TRUST ESTABLISHMENT IN COMPUTER NETWORKS: THEORIES AND APPLICATIONS

BY

YAFEI YANG

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

ELECTRICAL ENGINEERING

UNIVERSITY OF RHODE ISLAND

2008

DOCTOR OF PHILOSOPHY DISSERTATION

OF

YAFEI YANG

APPROVED:

Dissertation Committee:

Major Professor _____

_____

_____

_____

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2008

ABSTRACT

Trust, a subject extensively studied in sociology and psychology, has long been known as the driving force for collaboration in social communities. Recently, trust is recognized as a key component in new security tools to defend computer networks, such as rating system in e-commerce networks, peer-to-peer networks, mobile ad hoc networks, and wireless sensor networks, against possible attacks. This dissertation focuses on trust framework, theoretical analysis and applications in different computer networks. It provides in-depth understanding of trust establishment process and analyze the vulnerability of trust and reputation mechanisms.

Online feedback-based rating systems are gaining popularity. Dealing with collaborative unfair ratings in such systems has been recognized as an important but difficult problem. This problem is challenging especially when the number of honest ratings is relatively small and unfair ratings can contribute to a significant portion of the overall ratings. In addition, the lack of unfair rating data from real human users is another obstacle toward realistic evaluation of defense mechanisms. We propose a set of methods that jointly detect smart and collaborative unfair ratings based on signal modeling. Based on the detection, a framework of trust-assisted rating aggregation system is developed. Furthermore, we design and launch a Rating Challenge to collect unfair rating data from real human users. The proposed system is evaluated through simulations as well as experiments using real attack data. Compared with existing schemes, the proposed system can significantly reduce the impact from collaborative unfair ratings.

Based on the real attack data from the rating challenge, we discover important features in unfair ratings, build attack models, and develop an unfair rating generator. The models and generator can be used to test current rating aggregation systems, as well as to assist the design of future rating systems.

Similar to other security mechanisms, reputation systems can be under attack.

We report the discovery of a new attack, named RepTrap(Reputation Trap), against feedback-based reputation systems, such as those used in P2P file-sharing systems and E-commerce websites(e.g. Amazon.com). We conduct an in-depth investigation on this new attack, including analysis, case study, and performance evaluation based on real data and realistic user behavior models. We discover that the RepTrap is a strong and destructive attack that can manipulate the reputation scores of users, objects, and even undermine the entire reputation system. Compared with other known attacks that achieve the similar goals, the RepTrap requires less effort from the attackers and causes multi-dimensional damage to the reputation systems.

There have been many time synchronization protocols proposed for sensor networks. However, the security problems related to time synchronization have not been fully solved yet. If malicious entities manipulate time synchronization, failures of many functionalities in the sensor networks would occur. we identify various attacks against time synchronization and then develop a trust and self-healing scheme to defeat those attacks. The proposed scheme has three phases: (1) abnormality detection performed by individual sensors, (2) trust-based malicious node detection performed by the base station, and (3) self-healing through changing the topology of synchronization tree. Simulations are performed to demonstrate the effectiveness of the proposed scheme as well as the implementation overhead.

In the current literature, the methods proposed for trust establishment are always evaluated through simulation, but theoretical analysis is extremely rare. We present a suite of approaches to analyze trust establishment process. These analysis approaches are used to provide in-depth understanding of trust establishment process and quantitative comparison among trust establishment methods. The proposed analysis methods are validated through simulations.

ACKNOWLEDGMENTS

First and foremost, my earnest appreciation goes to my advisor Dr. Yan Sun. She is a remarkable advisor in many aspects. I not only thank for her critical role in my intellectual growth, but also for her consistent encouragement throughout my research. She was always patient and optimistic during the hard times, and was happy for me during the good times. Her enthusiasm in research and teaching, modesty and extraordinary kindness make her a terrific advisor and an irreplaceable role model for me. I will always remember many things she has taught me, from identifying a good research topic to children education; from developing presentation skills to writing things up in a coherent way. Without her help, I could not have completed this work.

Dr. Qing Yang has been like a second advisor to me. I am deeply grateful for his invaluable support and guidance. I benefit a lot from many discussions I had with him. His insights and advice not only helped me to find solutions to research problems but also developed me as a researcher. I also appreciate many terrific stories and life experience he shared with me. I cannot thank him enough for giving me such an inspiring experience at URI.

I am also indebted to Prof. Joan Peckham, Resit Sendag and Gerry Tyler. Their incredible knowledge on computer science and political science inspired me a lot. Their valuable comments helped me to improve my research and dissertation in many ways. Appreciation is also given to them for taking the time participating in my comprehensive exam and thesis defense.

I thank the faculty and staff of the Electrical Computer and Biomedical Engineering department, specifically Dr. Steven Kay, Dr. JC Lo, Dr. Boudreaux-Bartels, Dr. Godi Fischer, Phyllis Golden, Meredith Leach and Tim Toolan.

Last but not the least, I would like to thank my wife for her accompany and support.

PREFACE

This dissertation is organized in the manuscript format consisting of five manuscripts. The publications are the following:

- Manuscripts 1:

  Yafei Yang , Yan Sun, Steven Kay, and Qing Yang, "Defending Online Reputation Systems against Collaborative Unfair Raters through Signal Modeling and Trust," to appear, in *Proceedings of the 24th ACM Symposium on Applied Computing (SAC'09)*, Honolulu, Hawaii, Mar. 2009.

- Manuscripts 2:

  Yafei Yang, Qinyuan Feng, Yan Sun and Yafei Dai, "RepTrap: A Novel Attack on Feedback-based Reputation Systems," *Proceedings of International Conference on Security and Privacy in Communication Networks (SecureComm'08)*, Istanbul, Turkey, Sep. 2008.

- Manuscripts 3:

  Yafei Yang, Qinyuan Feng, Yan Sun, and Yafei Dai, "Dishonest Behaviors in Rating Systems: Cyber Competition, Attack Models, and Attack Generator," submitted to *Journal of Computer Science and Technology*.

- Manuscripts 4:

  Yafei Yang and Yan Sun, "Securing Time-synchronization Protocols in Sensor Networks: Attack Detection and Self-healing," to appear, in *Proceedings of IEEE Global Telecommunications Conference (Globecom'08*, New Orleans, LA, Dec. 2008.

- Manuscripts 5:

  Yan Sun and Yafei Yang, "Trust Establishment in Distributed Networks: Analysis and Modeling," *Proceedings of IEEE International Conference on Communications (ICC'07)*, Glasgow, UK, Jun. 2007.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

MANUSCRIPT 1

Defending Online Reputation Systems against Collaborative Unfair Raters through Signal Modeling and Trust

Abstract

Online feedback-based rating systems are gaining popularity. Dealing with collaborative unfair ratings in such systems has been recognized as an important but difficult problem. This problem is challenging especially when the number of honest ratings is relatively small and unfair ratings can contribute to a significant portion of the overall ratings. In addition, the lack of unfair rating data from real human users is another obstacle toward realistic evaluation of defense mechanisms. In this paper, we propose a set of methods that jointly detect smart and collaborative unfair ratings based on signal modeling. Based on the detection, a framework of trust-assisted rating aggregation system is developed. Furthermore, we design and launch a Rating Challenge to collect unfair rating data from real human users. The proposed system is evaluated through simulations as well as experiments using real attack data. Compared with existing schemes, the proposed system can significantly reduce the impact from collaborative unfair ratings.

1.1 Introduction

Word-of-mouth, one of the most ancient mechanisms in the history of human society, is gaining new significance in the Internet [1, 2]. The *online reputation systems*, also known as the *online feedback-based rating systems*, are creating large scale, virtual word-of-mouth networks in which individuals share opinions and experiences by providing ratings to products, companies, digital content and even other people. For example, Epinions.com encourages Internet users to rate practically any kind of businesses. Citysearch.com solicits and displays user ratings on restaurants, bars, and performances. YouTube.com recommends video clips based on viewers' ratings.

The **value** of reputation systems has been well proved by research as well as the success of reputation-centric online businesses [3]. A study [4] showed that eBay sellers with established reputation could expect about 8% more revenue than new sellers marketing the same goods. A recent survey conducted by comScore Inc. and the Kelsey Group revealed that consumers were willing to pay at least 20% more for services receiving an "Excellent," or 5-star, rating than for the same service receiving a "Good," or 4-star, rating[5]. Digg.com was built upon a feedback-based reputation system that rates news and articles based on user feedback. After only 3 years of operation, this company now has a price tag of 300 million dollars and has overtaken Facebook.com in terms of the number of unique visitors[6].

As reputation systems are having increasing influence on purchasing decision of consumers and online digital content distribution, **manipulation** of such systems is rapidly growing. Firms post biased ratings and reviews to praise their own products or bad-mouth the products of their competitors. Political campaigns promote positive video clips and hide negative video clips by inserting unfair ratings at YouTube.com. There is ample evidence that such manipulation takes place [7]. In February 2004, due to a software error, Amazon.com's Canadian site mistakenly revealed the true identities of some book reviewers. It turned out that a sizable proportion of those reviews/ratings were written by the books' own publishers, authors, and competitors [8]. The scammers are creating sophisticated programs that mimic legitimate YouTube traffic and provide automated ratings for videos they wish to promote[9]. Some eBay users are artificially boosting their reputation by buying and selling feedbacks [10]. *The online reputation systems are facing a major challenge: how to deal with unfair ratings from dishonest, collaborative, and even profit-driven raters.*

In the current literature, most of the **defense** schemes detect unfair ratings based on the *majority rule*. That is, mark the ratings that are far away from the majority's opinion as unfair ratings. The majority rule holds under two conditions. First, the number of unfair ratings is less than the number of honest ratings. Second, the bias of the unfair rat-

ings (i.e. the difference between the unfair ratings and the honest ratings) is sufficiently large. However, the number of ratings for one product can be small. For example, the majority of products at Amazon.com have less than 50 ratings. It is not difficult for the manipulator (also referred to as the attacker) to register/control a large number of user IDs that make the unfair ratings overwhelm the honest ratings. Furthermore, the attacker can introduce a relatively small bias in unfair ratings. Therefore, the smart attackers can defeat the majority-rule based detection methods by either introducing a large number of unfair ratings or introducing unfair ratings with relatively small bias.

Recognizing the limitation of the majority-rule based detection methods, we solve the unfair rating problem from a new angle.

- Whereas most existing methods treat the rating values as samples of a random variable, we exploit the *time-domain information* (i.e. the time when the ratings are provided) and model the ratings as a random process.

- We develop a suite of *novel detectors* based on signal modeling. In one detector, honest ratings are treated as *noise* and unfair ratings are treated as *signal*. We model the overall ratings using an autoregressive (AR) signal modeling technique and examine the model errors. The model error is proved to be a good indicator of whether the "signal" (i.e. collaborative unfair ratings) is present. Furthermore, we use hypothesis testing to detect mean change, histogram change, and arrival rate change in the rating process. These detectors are integrated to address a full range of possible ways of inserting unfair ratings.

- We design a *trust manager* to evaluate the trustworthiness of raters based on the detection results. The trust information is applied to a trust-assisted rating aggregation algorithm to calculate the final rating scores and to assist future unfair rating detection.

To evaluate the proposed methods in the real world, we design and launch a **rating challenge**[11] to collect attack data from real human users. The proposed method, as

well as several traditional methods, is tested against attacks from real human users. The proposed system shows excellent performance and significantly reduces the impact from unfair ratings.

The rest of the paper is organized as follows. Related work and attack models are discussed in Section 2. An overview of the proposed system is presented in Section 3, and the algorithms are described in Section 4. The rating challenge and experimental results are presented in Section 5, followed by discussion in Section 6 and conclusion in Section 7.

1.2 Related Work and Attack Models

1.2.1 Related Research

In the current literature, unfair rating detection is conducted from several perspectives. In [12], the unfair ratings and honest ratings are separated through clustering technique. In [13], a rater gives high endorsement to other raters who provide similar ratings and low endorsement to the raters who provide different ratings. The quality of a rating, which is the summation of the endorsements from all other raters, is used to separate unfair and honest ratings. In [14], a statistical filtering technique based on Beta-function is presented. The ratings that are outside the q quantile and $(1-q)$ quantile of the majority opinion are identified as unfair ratings, where q is a parameter describing the sensitivity of the algorithm. In [15], if a new rating leads to a significant change in the uncertainty in rating distribution, it is considered to be an unfair rating. These schemes work well when the majority rule holds. Their effectiveness, however, degrades significantly when the majority rule does not hold.

Trust establishment is another key element in the proposed scheme. There is a rich literature on trust establishment for authorization and access control, electronics commerce, peer-to-peer networks, distributed computing, ad hoc and sensor networks, and pervasive computing [16, 17, 18, 19]. For rating aggregation problem, simple trust models are used to calculated trust in raters in [13, 20, 21]. However, their effectiveness is restricted due to the limitation of the underlying detection algorithms.

Cyber competitions are effective ways to collect real user data. Related to the topic of rating, there is the Netflix Challenge [22] whose purpose is to build a better recommendation system based on user ratings. The data collected in the Netflix challenge is not suitable for studying the collaborative unfair rating problem.

1.2.2 Attack Models

In the design of unfair rating detectors, we focus on *collaborative unfair ratings*. That is, a group of raters provide unfairly high or low ratings to boost or downgrade the overall ratings of an object. As discussed in Section 5.1, this type of ratings results from strategic manipulation against online rating systems.

It is well known that modeling the human attackers' behavior is very difficult. Therefore, we launched a Rating Challenge [11] to collect dishonest rating behaviors from real human users. In this challenge, participants inserted unfair ratings into a regular rating data set. The participants who can mislead the final rating scores the most won a cash prize. The proposed scheme and several other schemes are tested against the attack data collected from the rating challenge, instead of against specific attack models.

1.3 Trust-enhanced Rating Aggregation System Design

The overall design of the trustworthy rating aggregation system is shown in Figure 1.1. In this section, we first provide a high-level description of the two core components: *Rating Aggregator* and *Trust Manager*, and then discuss some major design challenges. The specific algorithms will be presented in Section 1.4.

1.3.1 Rating Aggregator Overview

The rating aggregation process contains four steps. **First**, four detectors are applied independently to analyze raw rating data.

- Since the primary goal of the attacker is to boost or reduce the mean value, a *mean change detector* is developed to detect sudden changes in the mean of rating values.

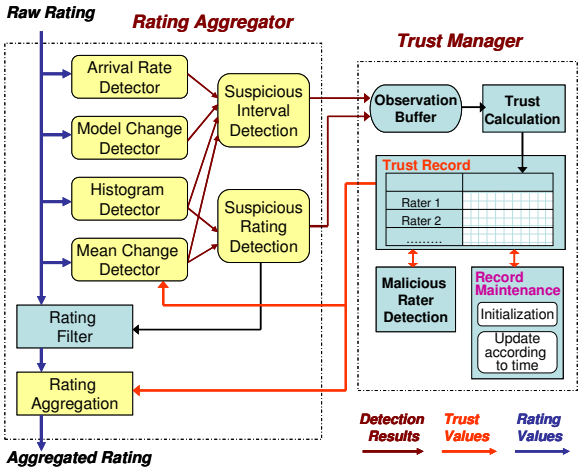

Figure 1.1. Block diagram of the trust-enhanced rating aggregation system

- When the attackers insert unfair ratings, they may cause an increase in the rating arrival rate. Thus, an *arrival rate detector* is designed to detect sudden increase in the number of ratings per time unit.

- A large number of unfair ratings can result in a change in the histogram of overall rating values, especially when the difference between unfair and honest ratings is large. Thus, a *histogram detector* is developed.

- The honest ratings can be viewed as a random *noise*. In some attacks, the unfair ratings can be viewed as a *signal*. Thus, a *signal model change detector* is used to detect whether a signal (i.e. unfair ratings) is presented.

Second, the outcomes of above four detectors are combined to detect the *suspicious time intervals* in which unfair ratings are highly likely. Additionally, the suspicious rating detection module can mark some specific ratings as suspicious.

Third, the trust manager uses the outcome of suspicious interval detection and suspicious rating detection to determine how much individual raters can be trusted.

Fourth, the rating filter removes highly suspicious ratings. The rating aggregation algorithm combines the remaining ratings using trust models.

1.3.2 Trust Manager Overview

Before discussing the trust manager, we introduce the relationship between trust establishment and rating aggregation.

A *trust relationship* is always established between two parties for a specific action. That is, one party trusts the other party to perform an action. The first party is referred to as the *subject* and the second party as the *agent*. A notation {subject: agent, action} is used to represent the trust relationship. For each trust relationship, one or multiple numerical values, referred to as *trust values*, describe the level of trustworthiness. In the context of rating aggregation,

- the *rating values* provided by the raters is just the trust value of {rater: object, having a certain quality};
- the *trust in raters* calculated by the system is just the trust value of {system: rater, providing honest rating};
- the *aggregated rating* (i.e. the overall rating score) is just the trust value of {system: object, having a certain quality}.

