 3.8. Average Download Time Based on # of Leeches

Leech Count	Nonsocial Time (sec)	Social Time (sec)
0	658	489
100	755	535
200	870	566

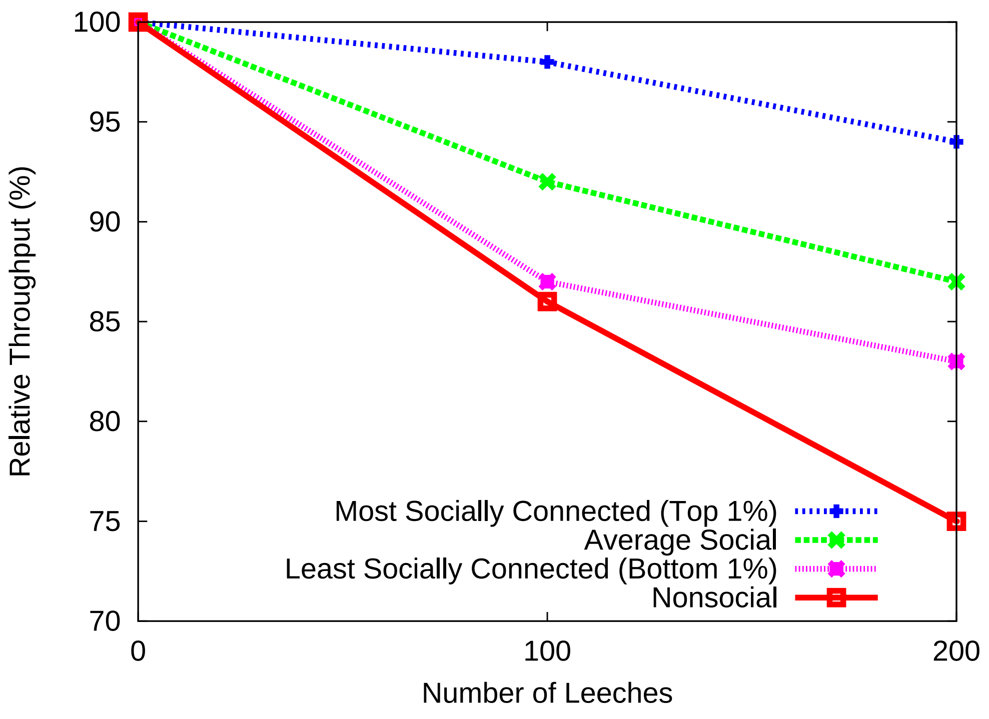


Figure 3.12. Effect of Leeches on Received Bandwidth

(25%) when faced with leeches. The most socially connected peers have the least performance degradation (6%). This performance degradation delta is attributed to the fact that peers with higher levels of social connectivity have a larger number of peers with which they may altruistically share bandwidth. Based on the assumption that social peers are less likely to exhibit malicious behavior than unknown nonsocial peers, SocialSwarm clients target known social peers when deciding those peers with which to establish outgoing and incoming connections. This may increase aversion to leeches. By unchoking a higher number of peers concurrently in comparison with standard BitTorrent, SocialSwarm distributes the upload and download bandwidth used across a larger number of peers, thus diversifying the risk that any individual malicious peer might adversely affect a client's performance.

3.5.7 Bandwidth Contribution and Unchoke Slot Allocation

Fig. 3.13 shows, for a given SocialSwarm, the average percentage of bandwidth used for interacting with nonsocial peers and, stacked on top of that, the additional percentage of bandwidth used for interacting with nonsocial peers. This figure shows that SocialSwarm does not replace interaction with nonsocial peers, but rather increases the percentage of bandwidth utilized on each peer. Fig. 3.13 also shows the average number of social peers a node will unchoke over time. An offer of bandwidth to a social peer in no way guarantees that the bandwidth will actually be used by the offer recipient. The number of social peers that actively use the offered bandwidth is lower than the number of nodes, also shown in Fig. 3.13, that are offered bandwidth. Clearly, out of the total number of social peers a node might have—as shown in Fig 3.2—only a very small percentage of those peers would need to be online to allow SocialSwarm to be effective.

3.6 Conclusions and Future Work

In this paper, we introduced SocialSwarm, a novel approach to flash file dissemination that exploits SocialDistance, which we extract from altruism between social peers, so as to relax the required, but inefficient, reservation of bandwidth for resource bartering in BitTorrent. We implemented SocialSwarm as an extension

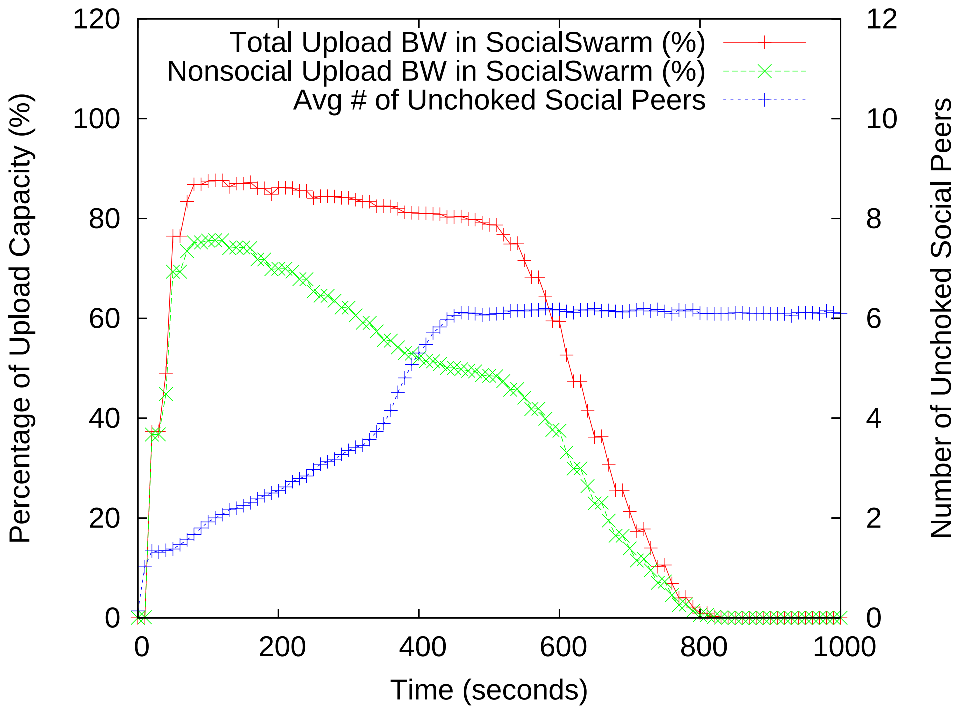


Figure 3.13. Bandwidth Allocation and Social Unchokes

to the libtorrent library, applied a social network topology and interaction history obtained from Facebook, and evaluated it on a testbed of 500 independent virtual clients. We showed that SocialSwarm reduces average file download time by 25% to 35% compared to that of standard BitTorrent under varied conditions—file sizes, max SocialDistance, and leech and seed counts.

In the future, we will investigate the effect of socially enabled leeches on SocialSwarm. Given that malicious code commonly uses social networks for propagation, clusters of social peers have the possibility of becoming infected. Our future work will also include finding a dynamic way to modify peer SocialDistance/altruism levels based on observed behavior between individual peers as well as among clusters of social peers.

SocialSwarm is a single application that was designed and implemented to directly use SocialDistance. Developing social relationship analysis and peer identification into individual applications such as SocialSwarm is not as efficient as leveraging a common framework for such functions. In the next chapter, we evaluate F3DS by using it to implement F3AV—a novel social network based distributed malware detection system. The results of our evaluation of F3AV are also found in the next chapter.

CHAPTER 4

LEVERAGING F3DS FOR DISTRIBUTED MALWARE DETECTION IN F3AV: F3DS ANTIVIRUS

4.1 Introduction

To demonstrate the applicability and value of F3DS, we leverage it to design, implement, and evaluate F3AV (F3DS Antivirus), a novel N-version distributed malware detection system. F3AV provides collaborative malware detection among social peers with the greatest levels or protection being provided to security-critical social hubs [17]—the most social users on the network, which are the most likely to propagate large quantities of malware when infected by socially aware malware[1]. Using F3AV we present and evaluate a novel method for varying the required diversity of virus scanners based on the age of the object being scanned so as to achieve a balance between high rates of malware detection and object scanning latency. Our evaluation using the Amazon Cloud shows that by concurrently leveraging diverse scanners across a social network, a user can achieve a 65% increase in the detection rate of 0- to 1-day-old malware as compared to the average detection rate of individual scanners. Our implementation of F3AV on top of F3DS is publicly available on the F3DS web site [18].

The rest of this chapter is organized as follows. Section 4.2 provides an overview of the F3AV design. The implementation of F3AV is detailed in Section 4.3. We evaluate F3AV using the Amazon EC2 cloud and provide the results in Section 4.4. Section 4.5

[1]Malware that propagates itself over social networks

gives an overview of related work. We provide our conclusions and delineate candidate areas of future work in Section 4.6.

4.2 F3AV Malware Detection System

Although F3DS can be used for applications in a variety of categories, we choose to demonstrate F3DS by applying it to malware detection, one of the key areas in the critical and challenging field of distributed system security. We build upon and contribute to the body of research on malware detection by enabling collaboration among social peers to detect malware.

4.2.1 The Challenge of Socially Aware Malware

Current malware is commonly designed to exploit existing altruism among social peers for malicious purposes. For example, malware running on compromised nodes uses altruism on social networks for self propagation in order to exploit CPU, memory, and bandwidth resources of social network participants. The creators of malware, recognizing the rise in popularity of social applications, have continuously tuned Botnet malware propagation and identity hijacking mechanisms to exploit trust among social network users. For instance, Facebook users are more willing to open a message with malicious links if that message was sent from a compromised account of a social peer. Users commonly are less vigilant with security while interacting with social peers. Security experts have claimed [55] [56] that cyber-espionage and social networking attacks are the top cyber security issues of 2012. The growth of malicious software that propagates overs social links has prompted security researchers to study methods for protecting social hubs. Zhu et al. found [17] that when new malicious code enters a social network, it commonly infects hub nodes—nodes with higher degrees of connectivity and malware exposure—more quickly than those users with relatively few social peers. Securing systems against malware—especially social hubs—have become critical areas of security research.

4.2.2 Motivation for F3AV

With their work on Cloud-AV [57], Oberheide et. al have shown the benefits of utilizing a cluster of servers running a heterogeneous set of antivirus software (with their respective signature sets). This is referred to as N-version virus scanning. The authors show that although the antivirus signature sets used by diverse vendors intersect, there is no single signature set that can effectively act as a superset of all other vendor signatures. Individual AV (antivirus) products have the potential to be compromised [58]. This provides motivation for using divergent signature sets and AV products to scan for malware. Individual home and small office users rarely have the budget or technical expertise to construct and maintain a cluster of N-version virus scanners, but users commonly have social peers that are willing to share resources (CPU, memory, bandwidth, etc). In many cases, a user's social peers run diverse antivirus scan engines and signature sets.

F3DS provides a means by which users can effectively share their resources and service requests from peers. We choose to evaluate the benefits of F3DS by implementing F3AV, an N-version distributed virus scanning application. An overview of F3AV is shown in Fig. 4.1. F3AV provides two important services: first, passive sharing

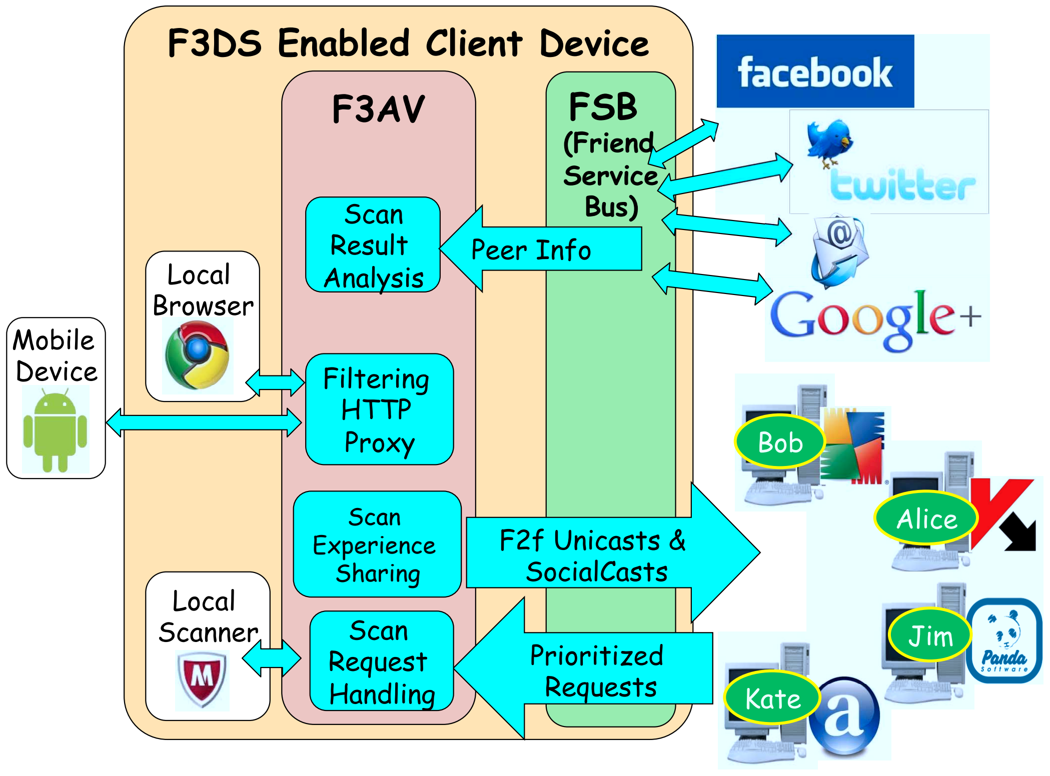


Figure 4.1. F3AV Overview

of the result of object scans among social peers; and second, providing a medium by which social peers can request immediate scans of particular objects; F3AV has the goal of improving malware detection accuracy while minimizing additional object scanning latency as described below.

4.2.3 Malware Detection Accuracy

By using the FSB services of F3DS, F3AV is designed to improve a node's accuracy in detecting malware from objects requested by the user. Utilizing the same N-version philosophy of the Cloud-AV [57] project, we design F3AV to increase the malware detection accuracy of individual nodes by encouraging the sharing of scan results among social peers running diverse virus scanners. With the level of resource sharing—i.e., servicing scan requests and sharing results—governed by levels of altruism among social peers, the FSB allows F3AV to maximize the antivirus resources of social-hubs. As the recipients of larger amounts of scanning resources, social-hubs will be armed with the highest levels of protection against malware.

4.2.4 Object Scanning Latency

By using the FSB services of F3DS, we design F3AV to minimize any latency penalty created by relying on social peers to assist with malware detection. In contrast with Cloud-AV, which relies on a dedicated local cluster of scanning hardware and software, F3AV relies on the sharing of surplus memory, bandwidth, and CPU resources across geographically dispersed nodes. Given the unpredictable availability of social peer resources as well as the lack of guarantees on the level of willingness of a peer to service scan requests, we design F3AV with a no-assurances approach as to scan request response times from peers.

4.2.5 Active Scan Requests

F3AV places all of its scanning logic into a central module called DecisionHandler. This module can be replaced or tuned given the preferences of the system user. The DecisionHandler module is able to use the FSB to promulgate scan requests by SocialCast to all social peers within a given Social Distance or to send unicast scan requests to specific peers. When a prioritized request is received from the FSB of F3DS, F3AV will service that request and send a response message to the requester containing the scan results, information on the scan vendor, and the date/time stamp of the scanner signature set used.

4.2.6 Passive Scan Experience Sharing

Over time, a node will accumulate scan results from both local user requests as well as active scan requests from peers. Scan results are stored in a local database table of scans. A node will also store the results in a quickly searchable hash table known as a Scan Log. The hash table key for one entry in the Scan Log is the SHA256 hash of: (URL + log creator ID + a nonce for the Scan Log). The hash table key for a second entry in the Scan Log is the SHA256 hash of: (file contents + log creator ID + a nonce for the Scan Log). The value for both entries in the hash table is the scan result with a 0 indicating the object was found to be benign and a 1 indicating the object was found to be malicious. The nonce associated with a Scan Log is part of the Scan Log metadata that is sent to peers along with the Scan Log. Peers thus can look up arbitrary URLs and objects in the Scan Log using the same nonce.

Both of these keys are also added to a Bloom Filter known as a Scan Digest. The Scan Digest contains an entry if the object has been scanned and the result stored in the Scan Log. The motivation for using a Bloom Filter to preshare information on URLs comes from Cache Digests [59] [60] which are widely used among caching HTTP proxy servers to identify peer proxy server contents. As with Cache Digests, Scan Digests are small in size compared with a full list of URLs such as the Scan Log. Scan Digests are small enough to reside in ram, whereas larger Scan Logs would typically reside on disk. Given the possibility of Bloom Filter collisions, Scan Digests may contain false positives, Scan Digests are always used in conjunction with Scan Logs. Scan Digests may be viewed as a manifest of the Scan Log keys with lossy compression.

We now describe the method used by F3AV for Scan Digest and Scan Log rotation, sharing, and eviction. Scan Digests by their nature do not allow existing entries to be deleted. Individual entries within a Bloom Filter do not contain semantic data about when the scans were conducted, what scan engine was used, and the date of the signature set used. We design F3AV Scan Digests to include scanner vendor and signature set information as accompanying metadata. Thus, Scan Digests and Scan Logs must be rotated at least each time these metadata change (e.g., when the scanner signature set is updated). To maintain a steady flow of new information on objects to social peers, Scan Digests and Scan Logs may be rotated several times each day. When Scan Digests and Scan Logs are rotated, the current "active write" pair of a Scan Digest and a Scan Log are closed and a new Scan Digest and a new Scan Log are opened for writing.

At the time of rotation, F3AV places the Scan Log in a hidden (retrievable but not searchable) location on a locally running web service provided by the FSB. F3AV then uses the FSB to send a Socialcast to trusted peers with the following pieces of information:

- The URL of the new Scan Log
- The Scan Digest associated with the Scan Log
- Metadata such as the scan engine that was used, the signature set that was used, and the date/time of the rotation creation.

Peers that receive this notice—over their FSB—retrieve the associated Scan Log when and if their F3AV instance so chooses. F3AV appraises the potential value of a peer's announced Scan Log by evaluating the perceived Social Distance to the creator of the Scan Log with Scan Logs generated by closer peers being considered as more reliable and valuable. Peers may also compare their own recent access history with the contents of the Scan Digest. Should the peer have a close-enough perceived Social Distance and should the Scan Digest indicate a sufficient level of correlated access history, F3AV will retrieve the full Scan Log associated with that Scan Digest.

When F3AV is queried to evaluate whether an object is malicious or not, it will first identify any scanning experiences for the object that it has already received from its social peers in the way of Scan Logs. Scan Digests are sufficiently small to be cached in ram, whereas larger Scan Logs are more suitably stored on slower and lower cost media. For this reason, F3AV will first perform a lightweight search of each memory-resident Scan Digest looking for the particular URL. If the URL is found in a particular Scan Digest, then F3AV will proceed to search for an entry in the disk-based Scan Log. Given the basic nature of hashtables, the Scan Log search is a O(log n) operation. If a value is found during the look up, then this value is the scan result claimed by the peer who performed the scanning. Scan result values that are found are passed to the DecisionHandler (described in Section 4.2.7). If no value is found during the hash table look up, then a Bloom Filter collision has occurred in the Scan Digest and no result is considered. The process of creating, sharing, and using Scan Digests and Scan Logs can be seen in Fig. 4.2.

Over time, a node will accumulate a variety of different Scan Digests and Scan Logs from both close and distant social peers. Each Scan Digest and Scan Log will eventually need to be evicted. To assist in Scan Digest and Scan Log eviction, F3AV maintains a maximum age limit for both Scan Digests and Scan Logs. Once a Scan Digest and Scan Log pair exceeds this age, the pair is automatically evicted. F3AV also keeps an EWMA utility rating for each Scan Digest and Scan Log based on the number of objects of interest that were found in a particular Scan Digest and Scan Log over a given time period. Scan Digests and Scan Logs with the lowest utility rating (those with the fewest objects of interest to the evaluator) are chosen as candidates for eviction, should disk- or ram-cache capacity limits mandate evictions. Eviction based on a utility rating will automatically bias the F3AV cache towards retaining objects from neighbors who are accessing similar content—those with correlated object-access behaviors. The greater the correlation of the objects accessed among peers the higher the probability will be that Scan Digests and Scan Logs received by a node from its peers will already contain scan results of value to that node.

4.2.7 Modular Scanning Logic

Fig. 4.3 shows the states used when an object request is passed through F3AV. At the core of F3AV is a DecisionHandler module that contains the majority of the logic around how much data are collected to make a decision as well as the result of the decision. F3AV provides the DecisionHandler module all of the available data on the particular object that is being requested, including all result records of previous local

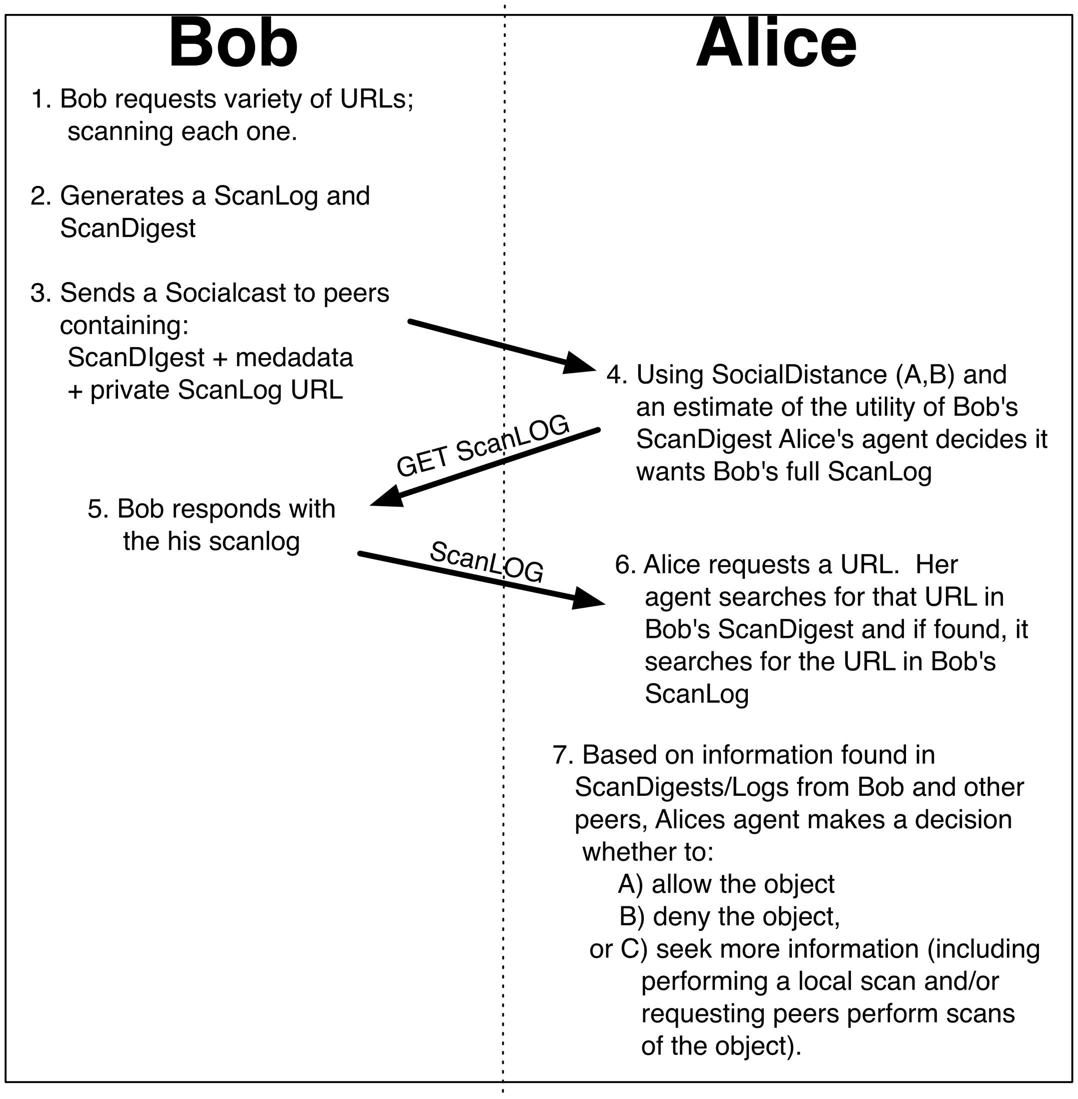

Bob
Alice
1. Bob requests variety of URLs; scanning each one.
2. Generates a ScanLog and ScanDigest
3. Sends a Socialcast to peers containing: ScanDIgest + medadata + private ScanLog URL
4. Using SocialDistance (A,B) and an estimate of the utility of Bob's ScanDigest Alice's agent decides it wants Bob's full ScanLog
GET ScanLOG
5. Bob responds with the his scanlog
ScanLOG
6. Alice requests a URL. Her agent searches for that URL in Bob's ScanDigest and if found, it searches for the URL in Bob's ScanLog
7. Based on information found in ScanDigests/Logs from Bob and other peers, Alices agent makes a decision whether to:
A) allow the object
B) deny the object,
or C) seek more information (including performing a local scan and/or requesting peers perform scans of the object).

Figure 4.2. F3AV Experience Sharing

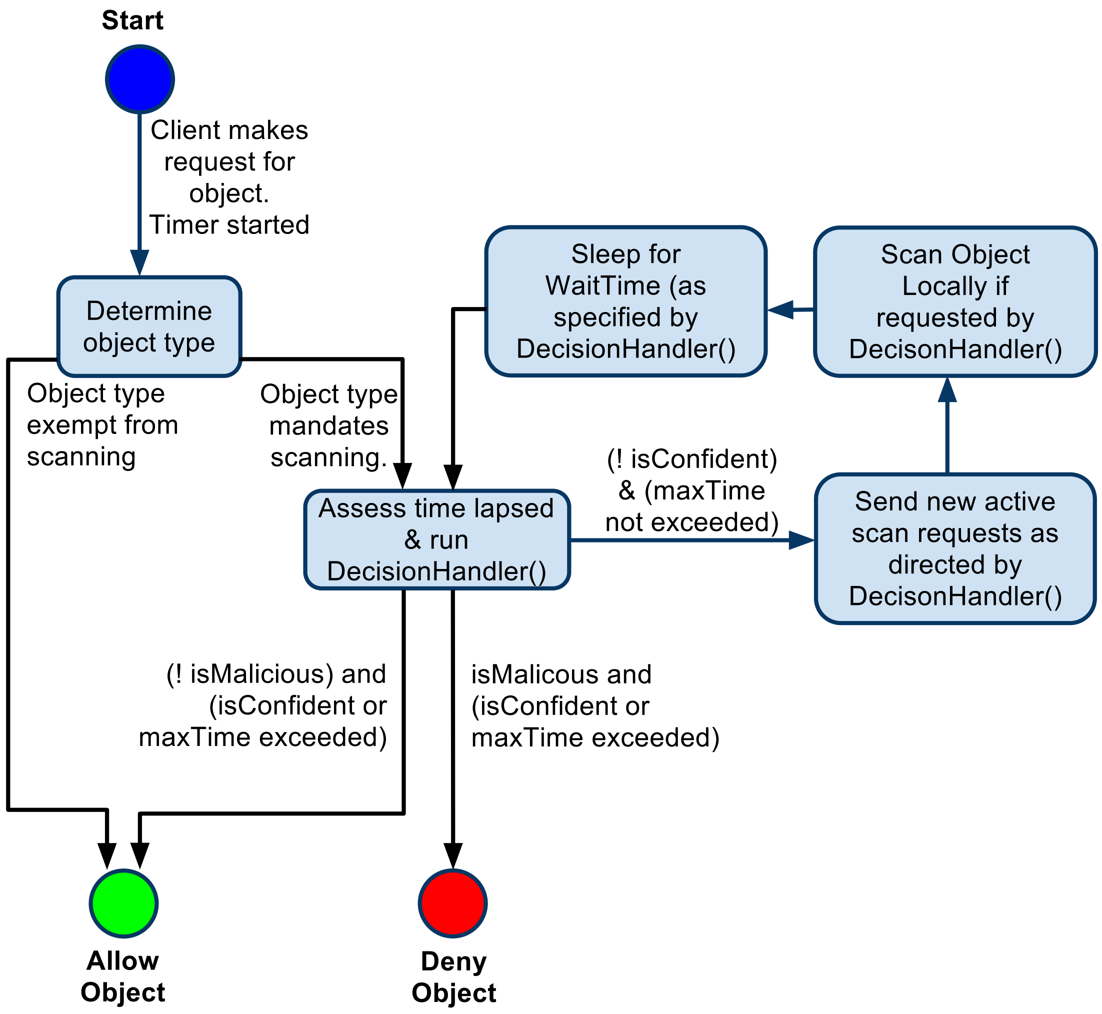

Start
Client makes request for object. Timer started
Determine object type
Object type exempt from scanning
Object type mandates scanning.
Sleep for WaitTime (as specified by DecisionHandler()
Scan Object Locally if requested by DecisonHandler()
Assess time lapsed & run DecisionHandler()
(! isConfident) & (maxTime not exceeded)
Send new active scan requests as directed by DecisonHandler()
(! isMalicious) and (isConfident or maxTime exceeded)
isMalicous and (isConfident or maxTime exceeded)
Allow Object
Deny Object

Figure 4.3. F3AV State

scans, peer scans found in Scan Logs, as well as peer scans obtained from directed scan requests. Along with each of the scan records, the DecisionHandler is provided with the scan engine name and date of signature set that created the scan results. The DecisionHandler returns the six data structures shown in Table 4.1.

DecisionHandler is thus provided with all possible information on an object and has full control over the behavior of F3AV. DecisionHandler can decide to make an immediate decision ($isConfident == True$) and either accept or reject the object. Alternatively DecisionHandler can decide to seek more information and specify a list of peers or a socialcast distance to send out active scan requests. There is a maximum decision delay time, however, and if the DecisionHandler call loop exceeds this maximum time, then the latest return value of $isMalcious$ is used to either accept or reject the object.

4.2.8 Privacy Considerations

The target user set for F3AV are those who have an existing willingness to relax their privacy requirements when collaborating with trusted and monitored social peers. The two primary areas of concern for privacy for F3AV users are those of

Table 4.1. F3AV Parameters

PARAMETER	PURPOSE
isConfident	Boolean indicating whether or not DecisionHandler is confident in its decision. If it is not confident yet in its decision, then it will loop and wait for additional data to be collected.
isMalicious	Boolean indicating the current decision by DecisionHandler (True=Malicious, False=Benign)
WaitTime	Number of uSeconds to sleep before calling DecisionHandler again if Confident ==False
doLocalScan	Boolean indicating whether a local scan should be performed on the object
ScanRequestRadius	Maximum Social Distance to which F3AV should send directed scan requests (via a FSB-based SocialCast)
ScanRequestPeers	List of specific Social Peers to which should send directed scan requests (via FSB-based unicast)

clickstream (object access sequences) and antivirus (vendor/version) privacy. There are a variety of potential methods—such as anonymous group multicast [61] among social peers—which F3DS could offer to applications to help address these concerns. We plan on evaluating such methods as part of our future work.

4.3 F3AV Implementation

In this section, we describe our F3AV implementation on top of F3DS. We publish our implementation of F3AV on the F3DS web site [18].

4.3.1 Messaging

F3AV uses the f2f messaging provided by the FSB of F3DS for active scan requests and responses, scan result confirmation requests and responses, as well as Scan Digest messages. To distribute Scan Digests, F3AV uses the FSB to send a socialcast containing the Scan Digest, the URL of the associated Scan Log, and the metadata for the Scan Digest. F3AV obtains information on the scan engines and signature sets used by peers from the Scan Digest metadata received from each peer.

4.3.2 DecisionHandler Modules

For the purpose of our evaluation, we create three interchangeable scanning logic (DecisionHandler) modules: local-only scanning, paranoid, and dynamic. We now describe each of these modules:

First, the local-only F3AV DecisionHandler behaves like a traditional single scanner client. This module is designed only for comparison with other DecisionHandler modules.

Second, the Paranoid F3AV DecisionHandler requires virus scan results from a tunable minimum number of diverse scanners—within a tunable social radius. Each of those scanners must have been updated within specific recent windows of time to be considered valid. Lack of sufficient result diversity as well as any single positive (malicious) result induces the Paranoid F3AV DecisionHandler module to block the object being requested. Intuitively, this module will likely yield the highest malware detection rates but will also require the greatest latency to certify benign objects.

Third, the Dynamic F3AV DecisionHandler module adjusts the required diversity of scanners based on the age of the object. The results of our experimentation as well as those of other researchers [57] show that as malicious objects age, their probability of being detected by one or more virus scanners increases. Protecting systems against recent malware can also be challenging given that new malware commonly attempts to exploit newly exposed and not-yet-patched software vulnerabilities. Software vendors attempt to patch such vulnerabilities quickly [62], but windows of exposure frequently exist. Although it is impossible to certify the age of objects from arbitrary sites on the Internet, there are certain software-as-a-service (SaaS) sites which may be considered trustworthy keepers of user submitted object upload and modification times. With the assumption that an object's age can be assessed with a high level of accuracy from a list of trusted sites via the $last-modified$ header of HTTP, the Dynamic F3AV DecisionHandler module dynamically adjusts the required diversity of scanners based on an object's age. Using the $last-modified$ HTTP header to determine the minimum age of an abject implies that the security of a SaaS application can be trusted to prevent date/time stamp modifications by application users of user submitted content.

4.3.3 AV Local Scan Handlers

An F3AV module known as the local scan handler is responsible for interacting with the antivirus scanner installed on the machine. The local scan handler retrieves a URL, scans it with a specific antivirus engine, and returns the result to F3AV. The local scan handler also returns information on the virus signature set used in the scanning. For our F3AV implementation, we implement scan handler modules for antivirus packages from the following six vendors: AVG, Avast, Microsoft, Clamwin, Avira, and Kaspersky. Based on existing research [63] on worldwide market share of scanning engines, we believe this set of engines represents around 65% of the global install base of antivirus software.

4.3.4 Browser Request Filtering

To maximize compatibility with a variety of desktop and mobile user agents (browsers), we implement an F3AV module for the squid caching http proxy server.

Applications that make outgoing http requests via squid automatically receive the scanning benefits of F3AV. As part of the implementation, we confirm compatibility with three locally installed browsers (IE, Firefox, and Chrome) as well as three mobile browsers (Opera, Firefox, and Android Browser) by configuring the mobile device to proxy http connections through an F3AV enabled squid instance.