When the subject can directly observe the agent's behavior, *direct trust* can be established. Trust can also transit through third parties. For example, if A and B have established a recommendation trust relationship and B and C have established a direct trust relationship, then A can trust C to a certain degree if B tells A its trust opinion (i.e. recommendation) about C. Of course, A can receive recommendation about C from multiple parties. This phenomenon is called *trust propagation*. *Indirect trust* is established through trust propagations. The ways to calculate the indirect trust are often called *trust models*.

In the context of rating aggregation, the system, the raters, and the object obviously form trust propagation paths. Thus, the aggregated rating, i.e. the indirect trust between the system and the object, can be calculated using trust models. *One important obser-*

vation is that the calculation in rating aggregation can be determined or inspired by existing trust models.

The design of the Trust Manager is shown in Figure 1.1. It contains the *observation buffer* that collects observations on whether specific ratings or time intervals are detected as suspicious, the *trust calculation module* and *trust record* that compute and store trust values of raters, and the *malicious rater detection* module that determines how to handle the raters with low trust values.

1.3.3 Design Challenges

The **first challenge** is to design detection methods. The trust manager determines how much a rater can be trusted based on observations. However, obtaining the observations, or in other words extracting features from the raw rating data is challenging. A very popular trust calculation method is the Beta-function based model proposed in [23]. In this method, trust value is calculated as $\frac{S+1}{S+F+2}$, where S denotes the number of previous successful actions and F denotes the number of previous failed actions. This method has been used in various applications [24, 25].

Assume that we are examining the trust in rater i. S is the number of honest ratings provided by i, and F is the number of dishonest ratings provided by i. However, it is impossible to perfectly monitor rater i's past behavior, and we must estimate S and F values through some detection methods.

The attacking behaviors can be very complicated. Several detection strategies must be used simultaneously. The **second challenge** is to understand the effectiveness of each detector against different attacks and to integrate multiple detectors such that a broad range of attacks can be handled.

1.4 Algorithm Description

1.4.1 Mean Change Detector

The mean change detector contains three parts.

Mean Change Hypothesis Test

For one product, let $t(n)$ denote the time when a particular rating is given, $x(n)$ denote the value of the rating, and $u(n)$ denote the IDs of the rater. That is, at time $t(j)$, rater $u(j)$ submits a rating for the product with rating value $x(j)$, where $j = 1, 2, \cdots, N$ and N is the total number of ratings for this product.

We first study the mean change detection problem inside a window. Assume that the window contains $2W$ ratings. Let X_1 denote the first half ratings and X_2 denote the second half ratings in the window. We model X_1 as an i.i.d Gaussian random process with mean A_1 and variance σ^2, and X_2 as an i.i.d Gaussian random process with mean A_2 and variance σ^2. Then, to detect the mean change is to solve the hypothesis testing problem

$$\mathcal{H}_0 : A_1 = A_2$$
$$\mathcal{H}_1 : A_1 \neq A_2.$$

It has been shown in [26] that the Generalized Likelihood Ratio Test (GLRT) for this problem is

$$\begin{aligned} &\text{Decide } \mathcal{H}_1 \text{ (i.e. there is a mean change), if} \\ &2 \ln L_G(x) = \frac{W(\hat{A}_1 - \hat{A}_2)^2}{2\sigma^2} > \gamma \end{aligned} \tag{1.1}$$

where $\hat{A}_1$ is the average of X_1 and $\hat{A}_2$ is the average of X_2, and γ is a threshold.

Mean Change Indicator Curve

Second, the detector constructs the mean change indicator curve using a sliding window with sliding size ($2W$). Based on (3.1), the mean change indicator curve is constructed as $MC(k)$ versus $t(k)$, where $MC(k)$ is the value of $W(\hat{A}_1 - \hat{A}_2)^2$ calculated for the window containing ratings $\{x(k - W), \cdots, x(k + W - 1)\}$. In other words, the test in (3.1) is performed to see whether there is a mean change at the center of the window.

The example of mean change indictor curve is shown in Figure 1.2. The top plot shows the rating data $x(n)$ vs. $t(n)$. The blue dots represent the rating values for a flat panel TV (the first data set) in the rating challenge, the red $\circ$ represent the unfair ratings added by simulation. On the MC curves (the 2nd plot), two peaks clearly show the beginning and end of the attack.

MC Suspiciousness

Based on the peak values on the mean change indicator curve, we detect the time interval in which abnormal mean change occurs. This interval is called *mean change (MC) suspicious interval*.

When there are only two peaks, the MC suspicious interval is just between the two peaks. When there are more than 2 peaks, it is not straightforward to determine which time interval is suspicious. We use trust information to solve this problem. In particular, we divide all ratings into several segments, separated by the peaks on the mean change indicator curve. Assume there are M segments. In each segment, the mean value of ratings are calculated as B_j for $j = 1, 2, \cdots, M$. And B_{avg} is the mean value of the overall ratings. A segment j is marked as ***MC suspicious*** if either of the following conditions is satisfied:

1. $|B_j - B_{avg}| > threshold_1$. That is, there is a very large mean change.

2. $|B_j - B_{avg}| > threshold_2$ and T_j/T_{avg} is smaller than a threshold, where T_j is the average trust value of the raters in the j^{th} segment, T_{avg} is the average trust value of the raters in all segments. Here, $threshold_2 < threshold_1$. This condition says that there is a moderate mean change and the raters in the segment is less trustworthy.

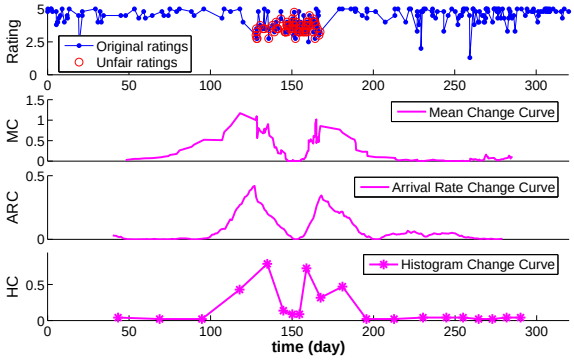

Figure 1.2. Illustration of MC, ARC and HC detection (attack duration: 40 days, bias: 0.2, variance: 0.5 $\times$ variance of honest ratings, arrival rate: 3 $\times$ arrival rate of the honest ratings)

1.4.2 Arrival Rate Change Detector

Arrival Rate Change Hypothesis Test

For one product, let $y(n)$ denote the number of ratings received on day n. We first study the arrival rate detection problem inside a window. Assume that the window covers $2D$ days, starting from day k. We want to detect whether there is an arrival rate change at day k', for $k < k' < k + 2D - 1$.

Let $Y_1 = [y(k), y(k+1), \cdots, y(k'-1)]$ and $Y_2 = [y(k'), y(k'+1), \cdots, y(k+2D-1)]$. It is assumed that $y(n)$ follow Poisson distribution. Then, the joint distribution of Y_1 and Y_2 is

$$p[Y_1, Y_2; \lambda_1, \lambda_2] = \prod_{j=k}^{k'-1} \frac{e^{-\lambda_1} \lambda_1^{y(j)}}{y(j)!} \prod_{j=k'}^{k+2D-1} \frac{e^{-\lambda_2} \lambda_2^{y(j)}}{y(j)!}, \tag{1.2}$$

where λ_1 is the arrival rate per day from day k to day $k' - 1$, and λ_2 is the arrival rate per day from day k' to day $k + 2D - 1$. To detect the arrival rate change is to solve to the hypothesis testing problem

$$\mathcal{H}_0 : \lambda_1 = \lambda_2$$

$$\mathcal{H}_1 : \lambda_2 \neq \lambda_1$$

It is easy to show that

$$p[Y_1, Y_2; \lambda_1, \lambda_2] = \frac{e^{-a\lambda_1} \lambda_1^{a\bar{Y}_1}}{\prod_{j=k}^{k'-1} y(j)!} \cdot \frac{e^{-b\lambda_2} \lambda_2^{b\bar{Y}_2}}{\prod_{j=k'}^{k+2D-1} y(j)!}. \tag{1.3}$$

where

$$\bar{Y_1} = \frac{1}{a}\sum_{j=k}^{k'-1} y(j), \quad \bar{Y_2} = \frac{1}{b}\sum_{j=k'}^{k+2D-1} y(j),$$
$$a = k' - k, \qquad b = k - k' + 2D.$$

A GLRT decides $\mathcal{H}_1$ if

$$\frac{p[Y_1, Y_2; \hat{\lambda_1}, \hat{\lambda_2}]}{p[Y_1, Y_2; \hat{\lambda}, \hat{\lambda}]} > \gamma, \tag{1.4}$$

where $\hat{\lambda_1} = \bar{Y_1}$, $\hat{\lambda_2} = \bar{Y_2}$, and $\hat{\lambda} = \frac{1}{2D}(\sum_{j=k}^{k+2D-1} y(j)) = \bar{Y}$. Taking logarithm at both sides of (3.4), we derive

Decide $\mathcal{H}_1$(i.e. there is an arrival rate change) if

$$\frac{a}{2D}\bar{Y_1}\ln\bar{Y_1} + \frac{b}{2D}\bar{Y_2}\ln\bar{Y_2} - \bar{Y}\ln\bar{Y} \geq \frac{1}{2D}\ln\gamma. \tag{1.5}$$

Arrival Rate Change Curve

Based on (3.5), the Arrival Rate Change (ARC) curve is constructed as $ARC(k')$ vs $t(k')$. Here, the k' value is chosen as the center of the sliding window, i.e. $k' = k+D$. When $D < k' < N - D + 1$, $ARC(k')$ is just the left-hand side of equation (3.5) with $a = b = D$. The example of the ARC curve is shown in Figure 1.2 (see the third plot), with two peaks showing the beginning and end of the attack.

ARC Suspiciousness

Based on the peaks on the ARC curve, we divide all ratings into several segments. If the arrival rate in one segment is higher than the arrival rate in the previous segment and the difference between the arrival rates is larger than a threshold, this segment is marked as *ARC suspicious*.

H-ARC and L-ARC

For some practical rating data, the arrival rate of unfair ratings is not very high or the poisson arrival assumption may not hold. For those cases, we design H-ARC, which detects the arrival rate change in high value ratings, and L-ARC, which detects the arrival rate change in low value ratings.

Let $y_h(n)$ denote the number of ratings that are higher than $threshold_a$ received on day n, and $y_l(n)$ denote the number of ratings that are lower than $threshold_b$ received on day n. The $threshold_a$ and $threshold_b$ are determined based on the mean of all ratings.

- H-ARC detector: replace $y(n)$ in the ARC detector by $y_h(n)$
- L-ARC detector: replace $y(n)$ in the ARC detector by $y_l(n)$.

Based on experiments, we found that H-ARC and L-ARC are more effective than the ARC detector when the arrival rate of unfair ratings is less than 2 times of the arrival rate of honest ratings.

1.4.3 Histogram Change Detector

Unfair ratings can change histogram of rating data. In this paper, we design a histogram change detector based on clustering technique. There are two steps.

- 1. Within a time window k with the center at t_k, constructed two clusters from the rating values using the simple linkage method. The Matlab function $clusterdata()$ is used in the implementation.
- 2. The Histogram Change (HC) curve, $HC(k)$ versus t_k, is calculated as

$$HC(k) = \min\left(\frac{n_1}{n_2}, \frac{n_2}{n_1}\right), \tag{1.6}$$

  where n_1 and n_2 denote the number of ratings in the first and the second cluster, respectively.

The example of the HC curve is shown in Figure 1.2. When the attack occurs, the $HC(k)$ increases.

1.4.4 Signal Model Change Detector

Let $E(x(n))$ denote the mean of $x(n)$, where $x(n)$ denote rating values. When there is no collaborative raters, ratings received at different time (also from different raters) should be independent. Thus, $(x(n) - E(x(n)))$ should approximately be a white noise.

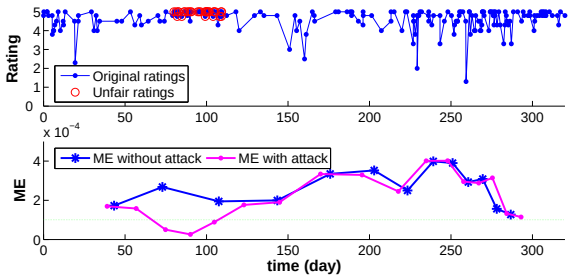

Figure 1.3. Illustration of ME detection(attack duration: 30 days, bias: 0.1, variance: 0.1 $\times$ variance of honest ratings, arrival rate: 2 $\times$ arrival rate of the honest ratings)

When there are collaborative raters, $(x(n) - E(x(n)))$ is not white noise any more. Instead, the ratings from collaborative raters can be looked at as a *signal* embedded in the white noise.

Based on the above argument, we develop an unfair rating detector through signal modeling.

- Model-error-based detection: the ratings in a time window are fit onto an autoregressive (AR) signal model. The model error is examined. When the model error is high, $x(n)$ is close to a white noise, i.e. honest ratings. When the model error is small, there is a "signal" presented in $x(n)$ and the probability that there are collaborative raters is high.

The *model error (ME) curve* is constructed with the vertical axis as the model error, and horizontal axis as the center time of the windows. The windows are constructed either by making them contain the same number of ratings or have the same time duration. The covariance method [27] is used to calculate the AR model coefficients and errors.

The example of ME curve is shown in Figure 1.3 (see lower plot). The curve marked with $*$ is the model error for the original rating data, the curve marked with $\cdot$ is the model error when unfair ratings are present. It can be seen that the model error drops when there is an attack. The time interval when the model error drops below a certain threshold is marked as the model error (ME) suspicious interval.

1.4.5 Integrated Detection

We have developed detectors for mean change, arrival rate change, histogram change, and model error change. The problem formulations for individual detectors are different. This is because that attack behaviors are very diverse and cannot be described by a single model. Different attacks have different features. For example, one attack may trigger MC and H-ARC detectors, another attack may trigger only L-ARC and HC detectors.

In addition, the normal behaviors, i.e. honest ratings, are not stationary. Even without unfair ratings, honest ratings can have variation in mean, arrival rate, and histogram. In smart attacks, the changes caused by unfair ratings and the normal changes in honest ratings are sometimes difficult to differentiate. Thus, using a single detector will cause a high false alarm rate. We have conducted experiments and compared these detectors quantitatively based on their Receiver Operating Characteristics (ROC) curves[26]. Based on ROC analysis and a study on real user attacking behavior, we develop an empirical method to combine the proposed detectors, as illustrated in Figure 3.1.

There are two detection paths. Path 1 is used to detect strong attacks. If the MC indicator curve has a U-shape, and H-ARC or L-ARC indicator curve also has a U-shape, the corresponding high or low ratings inside the U-shape will be marked as suspicious. If for some reasons, H-ARC (or L-ARC) indicator curve does not have such a U-shape, H-ARC (or L-ARC) alarm is issued. The alarm will be followed by the ME or HC detector. This is path 2. Path 2 detects suspicious intervals. Since there may be multiple attacks against one product, the ratings must go through both paths.

1.4.6 Trust in Raters

It is noted that we cannot perfectly differentiate unfair ratings and honest ratings in the suspicious intervals. Therefore, some honest ratings will be marked as suspicious. As a consequence, one cannot simply filter out all suspicious ratings. In our work,

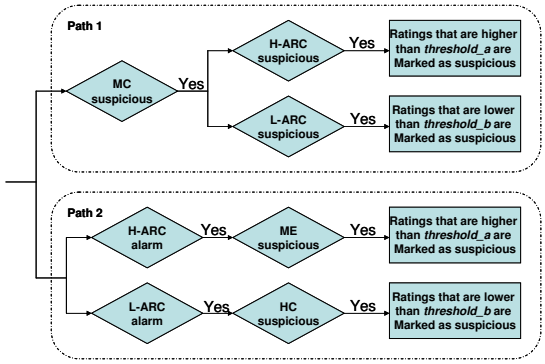

Figure 1.4. Joint Detection of Suspicious Ratings

this suspicious rating information is used to calculate trust in raters, based on the beta-function trust model[23]. The calculation is described in Procedure

Procedure 1 Computing Trust in Raters

For each rater i, initialize $S_i = 0$, and $F_i = 0$
for $k = 1 : K$ **do**
  % Let $\hat{t}(k)$ denote the time when we calculate trust in raters.
  % k is the index.
  for each rater i **do**
    Set $n_i = f_i = 0$,
    Considering all products being rated during time $\hat{t}(k-1)$ and $\hat{t}(k)$, determine:
      n_i: number of ratings that is provided by rater i
      f_i: number of ratings from rater i and being marked as suspicious
    calculate $F_i = F_i + f_i$ and $S_i = S_i + n_i - f_i$.
    calculate trust in rater i at time $\hat{t}(k)$ as: $(S_i + 1)/(S_i + F_i + 2)$.
  end for
end for

1.4.7 Rating Aggregation

Several trust models, including simple average and complicated ones in [23, 24], have been compared for rating aggregation in [28]. Based on the comparison in [28], we adopt the modified weighted average trust model to combine rating values from different raters.