4.4 Evaluation

In this section, we evaluate F3AV via both experimentation and simulation. We also evaluate the effectiveness of F3AV at providing the most protection to Social Hubs to reduce the probability they will become infected and serve as valuable launch locations for malware propagation. Due to limited space, any experimentation details—including specific parameters used for each of the various various experiment and simulation runs—are deferred to a longer version of this paper.

4.4.1 Malware Detection Accuracy

Using experimentation, we evaluate F3AV against its design goal of increasing malware detection accuracy by exploiting the functionality of F3DS to allow peers access to diverse scanning of objects. Given that the ratio of false negatives to false positives in signature-based commercial virus scanners is several orders of magnitude, our current experiments focus on rates of false negatives; we evaluate the detection rate of objects that have been verified to be malware (false negatives).

4.4.1.1 Malware repository. Malware is continuously changing and adapting in attempts to avoid security software signature sets. The authors of Cloud-AV have shown through experimentation that the older the piece of malware, the higher the probability is that it will be caught by a given antivirus signature set. For the purpose of evaluating F3AV, we choose to collect a large body of real malware from the Internet [64] and update this collection daily with newly discovered malicious code. We index each malware object using the SHA256 hash of the object's contents. Using a hash of each malware object's contents allows us to ensure that we are only storing a single copy of each given malware sample, even if that object is found in multiple locations on the Internet. Over the course of 150 days spanning from late

2011 into spring of 2012, we collected over 215,000 unique pieces of live malware from the Internet (more than 60GB of contents).

4.4.1.2 Local vs F3AV. Using the local DecisionHandler, we run each of six individual scanners through a random selection of approximately 3000 pieces of malware. We find similar results as found by Oberheide et al. [57] in that older objects had a higher probability of being detected by malware scanners. Fig. 4.4 shows the individual scanner results in relation to the age of the malware object being scanned as well as a weighted mean for those scans, with the weight based on the proportional global market share [63] of each scanner. The weighted mean provides a strong indication of the total effectiveness of populations at detecting malware based on the age of the object scanned. Fig. 4.4 also shows the results of the Paranoid and the Dynamic DecisionHandlers in F3AV. The Paranoid DecisionHandler uses collaboration with social peers and in this experiment requires responses from each of the six diverse scanners before a decision to allow the object is made. The Dynamic DecisionHandler also uses collaboration with social peers but varies the number of required responses based on the age of the object being scanned. For 0- to 1-day-old malware, this figure highlights a 44% to 74% change (a 65% increase) in the malware detection accuracy when using the Paranoid and Dynamic Decision handlers in comparison with the market-share average of individual scan results.

4.4.2 Object Scanning Latency

Even though diverse scans across peers occur in parallel, the aggregation of results in F3AV requires additional latency for communication overhead. We compare the mean, max, and average latency required across the local scanners with those required by the Paranoid DecisionHandlers. We also test at three different levels of a priori object access and Scan Digest/Log sharing (locality) by peers: 0, 50, and 100%. At the 100% level, each of the peers has already accessed an object, scanned it, and shared the result with the inquiring peer via passive Scan Digest/Log sharing. At the 50% level, half of the URLs have previously been scanned with their results shared and the other half are new/distinct to all of the peers in the system.

Fig. 4.5 shows that the Paranoid DecisionHandler at 0% object access/scan locality

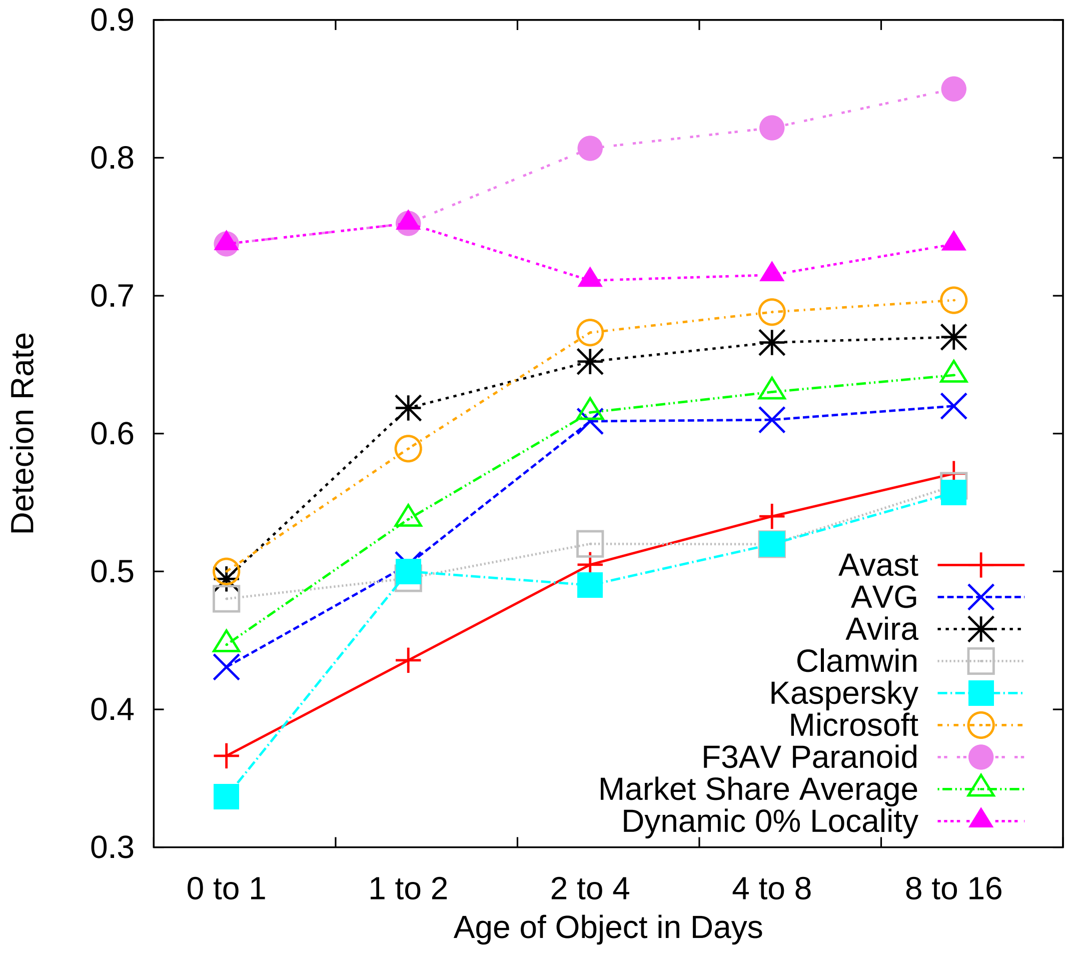


Figure 4.4. Local vs F3AV

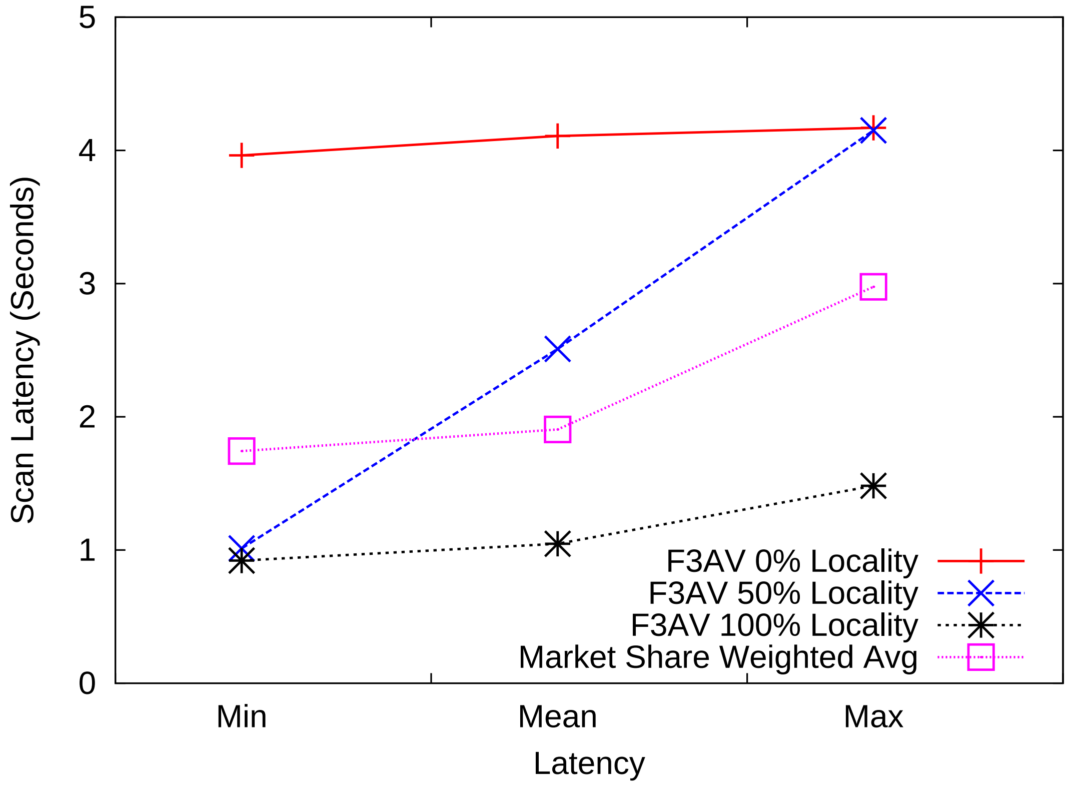


Figure 4.5. Latency vs Object Access Locality

requires the most time due to the real-time communication and collaboration that must happen between the different nodes. With 100% object access/scan locality, the Paranoid DecisionHandler clearly outperforms even the local scans due to the fact that with 100% object access/scan locality, F3AV is able to avoid local scans by relying exclusively on previously collected information from peers. For the purpose of this evaluation, we only provide the extremes and do not attempt to approximate or predict levels of object access locality among peers within real-world social networks.

Fig. 4.6 shows that the Dynamic DecisionHandler requires the greatest scan times for the newest objects. As objects age, the Dynamic DecisionHandler decreases the scanner diversity that it requires and thus, the time required to scan objects

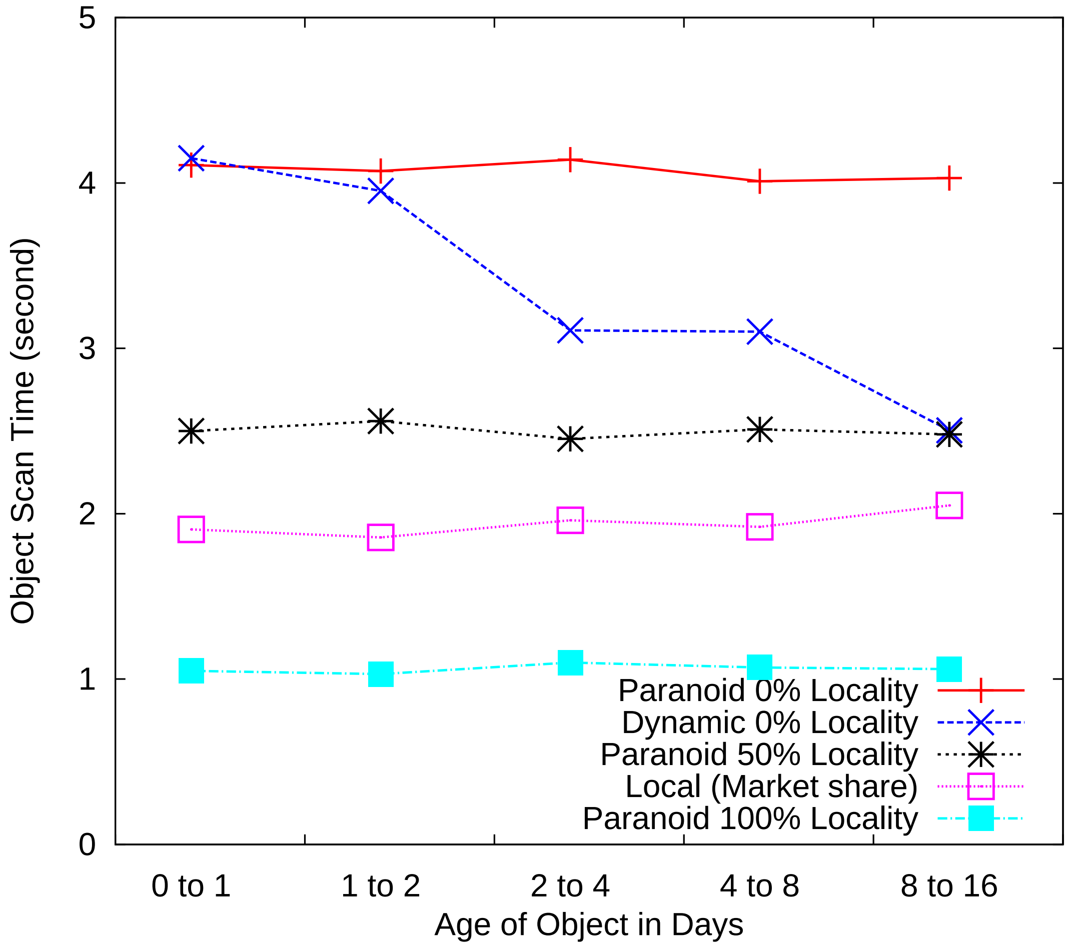


Figure 4.6. Latency vs Object Age

decreases. Though our current Dynamic DecisionHandler experiments only use a 0% object access locality, we expect that testing of greater than 0% object access locality would reduce the scan times proportionally with those results from the Paranoid DecisionHandler because of the passive scan experience caching and sharing within F3AV.

4.4.3 Scanner Availability and Diversity

We evaluate the effectiveness of F3AV at providing significant scanner diversity to all F3AV participants. We also measure the effective of F3AV at delivering the highest level of protection to Social Hubs to reduce the probability they will become infected and subsequently serve as valuable launch locations for social malware propagation.

Our implementation of F3AV supports social relationship analysis modules for Twitter, Facebook, and email. To evaluate the potential benefit real social network users would have when using F3AV, we conduct simulations by using real-world social connectivity metadata extracted from Facebook. We take an anonymized data set of real-world interactions—wall postings—from a network of 5,000 Facebook users [49] and feed that data through the Facebook peer relationship analysis module in F3DS to identify the nominal SocialDistance between each pair of social peers. We repeat several of our simulations while varying the Social Net Radius—the maximum SocialDistance whereby a peer will consider another node part of its social network and be willing to share resources with that node.

4.4.3.1 Diverse scanner availability. The social relationships derived from Facebook are fed into an F3AV simulator. We assign each of the 5,000 social peers a social identity. For each Social Net Radius tested, we perform 100 simulation runs. On each simulation run, we randomly assign each of the 5,000 social peers an antimalware scanner from a pool of 21 candidate scanners based on the proportional world-wide market share of each scanner. These 21 scanners represent 80% of the global market share for antimalware software. In our simulations, we do not consider the geographic distribution of social network users or variances in regional market share of antimalware software; we leave such considerations to future work.

After assigning each social peer an antimalware scanner, we then evaluate the level of scanner diversity that exists within each peers social network and extract the mean diversity level for each peer across the 100 simulation runs. Fig. 4.7 provides a CDF for each of the tested Social Net Radius values and shows that even for smaller Social Net Radius values, most peers have access to significant diversity of scanners across their social peers. These findings also imply that even if a significant portion of a user's social peers were offline, unwilling, or unable to share resources, there would still be sufficient peers online and available to supply a high diversity of scanners.

4.4.3.2 SocialHub protection – scanner diversity. Fig. 4.8 compares the level of scanner diversity available to three different sub groups of nodes: The top 250 nodes (95th percentile), the median 250 nodes, and the bottom 250 nodes (5th percentile) with respect to social connectivity. Fig. 4.8 clearly shows that F3AV participants offer the highest scanner diversity to the most socially connected nodes.