Let R denote the set of raters whose ratings are the inputs to the aggregation module. If rater $i \in R$, let r_i denote the rating from rater i and T_i denote the current trust value of rater i. In addition, each rater provides only one rating for one object and R_{ag}

denotes the aggregated rating. Then,

$$R_{ag} = \frac{1}{\sum_{i:i\in R} max(T_i - 0.5, 0)} \sum_{i:i\in R} r_i \cdot max(T_i - 0.5, 0). \tag{1.7}$$

1.5 Performance Evaluation

1.5.1 Rating Challenge and Experiment Description

For any online rating systems, it is very difficult to evaluate their attack-resistance properties in practical settings due to the lack of realistic unfair rating data. Even if one can obtain data with unfair ratings from e-commerce companies, there is no ground truth about which ratings are dishonest. To understand human users' attacking behavior and evaluate the proposed scheme against non-simulated attacks, we designed and launched a *Rating Challenge*[11]. In this challenge,

- We collected real online rating data for 9 flat panel TVs with similar features. The data are from a well-known online-shopping website. The numbers of fair ratings of the 9 products are 177, 102, 238, 201, 82, 87, 60, 53, and 97.

- The participants to the Rating Challenge download the rating dataset and control 50 biased raters to insert unfair ratings. In particular, the participants decide when the 50 raters rate, which products they rate for, and the rating values.

- The participants' goal is to boost the ratings of two products and reduce the ratings of two other products.

- The participants' attacks are judged by the *overall manipulation power*, called MP value. For each product, we calculate $\Delta_i = |R_{ag}(t_i) - R^o_{ag}(t_i)|$ during every 30 day period, where $R_{ag}(t_i)$ is the aggregated rating value with unfair ratings, and $R^o_{ag}(t_i)$ is the aggregated rating value without unfair ratings. The overall MP value is calculated as $\sum_k (\Delta^k_{max_1} + \Delta^k_{max_2})$, where $\Delta^k_{max_1}$ and $\Delta^k_{max_2}$ are the largest and 2nd largest among $\{\Delta_i\}'s$ for product k. The participant that can generate the largest MP value win the competition.

In the calculation of the MP values, the two "big-deltas" represent 2$\sim$3 months of persistent change in rating scores. The calculation also considers both boosting and downgrading.

We have collected 251 valid submissions, which correspond to 1004 set of collaborative unfair ratings for single products. Three observations are made. First, more than half of the submitted attacks were straightforward and did not exploit the features of the underlying defense mechanisms. Many of them spread unfair ratings over the entire rating time. Second, among the attacks that exploit the underlying defense mechanisms, many of them are complicated and previously unknown. Third, according to a survey after the challenge, many successful participants generated unfair ratings manually. This data set covers a broad range of attack possibilities.

In the performance evaluation, all raters' trust values are assigned as 0.5 initially. The window size of the MC detector, H-ARC/L-ARC detectors, HC detector, and ME detector are 30 (days), 30 (days), 40 (ratings), and 40 (ratings), respectively. In the H-ARC and L-ARC, $threshold_a = 0.5m$ and $threshold_b = 0.5m + 0.5$, where m is the mean of the ratings in the time window. For the purposed of comparison, we also evaluate the performances of three other schemes.

1. SA scheme - No attack detection, using simple averaging for rating aggregation.

2. BF scheme - Using the beta-function based filtering technique proposed in [14] to remove unfair ratings. Then, the trust value of rater i is calculated as $(S_i + 1)/(S_i + F_i + 2)$, where F_i is the number of ratings (from rater i) that have been removed, and $F_i + S_i$ is the total number of ratings provided by rater i.

3. DC scheme - Using the proposed detectors without trust establishment.

1.5.2 Results

We have conducted simulations and experiments using real attack data. Due to space limitation, we will only show the performance comparison among different

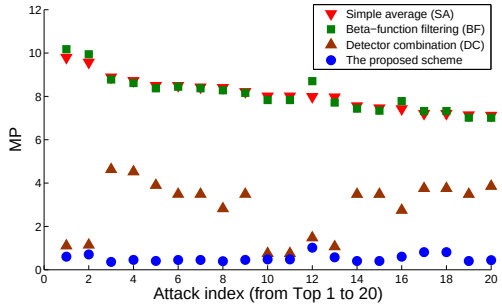

Figure 1.5. Performance comparison in terms of the MP resulting from Top 20 attacks against SA

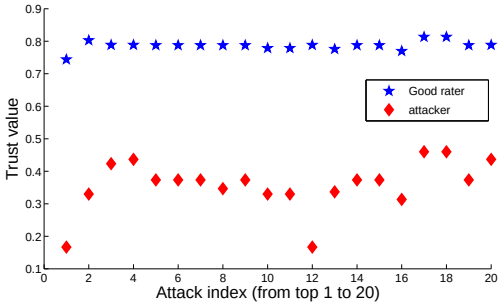

Figure 1.6. Trust values of the raters in the proposed scheme

schemes under the strongest attacks collected from the rating challenge.

In Experiment 1, we pick top 20 attacks against the SA scheme. That is, these 20 attacks generate the highest MP values when the SA scheme is used. In Figure 1.5, the four schemes are compared under these 20 attacks. The horizontal axis is the index of the attack data set from top 1 to top 20. The vertical axis is the overall MP value, which is the summation of individual products' MP values in each submission. Three observations are in order.

First, it is clear that the proposed scheme has the best performance. It can significantly reduce the MP values resulting from real users' attacks.

Second, the performance of the SA scheme is similar to that of BF scheme. The BF method is even slightly worse than the SA in some situations. There are two reasons. When the unfair ratings are concentrated in a short time interval, the majority ratings in this time interval can be unfair ratings. Using the majority rule, the beta filter will in fact remove good ratings. This is why the beta filter performs worse. In addition, when

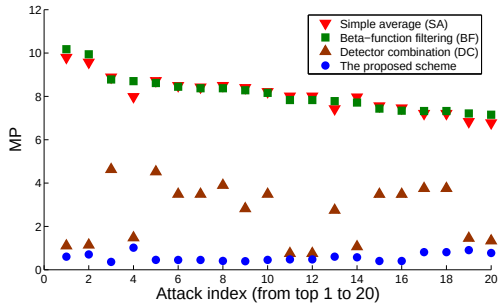

Figure 1.7. Performance comparison in terms of the MP resulting from Top 20 attacks against BF

the attacks do not have a large bias, the beta filter cannot detect unfair ratings. This is why the beta filter has almost the same performance as the simple averaging under some attacks. Therefore, the beta filter scheme, as well as other majority-rule based schemes, is not affective in detecting smart attacks from real human users.

Third, trust establishment plays an important role in the proposed scheme. Although the proposed detectors without trust models can reduce MP value greatly, the performance is still worse than that of our proposed scheme with trust. No matter how good the detectors are, there is a small amount of false alarm. With trust establishment, good users and bad users can be distinguished in several rounds rather than in one shot. In Figure 1.6, we show the trust values of the honest raters and the trust values of the raters inserted by the attackers. We see that the trust values of good raters are much higher than that of unfair raters. This partially explains the good performance of the proposed scheme.

In Experiment 2, we select the top 20 attacks that are strongest against the BF scheme. Figure 1.7 shows the performances of the four schemes. Again, the proposed scheme has the best performance; SA and BF have similar performance; DC can catch most of the unfair ratings but its performance is worse than the proposed scheme with trust.

In Experiment 3, we select 20 attacks that are strongest against the DC scheme. As shown in Figure 1.8, DC still performs much better than SA and BF. There is a big gap between DC and the proposed scheme, which represents the performance advantage

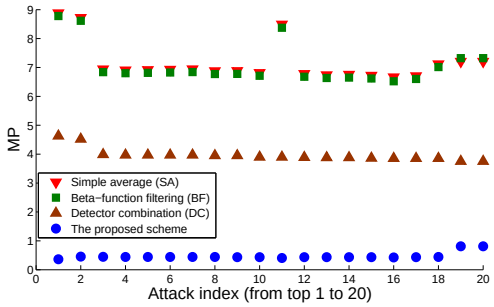

Figure 1.8. Performance comparison in terms of the MP resulting from Top 20 attacks against DC

Manipulation power	Min	Max	Average
Simple average	7.12	9.79	8.10
Beta-function filtering	7.15	10.18	8.14
Detector combination	3.75	4.63	3.96
The proposed scheme	1.83	2.73	2.11

Table 1.1. MP values resulting from top 20 strongest attacks against individual methods

resulting from trust establishment.

In Experiment 4, we show the minimum, maximum and average MP values of each method when they are facing the 20 strongest attacks against them. The advantages of the proposed scheme is clearly shown in Table 1.1. Compared with the majority-rule based methods and simple averaging, the proposed scheme reduces the MP value by a factor of 3 or more.

1.6 Discussion

When deriving the detectors, we assume that colluded profit-driven unfair ratings might have patterns that are somehow different from regular ratings. In the evaluation process, we do not make this assumption. Instead, all the unfair ratings were provided by real human users. In addition, the Poisson arrival assumption is only used to simplify the derivation of the ARC detector, and is not used in performance evaluation. Finally, the ARC detectors only detect rapid changes, and will not be triggered by slow variations in arrival rate. Similarly, the mean change detector is triggered only by rapid mean change, and does not require constant mean in honest ratings.

Also, we would like to point out that trust establishment is mainly for reducing the false alarm rate of the detectors. Even if some honest ratings are wrongly marked as suspicious by the proposed detectors for some reason, the proposed trust establishment mechanism will *correct* this false alarm after more observations are made.

1.7 Conclusion

In this paper, we addressed the problem of detecting and handling unfair ratings in on-line rating systems. In particular, we designed a comprehensive system for integrating trust into rating aggregation process. For detecting unfair ratings, we developed a model error based detector, two arrival rate change detectors, a histogram change detector, and adopted a mean change detector. These detectors cover different types of attacks. A method for jointly utilizing these detectors is developed. The proposed solution can detect dishonest raters who collaboratively manipulate rating systems. This type of unfair raters is difficult to be caught by the existing approaches. The proposed solution can also handle a variety of attacks. The proposed system is evaluated against attacks created by real human users and compared with majority-rule based approaches. Significant performance advantage is observed.

1.8 Acknowledgement

We sincerely thank Jin Ren and Nitish Reddy Busannagari for administrating Rating Challenge Website, and all participants to the Rating Challenge.

List of References

[1] C. Dellarocas, "The digitization of word-of-mouth: Promise and challenges of online reputation systems," *Management Science*, vol. 49, no. 10, pp. 1407–1424, October 2003.

[2] A. Jøsang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decis. Support Syst.*, vol. 43, no. 2, pp. 618–644, 2007.

[3] J. Livingston, "How valuable is a good reputation? a sample selection model of internet auctions," *The Review of Electronics and Statistics*, vol. 87, no. 3, pp. 453–465, August 2005.

[4] P. Resnick, R. Zeckhauser, J. Swanson, and K. Lockwood, "The value of reputation on ebay: A controlled experiment," *Experimental Economics*, vol. 9, no. 2, pp. 79–101, June 2006.

[5] comScore.com, "Press release: Online consumer-generated reviews have significant impact on offline purchase behavior," http://www.comscore.com/press/release.asp?press=1928, November 2007.

[6] J. Meattle, "Digg overtakes facebook; both cross 20 million u.s. unique visitors," compete.com, 2007.

[7] C. Dellarocas, "Strategic manipulation of internet opinion forums: Implications for consumers and firms," *Management Science*, October 2006.

[8] A. Harmon, "Amazon glitch unmasks war of reviewers," The New York Times, February 14 2004.

[9] M. Hines, "Scammers gaming youtube ratings for profit," InfoWorld, http://www.infoworld.com/article/07/05/16/cybercrooks_gaming_google_1.html, May 2007.

[10] J. Brown and J. Morgan, "Reputation in online auctions: The market for trust," *California Management Review*, vol. 49, no. 1, pp. 61–81, 2006.

[11] U. of Rhode Island, "Etan rating challenge," www.etanlab.com/rating.

[12] C. Dellarocas, "Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior," in *Proceedings of the 2nd ACM conference on Electronic commerce*, 2000.

[13] M. Chen and J. Singh, "Computing and using reputations for internet ratings," in *Proceedings of the 3rd ACM conference on Electronic Commerce*, 2001.

[14] A. Whitby, A. Jøang, and J. Indulska, "Filtering out unfair ratings in Bayesian reputation systems," in *Proc. 7th Int. Workshop on Trust in Agent Societies*, 2004.

[15] J. Weng, C. Miao, and A. Goh, "An entropy-based approach to protecting rating systems from unfair testimonies," *IEICE TRANSACTIONS on Information and Systems*, vol. E89-D, no. 9, pp. 2502–2511, September 2006.

[16] S. Kamvar, M. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *Proceedings of 12th International World Wide Web Conferences*, May 2003.

[17] A. Jøsang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decision Support Systems*, vol. 43, no. 2, pp. 618–644, 2005.

[18] R. Zhou and K. Hwang, "Powertrust: A robust and scalable reputation system for trusted peer-to-peer computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 5, May 2007.

[19] S. Ganeriwal and M. Srivastava, "Reputation-based framework for high integrity sensor networks," in *Proceedings of ACM Security for Ad-hoc and Sensor Networks (SASN)*, Washington, D.C., USA, Oct. 2004.

[20] L. Xiong and L. Liu, "Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 7, pp. 843–857, July 2004.

[21] K. Fujimura and T. Nishihara, "Reputation rating system based on past behavior of evaluators," in *Proceedings of the 4th ACM conference on Electronic commerce*, 2003.

[22] "Netflix prize dataset," www.netflixprize.com/download.

[23] A. Jsang and R. Ismail, "The beta reputation system," in *Proceedings of the 15th Bled Electronic Commerce Conference*, June 2002.

[24] Y. Sun, Z. Han, W. Yu, and K. J. R. Liu, "A trust evaluation framework in distributed networks: Vulnerability analysis and defense against attacks," in *Proceeding of the 27th Conference on Computer Communications (INFOCOM'06)*, Barcelona, Spain, April 2006.

[25] S. Buchegger and J.-Y. L. Boudec, "The effect of rumor spreading in reputation systems in mobile ad-hoc networks," in *Proceedings of Wiopt'03*, 2003.

[26] S. Kay, *Fundamentals of Statistical Signal Processing, Volume 2: Detection Theory*. Prentice Hall, 1998.

[27] M. Hayes, *Statistical Digital Signal Processing and Modeling*. John Wiley and Sons, 1996.

[28] Y. Yang, Y. Sun, J. Ren, and Q. Yang, "Building trust in online rating systems through signal modeling," in *Proceedings of IEEE ICDCS Workshop on Trust and Reputation Management*, 2007.

MANUSCRIPT 2

RepTrap: A Novel Attack on Feedback-based Reputation Systems

Abstract

Reputation systems are playing critical roles in securing today's distributed computing and communication systems. Similar to other security mechanisms, reputation systems can be under attack. In this paper, we report the discovery of a new attack, named RepTrap(Reputation Trap), against feedback-based reputation systems, such as those used in P2P file-sharing systems and E-commerce websites(e.g. Amazon.com). We conduct an in-depth investigation on this new attack, including analysis, case study, and performance evaluation based on real data and realistic user behavior models. We discover that the RepTrap is a strong and destructive attack that can manipulate the reputation scores of users, objects, and even undermine the entire reputation system. Compared with other known attacks that achieve the similar goals, the RepTrap requires less effort from the attackers and causes multi-dimensional damage to the reputation systems.

2.1 Introduction

Word-of-mouth, one of the most ancient mechanisms in the history of human society, is gaining new significance in the Internet [1, 2]. The *online reputation systems*, also known as the online feedback mechanisms, are creating large scale, virtual word-of-mouth networks in which individuals share opinions and experiences on a wide range of topics, including products, companies, digital content and even other people. For example, Epinions.com encourages Internet users to rate practically any kind of businesses. Citysearch.com solicits and displays user feedback on restaurants, bars, and performances. YouTube.com recommends video clips based on viewers' feedback and some P2P file-sharing systems[3] do the same for the files shared by the users.

The reputation systems play significant roles in (1) assisting users' decision-

making, (2) encouraging trustworthy behavior, and (3) deterring participation by unskilled or dishonest users. Reputation systems are having increasing influence on purchasing decision of consumers, online digital content distribution, and even political campaigns [1].

Meanwhile, the *manipulation* of such systems is rapidly growing. Firms post biased ratings and reviews to praise their own products or bad-mouth the products of their competitors. Political campaigns promote positive video clips and hide negative video clips by inserting unfair ratings at YouTube.com. There is ample evidence that such manipulation takes place [4]. In February 2004, due to a software error, Amazon.com's Canadian site mistakenly revealed the true identities of some book reviewers. It turned out that a sizable proportion of those reviews were written by the books' own publishers, authors, and competitors [5]. The scammers are creating sophisticated programs that mimic legitimate YouTube traffic and provide automated feedback for videos and other content they wish to promote[6]. Some eBay users are artificially boosting their reputation by buying and selling feedbacks[7]. Collaborative manipulation of feedback based reputation systems is a growing threat. This threat is hurting consumers and will also hurt the business hosting such systems [8].