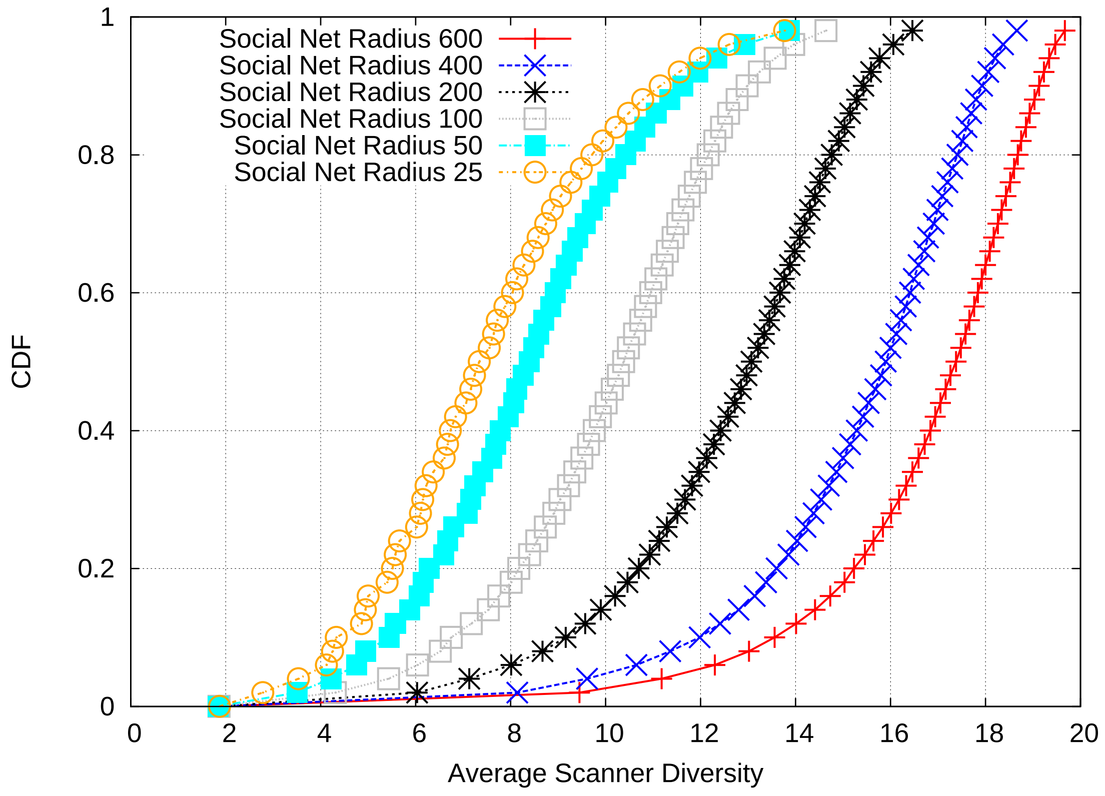


Figure 4.7. Scanner Diversity Availability by Social Radius

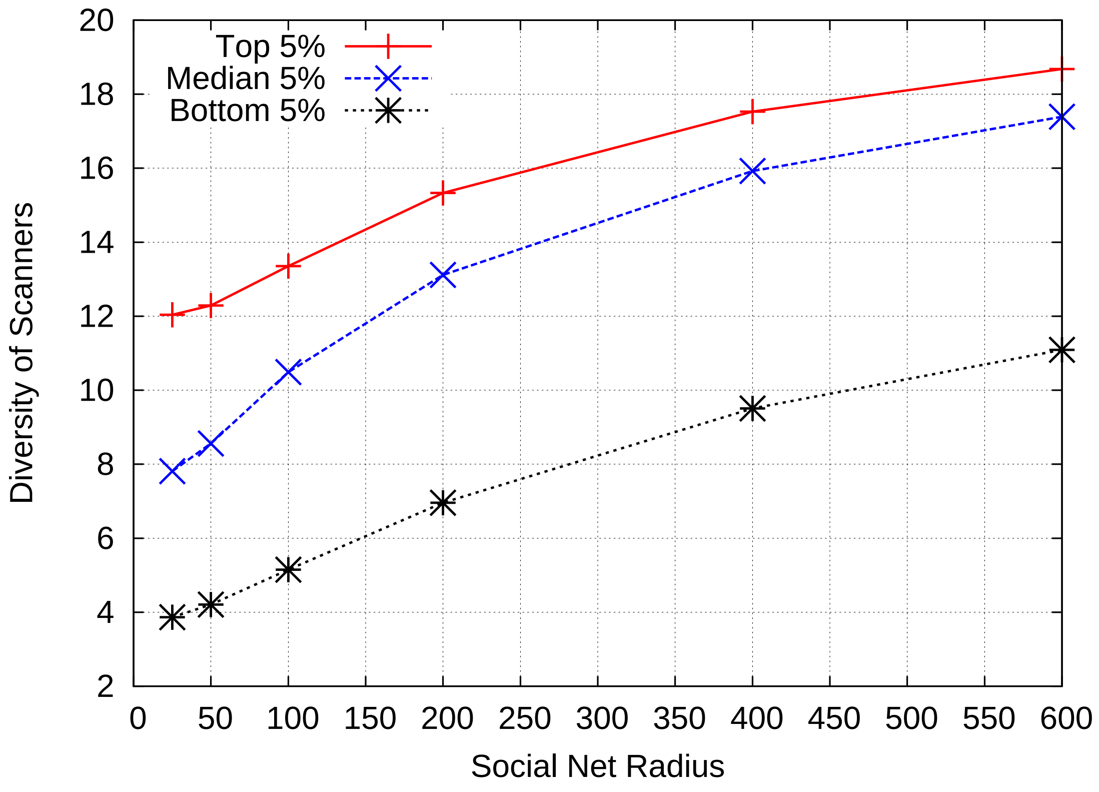


Figure 4.8. Scanner Diversity Availability by Social Connectivity

4.4.3.3 SocialHub protection – Scan Digests/Logs. We also run a simulation whereby every hour, each peer in the system offers a single Scan Digest/Log pair to one of its peers. The Scan Digest/Log offers are probabilistically granted based on the proportion of altruism a peer has towards each of its peers. We run the simulation for 500 virtual hours and sum up the total Scan Digest/Logs received by each node. Fig. 4.9 shows three CDF lines—one for each of the same 95th percentile, median, and 5th percentile 250 node peer sets used in the previous simulation. This figure clearly shows that the most socially connected peers are the recipients of the Scan Digest/Log offers and thus, they have access to more cached scan results in comparison with peers that have a lower social connectivity.

These results show that, as designed, F3AV properly prioritizes allocation of resources—including scan results sharing and availability of diverse scanners—among social network participants with social hubs receiving the greatest benefit (and thus protection) from F3AV.

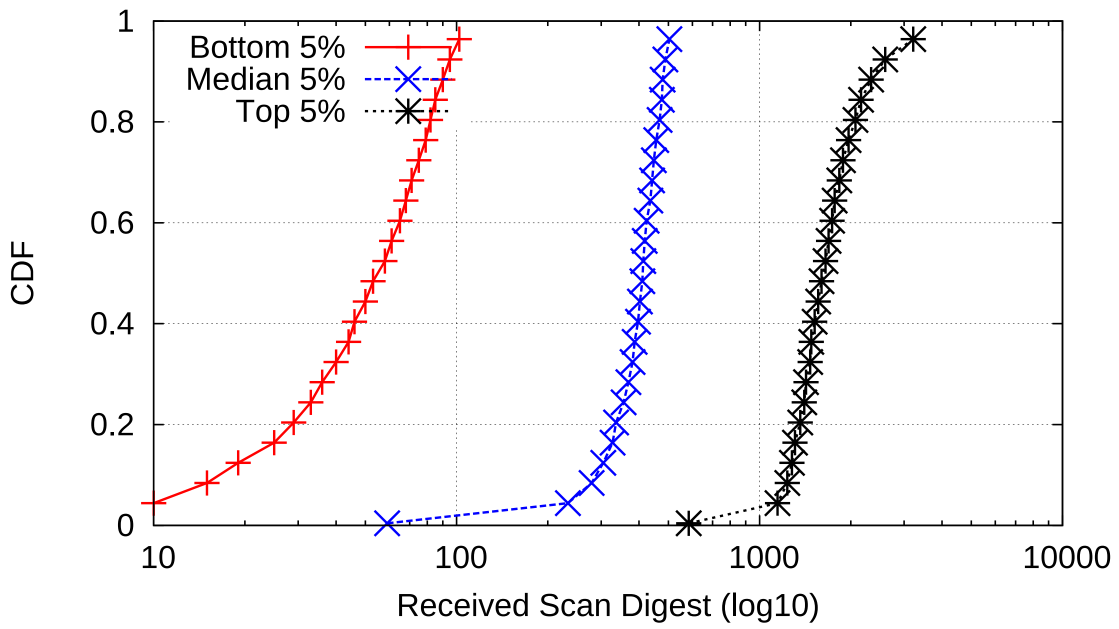


Figure 4.9. Scan Digest/Log offers by Social Connectivity

4.5 Related Work

For malware detection and isolation on distributed devices, researchers have proposed a variety [65] [66] [67] [68] [69] of methods based around the basic principle of workload offloading to a cloud-based service. F3AV is complementary to these approaches in that it provides a decentralized service based on existing resources among social peers for servicing scanning requests.

4.6 Conclusions and Future Work

To demonstrate the value of F3DS, we designed and implemented a novel malware detection application—F3AV (F3DS antivirus)—on top of F3DS and found that with f2f sharing of resources F3AV significantly enhances the ability of social peers to detect malware. We also showed that F3AV provides greatly enhanced malware protection to social hubs by ensuring that they receive the most Scan Digest/Log offers and have access to the greatest diversity of scanner engines.

Other areas of potential future research include enhancing F3AV with additional and more complex DecisionHandlers, evaluating the resilience of F3DS-enabled applications to both innocuous and malicious peer misbehavior, and investigating execution-based malware detection.

F3DS lacks sharing of detected peer behavior—be that proper peer behavior or misbehavior–among peers. We envision that F3DS could be enhanced to share peer behavior information among social network participants.

In the next chapter, we present novel method for establishing trust among distributed system participents based exclusively on monitoring of peer behavior and sharing of peer behavior information among peers. Such a system could be used in the future to enhance F3DS.

CHAPTER 5

STATISTICAL TRUST ESTABLISHMENT IN WIRELESS SENSOR NETWORKS

We present a new distributed approach that establishes reputation-based trust among sensor nodes in order to identify malfunctioning and malicious sensor nodes and minimize their impact on applications. Our method adapts well to the special characteristics of wireless sensor networks, the most important being their resource limitations. Our methodology computes statistical trust and a confidence interval around the trust based on direct and indirect experiences of sensor node behavior. By considering the trust confidence interval, we are able to study the tradeoff between the tightness of the trust confidence interval with the resources used in collecting experiences. Furthermore, our approach allows dynamic scaling of redundancy levels based on the trust relationship between the nodes of a wireless sensor network. Using extensive simulations, we demonstrate the benefits of our approach over an approach that uses static redundancy levels in terms of reduced energy consumption and longer life of the network. We also find that high confidence trust can be computed on each node with a relatively small memory overhead and used to determine the level of redundancy operations among nodes in the system.

5.1 Introduction

Due to the criticality of many WSN applications, including monitoring and early warning systems, it is crucial that the information obtained from these networks be trustworthy. Decisions based on the sensor network data can have serious economic and social impact. Therefore, nodes in a sensor network must perform their functions reliably. However, due to their limited capabilities for economic viability, deployment in "unfriendly" physical environments, and risk of physical attacks, not all sensor nodes can be expected to behave reliably at all times [70]. It then becomes necessary to identify malfunctioning and malicious or compromised nodes, and isolate them. Detecting such misbehaving nodes from a location external to the network is difficult. This is because sensor nodes perform in-network data processing and aggregation. Wireless sensor networks can be secured most effectively against misbehaving nodes if the nodes closest to the source of the problem themselves can detect such misbehavior and react accordingly.

Currently, to deal with node misbehavior, critical sensor network deployments require sufficient redundancy to meet the needs of the particular application. However, complete redundancy typically requires a minimum of triple the amount of hardware (and energy expenditures) to ensure that a 2/3 Byzantine consensus can be achieved when a sensing or aggregation discrepancy is encountered. Such full redundancy has traditionally required a constant level of energy expenditure irrespective of network threat and misbehavior levels.

Wireless sensor networks must protect themselves from a variety of threats. In WSNs, typically, a large number of sensors are deployed in some area of interest. These sensor nodes are expected to work unattended even in naturally harsh physical conditions. They are also often deployed in accessible areas where they could be physically attacked. The harsh physical conditions, or physical attacks, could result in malfunctioning of the sensor devices. Sensor nodes could also be compromised by tampering, and replicated. Additionally, malicious sensor nodes could be dropped into the area of deployment. These malicious sensors could eavesdrop on sensor communications, pose as legitimate nodes, disrupt the functioning of the sensor networks by imposing themselves as "nodes-in-the-middle", and disrupting service in a variety of ways. We have loosely classified the different types of misbehaviors in a WSN below. This classification is not intended to be comprehensive. Our goal here is to identify the type of misbehavior our research has targeted.

- *Misbehavior 1:* Sensor nodes malfunction but are not malicious.
- *Misbehavior 2:* Malicious attacker nodes (possibly dropped in the sensor field)

eavesdrop on communications between genuine nodes, impersonate genuine nodes, and generate denial-of-service traffic or signals. However, in this threat model, there are no compromised nodes.

- *Misbehavior 3:* Compromised nodes, although appearing to be genuine, malfunction maliciously. They are also likely to cause the second type of misbehavior.

Our research focuses on misbehavior 1. We have established a trust system in sensor networks where nodes could malfunction but are not malicious. However, our trust system can also be a useful component for any solution that addresses misbehavior 3. This is because a compromised node might be able to authenticate itself correctly and still malfunction maliciously. Our trust system detects malfunctioning, whether malicious or not. Our research, however, does not address misbehavior 2, which requires suitable authentication and privacy mechanisms. In addressing misbehavior 1, we have focused on the following three basic functions of WSNs - accurate data collection, data routing, and data processing and aggregation. In this chapter, we present a new distributed approach that establishes reputation-based trust among sensor nodes that allows the system to dynamically adapt its redundancy based on the confidence that nodes have between each other to behave correctly (trust). We show that a significant amount of energy can be conserved and the sensor network life extended when redundancy is varied according to the changing levels of trust between nodes.

The remainder of this chapter is structured as follows. Section 5.2 summarizes existing work on building trust. Section 5.3 describes our trust system, its various components, and our trust computation methodology. In Section 5.4, we evaluate our trust model and in Section 5.5, we present conclusions and potential future areas of research.

5.2 Related Work

Trust has been studied in a variety of networks and applications. A large number of trust models have been proposed in social networking. In this section, we review only those existing works that are somewhat related to our research. Golbeck and Hendler [71] extend the concept of the semantic web to include reputation ratings. The algorithm they present is based on voting to derive either a complete trust or complete lack of trust in an entity. No partial trust is derived with their algorithm. They do not account for history of interactions and they assume perfect connectivity. There is no discussion of how to cache reputation ratings. Cahill et al. have outlined the importance of considering both risk as well as trust when making decisions [72]. If risk is low, then the action threshold for trust can be low. The trust model component of our research is based on the hypothesis of Carbone et al., which introduces a model that takes uncertainty into account. Caching trust is discussed in [73] but only to the extent of caching ciphers. Cache eviction based on content is not discussed. Gray et al. introduce the importance of calculating trust based on the "Small Worlds" approach [74]. They recommend a cache eviction algorithm also based on a "Small Worlds" approach. Reputation-based trust has also been proposed for peer-to-peer systems. Ganeriwal et al. present a Bayesian-based approach for building WSN trust [75]. Bayesian methods, though memory efficient, are not suitable for delay-tolerant networks where significantly stale information may arrive at the same time as fresh information. Chen and Yeager have constructed a decentralized trust system for the Sun JXTA platform [76]. They also take a Bayesian approach and use discrete trust ratings which cannot provide the same level of accuracy as continuous trust ratings. The *confidence* levels that they propose are not based on statistical confidence intervals. Although the above existing research addresses the problem of trust, none looks at building trust specific to *resource-limited* and *delay-tolerant* wireless sensor networks. Theodorakopoulos and Baras in [77] present an algorithm for forming trust in Ad Hoc networks based on seminirings. This approach, however, lacks the ability to easily decay the usefulness of previous experiences based on the risk sensitivity (aversion level) of each node independent of other nodes. Our method for trust calculation allows each node in a system to independently evaluate and weigh the experiences of other nodes without reliance on summarized recommendations of other nodes. Avinash et al. in [78] present a reputation-based system that precludes the ability for nodes to perform their own assessment of original experience evidence.

Their system is also delay in-tolerant. Yu et al. in [79] present an information theoretic framework for trust evaluation. Their framework along with the work of Kraniewski et al. in TIBFIT [80] do not leverage statistical confidence intervals nor do they address energy consumption optimizations. There is also a growing amount of research on security in WSNs (e.g., [81, 70, 82, 83, 84, 85]. This research mainly addresses misbehavior 2, and to some extent misbehavior 3, as described in Section 5.1. Unlike this existing research, our focus is on building trust in the presence of malfunctioning nodes while reducing energy consumption. Trust is not a replacement for security nor is security a replacement for trust. Trust and security rather complement each other. Within a system, building trust may require the use of secure protocols and maintaining security may be aided by trust establishment and maintenance.

5.3 Trust System

Social networks serve as an example by which we created a trust model for wireless sensor networks. Social trust is built in two phases. Before we directly interact with an individual, we might postulate a preconceived level of trust in that individual. Preconceived trust is formed from evidence we are given from other individuals in our social network. We automatically discount the accuracy of the obtained information based on our trust in the individuals who are generating and passing the information. We tend to trust information received via our direct social peers more than information received from the second layer of our social peers. The second phase of building trust is to interact directly with the individual or observe the direct interaction of others with the individual and start to establish a history of trust with that individual. In the case of WSNs, these observations to the sensing, routing and aggregation behavior of other nodes may be made by overhearing the radio communications of these nodes.

5.3.1 Context-specific Trust

Social trust in relationships may be built over days, months, years, or even decades. Each individual might have a different valuation of trust built over time. *Earning trust* may take a different length of time depending on the circumstances. Again, the trust we form with other individuals is limited to specific contexts based on the interaction we have had with them. Typically, we do not trust our preferred automobile mechanic with legal questions nor do we trust our preferred lawyer with questions regarding fuel injection systems.

Following the social network model, we have postulated that, given a network of context-specific and directional-trust relationships connecting two entities, any entity can place a confidence rating on any piece of data/fact/statement generated by another entity that falls within the given context. In the case of WSNs, the data produced by other nodes can be sensor readings, routed data, or aggregated data. We have further presumed that a confidence interval about the trust rating may be established to allow the entities to make accurate decisions. In the case of WSNs, this confidence interval will assist nodes in making decisions for routing, sensing, and data aggregation. Data should not be routed through nodes that can not be trusted. Likewise, data collected from a misbehaving node or routed or aggregated through should not be propagated through the network. Nodes may need to expend more power sensing to take over for a neighbor node whose sensors can no longer be trusted. Nodes should not include sensor readings in aggregation processing from nodes that cannot be trusted to provide generally accurate readings.