In the current literature, the research on attacks against reputation systems is still immature. The existing threat models are rather straightforward[9, 10]. The known attacks can be classified according to their attack goals: *self-promoting* (i.e. falsely increasing reputation), *slandering* (i.e. falsely reducing reputation), *whitewashing* (i.e. repairing reputation after bad behaviors), and *denial-of-service* (i.e. making the system wrongly accuse honest users or objects). Attackers achieve the above goals through a variety of methods[9]. Each attacker can acquire multiple identities through a sybil attack[11, 12, 13]. All the identities under the control of the attackers can (1) provide positive feedbacks for self-promoting, (2) provide negative feedbacks for slandering; (3) behave honestly to the objects that the attackers are not interested in or simply register as a new user for whitewashing[14, 15]; and (4) subvert the underlying detection

algorithms to make honest users/objects look suspicious[10].

Whereas current research primarily focuses on individual attack strategies, the intelligent human attackers can create sophisticated attacks by integrating different manipulation strategies. In this paper, we report the discovery of a new attack, named *RepTrap*, against feedback-based reputation systems, and conduct an in-depth investigation on this new attack.

RepTrap is applicable to reputation systems that calculate key reputation scores based on *feedbacks* as well as metrics describing whether users provide honest feedbacks. These metrics are often referred to as *feedback reputation*. Although RepTrap is applicable to a broad range of systems, we choose P2P file-sharing system as the evaluation platform for three reasons. First, there are real trace data[16] and mature models describing user behaviors in P2P networks[17, 18]. Second, P2P networks, such as Kazaa [19], eMule [20] and Maze[16], are becoming the most important infrastructure to share content and services among users in a distributed way. Third, reputation systems play critical roles in P2P file-sharing systems as a method to filter out fake content (pollution) and malicious users [14, 21]. The simulation results have demonstrated that many current reputation systems are highly vulnerable to this new attack. With the same attack effort, RepTrap can significantly increase the attackers' chance of success. Furthermore, this attack can undermine the users' incentive to participate and contribute to the application systems.

As a summary, our *contributions* are (1) discovery of a new and powerful attack against feedback-based reputation systems, which reveals a severe vulnerability in prevalent reputation systems; (2) formal attack strategies to maximize the effectiveness of this attack; and (3) an in-depth study on the impact of the attack in P2P file-sharing systems. The rest of the paper is organized as follows. An overview of feedback-based reputation systems and related work are presented in Section 2. Section 3 presents the RepTrap attack, including basic ideas, case studies and detailed attack methods. Section 4 demonstrates the performance evaluation. Some additional discussions are provided

in Section 5, followed by the conclusion in Section 6.

2.2 Reputation System Review and Related Work

Reputation systems have been used in many application scenarios. Instead of studying a specific reputation system for a specific application, we would like to broaden the scope of this investigation by taking the following 3 steps.

- Step 1: We summarize the common properties of prevalent application scenarios of reputation systems, and build architecture of reputation systems without specifying the algorithms for reputation calculation.

- Step 2: Based on this architecture, we discover the new attack that can undermine the foundation of reputation systems.

- Step 3: To demonstrate the performance of the attack, specific algorithms and application scenario are put back into the picture. We will quantitatively evaluate the attack in a P2P file-sharing system based on real trace data.

Step 1 and 2 will be described in this section. Step 3 will be discussed in Section 2.3 and demonstrated in Section 4.5.

2.2.1 Application Scenarios

Feedback-based reputation systems are used in many online applications as discussed in Section 5.1. These application systems have two common properties.

First, they have reputation mechanisms that provide information to users for assisting their decision-making. For example, P2P file-sharing systems can provide scores representing the quality of files such that users can decide which file to download[3]; product rating systems provide rating scores describing the quality of the products to guide users' purchasing decisions; eBay maintains the reputation scores of sellers/buyers such that one can decide whether to trust a seller/buyer. All above scores can be viewed as reputation, and are referred to as the *primary reputation* in this work.

In addition, the party, whose quality is described by the primary reputation score, is referred to as the *object*. An object in P2P file-sharing, product rating, and user rating system is a file, a product, and a user, respectively. In other words, as the first common property, these application systems *calculate and maintain primary reputation scores describing the quality of the objects.* So in the rest of the paper, the primary reputation is referred to as the *object quality reputation*.

Second, users can provide *feedbacks* about the quality of the objects, and the feedbacks are the primary source to determine the object quality reputation.

It is important to point out that many applications provide services in a *distributed* manner, such as P2P file-sharing systems in which users can directly interact with each other. However, the reputation system can be *centralized*. That is, there are one or several central authorities collecting feedbacks, publishing reputation scores, and detecting abnormalities. The central authority is referred to as Trust Manager (TM). It does not mean the system must have a central server. The central authority can be formed by a group of users. For example, even in a pure distributed P2P file-sharing network, we can build a reputation system with DHT[22]. In this paper, we focus on the usage of RepTrap in centralized reputation systems. Its extension to distributed reputation systems will be investigated in the future work.

As a short summary, the key concepts in the application scenarios that we care about in this paper are: *users, objects, feedback, and object quality reputation*.

2.2.2 Reputation System Architecture

Figure 2.1 shows the basic building blocks of the reputation systems and the relationship among these building blocks.

Object quality reputation: In most reputation systems, the primary goal is to determine the object quality reputation, which will assist the users to select high-quality objects, and assist the system to detect low-quality objects. As illustrated in Figure 2.1, the object quality reputation is determined by feedbacks from users, feedback reputation of users,

and possibly other factors. The algorithm that calculates the object quality reputation from the above factors is referred to as the *QR Algorithm*.

Feedback and feedback reputation: Utilizing feedback has both advantages and disadvantages. As an advantage, feedbacks (if from honest users) can accurately reflect the quality of the objects and therefore allow a fast establishment of object quality reputation. On the other hand, the system must provide incentive to users such that they will provide honest feedbacks. More importantly, the usage of feedbacks leads to feedback-based attacks against reputation systems. In particular, malicious users can provide dishonest feedbacks and mislead the calculation of object quality reputation. More details will be discussed in Section 3.2.1.

The feedback reputation is calculated based on the users' previous feedback behaviors. The feedback behavior is often described by

- the number of honest feedbacks given by this user previously, denoted by G_{fb};
- the number of dishonest feedbacks given by this user previously, denoted by B_{fb};
- time when the feedbacks were given.

The algorithm that calculates user feedback reputation is referred to as the *FR Algorithm*.

Evidence collection: The Evidence Collection (EC) module provides inputs to the QR algorithm and the FR algorithm. The module needs to judge whether a user's feedback is honest and whether an object has high quality.

It is important to point out that the system usually does not know the ground truth about whether an object is high-quality or low-quality. Therefore, the EC often needs to estimate the ground truth. The common practice is to assume that the object quality reputation calculated by the reputation system is accurate. As we will discuss in the next section, this assumption introduces vulnerabilities into the reputation system.

Abnormal detection: This module detects low-quality objects, malicious users, attacks

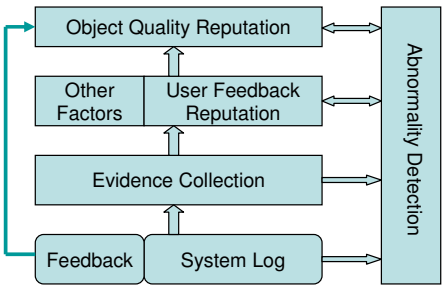

Figure 2.1. Architecture of Reputation System

against application systems, attacks against reputation systems, and any other abnormalities.

Finally, we would like to point out that the current literature already provides ample choices for QR and FR algorithms. Examples include the beta-function based method in [23], the simple summation method in eBay [1], the entropy-based methods in [24]. An overview of more algorithms can be found in [2]. Meanwhile, the research on abnormality detection is still in its early stage.

2.2.3 Related Work

Generally speaking, the basic goal of malicious attacks against a reputation system is to boost or reduce the reputation of certain objects. There is an *arms race* between the attack and defense efforts.

Attack: To boost or reduce the object quality reputation, the most straightforward method is to insert dishonest feedbacks. This is referred to as the bad-mouthing attack[10]. (This attack has other names. For example, in eBay-like systems, ballot stuffing refers to a seller collecting positive feedbacks from faked transactions with friends; bad-mouthing refers to deliberately lowering others' reputation through negative feedbacks[25, 26, 1].) In this paper, we adopt the broader definition of bad-mouthing in [9], and use **RepBad** to denote this attack.

Defense: To defeat the bad-mouthing attack, the reputation system adopts the feedback reputation [27]. Feedback reputation also has different names, such as "overall reputation score of users" in [28] and "trust in raters" in [29].

Attack: If feedback reputation is used, the attackers need to maintain their feedback reputation after providing dishonest feedbacks. The attackers can provide honest feedbacks to the objects that they are not interested in. This is referred to as *reputation recovery*[10] or *self-promoting* [9]. To avoid confusion, we use **RepSelf** to denote this attack in this paper.

Defense: To reduce the effectiveness of feedback reputation self-promoting, the system can limit the maximum number of feedbacks per user in a given time interval. This is reasonable because regular users do not use/evaluate a large number of objects in a short period of time. Therefore, the attackers must balance the usage of the limited number of feedbacks between attack and self-promoting, i.e. either to (1) provide a large number of dishonest feedbacks and have not-so-good feedback reputation or (2) to provide a small number of dishonest feedbacks and maintain a good feedback reputation. By limiting the number of feedbacks per user, the reputation system can suppress either the *impact* of each dishonest feedback or the *number* of dishonest feedbacks.

2.3 RepTrap: a New Attack against Reputation Systems

2.3.1 Basic Ideas

In self-promoting, the attackers only increase their own feedback reputation. How about reducing the honest users' feedback reputation? Can the attackers hurt honest users' feedback reputation and improve their own feedback reputation at the same time? In this paper, we move the attack-defense arms race one step further and propose a new attack strategy, called *Reputation Trap* (RepTrap).

RepTrap Attack: Attackers find some high quality objects that have a small number of feedbacks. Then, attackers provide a large number of negative feedbacks to these objects such that these objects are marked as low-quality by the system. That is, the system makes wrong judgement about the object quality. As a consequence, the system thinks the attackers' negative feedbacks agree with the object quality and the feedbacks from honest users disagree with the object quality. Therefore, the feedback reputation

of the attackers will be increased while the feedback reputation of honest users will be reduced.

RepTrap is applicable under three conditions. *First*, there exist high quality objects with a small number of feedbacks. Many existing applications, such as P2P file-sharing systems and Amazon.com, satisfy this condition. *Second*, the reputation system uses feedback reputation to calculate object quality reputation. *Third*, the system does not know the ground truth about the object quality and has to estimate the ground truth based on the object quality reputation. This is true in many large scale open systems. It is also noted that this condition may not be satisfied in some semi-centralized systems where a large number of experts are hired to evaluate product quality and the expert opinions override users' feedbacks.

As a summary, *the RepTrap attack hurts the feedback reputation of honest users and boosts the feedback reputation of attackers by undermining the system's estimation of object quality*. The RepTrap attack also has other effects, such as hurting good users' incentive to collaboration, which will be discussed in Section 2.5.

2.3.2 Case Studies

To illustrate the basic idea of RepTrap, we create an example in a simplified reputation system. In this section, we first describe this simple reputation system, then present the example, and finally compare the effectiveness of various attacks in this example scenario.

A Simple Reputation System

This simple reputation system takes binary feedbacks. Positive feedback indicates high quality and negative feedback indicates low quality. One user can provide at most one feedback to one object. The beta-function based trust model is adopted to calculate reputation scores in the QR algorithm and FR algorithm. The EC module uses thresholding method. Particularly,

1. The feedback reputation of user u_j is calculated as $R^{fr}(u_j) = \frac{G_{fb}(u_j)+1}{G_{fb}(u_j)+B_{fb}(u_j)+2}$, where G_{fb} and B_{fb} are defined in Section 2.2.2.

2. The quality reputation of object O_k is calculated as a weighted average of feedback values with the weight factors as users' feedback reputation. In particular,

$$R^{qr}(O_k) = \frac{\sum_{u_j \in U^k_{pos}} R^{fr}(u_j)}{\sum_{u_j \in U^k_{pos}} R^{fr}(u_j) + \sum_{u_i \in U^k_{neg}} R^{fr}(u_i)}, \tag{2.1}$$

   where U^k_{pos} and U^k_{neg} denote the set of users who give object O_k positive and negative feedbacks, respectively.

3. Update the G_{fb} and B_{fb} values using the following rules.

   – If $R^{qr}(O_k) \geq threshold$, O_k is marked as a high-quality object; if $R^{qr}(O_k) < threshold$, O_k is marked as a low-quality object.

   – If a user u_j provides a positive (or negative) feedback to an object marked with high (or low) quality, the value of $G_{fb}(u_j)$ is increased by 1. Otherwise, if the feedback and the object quality reputation do not match, $B_{fb}(u_j)$ is increased by 1.

4. If the values of G_{fb} and B_{fb} for any users are changed in step 3, go to step 1.

A Simple Application Scenario

We create a simple scenario with 5 objects: O_1, O_2, O_3, O_4, and O_5, and 3 honest users: u_1, u_2, and u_3. All the objects have high quality. u_1 gives four feedbacks; while u_2 and u_3 each gives two feedbacks.

$$O_1 \leftarrow u_1,\ O_2 \leftarrow u_1,\ \ O_3 \leftarrow u_1,\ O_4 \leftarrow u_1$$

$$O_1 \leftarrow u_2,\ O_2 \leftarrow u_2;\ \ O_1 \leftarrow u_3,\ O_5 \leftarrow u_3$$

where $O_k \leftarrow u_j$ denotes that user u_j provides a feedback for object O_k. The relationship between objects and users is illustrated in Figure 2.2. To concisely describe the status

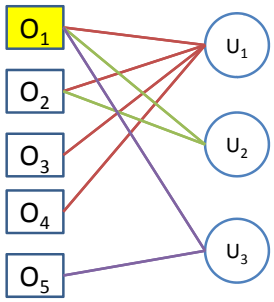

Figure 2.2. Object-User Graph

of an object, we introduce the following notations.

$$R^{fr}_{pos}(O_k) = \sum_{u_j \in U^k_{pos}} R^{fr}(u_j), \quad R^{fr}_{neg}(O_k) = \sum_{u_i \in U^k_{neg}} R^{fr}(u_i)\ .$$

Then, (2.1) becomes

$$R^{qr}(O_k) = \frac{R^{fr}_{pos}(O_k)}{R^{fr}_{pos}(O_k) + R^{fr}_{neg}(O_k)}\ . \tag{2.2}$$

For the example in this section, we set $threshold = 0.3$. And we can then calculate the reputations scores using the algorithms described in Section 2.3.2. When there are no malicious users, the feedback reputation of the honest users is:

$$u_1 : 0.83; \quad u_2 : 0.75; \quad u_3 : 0.75$$

and the object quality reputation is

$$O_1 : 1.00; \quad O_2 : 1.00; \quad O_3 : 1.00; \quad O_4 : 1.00; \quad O_5 : 1.00$$

Thus, all of the objects are marked as high quality.

Goal of Attacker

In this example, the **attacker's goal** is to make the system mark O_1 as a low-quality object while minimizing the *attack effort*.

The attack effort is described by two values (K, F). Here, K is the number of user IDs under the attacker's control. These user IDs are also referred to as **malicious users**.

F is the total number of feedbacks from the malicious users. Higher are the K and F values, more resources the attacker needs to spend. For an attacker, acquiring user IDs is usually harder than providing feedbacks, for two reasons. First, many systems limit the number of user IDs per IP address. Second, there are many techniques to defense against the Sybil attacks [12, 13] such that the cost for an attacker to control many user IDs increases greatly. Therefore, reducing the K value has higher priority than reducing the F value, from the attacker's point of view. Thus, the attacker first finds the minimal K value, and then finds the minimum F value given the minimal K value.

The attacker's goal is then translated into finding a way to provide feedbacks such that (1) the K value is minimized and (2) the F value is minimized given the minimal K value, under the constraint

$$R^{qr}(O_1) < threshold \tag{2.3}$$

Comparison among Different Attack Strategies

In this subsection, we compare RepTrap with RepBad and RepSelf, which are defined in Section 3.2.1. From (2.2) and (2.3), we derive the necessary and sufficient condition for making the reputation system mark an object (O_k) as low quality. The condition is:

$$R^{fr}_{neg}(O_k) > R^{fr}_{pos}(O_k) \cdot \frac{1 - threshold}{threshold} \tag{2.4}$$

In this case, if the attacker wants O_k to be marked as low-quality when there are no previous negative feedbacks given to O_k, the attacker needs to make some malicious users provide negative feedbacks to O_k and the summation of the malicious users' feedback reputation needs to be higher than $\frac{7}{3}R^{fr}_{pos}(O_k)$. We define the value of $\frac{7}{3}R^{fr}_{pos}(O_k)$ as the *hardness* value of an object. In this example, the hardness values of the objects are:

$$O_1 : 5.44; \;\; O_2 : 3.69; \;\; O_3 : 1.94; \;\; O_4 : 1.94; \;\; O_5 : 1.75$$

RepBad: In the RepBad attack, the feedback reputation for each malicious user is 0.5, and the hardness of O_1 is 5.44. Since $5.44/0.5 = 10.88$, the attacker needs at least 11 malicious users to successfully attack O_1. In this case, the attack power is $(N = 11, K = 11)$.