5.3.2 Collection of Experiences

We describe four types of experiences below. For each type of experience, we list the methodology we used to enable a node to turn the experience into a useful piece of information.

5.3.2.1 Sensor readings. Nodes follow the process of overhearing sensor readings of nearby nodes and then comparing them to their local sensor readings. If the remote sensor readings are correlated closely enough with the local sensor readings (they are within a threshold set by decaying the correlation of values based on the distance between the sensor ranges), then the remote sensor reading is considered to be valid. Overhearing the source node of sensor readings is not the only way to evaluate a sensor reading. Intermediary nodes that have been requested to route raw sensor information (and their respective neighbors that can overhear them) can also evaluate each sensor reading for accuracy, albeit with more limited ability given the increased distance from the source nodes sensors. The greater the perceived degradation in sensor accuracy, the less the source sensor node is trusted to accurately sense in the near future.

5.3.2.2 Experience generation accuracy. This is the evaluation of a neighbor's accuracy in recording direct experiences. To evaluate a neighbor's ability to generate experiences, a node listens to experiences generated and communicated by that neighbor and compares these experiences to its own. The larger the discrepancies in perceptions of experiences, the less trust a node will have in its neighbor's ability to accurately generate experiences. Examples of collection misbehavior include improperly weighing or evaluating an experience.

5.3.2.3 Observed data propagation accuracy (routing). Neighboring nodes within a certain proximity to a node performing some routing action are able to overhear both the incoming packet and the outgoing packet. These neighboring nodes can compare the outgoing routing destination of each overheard packet to its information in its own routing tables. If the packet apparently advances toward its intended destination, then the routing behavior of the overheard node is considered correct. If the packet does not get routed, gets corrupted or modified, or gets routed along an incorrect path, then the experience is recorded as a misbehavior by the overhearing node.

5.3.2.4 Observed accuracy of data aggregation. We examined two types of aggregation behavior observance. The first is one where a node is close enough to a neighbor to overhear all aggregation communication (inputs and outputs). In this case, a node will simply verify that the aggregation function behaved correctly. The second and more complex case is where the aggregation behavior of a node is to be evaluated for a node that is far enough away not to be able to overhear all its inputs. In this case, we relied on nodes that can compare the result of multipath propagation schemes for data aggregation. Examples of aggregation misbehavior include inaccurately aggregating data due either to processor error or to intentional bias.

5.3.3 Trust Computation Methodology

In this section, we present our trust computation methodology using experience records as input and providing as an output a trust value and a confidence interval based on those experiences. We present this methodology in the context of one entity, E1, that wishes to form trust in another entity, E2. Although a typical motivation for trust formation between nodes is decision-making, we do not explore different motivations here because the methodology is indifferent to motivation. Before trust is formed, entity E1 observes the behavior of E2 and judges whether the behavior is correct. Each opportunity E1 has of observing and judging the behavior of E2 is recorded in an *experience record*. An experience record contains at a minimum the following pieces of information:

- Identification of the entity (node) being observed. In our example, this is the identity of E2. This may be a unique node-id, unique location, or some other type of entity identifier.

- Identification of the entity (node) making the observation. In our example, this is the identity of E1. This may be a digital signature. The identity of the observer is necessary in the cases where experience records are shared between nodes.

- The context type of the experience/observation. If, for example, E1 judges E2's ability to sense temperature accurately, the context of the experience would be data sensing. In WSNs, data sensing is one important responsibility that nodes fulfill and thus is well served by neighbor observation. Two other important responsibilities are a) data routing, propagation and aggregation, and b) generation of an experience record. Experience record includes E2's ability to observe other nodes accurately, and generate experiences itself.

- A timestamp indicating how long ago the experience took place. This information is important given that experiences become stale over time (nodes may change behavior in the interim).

- The trust value. This is the actual rating of trustworthiness that the observer (E1) assigns the node being observed (E2) for this particular experience. We use x_i to represent the trust value of experience (sample) i

- A weight that the observer (E1) assigned to the experience record indicating the amount of observation that went into generating the experience record. A limited or brief experience would be weighted lower than a longer lasting or more intense experience. We use w_i^c to represent the context specific weight that an observer assigns to the experience i.

E1 thus observes the behavior of E2 and records these experiences in a local *experience cache*. Over time, these experiences will become stale and E1 may find it necessary to evict an existing record in the trust cache to make room for a newer record. E1 uses this trust cache to store both experiences that it recorded itself as well as experience records it receives from other nodes in the network.

5.3.3.1 Initial evaluation of experience records. When E1 wishes to form a trust confidence interval for E2, it first identifies the context of the desired trust confidence interval (ability to sense data, etc). It queries its experience cache for records that have E2 being evaluated in this context. The goal of E1 is to find the mean of these trust values and to identify a confidence interval about this mean. The typical method for finding a mean ($\overline{x}$) of the sample values is simply to add up all of the values and divide by the number of samples:

$$\overline{x} = \frac{\Sigma x_i}{n}$$

The typical method for finding a confidence interval about this mean is to first estimate the variance of the population σ^2:

$$\sigma^2 = \frac{\Sigma\left(x_i - \overline{x}\right)^2}{n-1}$$

This estimated variance is used to create a confidence interval about the mean [86]:

$$\overline{x} \pm t_{n-1,1-\alpha/2}\sqrt{\sigma^2/n} \qquad \text{(eq.1)}$$

where α is 0.10 for a 90% confidence interval, 0.05 for a 95% confidence interval, etc. The t in the above equation represents the $student - t$ distribution. If the confidence interval is sufficiently narrow (as determined by the context), E1 proceeds with its decision-making process. However, if the confidence interval is too wide, then additional experiences are collected at the expense of additional resources.

The above method constitutes the basis of our trust computation methodology. Experience records may be received after a significant delay. The significance of an event may be different between observing nodes. A node that creates an experience may be unreliable or malicious. For these reasons, our trust system establishes a confidence interval around a weighted mean [87, 88] to overcome this problems rather than taking a Bayesian approach.

To create the confidence interval around a weighted mean, E1 first calculates a weight W_i for each sample point i. It does this by combining the context specific, level of observation weight w_i^c with a new weight w_i^t that is based on the age of the experience record. The formula behind w_i^t may be chosen at the discretion of E1, but the idea is that the older the sample point (experience record) is, the lower the weight should be. This may be, for instance, some constant chosen from the interval [0,1] raised to the power of the age. These two weights are combined as follows:

$$W_i = w_i^t w_i^c$$

Using this total weight for each sample point E1 then determines the weighted mean ($\overline{x}_w$) of all of the experiences with E2:

$$\overline{x} = \Sigma\left(\frac{W_i}{\Sigma W_i} x_i\right)$$

From this weighted mean, the unweighted variance (σ^2) is then calculated as usual:

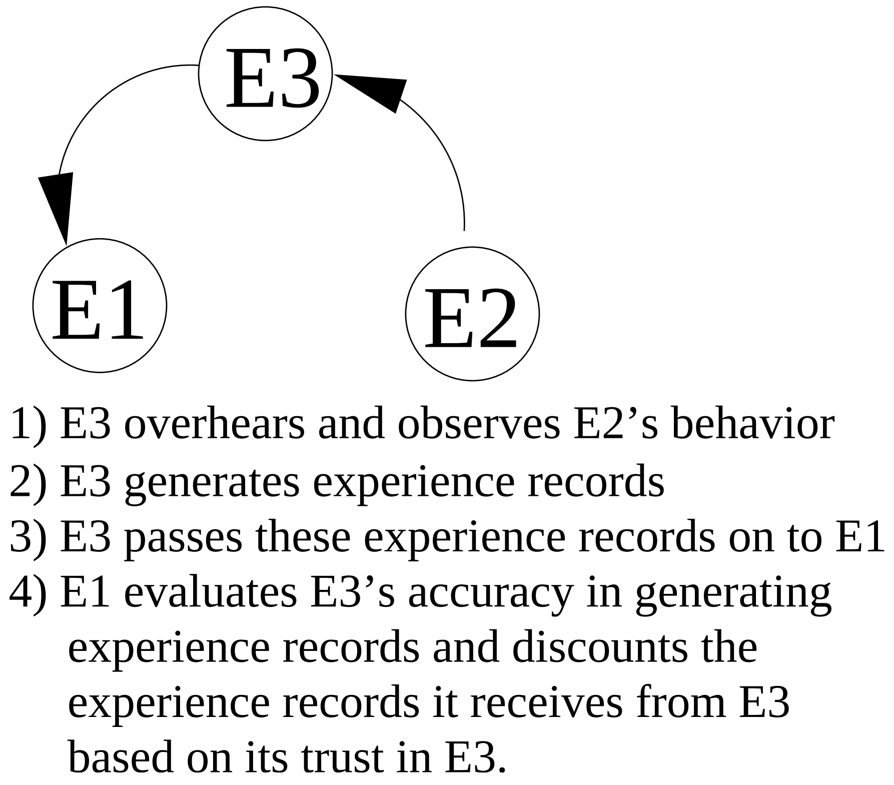
$$\sigma^2 = \frac{\Sigma\left(x_i - \overline{x_w}\right)^2}{n-1}$$

and then turned into a weighted variance (σ_w^2) by E1 via the following manner:

$$\sigma_w^2 = \frac{\sigma^2 \Sigma W_i^2}{\left(\Sigma W_i\right)^2}$$

Armed with the weighted mean $\overline{x}_w$ and the weighted variance σ_w^2, E1 then forms a confidence (eq.1) interval about the weighted mean. To reduce the effect of stale samples and to reduce bias created by correlated samples, the $t_{n-1,1-\alpha/2}$ value is established not by using the usual total number of samples points (n) but instead by using a deflated number of degrees of freedom. This deflated number of degrees of freedom is obtained by simply adding up the sum of all of the total experience weights: ΣW_i. When all total experience weights are in the interval [0,1], the net effect of using this deflated number for degrees of freedom is a widening of the confidence interval. This widening is important due to the reduced confidence we have in correlated and stale values.

5.3.3.2 Incorporating experiences collected by third parties. Although first-hand experiences are the most valuable, it is also valuable for E1 to collect and weigh experiences generated by neighbors of E2. To use experience data produced by other neighbors of E2, E1 must first establish its own trust in the ability of those neighbors to generate experiences. We will use Fig. 5.1 as an example. E3 has generated experiences relating to E2, but until E1 has established its own trust in E3, these experience records cannot be used by E1. E1 starts the process of establishing



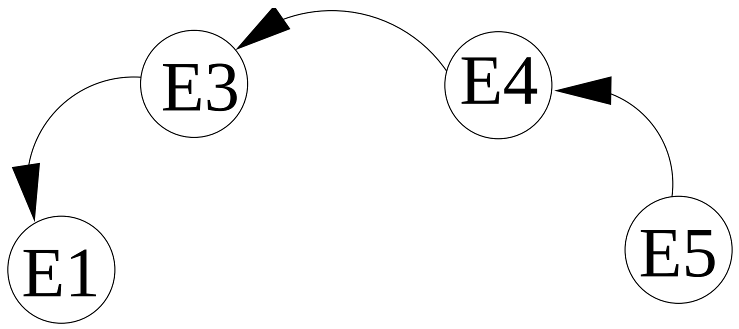
1) E3 overhears and observes E2's behavior
2) E3 generates experience records
3) E3 passes these experience records on to E1
4) E1 evaluates E3's accuracy in generating experience records and discounts the experience records it receives from E3 based on its trust in E3.

Figure 5.1. Building Trust via a Third Party trust in E3 by comparing experience records it (E1) has created while observing certain sensor network/node behavior to those created by E3 for the same behavior. In this process, E1 is able to collect experiences of E3's ability to generate experiences. It then calculates the confidence interval about the weighted mean of these experience using the equations for weighted mean, weighted variance, and a deflated number of degrees of freedom, described above. The resulting confidence interval in the context of E3's *Experience Generation* accuracy is formed by using eq.1. This confidence interval is then transformed by E1 into a fixed point τ_3^{EG} which represents the level of trust E1 places in E3's ability to accurately generate experiences. τ stands for "Trust". The number 3 is the id of the neighbor being evaluated, and EG stands for "Experience Generation". Specifically, the trust level, τ_3^{EG}, is calculated based on the following equation.

$$\tau_3^{EG} = \overline{x}_w - \rho * t_{n-1,1-\alpha/2} * \sqrt{\sigma_w^2/n} \tag{eq.2}$$

Here, ρ, ($\epsilon[-1, 1]$) is the level of aversion held by E1 (the node doing the evaluation). It identifies the risk E1 is willing to take in E3's experience generation ability. In the worst case, $\rho = 1$, implying that E3 chooses the lowest possible trust value in E3's experience generation ability. In the best case, $\rho = -1$, implying that E1 is willing to accept the highest possible trust value in E3's experience generation capability.

5.3.3.3 Incorporating distant observations. So far, we have covered how a node (E1 in our example) can use experience records generated by both itself and immediate neighbors. We will now explain how experience records generated by nodes farther than one hop away (nondirectly connected), which can be used in this trust system. This is experience data produced by entities that have potentially never interacted or communicated directly with the node doing the evaluation. For example: E1 wishes to establish a trust confidence interval about E5's sensing accuracy. E1 is not near enough to watch and evaluate E5. An additional entity E4, however, is near enough to evaluate and generate experience data on E5 and it happens to be near enough to E3 for radio communication as illustrated in Fig. 5.2. For E1 to utilize the experience data generated by E4, experience data must be accurately generated and propagated by E4. E1 thus must first evaluate E4's ability to accurately generate



1) E4 overhears and observes E5’s behavior
2) E4 generates experience records
3) E4 passes these experience records on to E3
4) E3 passes these experience records on to E1
5) E1 calculates its trust in E4’s ability to accuratly generated experience records and E3’s ability to accurately route data. E1 then discounts the experience records generated by E4 based on this trust in E4 and E3.

Figure 5.2. Building Trust in a Remote Node experiences in the same manner as described above. However instead of directly being able to evaluate E4, one must instead rely in experience records generated by E3. After receiving experience records relating to E4's ability to accurately generate experiences, E1 combines these into a confidence interval and then in turn into a single trust value, τ_4^{EG}, using eq.2. Likewise, E1 evaluates E3's ability to accurately propagate data. There may exist cases where nodes can accurately generate experiences and sense data, but due to faulty (or malicious) software, fail to route and propagate data accurately. E1 uses all of its available experience records related to E3's ability to propagate data and creates a confidence interval and then a single trust value, τ_3^{DP}, also using eq.2. where τ_3^{DP} in this case represents E1's trust in E3 in the context of "Data Propagation". E1 then uses the trust values it has established in E4 (ability to accurately generate experiences) and E3 (ability to accurately propagate data) to discount the weight of the experiences recorded by E4:

$$W_i = w_i^t w_i^c \tau_4^{EG} \tau_3^{DP}$$

If a certain piece of experience data must be propagated through multiple nodes, then that piece of experience data is discounted by the evaluator's trust in each intermediate node to propagate the data accurately. Hence, the generic equation for assessing the weight of any arbitrary experience record is:

$$W_i = w_i^t w_i^c \tau_{generator-id}^{EG} [\tau_{router1}^{DP} * \tau_{router2}^{DP} * ...]$$

where $\tau_{generator-id}^{EG}$ is the evaluators trust in the node that recorded the experience (in the context of Experience generation) and $\tau_{routerX}^{DP}$ is repeated for each node through which the experience record was propagated. With this method, an evaluator establishes trust through a chain of nodes and can use experience data generated by distant nodes.

5.3.3.4 Initial bootstrap of the trust system. Initialization of the system starts by nodes recording their own direct experiences with their physical neighbors. These experiences should include the evaluation of neighbors in at least the two special contexts of Experience-generation and Data-propagation. Each of these direct experiences is used for calculating the trust the evaluator node has in its neighbors.

The evaluator calculates the weight (W_i) for each of these experiences as: $w_i^t w_i^c$ Here w_i^t is an age-based weight (described in the previous section) and w_i^c is a weight assigned by the evaluator based on the level of contact the experience represents. For each neighbor, the experiences in each of the above contexts are grouped together to form a trust confidence interval in that particular context. For example, a node wishing to form a trust confidence interval in the context of Experience-generation for a particular neighbor will follow this protocol:

1. Observe experience data generated by the neighbor. This experience data would be in some context other than experience generation.

2. Compare that experience data to locally generated experience data and rate the accuracy of the experience data generation. From this comparison, a new experience point is generated.

3. Each of these experience points are weighed based on their age and context-weight to form a confidence interval: $\overline{x}_w \pm t_{n-1,1-\alpha/2}\sqrt{\sigma_w^2/n}$.

5.3.3.5 Limited memory for experience data. Sensor nodes usually have limited memory for storing experience data. If a new and apparently useful piece of experience data is acquired and must be added to a completely full experience data store, then an existing piece of experience data must be evicted. The eviction only takes place if the new piece of experience data has a higher "usefulness" than a piece of information already in the cache. We find that the resulting cache replacement pattern is similar to the "small worlds" replacement method, as recommended in [74]. In order to gain unfair advantage, certain entities could attempt to flood all receptive nearby entities with messages and requests for interaction in attempt to boost their own trust rating. For this reason, entities would find it beneficial to throttle the rate of new experiences from other entities.