RepSelf: In the RepSelf attack, the malicious users first provide honest feedbacks to the objects that they do not want to attack. In this example, they can provide positive feedbacks to O_2, O_3, O_4 and O_5, and accumulate their feedback reputation up to $(4 + 1)/(4 + 2) = 0.83$. Since $5.44/0.83 = 6.55$, at least 7 malicious users are needed to successfully attack O_1.

Not all 7 malicious users need to provide 4 honest feedbacks. After enumerating all possible ways to perform self-promoting, we find that the malicious users need to provide at least 25 feedbacks. To achieve this,

- 3 malicious users provide positive feedbacks to 2 objects, and their feedback reputation becomes $(2 + 1)/(2 + 2) = 0.75$.
- 4 malicious users provide positive feedbacks to 3 objects, and their feedback reputation becomes $(3 + 1)/(3 + 2) = 0.8$.

Then, the summation of the feedback reputation of malicious users is $0.75 \times 3 + 0.8 \times 4 = 5.45 > 5.44$, which means they can successfully attack O_1. Thus, total 18 honest feedbacks and 7 dishonest feedbacks are given. In this case, the attack power is $(N = 7, K = 25)$.

RepTrap: In the RepTrap attack, the malicious users provide dishonest feedbacks to unpopular high-quality objects. If a high-quality object is marked as low quality by the reputation system, this object is turned into a **"trap"**, which will mislead the feedback reputation of good users who give honest feedback to this trap.

It is difficult to find the optimal way to conduct the RepTrap attack. Instead, we just show one method to successfully attack O_1 using RepTrap. The attack power of this method is $(N = 4, K = 18)$.

	Initial state	After 1^{st} round	After 2^{nd} round	After 3^{rd} round	After 4^{th} round
$R^{fr}(u_1)$	0.83	0.83	0.67	0.50	0.33
$R^{fr}(u_2)$	0.75	0.75	0.75	0.75	0.50
$R^{fr}(u_3)$	0.75	0.50	0.50	0.50	0.50
$R^{fr}(X_1)$	0.50	0.67	0.75	0.80	0.83
$R^{fr}(X_2)$	0.50	0.67	0.75	0.80	0.83
$R^{fr}(X_3)$	0.50	0.67	0.75	0.80	0.83
$R^{fr}(X_4)$	0.50	0.67	0.67	0.67	0.75
$\sum_{i=1}^{4} R^{fr}(X_i)$	2.00	2.68	2.92	3.07	3.24
$Hardness(O_1)$	5.44	4.86	4.47	4.080	**3.11**
$Hardness(O_2)$	3.69	3.69	3.31	**2.92**	#
$Hardness(O_3)$	1.94	1.94	**1.56**	#	#
$Hardness(O_4)$	1.94	**1.94**	#	#	#
$Hardness(O_5)$	**1.75**	#	#	#	#

Table 2.1. Reputation and hardness values in case study

1. The initial feedback reputation of the four malicious users is 0.5. Since $0.5 \times 4 = 2$ is larger than the hardness of O_5 (i.e. 1.75), they can turn O_5 into a trap by providing negative feedbacks to O_5. This has three consequences. First, the feedback reputation of the malicious users increases to $(1+1)/(1+2) = 0.67$. Second, the feedback reputation of good user u_3 is reduced to $(1+1)/(2+2) = 0.5$. Third, the value of $R^{fr}_{pos}(O_1)$ and the hardness of O_1 is reduced. The results are shown in the second column in Table 2.1, and we use X_1, X_2, X_3 and X_4 to denote the four malicious users.

2. After the first step, three malicious users can turn O_4 into a trap because the summation of their feedback reputation is larger than the hardness of O_4, i.e. $0.67 \times 3 > 1.94$. Then, the feedback reputation of these three malicious users are increased to $(2+1)/(2+2) = 0.75$, and the feedback reputation of the honest user u_1 is reduced to $(3+1)/(4+2) = 0.67$. The hardness of O_1, O_2 and O_3 is also reduced, because their hardness depends on the feedback reputation of u_1. See the third column in Table 2.1 for details.

3. The three malicious users in the second step can further turn O_3 into a trap because $0.75 \times 3 = 2.25$ is larger than the current hardness of O_3. Then, the feedback reputation of these three malicious users are increased to $(3+1)/(3+2) = 0.8$, the feedback reputation of u_1 is reduced to $(2+1)/(4+2) = 0.5$, and the hardness of O_1 and O_2 continues to drop.

4. Similarly, all four malicious users can turn O_2 into a trap. Then, the feedback reputation of malicious users becomes 0.83 or 0.75, the feedback reputation of u_1 becomes 0.33, and the hardness of O_1 is reduced to 3.11.

5. Finally, the summation of feedback reputation of the four malicious users is $0.83 \times 3 + 0.75 = 3.24$, which is larger than 3.11. This means that the malicious users can now make the system mark O_1 as low quality. In total, the malicious users give $4 + 3 + 3 + 4 + 4 = 18$ feedbacks.

In this case study, RepTrap reduces the requirement on the number of malicious users by 64% when compared with RepBad, and by 43% when compared with RepSelf. RepTrap also requires 28% less feedbacks than RepSelf. In other words, RepTrap is a much stronger attack than the existing ones in manipulating the reputation system.

It is noted that malicious users provide many negative feedbacks in RepTrap. In systems such as eBay, this can be detected because positive feedbacks overwhelm negative feedbacks in such systems. However, in many systems, users give the negative feedbacks much more frequently than the positive feedbacks due to lack of incentive [1, 2]. In general, RepTrap cannot be effectively detected simply based on the number of negative feedbacks provided by individual user IDs.

2.3.3 Basic Concepts in RepTrap

As introduced in Section 2.3.1, the core idea of RepTrap is to degrade the feedback reputation of honest users by undermining the system's estimation on object quality. In Section 2.3.2, a case study shows the effectiveness of RepTrap. To fully understand this

new attack, we develop a *systematic procedure* to conduct the RepTrap attack. Whereas the formal procedure will be described in Section 2.3.4, we introduce the basic concepts in this subsection.

Object-User Graph

Figure 2.2 is created to represent the relationship between objects and users. On the left side of the graph, each vertex represents an object. On the right side of the graph, each vertex represents a user. Each object has a popularity score (a, b), where a is the number of positive feedbacks and b is the number of negative feedbacks given to this object. If a user provides feedback to an object, a link is created between this user and the object. The link has weight 1, if the feedback from the user agrees with the quality of the object which is marked by the reputation system. Otherwise, the weight of the link is -1. This graph is referred to as the *object-user graph*.

Correlation Between Objects

Based on the object-user graph, we can calculate the correlation between object O_i and O_j, denoted by $Cor(O_i, O_j)$. This correlation describes how O_j's reputation is affected if O_i becomes a trap. $Cor(O_i, O_j)$ is calculated in three steps.

1. Between O_i and O_j, find all the paths with length 2. For example, if user u_k provides feedbacks to both O_i and O_j, there exists a path $O_i - u_k - O_j$ and the path length is 2.

2. For each path, calculate the $path\_effect$ value as the product of the two links' weights. Thus, the $path\_effect$ value can be either 1 or -1. When $path\_effect = 1$, the quality reputation of O_j will move opposite to its original value if O_i is turned into a trap. That is, high object reputation will become lower, and low object reputation will become higher. When $path\_effect = -1$, the object reputation will be maintained. That is, the effect of this path is to make high object reputation become higher and low object reputation become lower.

$R^{qr}(O_i)$	u_k's feedback to O_i	weight of link $O_i - u_k$	if O_i becomes trap, change in $R^{fr}(u_k)$	$R^{qr}(O_j)$	u_k's feedback to O_j	weight of link $O_j - u_k$	change in $R^{qr}(O_j)$ due to change in $R^{fr}(u_k)$	maintain original $R^{qr}(O_j)$?	path effect value
high	positive	1	reduced	high	positive	1	reduced	opposite	1
high	positive	1	reduced	low	negative	1	increased	opposite	1
high	positive	1	reduced	high	negative	-1	increased	maintain	-1
high	positive	1	reduced	low	positive	-1	reduced	maintain	-1
high	negative	-1	increased	high	positive	1	increased	maintain	-1
high	negative	-1	increased	low	negative	1	reduced	maintain	-1
high	negative	-1	increased	high	negative	-1	reduced	opposite	1
high	negative	-1	increased	low	positive	-1	increased	opposite	1

Table 2.2. Truth table for verifying that $path\_effect$ value represents the effects of a trap upon O_j's object reputation.

The truth table (Table 2.2) supports the above statements, which will be explained later.

3. Let N_1 denote the number of paths with the positive $path\_effect$ values, and N_2 denote the number of paths with negative $path\_effect$ values. Then, the correlation is calculated as

$$Cor(O_i, O_j) = N_1 - N_2. \tag{2.5}$$

It is noted that $Cor(O_i, O_j)$ can be used for trap selection. If $Cor(O_{i_1}, O_j) > Cor(O_{i_2}, O_j)$, turning O_{i_1} into a trap is more likely to have a greater impact on the reputation of O_j than turning O_{i_2} into a trap. In the example shown in Figure 2.2, $Cor(O_2, O_1) = 2$ and $Cor(O_5, O_1) = 1$.

To understand table 2.2, we examine the path $O_2 - u_1 - O_1$ shown in Figure 2.2 with the elements in the first row of Table 2.2 explained one by one, from left to right.

1. Initially, O_2 is marked as *high* quality by the system;
2. u_1 gives an honest feedback to O_2 which is *positive*;
3. Since u_1's feedback agrees with O_2's quality reputation, the weight of link $O_2 - u_1$ is *1*;
4. If O_2 is turned into a trap, the system will mark O_2 as low quality and also think u_1 gives a wrong feedback, which leads to a *reduction* in u_1's feedback reputation;

5. Initially, O_1 is marked as *high* quality by the system;
6. u_1 gives an honest feedback to O_1 which is *positive*;
7. Since u_1's feedback agrees with O_1's quality reputation, the weight of link $u_1 - O_1$ is *1*;
8. Since the feedback reputation of u_1, who gives a positive feedback, is reduced (see column 4), the object quality reputation of O_1 could also be *reduced*;
9. Making O_2 into a trap will make a high object quality reputation lower, that is, the object quality reputation of O_1 moves toward the *opposite* direction.
10. The $path\_effect$ value which is the product of two link weights, is *1*.

The eight rows in Table 2.2 represent all different cases. It is seen that the $path\_effect$ value agrees with the direction of reputation change. It is a good indicator showing the effect of the traps.

Object-targeted RepTrap and User-targeted RepTrap

The RepTrap attack can mislead the object quality reputation as well as the feedback reputation of users.

- When the primary attack goal is to mislead the quality reputation of one or multiple specific objects, we call it the *object-targeted RepTrap*.
- When the primary attack goal is to mislead the feedback reputation of one or multiple specific users, we call it the *user-targeted RepTrap*.

Using the object-user graph, we can unify these two types of RepTrap attacks. In the user-targeted RepTrap, we create a *virtual object*. This virtual object is linked to all the users who are the target of the attacker. The weight of the links is 1. It is easy to see that reducing the feedback reputation of the users is equivalent to reducing the quality reputation of the virtual object. By creating the virtual object, we can convert

the user-targeted RepTrap attack into the object-targeted RepTrap attack. Therefore, in the rest of the section 2.3, we only describe how to conduct the object-targeted RepTrap.

Knowledge Level of Attackers

In different application scenarios, the attackers may have different amount of knowledge about the reputation systems, which affects their ways to conduct the RepTrap attack. In this subsection, we discuss a few typical cases about the attackers' knowledge level.

The attacker wants to know all of the following information in the reputation system, including

- $info_1$: specific algorithms and parameters (especially thresholds) used in the reputation system
- $info_2$: the information in the object-user graph
- $info_3$: feedback reputation of malicious users
- $info_4$: feedback reputation of other users

First of all, we cannot assume that the algorithms used in the reputation systems are secrets. In other words, the security and robustness of the reputation system should not rely on the secrecy of the algorithms. This is a general principle in security research. Therefore, we assume that the attacker knows the algorithms in the reputation system.

Second, we assume that the attacker can judge whether an object has been marked as low-quality by the system. This assumption is reasonable because the system must take some actions to deal with low-quality objects, such as removing them from the system or ranking them extremely low compared to other normal objects. Based on this assumption, the attacker can insert positive and negative feedbacks to a newly published object, and examine how the reputation changes and when this object is detected as low-quality. By doing this for multiple times, the attacker can estimate the detection

thresholds. Therefore, it leads to another assumption that the attacker can estimate the parameters in the algorithms, e.g. $threshold$ in (2.3).

Third, in many reputation systems, the information about who provides what feedbacks to which objects is open to the public. In addition, the object quality information is obviously known to all users. In those systems such as Epinion, the object-user graph is known to the attacker. Of course, there are systems that do not publish details about the feedback information.

Furthermore, since the attacker knows the algorithms and the behavior of malicious users, the attacker surely knows the feedback reputation of malicious users. Some systems publish information about the trustworthiness of users, which is directly related to the users' feedback reputation. Examples include Epinion and Amazon. In some other systems, the attacker can easily dig out the history of users giving feedbacks. Knowing this history and the algorithms, the attacker can estimate the feedback reputation of good users. Of course, it is possible that the attackers cannot obtain this information in some systems.

Based on the above discussion, we define three knowledge levels:

- Knowledge Level 1 (KL1): the attacker knows $info_1$ and $info_3$.
- Knowledge Level 2 (KL2): the attacker knows $info_1$, $info_2$ and $info_3$.
- Knowledge Level 3 (KL3): the attacker knows $info_1$, $info_2$, $info_3$, and $info_4$.

In Section 2.3.4, we will first present the specific attack procedure for the KL3 case, and then extend the procedure to KL2 and KL1 cases.

2.3.4 RepTrap Attack Procedure

Before we give a formal RepTrap Attack procedure, some important terminologies and notations are summarized.

- *Object Correlation* between object O_i and O_j is denoted by $Cor(O_i, O_j)$ (see Section 2.3.3).

- *Malicious Users* are the user IDs under the control of the attacker. We use m_i to denote a malicious user.

- *Attack Capability of a Malicious User* describes the impact of the feedback from this particular user. Let a_i denote the attack capability of the malicious user m_i. For example, we can choose a_i as the feedback reputation of m_i.

- *Attack Capability of a Set of Malicious User* describes the overall impact when every user in this set gives feedback to the same object. Let $A(S)$ denote the attack capability of user set S. The simplest way to calculate $A(S)$ is $A(S) = \sum_{i:m_i \in S} a_i$.

- *Hardness Value* of an object describes the difficulty level in turning this object into a trap. Let $H(O_k)$ denote the hardness of object O_k. When the object quality reputation is calculated using (2.2), one reasonable way to calculate $H(O_k)$ is

$$H(O_k) = R^{fr}_{pos}(O_k) \cdot \frac{1 - threshold}{threshold} - R^{fr}_{neg}(O_k), \tag{2.6}$$

  where $R^{fr}_{pos}(O_k)$ (or $R^{fr}_{neg}(O_k)$) is the summation of feedback reputation of the users who have provided positive (or negative) feedbacks to O_k. In the case of KL3, the attacker can calculate $H(O_k)$ using (2.6).

- *Malicious User Set*, denoted by S_m, contains all the malicious users that are under the control of the attacker.

- *Target Object Set*, denoted by S_t, is the set of objects that are the attacker's target. That is, the attacker wants to reduce or boost the reputation of the objects in S_t.

- *Related Object Set*, denoted by S_r, contains the objects that have positive correlation with at least one object in S_t. In other words, turning an object in S_r into a trap will help to attack the objects in S_t.

- *Candidate Object Set*, denoted by S_c, contains the objects that satisfy two conditions:

- They are marked as high-quality objects.
- The objects' hardness values are smaller than the attack capability of the malicious user set. That is, if $O_k \in S_c$, then $H(O_k) < A(S_m)$.

In other words, the attacker only turns high-quality objects into traps, and the attacker does not try to create a trap unless the attacker has enough attack capability. Note that all traps are marked as low-quality. Since regular users usually do not use low-quality objects, traps will not receive many new feedbacks from honest users. On the other hand, if the system marks a low-quality object as high-quality, this object will be used by many users and quickly receives a large number of negative feedbacks. The system will correct its wrong judgement quickly.

- *Effectiveness Value* of an object describes how much influence the attacker can have on the target set if the attacker turns this particular object into a trap. The effectiveness value of the object O_k can be calculated as

$$E(O_k) = \sum_{j:O_j \in S_t} Cor(O_k, O_j) \tag{2.7}$$

- *Feedback Allocation* describes the specific behavior of the malicious users. From the attacker's point of view, *the most challenging job is to determine how many negative feedbacks should be given to which objects in order to create traps*, i.e. how to allocate the dishonest feedbacks. We introduce the notation

$$FA(S_c; S^m_{neg}(O_k) \text{ for } k : O_k \in S_c)$$

which means that the malicious users in set $S^m_{neg}(O_k)$ provide negative feedbacks to O_k, and O_k is in the candidate set S_c.

- *Round* is the time unit in which the attacker can adjust the feedback allocation. Without losing generality, one malicious user ID can provide up to one feedback in each round. The concept of round simplifies the description of the RepTrap attack procedure.

The core task of the RepTrap attack is to determine the feedback allocation in each round such that the quality reputation of the objects in the target set (S_t) is affected most. One round of the RepTrap attack is conducted in 9 steps.

In **Step 1**, construct the related object set (S_r) based on the definition given previously.

In **Step 2**, calculate the hardness value for each object in S_r, i.e. $H(O_j)$, for $j : O_j \in S_r$.

In **Step 3**, calculate the attack capability of each malicious user, as well as the malicious user set, i.e $A(S_m)$.

In **Step 4**, construct the candidate object set (S_c) by comparing candidate $A(S_m)$ and the hardness values of the objects in S_r.

In **Step 5**, for each object in the candidate set (S_c), calculate its correlation with all other objects in the target object set (S_t).

In **Step 6**, calculate the effectiveness value for each object in S_c, i.e. $E(O_j)$, for $j : O_j \in S_c$.

In **Step 7**, determine the feedback allocation in this round by solving the following optimization problem.

$$\begin{aligned} \text{inputs:} \quad & S_c; E(O_j) \text{ and } H(O_j) \text{ for } j : O_j \in S_c; a_i, m_i \in S_m \\ \text{outputs:} \quad & \text{feedback allocation}, i.e. \\ & FA(S_c; S^m_{neg}(O_j) \text{ for } O_j \in S_c) \end{aligned}$$

The optimization problem is:

$$\max_{FA(.)} \sum_{j:\ S^m_{neg}(O_j) \neq \phi} E(O_j) \tag{2.8}$$

under constraints:

$$A(S^m_{neg}(O_j)) > H(O_j) \ \text{ if } S^m_{neg}(O_j) \neq \phi \tag{2.9}$$

$$S^m_{neg}(O_j) \cap S^m_{neg}(O_i) = \phi, \forall i, j : O_i \in S_c, O_j \in S_c, i \neq j \tag{2.10}$$

$$\bigcup_{j:\ O_j \in S_c} S^m_{neg}(O_j) \subseteq S_m \tag{2.11}$$

Here, (2.8) states that the attacker finds the feedback allocation, which maximizes the summation of the effectiveness values of the objects that will be turned into traps. (2.10) and (2.11) state that one malicious user can provide at most one feedback in one round. The constraint (2.9) is for ensuring the success of creating traps. That is, if the attacker provides negative feedbacks to an object, the attacker aims to turn the object into a trap in this round. In practice, we often overestimate the hardness value because it is possible that some good users provide honest feedbacks to the object in S_c during the process of the RepTrap attack. Based on the previous feedback history, the attacker can estimate the number of new feedbacks from honest users, and adjust the hardness value estimation accordingly.

In **Step 8**, the attacker inserts feedbacks according to the feedback allocation calculated in step 7.

In **Step 9**, at the end of each round, the attacker checks whether the traps are successfully created. If $O_i \in S_c$ is turned into a trap, set the trap indicator T_i as 1. (The default T_i value is 0). If O_i is not successfully turned into a trap for some reasons, O_i is put into the candidate set in the next round. This might happen only with a very small probability.

After one round the attack, the attacker can go back to step 1 and do more rounds, until the ending condition is met.

After reducing the good users' feedback reputation and enhancing their own feedback reputation through the RepTrap attack, the attacker is ready to attack the objects in the target set S_t. According to the specific attack goals, the attacker provides either positive or negative feedbacks to the objects in S_t.

In this 9-step procedure, two issues are not specified: the ending condition and the solution to the optimization problem in step 7. Next, we address these two issues.

The **ending condition** depends on the specific goal of attacker. Here are two examples.

- If the attacker's goal is to reduce or boost the quality reputation of the objects in

S_t as much as possible, the ending condition is

  - the candidate set S_c is empty in a new round.

  This ending condition means that no more traps can be created. In this case, the attacker creates as many traps as possible before giving dishonest feedbacks to the objects in S_t.

- If the attacker's goal is to make the system mark some high-quality object as low-quality, the ending condition is

  - the candidate set S_c is empty;
  - or the attacker estimates that he can achieve the attack goal given the users' feedback reputations in a new round.

  If the attacker has knowledge KL3, he can easily perform this estimation. If the attacker has knowledge KL1 and KL2, he may try to attack the objects in S_t and see how the object reputation changes.

The **optimization** problem in step 7 is NP-hard. Therefore, we developed a sub-optimal solution as shown in Procedure 4.

To further enhance the power of RepTrap, we introduce the concept of *enhancement round*. In the enhancement round, the attacker finds an object O_e that is not in S_r. Then, the malicious users give honest feedback to O_e. This will not affect the quality reputation of the objects in the target set, but can increase the feedback reputation of malicious users, as well as their attack capability. Then the malicious users may turn more related objects into traps. In fact, the malicious users can also turn O_e into a trap, but it is easier to give honest feedback.

So after the ending condition of the regular RepTrap rounds is met, the attacker checks whether the hardness of the objects in the related candidate set S_r can be reached if the feedback reputation of the malicious users is increased. If not, there is no room

Procedure 2 Feedback Allocation Algorithm

1: Set $S^m_{neg}(O_k) = \phi$ for $k: \ O_k \in S_c$.
2: Let S_a denote the set of unused malicious users. Set $S_a = S_m$.
3: Let S_o denote the set of remaining objects that can be turned into traps. Set $S_o = S_c$.
4: Calculate the benefit value as $B_k = E(O_k)/H(O_k)$, for $k: \ O_k \in S_c$.
5: **while** $A(S_o)$ is not empty **do**
6: From S_o, select the object that has the largest benefit value. Assume that this object is O_r
7: From S_a, select a set of the malicious users, denoted by S_s, such that $A(S_s) > H(O_r)$ and the number of malicious users in S_s is minimized.
8: **if** the selection of S_s is successful **then**
9: set $S^m_{neg}(O_r) = S_s$
10: remove S_s from S_a
11: **end if**
12: Remove O_r from S_o
13: Remove the objects whose Hardness is not less than $A(S_a)$ from S_o
14: **end while**

for improvement. Otherwise, the attacker runs a few enhancement rounds to boost the feedback reputation of malicious users, which will improve their attack capability. Then, the attacker conducts regular RepTrap rounds until the ending condition is met again. The usage of the enhancement rounds can slightly improve the effectiveness of RepTrap.

2.3.5 RepTrap Attack for KL1 and KL2 Cases

The RepTrap attack procedure in Section 2.3.4 is obviously suitable for the case that the attacker has KL3. In this section, we briefly discuss how to conduct RepTrap for the KL1 and KL2 cases.

When the attacker does not know who provide feedbacks to the target set S_t (i.e. KL1), it is difficult to determine an effective strategy for creating traps. In this case, the attacker can simply try to create as many traps as possible. Note that creating traps not only make the objects in S_t easier to attack, but also make the system publish wrong object quality reputation.

When the attacker does not know the feedback reputation of good users, the attacker cannot calculate the hardness value $H(O_k)$ accurately. Instead, the attacker needs to

estimate $H(O_k)$. For example, the attacker can estimate $H(O_k)$ as

$$\hat{H}(O_k) = N(O_k) \cdot R_{avg}^{fr} \cdot \frac{1 - threshold}{threshold}, \tag{2.12}$$

where $N(O_k)$ is the number of users who have provided feedbacks to O_k (excluding the malicious users), and R_{avg}^{fr} is the average feedback reputation of good users. In most systems, $N(O_k)$ is known and the attacker only needs to estimate R_{avg}^{fr}. To estimate R_{avg}^{fr}, the attacker can use some additional user IDs that provide honest feedbacks to some other objects, just like good users. These additional users' feedback reputation can be used as a guideline for estimating R_{avg}^{fr}. Usually, R_{avg}^{fr} should be overestimated to ensure the success of creating traps.

2.4 Performance Evaluation

2.4.1 Experiment Description

As discussed in Section 5.1, we evaluate the performance of RepTrap attack in P2P file-sharing systems. We build a simulator using models similar to [17] and the parameters from Maze [16]. Maze is a popular P2P file-sharing system with more than 2 million registered users and 40,000 users online at peak time. In the simulation, there are 1,000 good users and 100 high quality files. Users follow Poisson process to download files and an important Zipf parameter, which is 0.676, is used to drive the downloading frequency. While another Zipf parameter, which is 0.890, is used to drive the popularity of the files [17]. Without loss of generality, we assume good users always provide honest feedbacks after successfully file downloads. (Of course, we can assume that good users provide honest feedbacks with a certain probability, but this will not have meaningful impact on the simulation results.) Each user can give no more than one feedback to a file. The reputation system model in the simulator is the same as that in Section 2.3.2.

We compare RepTrap with two other schemes: RepBad and RepSelf, which are described in Section 2.3.2. The attacks are performed in rounds and 50 malicious users are added. In one round of attack, each malicious user can only provide one feedback. With the RepBad strategy, malicious users attack the target files by providing false feed-

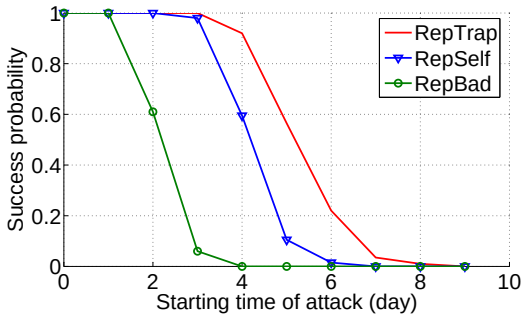

Figure 2.3. Success probability when attacking the 1^{st} popular file

backs in the first round. With the RepSelf strategy, malicious users give honest feedbacks to non-targeted files and gain feedback reputation. After getting enough feedback reputation, they provide false feedbacks to the target files. With the RepTrap strategy, malicious users create traps first and then attack the target files.

2.4.2 Attack model 1: Targeting a popular file

In a P2P file-sharing system, the popularity of files is driven by the Zipf parameter. More popular is a file, more feedbacks from good users are given to this file, and therefore more difficult for malicious users to conduct the attack. In this experiment, We attack the 1^{st} (i.e. top 1) and the 10^{th} (i.e. top 10) popular files and aim to make the system identify them as low quality files (i.e. faked files or pollution).

If the malicious users start an attack at the beginning of the simulation, the attack is much easier to succeed. This is because there are few feedbacks for each file and even a small number of false feedbacks can easily slander a good file. As the time goes, more feedbacks are provided to the files and the attack gets harder and harder to succeed with limited number of malicious users. We therefore change the starting time of the attack to show the attack performance measured by success probability. For each starting time, we conduct 1,000 simulations and success probability is calculated as the number of successful attacks divided by 1000.

Figure 2.3 and 2.4 are for the scenarios that malicious users attack the 1^{st} and the 10^{th} popular files respectively. The horizontal axis is attack starting time and the

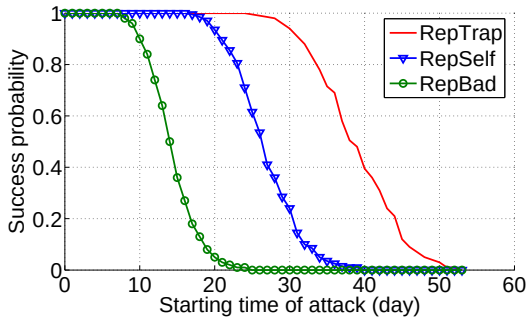

Figure 2.4. Success probability when attacking the 10^{th} popular file

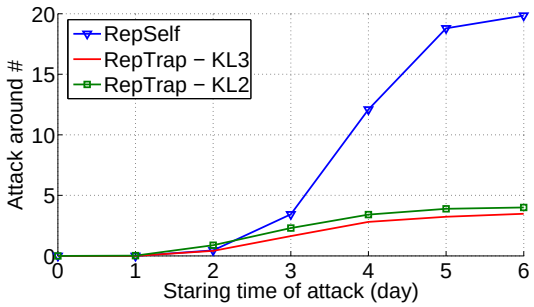

Figure 2.5. The number of rounds needed to successfully attack the 1^{st} popular file

vertical axis is the success probability. It is obvious that the proposed scheme has the best performance and RepBad has the worst performance. RepBad can only successfully attack the 1^{st} popular file within 2 days and the 10^{th} popular file within 10 days after the file is published. After that time, the success probability decreases sharply. RepSelf has better success probability than RepBad because in RepSelf malicious users can improve their feedback reputation and have more attack power. However, the performance of RepSelf is still much worse than RepTrap. For example, when RepSelf has only 10% chance of success, RepTrap has 60% and 80% of success probability in Figure 2.3 and 2.4 respectively. These results clearly show the advantage of reducing honest users' feedback reputation, which can only be achieved in RepTrap.

In the above experiment, both RepTrap and RepSelf need several attack rounds for the malicious users to gain feedback reputation. To compare the efficiency of RepTrap and RepSelf, we examine the *round number*, which is the number of rounds needed to gain enough feedback reputation to successfully attack a file. Please note that in

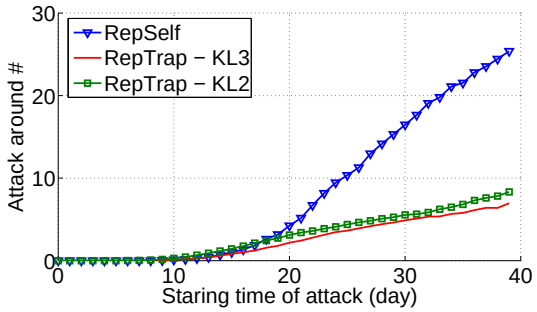

Figure 2.6. The number of rounds needed to successfully attack the 10^{th} popular file

each attack round, 50 malicious users provide 50 feedbacks (each user provides one feedback). The number of attack rounds therefore can represent the total amount of feedbacks provided by the malicious users.

Figure 2.5 and Figure 2.6 show the number of rounds needed to successfully attack the 1^{st} and the 10^{th} files, respectively. Three schemes are compared: RepSelf, RepTrap in KL3 case, and RepTrap in KL2 case. Recall that KL2 and KL3, defined in Section 2.3.3, represent different knowledge levels of the attacker. In KL2, the attacker has to estimate the good users' feedback reputation that affects the hardness value calculation. We assume it overestimates the hardness values of the objects by 20% in KL2. Compared with RepSelf, both RepTrap methods need much less number of rounds. This means that the RepTrap attackers spend much less resources. Compared with the KL3 case, the KL2 case needs slightly more rounds. It indicates that imperfect knowledge about the good users' feedback reputation will not largely reduce the power of the RepTrap attack.

2.4.3 Attack model 2: Blind attack

In this experiment, we consider the KL1 case, in which malicious users know neither the object-user graph nor the good users' feedback reputation. In this case, RepTrap does not target any specific files or users. Instead, it aims to disturb the entire reputation system by creating as many traps as possible. The malicious users will continue to make traps, starting from the least popular files to more popular files, until they cannot make

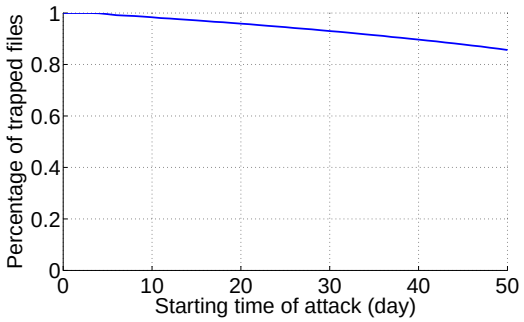

Figure 2.7. Maximum percentage of files turned into traps by 50 malicious users

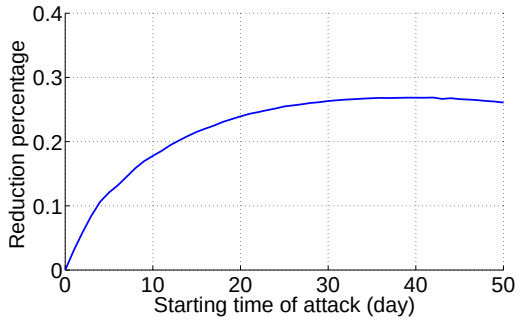

Figure 2.8. Reduction percentage of the average of feedback reputation for good users

any more traps.