5.3.3.6 Experience correlation. Our statistical methodology for computing confidence intervals expects independent samples. If the experience data are correlated, several samples must be aggregated to generate a single sample [86].

5.3.3.7 Location awareness. Sensor nodes must have a good sense of their environment in order to evaluate experiences such as sensor values received from other sensor nodes. Location awareness is necessary for extrapolation of sensor data. Typically node location is not known beforehand; thus, an in-network location awareness system [89] must be used. The location information required for evaluating experiences will be no more than that already required for efficiently routing and aggregating data in a deployed WSN application. Thus, location awareness for the purpose of trust computation should not require any additional resources.

5.3.3.8 Energy considerations. Our trust system requires sensor nodes to "overhear" messages from neighboring sensor nodes, and also collect trust data from the neighboring sensor nodes. We piggyback trust data on transmission and reception of application messages wherever possible. The system may be combined with other optimizations designed to reduce the overhead of trust formulation. An example of one such optimization is the MIT Leach protocol [90].

5.4 Evaluation of the Trust System

Using a custom discrete event simulator, "Trust-Sensim", we have simulated a cluster of nodes with the ability to formulate trust between each other. Here we describe the different aspects of this simulator.

5.4.1 Energy

Each node in the simulation has the energy consumption characteristics for transmitting, receiving, and processing data as well as sleeping of a Telos Rev B WSN node (model TPR2420CA). Each node is powered by a simulated energy capacity of two AA batteries. At the beginning of each simulation, these batteries are given an initial charge normally distributed around 2400 mAh (with a standard deviation of 200mAh).

5.4.2 Positioning

The nodes are positioned in relatively close proximity so as to allow for overlap between their sensing and communication ranges. This allows for all nodes to overhear each other's communications and build trust based on their observations. For this simulation the nodes are statically positioned 5 meters apart in a square grid of NxN nodes that act as a cluster of nodes. We simulate cluster sizes of 2x2, 3x3, 4x4, and 5x5 nodes. The 5x5 cluster having 25 nodes and an edge length of 25 meters. Nodes are aware of each other's identities and positions.

5.4.3 Simulation Events

As in the MIT Leach protocol [90], each node is assigned TDMA time slots for communication with the cluster lead during each round of sensing. For each round of sensing, each node: A) Wakes up on its assigned slot. B) Senses the current temperature and estimated remaining battery charge. C) Transmits the readings to the current cluster lead. D) Receives an ACK message back from the cluster lead which may include system management information such as the identity of the next cluster lead. E) Sleeps until the next round.

At the end of each round, the cluster-lead has the responsibility of doing a single high power transmission of the aggregated results to the base station. Any node that disagrees with the aggregated results can do its own high power transmission to the base station. For this simulation, the round period was 2 hours. Each cluster lead serves for a total of 25 rounds (50 hours) before a new cluster lead is chosen. During the final round of a cluster lead's tenure, the cluster lead advertises to all of the system nodes the node identify which will serve as the next cluster lead, which is the node with the most available remaining energy. The simulation ends when any node in the system is fully depleted of its energy capacity.

5.4.4 Trust Formulation

Each node can enter a mode called "neighborhood watch mode" which causes a node to first wake up at the very beginning of each round of sensing and listen on the radio for transmission of all other nodes. After overhearing the sensing data transmission of other nodes, the node will aggregate the data and transmit the aggregation result to the cluster lead (and any other listening nodes). A node will enter the low-power sleep mode at the very end of the "round" after all other nodes have concluded their aggregated data transmissions.

Neighborhood watch mode allows the system to act in a fully redundant manner.

All nodes not only sense and transmit the current cluster lead, but they also monitor each other's sensing and aggregation behavior and transmit their own aggregation result to the current cluster lead. Each node also has the ability to enter the "trust formulation mode" which adds the following actions onto the "neighborhood watch mode".

1. Evaluate the sensor readings, data aggregation results, and trust experiences overheard from other nodes; based on this evaluation, add new experiences to the local cache of trust experiences.

2. Communicate highly weighted experiences from the local experience cache to neighbors that may be listening via piggy-backing trust experiences on top of sensor reading transmissions to that node acting as the current cluster lead.

3. Probabilistically sleep and skip one or more sensing rounds based on each node's trust in the current cluster lead and its own neighbors.

On initial system startup, no trust exists between nodes. As nodes interact with each other, trust is formed. Figs. 5.3 and 5.4 show the effect of varying levels of link loss on the trust formation process. High levels of link loss prevent narrow confidence intervals from forming and delay, though they do not prevent, the formation of a high trust value. Nodes that are able to dedicate more memory resources to the formation of trust are able to achieve narrower confidence intervals, as shown in Fig. 5.5. There exists a tradeoff to the amount of memory dedicated to the formation of trust and the level of trust achievable (and therefore the life expectancy of a network that uses dynamic redundancy).

5.4.5 Findings

Though building trust requires more memory and computational resources than the neighborhood watch mode alone (due to the trust cache and trust formulation processing), the potential is created for dynamically sleeping through sensing rounds based on current trust in other system nodes. Thus, with building trust, the potential is created to save energy and extend the life span of the sensor network. We first establish a baseline by comparing three different systems: A) One with no redundancy

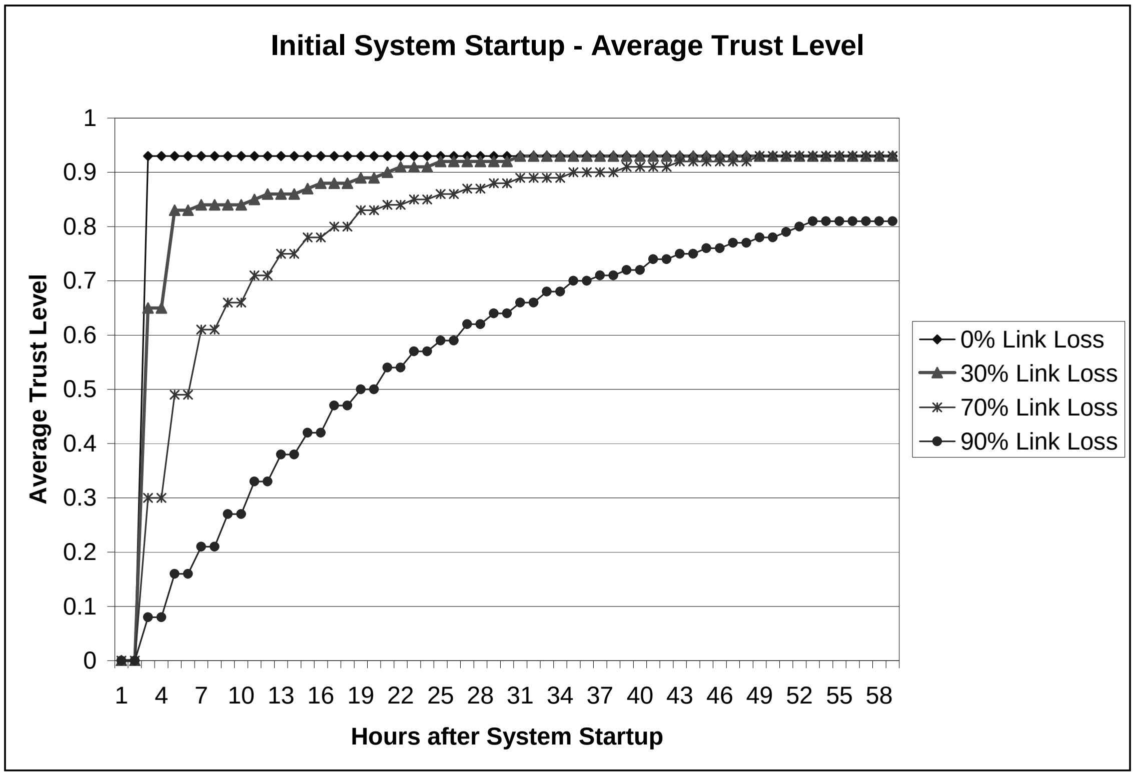


Figure 5.3. Effect of Link Loss on Startup Trust Level

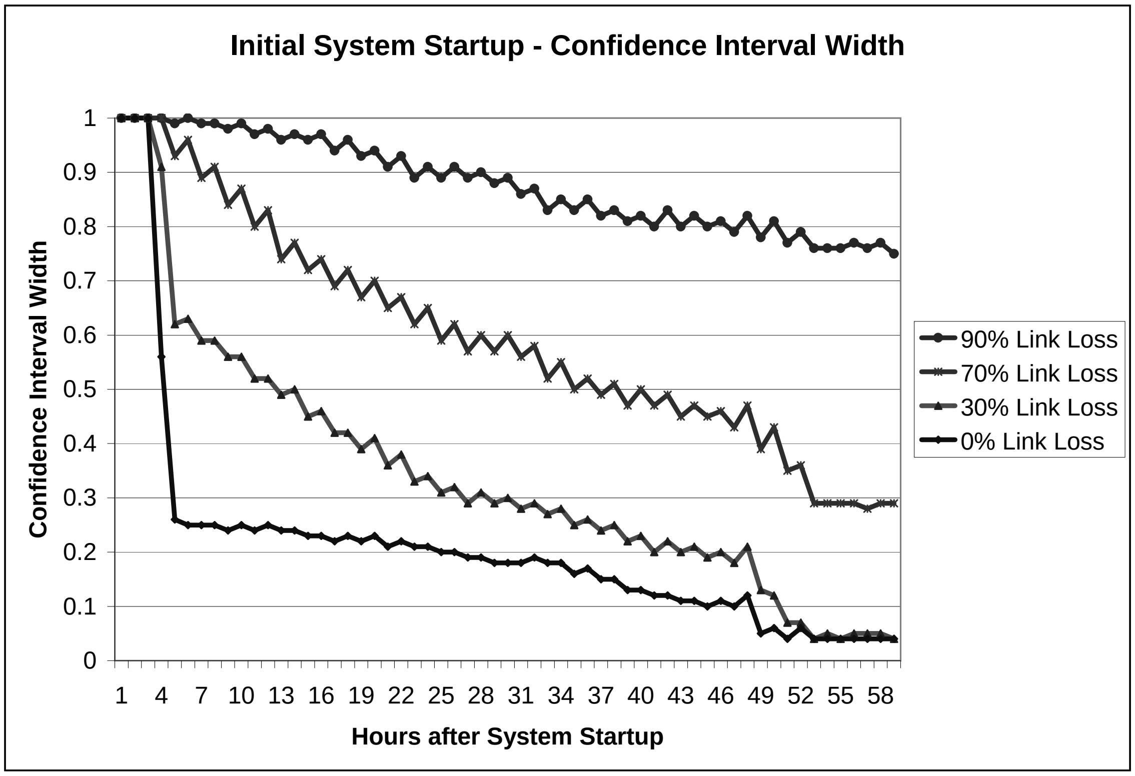


Figure 5.4. Effect of Link Loss on Startup Trust Confidence Interval Width

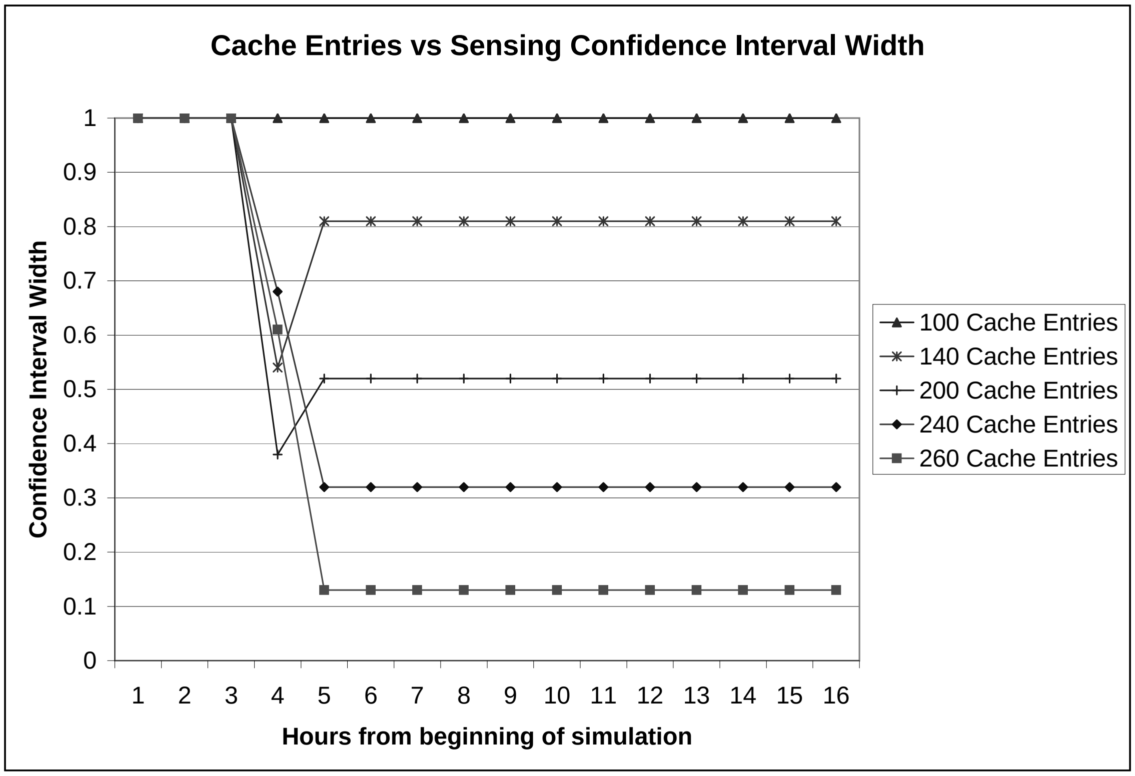


Figure 5.5. Cache Size Effect on Startup Confidence Interval Width

and neighbor monitoring. In this case, nodes perform only their own duties and never attempt to overhear or monitor the communications of other nodes, including the aggregation function performed by the base station. B) A system with full redundancy where all nodes monitor the communications and actions of all other nodes. C) A system with dynamic redundancy based on current levels of trust between nodes. In this simulation, we allow each node to have a relatively large trust cache (1000 entries). Fig. 5.6 shows that the achievable life expectancy of a system with trust enabled is well higher than a statically redundant system without trust. We ran the simulator varying the number of nodes in the system as well as the amount of memory dedicated to the trust cache in each system. Fig. 5.7 shows the change in expected minimum life of the system (in hours) as the amount of memory available to the trust cache on each node is increased. We find that full redundancy within larger networks requires a considerably higher amount of memory on each node in order to establish trust among nodes. Dedicating a relatively small amount of memory to each node can

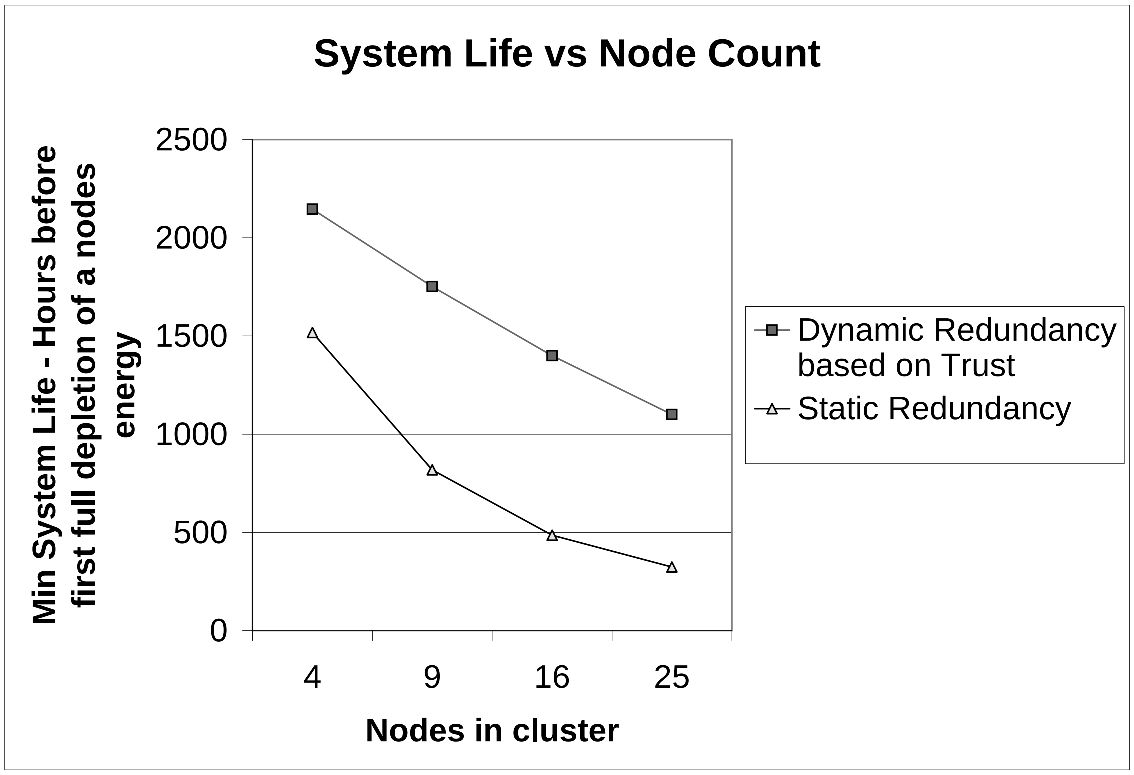


Figure 5.6. System Life with Different Node Counts

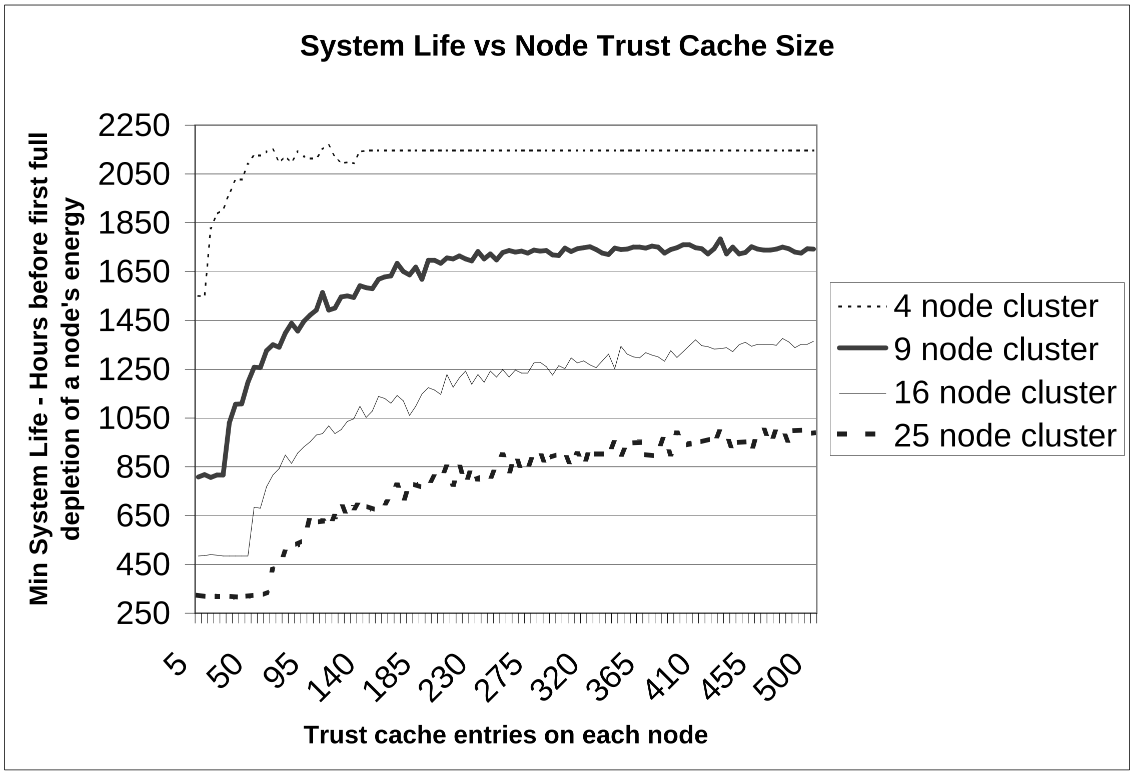


Figure 5.7. Cache Size Comparison

produce a significant improvement in the life expectancy of the system. As with any typical cache, however, there are diminishing returns associated with adding memory to the trust cache on each node. The simulator was also used to test the reaction time for injecting node failures into the system. At approximately 120 hours into the simulation, after trust had been established between system nodes, a sensor failure on one of the nodes was simulated. Fig. 5.8 shows the time required for the system to react (the confidence interval to widen) when a node fails. This graph represents the average trust confidence interval width in the failed node among all other nodes in the system. It is interesting to note that larger trust caches on each node are able to maintain narrower confidence intervals in the event of node failure. This is because with larger caches, a higher number of new experiences are accepted into the cache and usable in calculating trust. Though not shown here, the cluster lead is able to react more quickly than other nodes in the system given that it is awake and thus has one of the first available opportunities to detect node failure. Other nodes in

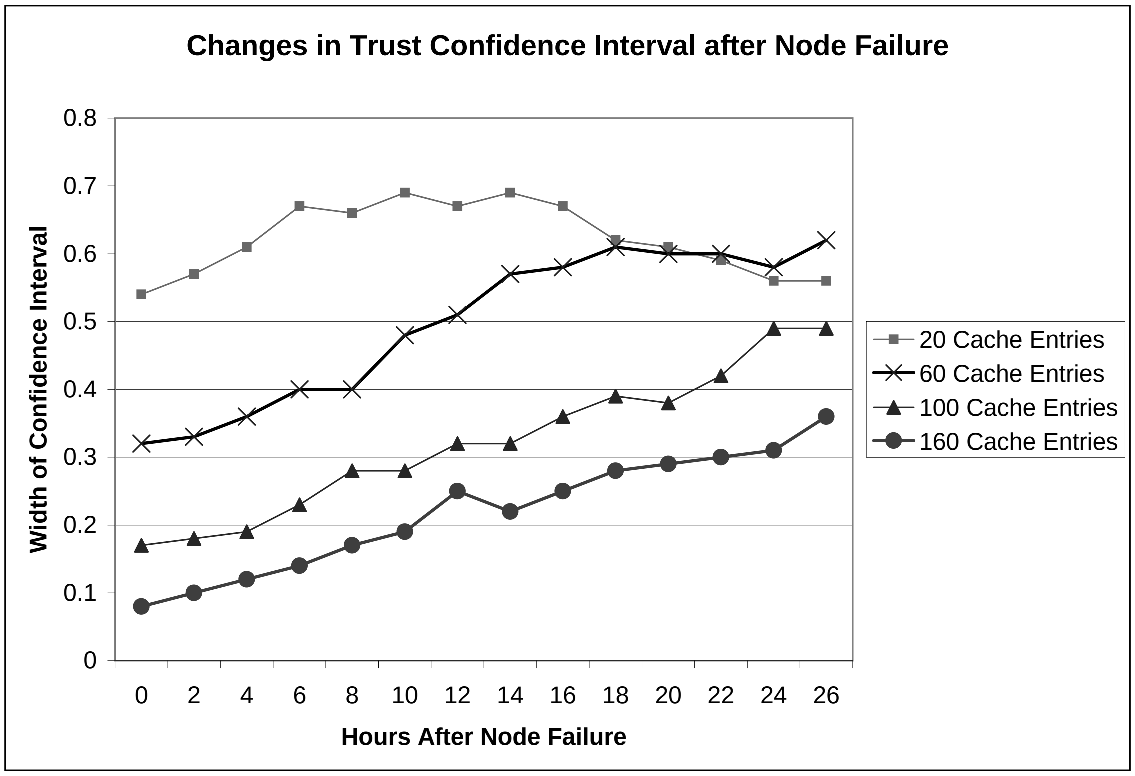


Figure 5.8. Simulated Sensor Failure Reaction Time

the system that are sleeping at the time of a node failure are unable to detect the node misbehavior until they wake up and are informed by the cluster lead (or their neighbors). It takes more time for the entire system to react with a higher number of sleeping nodes. The node with the failed sensor itself may have been sleeping during some of these hours, delaying detection by neighbors until it wakes up. A future enhancement to the system might be to assign higher weights to experiences where nodes behavior changes rather than give the same weight to all new experiences. Such a change would assist in drawing the attention to sudden changes in behavior.

5.5 Conclusions and Future Work

In this chapter, we presented a new distributed approach that establishes reputation-based trust among sensor nodes in order to identify sensor node misbehavior, minimize their impact on applications, and maximize energy conservation. We demonstrated the benefits of our approach using extensive simulations. However, we have only tested simple node failures and levels of link loss. We plan to investigate the responsiveness of the trust model to malicious misbehavior, including both external attackers and existing nodes that have been compromised. We also plan to experiment with node mobility.

We envision that our proposed novel approach to establish trust among nodes based on distributed monitoring, assessment, and sharing of peer behavior information among peers can be used to enhance the capabilities of F3DS. In the next chapter, we present the overall conclusions of this dissertation as well as ideas for future research.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In Chapters 2 through 4 of this dissertation, we have presented novel methods that use social metrics to enhance p2p collaboration, including describing, implementing, and evaluating social collaborative systems that enhance file distribution (SocialSwarm) and malware detection (F3AV) via the novel F3DS framework.

F3AV can be incrementally improved via such varied methods as adding a "multi-factor" DecisionHandler that takes into account multiple factors including each peer's SocialDistance, the age of each signature set used, the age of each scan, the commonality and diversity among scan engines and signatures sets to form a combined, and weighted confidence interval to decide whether to trust an object, to reject an object, or to seek more scans on an object.

F3AV could also be enhanced to leverage cloud-based scanning when absolutely necessary—when f2f scanning services are unavailable or the accuracy thereof is questionable.

The F3DS framework that we presented and implemented can serve as a basis for enabling social awareness in many other p2p applications. We now present three examples of such applications—Distributed f2f backups, content distribution, and IDS/IPS.

6.1 Other Applications of F3DS

In this section, we describe some of the other candidate applications that can be built upon F3DS.

- *Distributed f2f Backup:* F3DS could be used to build a fully distributed erasure code–based f2f backup service using capacity and bandwidth from trusted social peers to store as well as retrieve (when necessary) encrypted backup sets. A goal of this system would be to dynamically adjust the encoding of backup sets based on the availability of social peers as well as our confidence in each of those peers to be willing and able to retain the data over the desired retention window.

- *Distributed f2f Content Cache:* Social peers commonly have correlated interests in online content [91]. F3DS could be used as the basis for a f2f-distributed web cache that prefetches, caches, and distributes content based on correlated content access behavior among social peers.
- *Distributed f2f IDS/IPS:* A significant challenge in the area of distributed intrusion detection [92] [93] is that of finding and maintaining sufficient diversity of trusted nodes and resources to participate in the collection and analysis of data on network attacks. F3DS could be used to allow f2f collaboration on attack monitoring and analysis with the goal of protecting distributed p2p application services from malicious entities.

6.2 Application Level Misbehavior Monitoring

In Chapter 4, we presented a statistical method for establishing trust in the behavior of peer devices. This method can be adapted and implemented in F3DS to allow for trust to be formed and maintained within a network of F3DS-enabled applications using peer behavior monitoring and analysis.

Behavioral analysis can leverage both the detection of peer misbehavior as well as the observation of sudden changes in normal behavior. The hypothesis is that if the correlation of a peer's object access or malware scanning patterns to those of his/her social peers rapidly changes, then the peer may have been compromised.

In the remainder of this chapter, we present a number of open challenges related to social networks and potential areas for future research.

6.3 Mobility

The price, weight, and energy consumption of sensors and processors on mobile devices will continue to drop, thus permitting a continuous increase in collected data which will be fed to friends and followers on social networks. Under such circumstances, the percentage of people using such mobile devices for collaboration with their social peers will increase.

Mobile devices will continue to increase in their utility. Users will commonly use their mobile devices to store financial data and make direct device-to-device financial transactions. Such data will also pose as valuable targets for attack by malware creators.

As the use of mobile devices increases, so will the use of location-aware and environment-aware software applications also increase. The information collected by these applications and then shared with other social network users will add to the volume of data that must be secured and kept private. Currently, there are races among mobile device vendors for increased sensing capabilities as well as among mobile application developers seeking for increased simplicity in sharing data with social peers. At the same time, little emphasis is being placed on security and privacy. Thus, the new mobile hardware and software seem to be generating more security and privacy problems than they are solving.

With the growth of mobile applications, the need to detect mobile malware will rise commensurately. Given the limited resources (memory, cpu, bandwidth) of current mobile devices, detection of sophisticated mobile and socially aware malware is a significant challenge.

6.4 Trends in Malware

Malware is continuously increasing in sophistication. Recent techniques have emerged to create malware by pragmatically stitching together [94] legitimate system binaries into malicious code. Other techniques leverage GPUs to perform run time unpacking and polymorphism [95]. Such self-camouflaging worms are almost impossible to detect using traditional signature-based virus detection.

The processing resources required to detect this new generation of malware are significantly higher than traditional signature-based virus detection. Some of the proposed effective solutions include GPU-assisted antimalware [96] [97] [98], and execution-based virus detection using lightweight virtual machines [99] [100].

The cpu and memory available to individual users—especially those with mobile devices—are not sufficient to scan and filter all objects that a user might access in real time. For this reason, a valuable enhancement to F3AV would be to incorporate GPU antimalware techniques and/or execution-based scanning so as to distribute the computation required to scan a group of objects that are of mutual interest to a social peers across the resources available to those peers.

IDS is another area that could benefit from distributed social peer collaboration. Distributed IDS has long been an area of interesting research [92] [93] but has never achieved practical implementation given the challenge of distributed trust and available resources. Also, new techniques for real-time IDS are computationally intensive [101]. F3DS could serve as a platform on which to build distributed IDS utilizing the trust and resources available among peers within a social network.

6.5 Centralized vs Distributed Social Networks

Centralized as well as distributed social networks create mechanisms for users to interact with each other. The information shared as part of those interactions must have its privacy and security preserved.

Centralized social networks (Facebook, MySpace, etc.) typically use proprietary and commonly undisclosed mechanisms to provide privacy and security between users. The strength of those undisclosed mechanisms is questionable. Malicious entities are continuously attempting to compromise social network accounts. Facebook has disclosed that it detects over 600,000 attempts daily to compromise accounts on its network [102]. Centralized social networks are commonly sponsored by corporations which have incentives to mine the networks for information and statistics that hold commercial value. A user of a Centralized social network must be concerned about not only 3rd-party attacks against the user's social network account, but also concerned about the commercial incentives and trust-ability of the corporation running the social network.

Distributed social networks (Safebook, LifeSocial, SMS, SMTP email, PeerSoN, Diaspora, etc.), by their nature, must use published (and commonly standardized) protocols for interaction between users. Therefore, the protocols and mechanisms used by distributed networks are commonly reviewed and scrutinized by a larger number on individuals that those of Centralized social networks. Distributed social networks avoid central databases and thus no single entity has control of or access to the relationship and interaction information which defines a social network.

6.6 Social Motivators for Strong Security

The human relationships within social networks may also be leveraged as strong motivators for behavioral change in users. Automatically generated security warnings to, or communication restrictions (ostracism) [103] from, social peers might help stigmatize those who are lax in deploying protection mechanisms. Fear of such ostracism likely would promote proactive and preventative security practices among social network users.

REFERENCES

[1] A. Mislove, H. S. Koppula, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Growth of the flickr social network," in *Proceedings of the first workshop on Online social networks*, ser. WOSN '08. New York, NY, USA: ACM, 2008, pp. 25–30. [Online]. Available: http://doi.acm.org/10.1145/1397735.1397742

[2] Kissmetrics, "Social media growth statistics," 2012. [Online]. Available: http://blog.kissmetrics.com/social-media-statistics/

[3] Flickr, "Flickr social network," 2012. [Online]. Available: http://www.flickr.com/

[4] zynga, "Zynga social games," 2012. [Online]. Available: http://www.zynga.com/

[5] linkedin, "Linkedin professional network," 2012. [Online]. Available: http://www.linkedin.com/

[6] pinterest, "Pinterest online pinboard," 2012. [Online]. Available: http://www.pinterest.com/

[7] S. Turkle, *Alone Together: Why We Expect More from Technology and Less from Each Other.* Basic Books, 2012.

[8] S. Marche, "Is facebook making us lonely?" 2012. [Online]. Available: http://www.theatlantic.com/magazine/archive/2012/05/is-facebook-making-us-lonely/308930/

[9] L. Indvik, "U.s. internet piracy is on the decline," 2011. [Online]. Available: http://mashable.com/2011/03/25/internet-music-piracy-study/

[10] A. Felt and D. Evans, "Privacy protection for social networking apis," 2008. [Online]. Available: http://www.eecs.berkeley.edu/~afelt/privacybyproxy.pdf

[11] J. Shneidman and D. Parkes, "Rationality and self-interest in peer to peer networks," *Peer-to-Peer Systems II*, pp. 139–148, 2003.

[12] S. Kamvar, B. Yang, and H. Garcia-Molina, "Addressing the non-cooperation problem in competitive p2p systems," in *Workshop on Economics of Peer-to-Peer Systems, jun.* Citeseer, 2003.

[13] B. Cohen, "Incentives build robustness in BitTorrent," in *2003 Workshop on Economics of Peer-to-Peer Systems (P2P Econ'03)*, 2003.

[14] D. Cabanillas, "Peer-to-Peer Bartering: Swapping Amongst Self-interested Agents," 2009. [Online]. Available: http://www.tdx.cat/bitstream/handle/10803/6658/01DCcb01de01.pdf?sequence=1

[15] Z. Zhu, G. Cao, S. Zhu, S. Ranjan, and A. Nucci, "A social network based patching scheme for worm containment in cellular networks," *IEEE INFOCOM, Rio de Janeiro, Brazil*, 2009.

[16] M. J. Probst, J. C. Park, R. Abraham, and S. K. Kasera, "Socialswarm: Exploiting distance in social networks for collaborative flash file distribution," in *Proceedings of The 18th IEEE International Conference on Network Protocols*, ser. ICNP '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 263–274. [Online]. Available: http://dx.doi.org/10.1109/ICNP.2010.5762775

[17] Z. Zhu, G. Cao, S. Zhu, S. Ranjan, and A. Nucci, "A social network based patching scheme for worm containment in cellular networks," in *2009 IEEE Conference on Computer Communications (INFOCOM'09)*, 2009.

[18] M. Probst, "F3ds web site," 2012. [Online]. Available: http://www.f3ds.org

[19] M. Probst and S. Kasera, "Statistical trust establishment in wireless sensor networks," in *2007 International Conference on Parallel and Distributed Systems*, vol. 2, 2007.