Figure 2.7 shows the number of traps created by the 50 malicious users with different attack starting time. It is seen that at the beginning, all files can be turned into traps. The trap number decreases gradually with the starting time. However, even after day 50, more than 80% of the files can be trapped. This is because the popularity of the files in P2P file-sharing systems is driven by the power law distribution, so there are a large number of files downloaded by only a few users. From Figure 2.3, 2.4 and 2.7, we can conclude that RepTrap is a very powerful attack, which can attack most of the files no matter the starting time. The attack performance is only sensitive to the starting time when the attack targets very popular files.

Let R_{avg}^{wo} and R_{avg}^{w} denote the average of good users' feedback reputation without and with the RepTrap attack. In Figure 2.8, the vertical axis is *reduction percentage*, defined as $(R_{avg}^{wo} - R_{avg}^{w})/R_{avg}^{wo}$. The horizontal axis is still the attack starting time. There observations are made.

First, at the very beginning, the reduction percentage is very low. This is because honest users have not given many feedbacks. The attacker cannot hurt a user's feedback reputation if this user does not provide feedbacks.

Second, as time goes on, more feedbacks are provided by honest users and the traps are more powerful to reduce honest users' feedback reputation. Therefore, we see that the reduction percentage gradually increases to 28% within 30 days.

Third, after day 30, the reduction percentage is not changing any more. This is because making traps becomes more difficult. Recall that it gets harder to attack the 10^{th} popular file after day 30 (see Figure 2.4). Considering **there are 1,000 good users and only 50 malicious users** in the simulation, 28% reduction in average feedback reputation of all honest users is a big achievement. In practice, the attacker can control more user IDs as the time goes on, and therefore can even achieve better performance.

2.5 Discussion

Through analysis and simulations, we have seen that RepTrap can severely degrade the quality reputation of objects and feedback reputation. Comparing RepBad and RepSelf, RepTrap needs the fewest user IDs to achieve the attack goal.

In fact, RepTrap has even broader influence. We use the P2P file-sharing system as an example to illustrate it.

First, RepTrap hurts the users' incentive to contribute. Nobody would like to see that the files they published are marked as faked files. The system with many traps would discourage people to publish high-quality files.

Second, many reputation systems calculate the reputation of the users in terms of publishing high quality files. This reputation can be referred to as the *user service reputation*. RepTrap obviously damage the user service reputation by making the system mark high-quality files as low-quality.

Third, many reputation systems have mechanisms to encourage good users and punish bad users. The target users of RepTrap would not receive the reward they deserve.

Especially, the malicious users will not be punished. This will undermine the incentive for participation in the service or system.

Fourth, in this paper, we study the reputation systems using feedback reputation. If RepTrap is not solved, the system may have to give up using feedback reputation, which open doors to simple bad-mouthing attacks. Even if the system does not use feedback reputation, RepTrap still hurts the object quality reputation by slandering good files directly.

Therefore, a throughout understanding on this attack, as well as developing defense mechanisms, will have significant impact on the security and usability of reputation systems used today and in the future. This paper is the first effort toward understanding and solving this security problem in reputation systems. More research on defense is expected.

2.6 Conclusion

In this paper, we reported the discovery of a new and powerful attack, named RepTrap, against feedback-based reputation systems. Through an in-depth investigation, we presented the scenarios that this attack can be applied to, the ways to effectively conduct this attack, and the consequence of the attack. The performance evaluation is based on real data and realistic user models in a popular P2P file-sharing system. The simulation results as well as the case studies have demonstrated that the RepTrap attack can significantly reduce the resources required to attack popular objects. With the same resources, the success probability of attacks was greatly increased. The broader impact of this attack and several variations of RepTrap were also discussed. The investigation in this paper has moved the arms race between attack and defense in reputation systems to a new level.

List of References

[1] C. Dellarocas, "The digitization of word-of-mouth: Promise and challenges of online reputation systems," *Management Science*, vol. 49, no. 10, pp. 1407–1424,

October 2003.

[2] A. Jøsang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decis. Support Syst.*, vol. 43, no. 2, pp. 618–644, 2007.

[3] K. Walsh and E. G. Sirer, "Experience with an object reputation system for peer-to-peer filesharing," in *USENIX NSDI*, 2006.

[4] C. Dellarocas, "Strategic manipulation of internet opinion forums: Implications for consumers and firms," *Management Science*, October 2006.

[5] A. Harmon, "Amazon glitch unmasks war of reviewers," The New York Times, February 14 2004.

[6] M. Hines, "Scammers gaming youtube ratings for profit," InfoWorld, http://www.infoworld.com/article/07/05/16/cybercrooks_gaming_google_1.html, May 2007.

[7] J. Brown and J. Morgan, "Reputation in online auctions: The market for trust," *California Management Review*, vol. 49, no. 1, pp. 61–81, 2006.

[8] D. Cosley, S. Lam, I. Albert, J. Konstan, and J. Riedl, "Is seeing believing? how recommender systems influence users' opinions," in *Proceedings of CHI 2003 Conference on Human Factors in Computing Systems*, Fort Lauderdale, FL, 2003, pp. 585–592.

[9] K. Hoffman, D. Zage, and C. Nita-Rotaru, "A survey of attack and defense techniques for reputation systems," Purdue University, Tech. Rep. CSD TR #07-013, 2007.

[10] Y. Sun, Z. Han, W. Yu, and K. J. R. Liu, "A trust evaluation framework in distributed networks: Vulnerability analysis and defense against attacks," in *Proceeding of the 27th Conference on Computer Communications (INFOCOM'06)*, Barcelona, Spain, April 2006.

[11] J. R. Douceur, "The sybil attack," in *Proceedings of First International Workshop on Peer-to-Peer systems (IPTPS'02)*, March 2002.

[12] H. Yu, M. Kaminsky, P. Gibbons, and A. Flaxman, "Sybilguard: Defending against sybil attacks via social networks," in *Proceedings of ACM SIGCOMM*, September 2006.

[13] H. Yu, P. Gibbons, M. Kaminsky, and F. Xiao, "Sybillimit: A near-optimal social network defense against sybil attacks," in *Proceedings of IEEE Symposium on Security and Privacy*, May 2008.

[14] S. Marti and H. Garcia-Molina, "Taxonomy of trust: categorizing p2p reputation systems," *Comput. Networks*, vol. 50, no. 4, pp. 472–484, 2006.

[15] P. Resnick, R. Zeckhauser, E. Friedman, and K. Kuwabara, "Reputation systems," *Communications of the ACM*, vol. 43, no. 12, pp. 45–48, 2000.

[16] Q. Lian, Z. Zhang, M. Yang, B. Y. Zhao, Y. Dai, and X. Li, "An empirical study of collusion behavior in the maze p2p file-sharing system," in *ICDCS*, 2007.

[17] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *Proceedings of ACM SOSP*, 2003.

[18] M. Yang, Q. Feng, Y. Dai, and Z. Zhang, "A multi-dimensional reputation system combined with trust and incentive mechanisms in p2p file sharing systems," in *Proceedings of ICDCS Workshops*, 2007.

[19] "Kazaa," http://www.kazaa.com.

[20] "Emule specification," http://www.emule.com.

[21] Q. Feng and Y. Dai, "Lip: A lifetime and popularity based ranking approach to filter out fake files in p2p file sharing systems," in *Proceedings of IPTPS*, 2007.

[22] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable Peer-To-Peer lookup service for internet applications," in *Proceedings of ACM SIGCOMM*, 2001.

[23] A. Josang and R. Ismail, "The beta reputation system," in *Proceedings of the 15th Bled Electronic Commerce Conference*, June 2002.

[24] J. Weng, C. Miao, and A. Goh, "An entropy-based approach to protecting rating systems from unfair testimonies," *IEICE TRANSACTIONS on Information and Systems*, vol. E89-D, no. 9, pp. 2502–2511, September 2006.

[25] C. Dellarocas, "Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior," in *Proceedings of the 2nd ACM conference on Electronic commerce*, 2000.

[26] C. Dellarocas, "Mechanisms for coping with unfair ratings and discriminatory behavior in online reputation reporting systems," in *Proceddings of the twenty first international conference on Information systems (ICIS)*, Brisbane, Queensland, Australia, December 2000.

[27] G. Swamynathan, B. Y. Zhao, and K. C. Almeroth, "Decoupling service and feedback trust in a peer-to-peer reputation system," in *Proceedings of ISPA Workshops*, 2005.

[28] S. Kamvar, M. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *Proceedings of 12th International World Wide Web Conferences*, May 2003.

[29] Y. Yang, Y. Sun, J. Ren, and Q. Yang, "Building trust in online rating systems through signal modeling," in *Proceedings of ICDCS Workshops*, 2007.

MANUSCRIPT 3

Dishonest Behaviors in Online Rating Systems: Cyber Competition, Attack Models, and Attack Generator

Abstract

Recently, online rating systems are gaining popularity. Dealing with unfair ratings in such systems has been recognized as an important but challenging problem. Many unfair rating detection approaches have been developed and evaluated against simple attack models. However, The lack of unfair rating data from real human users and realistic attack behavior models has become an obstacle toward developing reliable rating systems. To solve this problem, we design and launch a rating challenge to collect unfair rating data from real human users. In order to broaden the scope of the data collection, we also develop a comprehensive signal-based unfair rating detection system. Based on the analysis of real attack data, we discover important features in unfair ratings, build attack models, and develop an unfair rating generator. The models and generator developed in this paper can be directly used to test current rating aggregation systems, as well as to assist the design of future rating systems.

3.1 Introduction

With the evolutionary development of E-Commerce, online feedback-based rating systems are gaining popularity. While these systems are increasing influence on today's consumers, ensuring trustworthiness of such systems remains as an important and challenging task [1, 2, 3, 4]. The major challenge toward building a trustworthy online rating system is to deal with *unfair ratings* from *dishonest raters*. In commercial systems, it has been observed that collaborative dishonest raters provide unfair ratings intentionally to boost or downgrade the rating scores of certain products or reputation of other users[5].

There have been many approaches proposed to deal with unfair ratings[6, 2, 7, 3, 8, 9]. However, most of them are evaluated against simple attack behaviors models, in

which assumptions are made to greatly simplify the behavior of dishonest raters. For example, the arrival of unfair ratings is often assumed to be a Poisson process with a fixed arrival rate, and the unfair rating values are often assumed to follow a Gaussian or uniform distribution. However, most of the assumptions have not been validated by real user data and the simplification has not been well justified. More importantly, these simple models cannot reflect the smart attacks from real human users who can always adjust the attack strategies based on their observation of original rating data and gain knowledge about the rating system. More realistic and possibly complicated models of dishonest raters need to be developed.

The development of dishonest behavior model faces two challenges. The *first* is the lack of unfair rating data from real human users. Although there are plenty of unfair ratings in commercial systems, there is no ground truth telling which ratings are unfair ratings and which are not. *Second*, the attack behaviors are affected by the defense system. In other words, the dishonest raters may behave differently when different rating aggregation algorithms are used. For example, when simple averaging is used as the rating aggregation algorithm, providing the largest or smallest possible rating values is the most effective attack strategy. When majority-rule based detection algorithms, such as [2], are used, dishonest raters may provide rating values not too far away from the majority's opinion. When the signal-based detection algorithm, such as [9], are used, more complicated attack behaviors are expected.

To address the above challenges, we first design and launch a *rating challenge* to collect attack data from real human users. Moreover, we extend the approach in [9] and design a new rating aggregation algorithm, which includes most of the latest and complex defense strategies, such as feedback reputation and signal-based unfair rating detection. By using the new rating aggregation algorithm in the rating challenge, we are able to collect the real attacks against it such that the collected data cover a broad range of smart attack behaviors. Then, we analyze the attack behavior of real human users and evaluate the performance of both the complex and simple defense schemes

against real attacks. Many important results are obtained. Especially, we were able to classify attacks according to the unfair rating values as well as the time when the unfair ratings are provided. Finally, we build novel attack models as well as a comprehensive unfair rating generator. The attack models provide an in-depth understanding on the behavior of dishonest raters, which provides guidelines for future system design. The unfair rating generator is also a great tool for the research community and industry to perform realistic evaluation of rating aggregation systems.

The rest of the paper is organized as follows. Related work and the rating challenge are discussed in Section 2. The algorithms used in the challenge is described in Section 3. The attack data analysis is presented in Section 4, followed by the conclusion in Section 5.

3.2 Related Work and Rating Challenge

3.2.1 Related Work

As discussed in Section 5.1, the evaluation of rating algorithms mostly rely on simple attack models. For example, [10] [11] [12] only considered the probability of lying; [13] only considered the percentage of unfair raters and the strategies of either bad-mouthing and ballot-stuffing; [9] [14] considered the fraction of unfair rating and the unfair rating following a simple distribution;[15] considered some types of collusion. However, none of these models are built from real world data, and the parameters in their probability models are seldom changed. As we discussed later, these are critical in the evaluation of the rating algorithms. In addition, [4] built a statistical model for honest entities in decentralized systems, which could help to identify the suspicious entities. Relatively, our work is focusing on the dishonest behavior modeling.

Trust establishment is another key element in some rating aggregation systems. There is a rich literature on trust establishment for electronics commerce, peer-to-peer networks, distributed computing, ad hoc and sensor networks, and social networking systems [16, 17, 18, 19, 20, 21]. For rating aggregation problem, simple trust models are used to calculated trust in raters in [7, 22, 23]. However, their effectiveness is restricted

due to the limitation of the underlying detection algorithms.

Cyber Challenges are effective ways to collect real user data. For example, there is the Netflix Challenge [24] whose purpose is to build a better recommendation system based on user ratings. This purpose is very different from the focus of this work and the data collected in the Netflix challenge is not suitable for studying the unfair rating problem. So we launched a **Rating Challenges**[25], which is described in Section 3.2.2.

3.2.2 Rating Challenge

As pointed in [9], the unfair ratings can be classified into two categories:

- *Individual unfair ratings*: an individual rater provides unfairly high or low ratings without collaborating with other raters. This type of ratings may result from raters' personality/habit (i.e. dispositional trust [26]), irresponsibleness, and randomness.

- *Collaborative unfair ratings*: a group of raters provide unfairly high or low ratings to boost or downgrade the overall ratings of an object. This type of rating may due to the strategic manipulation from the owner of the object[5].

Compared with collaborative unfair ratings, individual unfair ratings are much less harmful. Therefore, *the focus of this work is to deal with collaborative unfair ratings.*

In order to investigate dishonest rating behaviors of real human users, we launched a **Rating Challenges**[25], in which participants insert collaborative unfair ratings into a regular rating data set. The participants who mislead the aggregated rating scores the most can win a cash prize. In this challenge,

- We collected real online rating data for 9 flat panel TVs with similar features. The data are from a well-known online-shopping website.

- The participants download the rating dataset and control 50 biased raters to insert unfair ratings. In particular, the participants decide when the 50 raters rate, which products they rate for, and the rating values.

- The participants' goal is to boost the ratings of two products and reduce the ratings of another two products.

- The successfulness of the participants' attack is determined by the **overall manipulation power**, i.e MP value. For each product, we calculate $\Delta_i = |R_{ag}(t_i) - R^o_{ag}(t_i)|$ during every 30 day period, where $R_{ag}(t_i)$ is the aggregated rating value with unfair ratings, and $R^o_{ag}(t_i)$ is the aggregated rating value without unfair ratings. The overall MP value is calculated as $\sum_k(\Delta^k_{max_1} + \Delta^k_{max_2})$, where $\Delta^k_{max_1}$ and $\Delta^k_{max_2}$ are the largest and 2nd largest among $\{\Delta_i\}'s$ for product k.

- The participants that can generate the largest MP value win the competition.

In the real world, when merchants intend to make more profits through unfair ratings, they have many options. They may boost their own products or downgrade the rivals' products. Here, we use MP instead of aggregation score of one product to rank the submitted attacks. Though MP metrics seems complex, it can evaluate attackers' power and countermeasure schemes more accurately.

The way to calculate the MP score will obviously influence the participants' attack strategy. For example, since the MP score depends on how much the attacker can mislead the monthly calculated rating scores, smart participants would tend to concentrate their attacks in 1 or 2 month duration. However, after analyzing the data collected, we found that not all participants were smart and even the smart participants tried many different attack strategies. Therefore, the collected attack data set covers a wide range of attack methods, and the influence from the specific MP score calculation method does not hurt the comprehensiveness of the data set.

3.3 Reliable Rating Aggregation System in the Rating Challenge

Although it is desirable to collect attack data targeting many rating aggregation systems, we must choose one rating aggregation system for the rating challenge. During our investigation, we observe that the attacks against majority-rule-based unfair rating

detection systems are *straightforward*.

- As pointed in [9], when there are a sufficient number of dishonest raters, the unfair ratings can become the majority and totally disable the majority-rule based methods. In other words, when the majority of ratings are unfair in certain time intervals, the majority-rule-based methods would totally *fail*.