[20] E. Bogardus, "Measuring social distances," *Journal of Applied Sociology*, vol. 9, pp. 299–308, 1925.

[21] M. S. Granovetter, "The strength of weak ties," *American Journal of Sociology*, vol. 78, pp. 1360–1380, 1973. [Online]. Available: http://sociology.stanford.edu/people/mgranovetter/documents/granstrengthweakties.pdf

[22] S. Buchegger, D. Schiberg, L. hung Vu, and A. Datta, "Peerson: P2p social networking – early experiences and insights," in *In Proc. ACM Workshop on Social Network Systems*, 2009.

[23] L. A. Cutillo, R. Molva, and T. Strufe, "Safebook: A privacy-preserving online social network leveraging on real-life trust," *Comm. Mag.*, vol. 47, no. 12, pp. 94–101, Dec. 2009. [Online]. Available: http://dx.doi.org/10.1109/MCOM.2009.5350374

[24] S. Marti, P. Ganesan, and H. Garcia-Molina, "Sprout: P2p routing with social networks," in *EDBT Workshops*, 2004, pp. 425–435.

[25] S. Yang and I. Chen, "A social network-based system for supporting interactive collaboration in knowledge sharing over peer-to-peer network," *International Journal of Human-Computer Studies*, vol. 66, no. 1, pp. 36–50, 2008. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1071581907001139

[26] F. Walter, S. Battiston, and F. Schweitzer, "A model of a trust-based recommendation system on a social network," *Autonomous Agents and Multi-Agent Systems*, vol. 16, no. 1, pp. 57–74, 2008.

[27] D. Brumley, P. Poosankam, D. Song, and J. Zeng, “Automatic patch-based exploit generation is possible: Techniques and implications,” in *IEEE Symposium on Security and Privacy*, May 2008.

[28] I. Norros, B. Prabhu, and H. Reittu, “On uncoordinated file distribution with non-altruistic downloaders,” *Managing Traffic Performance in Converged Networks*, pp. 606–617, 2007.

[29] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee, “Bittorrent is an auction: analyzing and improving bittorrent’s incentives,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 243–254, 2008.

[30] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, “Do incentives build robustness in bittorrent,” in *2007 Symposium on Networked Systems Design and Implementation (NSDI’07)*, 2007.

[31] A. Norberg, “The opensource libtorrent library.” [Online]. Available: http://www.rasterbar.com/products/libtorrent/

[32] C. Gkantsidis, T. Karagiannis, and M. Vojnovic, “Planet scale software updates,” *SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 423–434, 2006.

[33] L. J. Camp and A. Friedman, “Good neighbors can make good fences: A peer-to-peer user security system,” in *Telecommunications Policy and Research Conference*, Sep 2004.

[34] S. Shakkottai and R. Srikant, “Peer to peer networks for defense against internet worms,” in *Interperf ’06: Proceedings from the 2006 workshop on Interdisciplinary systems approach in performance evaluation and design of computer & communications sytems.* New York, NY, USA: ACM, 2006, p. 5.

[35] M. Deshpande, B. Xing, I. Lazardis, B. Hore, N. Venkatasubramanian, and S. Mehrotra, “Crew: A gossip-based flash-dissemination system,” *Distributed Computing Systems, International Conference on*, vol. 0, p. 45, 2006.

[36] P. G. Lind, L. R. da Silva, Jr, and H. J. Herrmann, “Spreading gossip in social networks,” *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, vol. 76, no. 3, 2007. [Online]. Available: http://dx.doi.org/10.1103/PhysRevE.76.036117

[37] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer, “Free riding in bittorrent is cheap,” in *2006 ACM Workshop on Hot Topics in Networks (HotNets’06)*, 2006.

[38] P. Shah and J.-F. Paris, “Incorporating trust in the bittorrent protocol,” in *2007 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS’07)*, 2007.

[39] M. Piatek, T. Isdal, A. Krishnamurthy, and T. Anderson, “One hop reputations for peer to peer file sharing workloads,” in *NSDI’08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation.* Berkeley, CA, USA: USENIX Association, 2008, pp. 1–14. [Online]. Available: http://portal.acm.org/citation.cfm?id=1387590

[40] A. Friedman, “Good Neighbors Can Make Good Fences,” *IEEE Technology and Society Magazine*, vol. 278, no. 0079/07, 2007.

[41] P. Garbacki, A. Iosup, D. Epema, and M. van Steen, “2fast: Collaborative downloads in p2p networks,” in *P2P ’06: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing.* Washington, DC, USA: IEEE Computer Society, 2006, pp. 23–30.

[42] V. Vishnumurthy, S. Chandrakumar, and E. Sirer, “KARMA: A secure economic framework for peer-to-peer resource sharing,” in *Workshop on Economics of Peer-to-Peer Systems (P2P Econ’03)*, 2003.

[43] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, and H. J. Sips, “Tribler: a social-based peer-to-peer system: Research articles,” *Concurr. Comput. : Pract. Exper.*, vol. 20, no. 2, pp. 127–138, February 2008. [Online]. Available: http://dx.doi.org/10.1002/cpe.v20:2

[44] O. Ozkasap, M. Caglar, and A. Alagoz, “Principles and performance analysis of second: A system for epidemic peer-to-peer content distribution,” *Journal of Netwwork and Computer Applications*, vol. 32, no. 3, pp. 666–683, 2009.

[45] G. Karame, M. Cagalj, and S. Capkun, “Small coalitions: Lightweight collaboration for efficient p2p downloads,” *Network Computing and Applications, IEEE International Symposium on*, vol. 0, pp. 278–283, 2009.

[46] R. Izhak-Ratzin, N. Liogkas, and R. Majumdar, “Team incentives in bittorrent systems,” in *ICCCN ’09: Proceedings of the 2009 Proceedings of 18th International Conference on Computer Communications and Networks.* Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–8.

[47] W. Galuba, K. Aberer, Z. Despotovic, and W. Kellerer, “Self-organized fault-tolerant routing in peer-to-peer overlays,” in *7th Annual IEEE Consumer Communications and Networking Conference (CCNC’10)*, 2009.

[48] C. Dumez, “qbittorrent client.” [Online]. Available: http://qbittorrent.sourceforge.net/

[49] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, “On the evolution of user interaction in facebook,” in *Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (WOSN’09)*, August 2009.

[50] blogspot, “Myspace trackers.” [Online]. Available: http://myspacetracker.blogspot.com/

[51] J. Cheng, J. Hoffman, T. LaMarche, A. Tavil, A. Yavad, and S. Kim, “Forensics tools for social network security solutions,” 2009.

[52] X. Jardin, “Yet another Facebook privacy risk: Emails Facebook sends leak user ip address,” 2010. [Online]. Available: http://boingboing.net/2010/05/07/yet-another-privacy.html

[53] S. Koolen, “Creating and maintaining relationships in social peer-to-peer networks,” *Delft University of Technology*, 2007.

[54] ”Industry Analysis and Technology Division - Wireline Competition Bureau of the US FCC”, “High-speed services for internet access,” 2010. [Online]. Available: http://www.fcc.gov/Daily_Releases/Daily_Business/2010/db0722/FCC-10-129A7.pdf

[55] P. Labs, “2012 security trends.” [Online]. Available: http://pandalabs.pandasecurity.com/2012-security-trends/

[56] S. C. E. Salem, “Social media: The new battlefront for cyber security.” [Online]. Available: http://video.foxbusiness.com/v/1367823707001/social-media-the-new-battlefront-for-cyber-security/

[57] J. Oberheide, E. Cooke, and F. Jahanian, “Cloudav: N-version antivirus in the network cloud,” in *Proceedings of the 17th conference on Security symposium.* USENIX Association, 2008, pp. 91–106.

[58] S. Musil, “Symantec says source code stolen in 2006 hack.” [Online]. Available: http://news.cnet.com/8301-1009_3-57360662-83/symantec-says-source-code-stolen-in-2006-hack/

[59] M. H. Hamilton, A. Rousskov, and D. Wessels, “Cache digest specification - version 5.” [Online]. Available: http://www.squid-cache.org/CacheDigest/cache-digest-v5.txt

[60] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, “Summary cache: a scalable wide-area web cache sharing protocol,” *IEEE/ACM Trans. Netw.*, vol. 8, pp. 281–293, June 2000. [Online]. Available: http://dx.doi.org/10.1109/90.851975

[61] N. Weiler, “Secure anonymous group infrastructure for common and future internet applications,” in *Proceedings of In Proceedings of 17th Annual Computer Security Applications Conference (ACSAC’01)*, 2001.

[62] E. Zurich, “Security econometrics: The dynamics of (in)security,” 2009. [Online]. Available: http://e-collection.library.ethz.ch/eserv/eth:154/eth-154-02.pdf

[63] OPSWAT, “Opswat security industry market share analysis,” June 2011. [Online]. Available: http://www.opswat.com/sites/default/files/OPSWAT-Market-Share-Report-June-2011.pdf

[64] NetPilot, “Clean mx realtime virus database,” 2012. [Online]. Available: http://support.clean-mx.de/clean-mx/viruses.php

[65] J. Oberheide, K. Veeraraghavan, E. Cooke, J. Flinn, and F. Jahanian, "Virtualized in-cloud security services for mobile devices," in *In Proc. of MobiVirt*, 2008.

[66] A. Bose, "Propagation, detection and containment of mobile malware," Ann Arbor, MI, USA, 2008, aAI3328771. [Online]. Available: "http://deepblue.lib.umich.edu/bitstream/2027.42/60849/1/abose_1.pdf"

[67] B.-G. Chun and P. Maniatis, "Augmented smartphone applications through clone cloud execution," in *Proceedings of the 12th conference on Hot topics in operating systems*, ser. HotOS'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 8–8. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855568.1855576

[68] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, "Paranoid android: versatile protection for smartphones," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC '10. New York, NY, USA: ACM, 2010, pp. 347–356. [Online]. Available: http://doi.acm.org/10.1145/1920261.1920313

[69] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 49–62. [Online]. Available: http://doi.acm.org/10.1145/1814433.1814441

[70] A. Perrig, J. Stankovic, and D. Wagner, "Security in wireless sensor networks," *Communications of the ACM*, vol. 47, no. 6, pp. 53–57, June 2004.

[71] J. Golbeck and J. Hendler, "Inferring reputation on the semantic web," 2004, http://www.mindswap.org/papers/GolbeckWWW04.pdf.

[72] V. Cahill, E. Gray, J.-M. Seigneur, C. D. Jensen, Y. Chen, B. Shand, N. Dimmock, A. Twigg, J. Bacon, C. English, W. Wagealla, S. Terzis, P. Nixon, G. di Marzo Serugendo, C. Bryce, M. Carbone, K. Krukow, and M. Nielsen, "Using trust for secure collaboration in uncertain environments," *Pervasive Computing*, vol. 2, pp. 52–61, 2003.

[73] M. Sathyanarayanan, "Caching trust rather than content," *Operating Systems Review*, vol. 34, no. 4, pp. 32–33, 2000. [Online]. Available: citeseer.ist.psu.edu/satyanarayanan00caching.html

[74] E. Gray, J. Seigneur, Y. Chen, and C. Jensen, "Trust propagation in small worlds," in *Proceedings of the First International Conference on Trust Management (iTrust2003)*, 2003. [Online]. Available: citeseer.ist.psu.edu/gray03trust.html

[75] S. Ganeriwal and M. Srivastava, "Reputation-based framework for high integrity sensor networks," in *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, 2004.

[76] R. Chen and W. Yeager, “Poblano: A distributed trust model for peer-to-peer networks,” *Sun Microsystems Technical Paper*, 2000, http://www.jxta.org/docs/trust.pdf.

[77] G. Theodorakopoulos and J. S. Baras, “On trust models and trust evaluation metrics for ad hoc networks,” *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 2, pp. 318–328, February 2006.

[78] A. Srinivasan, J. Teitelbaum, and J. Wu, “Drbts: Distributed reputation-based beacon trust system.” in *DASC*, 2006, pp. 277–283.

[79] Y. L. Sun, W. Yu, Z. Han, and K. Lui, “Information theoretic framework of trust modeling and evaluation for ad hoc networks,” *Selected Areas in Communications, IEEE Journal on*, vol. 24, pp. 305–317, 2006.

[80] M. Krasniewski, P. Varadharajan, B. Rabeler, S. Bagchi, and Y. C. Hu, “Tibfit: Trust index based fault tolerance for arbitrary data faults in sensor networks,” *dsn*, vol. 00, pp. 672–681, 2005.

[81] C. Karlof, N. Sastry, and D. Wagner, “Tinysec: A link layer security architecture for wireless sensor networks,” in *Proceedings of Sensys 2004*, November 2004.

[82] B. Parno, A. Perrig, and V. Gligor, “Distributed detection of node replication attacks in sensor networks,” in *IEEE Computer Society Symposium on Security and Privacy*, May 2005.

[83] J. McCune, E. Shi, A. Perrig, and M. Reiter, “Detection of denial-of-message attacks on sensor network broadcasts,” in *IEEE Computer Society Symposium on Security and Privacy*, May 2005.

[84] Z. Benenson, N. Gedicke, and O. Raivio, “Realizing robust user authentication in sensor networks,” in *In Real-World Wireless Sensor Networks (REALWSN)*, 2005, intranet.sics.se/realwsn05/papers/benenson05realizing.pdf.

[85] V. Gupta, M. Millard, S. Fung, Y. Zhu, N. Gura, H. Eberle, and S. C. Shantz, “Sizzle: A standards-based end-to-end security architecture for the embedded internet (best paper),” in *PerCom*, 2005, pp. 247–256.

[86] A. Law and W. Kelton, “Simulation modeling and analysis,” in *McGraw Hill Series in Industrial Engineering and Management Science*, 2000.

[87] D. Krouse and C. Withers, “A visual basic program giving weighted confidence intervals for mean and variance,” *Industrial Research Limited Report 1581*, June 2004.

[88] J. M. Bland and S. Kerry, “Weighted comparison of means,” 1998. [Online]. Available: www.bmj.com/content/316/7125/129.full

[89] L. Lazos and R. Poovendran, “Serloc: Robust localization for wireless sensor networks,” *ACM Trans. Sen. Netw.*, vol. 1, no. 1, pp. 73–100, 2005.

[90] M. Handy, M. Haase, and D. Timmermann, "Low energy adaptive clustering hierarchy with deterministic cluster-head selection," *Proceedings of IEEE International Conference on Mobile and Wireless Communications Networks, Stockholm, 2002.*, 2002. [Online]. Available: citeseer.ist.psu.edu/handy02low.html

[91] X. Cheng and J. Liu, "Exploring interest correlation for peer-to-peer socialized video sharing," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 8, no. 1, pp. 5:1–5:20, Feb. 2012. [Online]. Available: http://doi.acm.org/10.1145/2071396.2071401

[92] M.-Y. Huang and T. M. Wicks, "A large-scale distributed intrusion detection framework based on attack strategy analysis," in *in Recent Advances in Intrusion Detection (RAID98)*, 1998.

[93] O. Oriola, A. Adeyemo, and A. Robert, "Distributed intrusion detection system using p2p agent mining scheme," 2012. [Online]. Available: http://www.ajocict.net/uploads/Oriola_Adeyemo_and_Robert_-_Distributed_Intrusion_Detection_System_Using_P2P_Agent_Mining_Scheme.pdf

[94] V. Mohan and K. W. Hamlen, "Frankenstein: Stitching Malware from Benign Binaries," in *Proc. of 6th Usenix Workshop on Offensive Technologies (WOOT 2012)*, 2012. [Online]. Available: https://www.usenix.org/conference/woot12/frankenstein-stitching-malware-benign-binaries

[95] G. Vasiliadis, M. Polychronakis, and S. Ioannidis, "Gpu-assisted malware," in *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*, oct. 2010, pp. 1 –6.

[96] E. Seamans and T. Alexander, "Fast virus signature matching on the gpu," 2009. [Online]. Available: http://http.developer.nvidia.com/GPUGems3/gpugems3_ch35.html

[97] H. Nguyen, "Kapersky labs releases gpu anti-virus that runs 360x faster than on core 2 duo cpu," 2009. [Online]. Available: http://www.ubergizmo.com/2009/12/virus-detection-claims-to-be-360x-faster-on-gpus-than-on-core-2-duo/

[98] F. Y.-F. Wang, "Offloading critical security operations to the gpu," 2011. [Online]. Available: http://web.mit.edu/frankw/www/papers/thesis.pdf

[99] A. Moshchuk, T. Bragin, D. Deville, S. Gribble, and H. Levy, "Spyproxy: Execution-based detection of malicious web content," in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium.* USENIX Association, 2007, pp. 1–16.

[100] L. Liu, S. Chen, G. Yan, and Z. Zhang, "Bottracer: Execution-based bot-like malware detection," in *Proceedings of the 11th international conference on Information Security*, ser. ISC '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 97–113. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-85886-7_7

[101] R. Borgohain, "Fugeids: Fuzzy genetic paradigms in intrusion detection systems," *CoRR*, vol. abs/1204.6416, 2012.

[102] B. Sullivan, "Facebook says 600,000 account logins compromised every day," 2011. [Online]. Available: http://redtape.nbcnews.com/_news/2011/10/28/8527819-facebook-says-600000-account-logins-compromised-every-day?lite

[103] K. Williams, T. Case, and C. Govan, "Impact of ostracism on social judgments and decisions: Explicit and implicit responses," *Responding to the social world: Implicit and explicit processes in social judgments and decisions*, pp. 325–342, 2003.