- When there is no sufficient dishonest raters, the best attack strategy is to provide higher or lower ratings that are not too far away from the majority.

Therefore, if the rating challenge adopts one of the majority-rule-based methods as the underlying unfair rating detection algorithm, we can predict the participants' attack methods very well. There is no incentive for the participants to create more complicated attacks.

On the other hand, the investigation on the signal-based unfair rating detection methods is still in its early stage. Signal-based methods can handle straightforward attacks but the effective attacks against them are unknown. If the signal-based methods are used in the rating challenge, the participants can be encouraged to exploit new and complicated attacks. This will make the rating challenge more meaningful and useful.

However, the current signal-based methods are still not mature. The method reported in [9] is designed for a specific type of attack but cannot handle a wide range of attacks. This motivates us to design an advanced signal-based reliable rating aggregation system, and use it in the rating challenge. This advanced system is described in the following subsections. Due to space limitation, the specific design considerations and derivations are omitted. The readers who are only interested in attack modeling can go to Section 3.4.1 first to read the attack data analysis and then go back to Section 3.3 for details in the underlying unfair rating detection algorithms in the rating challenge.

3.3.1 Rating Aggregation Overview

The rating aggregation process contains four steps.

First, raw ratings are analyzed. Four analysis methods, arrival rate detection, model change detection, histogram detection and mean change detection, are applied independently.

Second, the outcomes of four detectors are combined to detect the time intervals in which unfair ratings are highly likely. Additionally, the suspicious rating detection module can mark some specific ratings as suspicious.

Third, we design a trust manager by simplifying the generic framework of trust establishment proposed in [27]. The trust manager determines how much individual raters can be trusted.

Fourth, the highly suspicious ratings are removed from the raw ratings by a rating filter. Then, the ratings are combined using trust information by the rating aggregation algorithm.

3.3.2 Mean Change Detector

The mean change detector contains three parts.

Mean Change Hypothesis Test

For one product, let $t(n)$ denote the time when a particular rating is given, $x(n)$ denote the value of the rating, and $u(n)$ denote the IDs of the rater. That is, at time $t(j)$, rater $u(j)$ submits a rating for the product with rating value $x(j)$, where $j = 1, 2, \cdots, N$ and N is the total number of ratings for this product.

We first study the mean change detection problem inside a window. Assume that the window contains $2W$ ratings. Let X_1 denote the first half ratings and X_2 denote the second half ratings in the window. We model X_1 as an i.i.d Gaussian random process with mean A_1 and variance σ^2, and X_2 as an i.i.d Gaussian random process with mean A_2 and variance σ^2. Then, to detect the mean change is to solve the hypothesis testing

problem

$$\mathcal{H}_0 : A_1 = A_2$$

$$\mathcal{H}_1 : A_1 \neq A_2.$$

It has been shown in [28] that the Generalized Likelihood Ratio Test (GLRT) is

$$\begin{aligned} &\text{Decide } \mathcal{H}_1 \text{ (i.e. there is a mean change), if} \\ &2 \ln L_G(x) = \frac{W(\hat{A}_1 - \hat{A}_2)^2}{2\sigma^2} > \gamma \end{aligned} \tag{3.1}$$

where $\hat{A}_1$ is the average of X_1 and $\hat{A}_2$ is the average of X_2, and γ is a threshold.

Mean Change Indicator Curve

Second, the detector constructs the mean change indicator curve using a sliding window with window size W. Based on (3.1), the mean change indicator curve is constructed as $MC(k)$ versus $t(k)$, where $MC(k)$ is the value of $W(\hat{A}_1 - \hat{A}_2)^2$ calculated for the window containing ratings $\{x(k - W), \cdots, x(k + W - 1)\}$. In other words, the test in (3.1) is performed to see whether there is a mean change at the center of the window.

MC Suspiciousness

Based on the peak values on the mean change indicator curve, we detect the time interval in which abnormal mean change occurs. This interval is called *mean change (MC) suspicious interval*.

3.3.3 Arrival Rate Change Detector

Arrival Rate Change Hypothesis Test

For one product, let $y(n)$ denote the number of ratings received on day n. We first study the arrival rate detection problem inside a window. Assume that the window covers $2D$ days, starting from day k. We want to detect whether there is an arrival rate change at day k', for $k < k' < k + 2D - 1$.

Let $Y_1 = [y(k), y(k+1), \cdots, y(k'-1)]$ and $Y_2 = [y(k'), y(k'+1), \cdots, y(k+2D-1)]$. It is assumed that $y(n)$ follow Poisson distribution. Then, the joint distribution of Y_1 and Y_2 is

$$p[Y_1, Y_2; \lambda_1, \lambda_2] = \prod_{j=k}^{k'-1} \frac{e^{-\lambda_1} \lambda_1^{y(j)}}{y(j)!} \prod_{j=k'}^{k+2D-1} \frac{e^{-\lambda_2} \lambda_2^{y(j)}}{y(j)!}, \tag{3.2}$$

where λ_1 is the arrival rate per day from day k to day $k'-1$, and λ_2 is the arrival rate per day from day k' to day $k+2D-1$. To detect the arrival rate change is to solve the hypothesis testing problem

$$\mathcal{H}_0 : \lambda_1 = \lambda_2$$

$$\mathcal{H}_1 : \lambda_2 \neq \lambda_1$$

It is easy to show that

$$p[Y_1, Y_2; \lambda_1, \lambda_2] = \frac{e^{-a\lambda_1} \lambda_1^{a\bar{Y_1}}}{\prod_{j=k}^{k'-1} y(j)!} \cdot \frac{e^{-b\lambda_2} \lambda_2^{b\bar{Y_2}}}{\prod_{j=k'}^{k+2D-1} y(j)!}. \tag{3.3}$$

where

$$\bar{Y_1} = \frac{1}{a} \sum_{j=k}^{k'-1} y(j), \quad \bar{Y_2} = \frac{1}{b} \sum_{j=k'}^{k+2D-1} y(j),$$
$$a = k' - k, \quad b = k - k' + 2D.$$

A GLRT decides $\mathcal{H}_1$ if

$$\frac{p[Y_1, Y_2; \hat{\lambda_1}, \hat{\lambda_2}]}{p[Y_1, Y_2; \hat{\lambda}, \hat{\lambda}]} > \gamma, \tag{3.4}$$

where $\hat{\lambda_1} = \bar{Y_1}$, $\hat{\lambda_2} = \bar{Y_2}$, and $\hat{\lambda} = \frac{1}{2D}(\sum_{j=k}^{k+2D-1} y(j)) = \bar{Y}$. Taking logarithm at both sides of (3.4), we derive

Decide $\mathcal{H}_1$(i.e. there is an arrival rate change) if

$$\frac{a}{2D} \bar{Y_1} \ln \bar{Y_1} + \frac{b}{2D} \bar{Y_2} \ln \bar{Y_2} - \bar{Y} \ln \bar{Y} \geq \frac{1}{2D} \ln \gamma. \tag{3.5}$$

Arrival Rate Change Curve

Based on (3.5), the Arrival Rate Change (ARC) curve is constructed as $ARC(k')$ vs $t(k')$. Here, the k' value is chosen as the center of the sliding window, i.e. $k' = k+D$. When $D < k' < N - D + 1$, $ARC(k')$ is just the left-hand side of equation (3.5) with $a = b = D$. When $k' \leq D$ or $k' \geq N - D + 1$, $ARC(k')$ can be calculated using a smaller window size, similar as the approach used in Section 3.3.2.

ARC Suspiciousness

Based on the peaks on the ARC curve, we divide all ratings into several segments. If the arrival rate in one segment is higher than the arrival rate in the previous segment and the difference between the arrival rates is larger than a threshold, this segment is marked as *ARC suspicious*.

H-ARC and L-ARC

For some practical rating data, the arrival rate of unfair ratings is not very high or the poisson arrival assumption may not hold. For those cases, we design H-ARC, which detects the arrival rate change in high value ratings, and L-ARC, which detects the arrival rate change in low value ratings.

Let $y_h(n)$ denote the number of ratings that are higher than $threshold_a$ received on day n, and $y_l(n)$ denote the number of ratings that are lower than $threshold_b$ received on day n. The $threshold_a$ and $threshold_b$ are determined based on the mean of all ratings.

- H-ARC detector: replace $y(n)$ in the ARC detector by $y_h(n)$
- L-ARC detector: replace $y(n)$ in the ARC detector by $y_l(n)$.

3.3.4 Histogram Change Detector

Unfair ratings can change histogram of the rating data. We design a histogram change detector based on clustering technique. There are two steps.

- 1. Within a time window k with the center at t_k, constructed two clusters

from the rating values using the simple linkage method. The Matlab function $clusterdata()$ is used in the implementation.

- 2. The Histogram Change (HC) curve, $HC(k)$ versus t_k, is calculated as

$$HC(k) = \min\left(\frac{n_1}{n_2}, \frac{n2}{n1}\right), \tag{3.6}$$

where n_1 and n_2 denote the number of ratings in the first and the second cluster, respectively.

3.3.5 Signal Model Change Detector

The signal model change detector is just the detector used in [9].

- Model-error-based detection: the ratings in a time window are fit into an autoregressive (AR) signal model. The model error is examined. When the model error is high, $x(n)$ is close to a white noise, i.e. honest ratings. When the model error is small, a 'signal' is present in $x(n)$ and the probability that there are collaborative raters is high.

The *model error (ME) curve* is constructed with the vertical axis as the model error, and horizontal axis as the center time of the windows. The windows are constructed either by making them contain the same number of ratings or have the same time duration. The covariance method [29] is used to calculate the AR model coefficients and errors. The time interval when the model error drops below a certain threshold is marked as the model error (ME) suspicious interval.

3.3.6 Integration of Multiple Detectors

We have developed detectors for mean change, arrival rate change, histogram change, and model error change. The problem formations for individual detectors are different. This is because that attack behaviors are very diverse and cannot be described by a single model. Different attacks have different features.

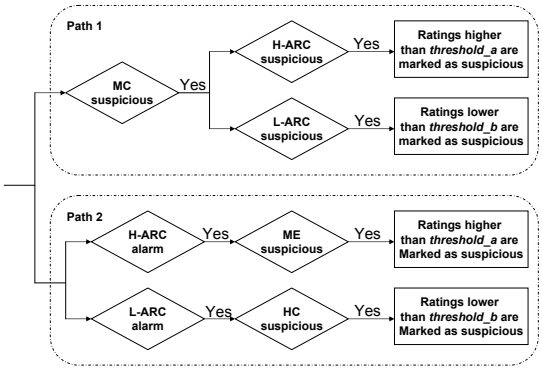

Figure 3.1. Join Detection of Suspicious Ratings

In addition, the normal behaviors, i.e. fair ratings, are not stationary. Even without unfair ratings, fair ratings can have variation such as in mean and arrival rate. In smart attacks, the changes caused by unfair ratings and the nature changes in fair ratings are sometimes difficult to differentiate. Thus, using a single detector will cause a high false alarm probability. Based on above discussions, we develop an empirical method to combine the proposed detectors, as illustrated in Figure 3.1.

There are two detection paths. Path 1 is used to detect strong attacks. If the MC indicator curve has a U-shape, and H-ARC or L-ARC indicator curve also has a U-shape, the corresponding high or low ratings inside the U-shape will be marked as suspicious. If for some reason, H-ARC (or L-ARC) indicator curve does not have such a U-shape, H-ARC (or L-ARC) alarm is issued. The alarm will be followed by the ME or HC detector. This is path 2. Path 2 detects suspicious intervals. Since there may be multiple attacks against one product, the ratings must go through both paths. Path 1 and Path 2 are in parallel.

3.3.7 Trust in Raters and Rating Aggregation

It is noted that we cannot perfectly differentiate unfair ratings and fair ratings in the suspicious intervals. Therefore, some fair ratings will be marked as suspicious. As a consequence, one cannot simply filter out all suspicious ratings. In this work, this suspicious rating information is used to calculate trust in raters, based on the beta-function trust model[30]. The calculation is described in Procedure 3.

Procedure 3 Computing Trust in Raters

1: For each rater i, initialize $S_i = 0$, and $F_i = 0$
2: **for** $k = 1 : K$ **do**
3: **for** each rater i **do**
4: Set $n_i = f_i = 0$,
5: Considering all products being rated during time $\hat{t}(k-1)$ and $\hat{t}(k)$, determine:
 n_i: the number of ratings that is provided by rater i
 f_i: the number of ratings from rater i and being marked as suspicious
6: calculate $F_i = F_i + f_i$ and $S_i = S_i + n_i - f_i$.
7: calculate trust in rater i at time $\hat{t}(k)$ as: $(S_i + 1)/(S_i + F_i + 2)$.
8: **end for**
9: **end for**

Similar as in [9], we adopt the weighted average trust model to combine rating values from different raters. In particularly, let R denote the set of raters whose ratings are the inputs to the aggregation module. If rater $i \in R$, let r_i denote the rating from rater i and T_i denote the current trust value of rater i. In addition, each rater provides only one rating for one object and R_{ag} denotes the aggregated rating. Then,

$$R_{ag} = \frac{1}{\sum_{i:i\in R} max(T_i - 0.5, 0)} \sum_{i:i\in R} r_i \cdot max(T_i - 0.5, 0) \tag{3.7}$$

3.4 Attack data analysis

3.4.1 Experiment Setup

We collected 251 valid submissions of unfair rating data in the rating challenge. Three observations are made. First, more than half of the submitted attacks were straightforward and did not exploit the features of the underlying defense mechanisms. These attacks are effective against majority-rule-based defense methods. Second, among the attacks that exploit the underlying defense mechanisms, many of them are complicated and unexpected. Third, according to a survey after the challenge, most successful participants either generated unfair rating manually or modified computer generated unfair rating data manually. As we expected, this data set covers a broad range of attack possibilities.

In this section, we first analyze the attack data set from multiple angles. Then an

attack data generator based on the analysis is presented in Section 3.4.5.

Three unfair rating detection systems are considered.

- P-scheme: the proposed system described in Section 3.3. The parameters of the proposed system are as follows. The initial trust value of all raters is 0.5. The window size of the MC detector, H-ARC/L-ARC detectors, HC detector, and ME detector are 30 (days), 30 (days), 40 (ratings), and 40 (ratings), respectively. In the H-ARC and L-ARC, $threshold_a = 0.5m$ and $threshold_b = 0.5m + 0.5$, where m is the mean of the rating values in the time window.

- SA-scheme: using simple averaging for rating aggregation and not applying any unfair rating detection.

- BF-scheme: using the beta-function based filtering technique proposed in[2] to remove unfair ratings. Then, the trust value of rater i is calculated as $(S_i + 1)/(S_i + F_i + 2)$, where F_i is the number of ratings (from rater i) that have been removed, and $F_i + S_i$ is the total number of ratings provided by rater i. This is a representative majority-rule-based scheme.

The above defense schemes are tested against all attack data collected in the rating challenge. When using the P-scheme, the maximum MP value that the attackers achieve is about 1/3 of the maximum MP value when using the other two schemes. Thus, *from the defense points of view, the proposed P-scheme has significant advantage over the traditional majority-rule based schemes.* Since this paper only focuses on attack data analysis, we do not report this comparison results. We would like to point out that studying the attacks against the P-scheme is essential because of its significance as a new and powerful defense technique.

3.4.2 Unfair Rating Value Analysis

Each individual unfair rating is uniquely determined by the rating value and the time when this rating is provided. We examine the rating values in this subsection, and

the time in the next subsection. Furthermore, the most effective unfair ratings might be correlated with the fair ratings. The issues related to correlation will be investigated in subsection 3.4.4.

When examining the rating values for one product, we define the difference between the mean of all unfair ratings and the mean of fair ratings as *bias*. When the bias value is positive, the purpose of the unfair ratings is to boost the final rating score. When the bias value is negative, the purpose is to downgrade the final rating score. The original rating value is between 0 and 5, and the mean of fair ratings is around 4 in the rating challenge. Thus, the bias is between -4 and 1. Besides bias, another important feature is the variance of the unfair rating values.

We are particularly interested in the unfair rating data that generate large MP values. The following data points are selected.

- We first compare the overall MP values. If one submission generates the top 10 overall MP values, this submission is marked as AMP. We only mark the AMP points with P-scheme because the SA-scheme and BF-scheme don't consider the correlation between different products.

- Then, for each product k, we compare the MP value gained from this product among all the submissions that have *negative* bias for product k. If one submission makes top 10, this submission is marked as UMP for product k.

- Finally, for each product k, we compare the MP value gained from this product among all the submissions that have *positive* bias for product k. If one submission makes top 10, this submission is marked as LMP for product k.

In Figure 3.2, the horizontal axis is the unfair rating bias for the first product and the vertical axis is the standard derivation of the unfair rating values. Each dot represents one submission. We use different colors to differentiate the different types.

- Black: without AMP, LMP, or UMP marks

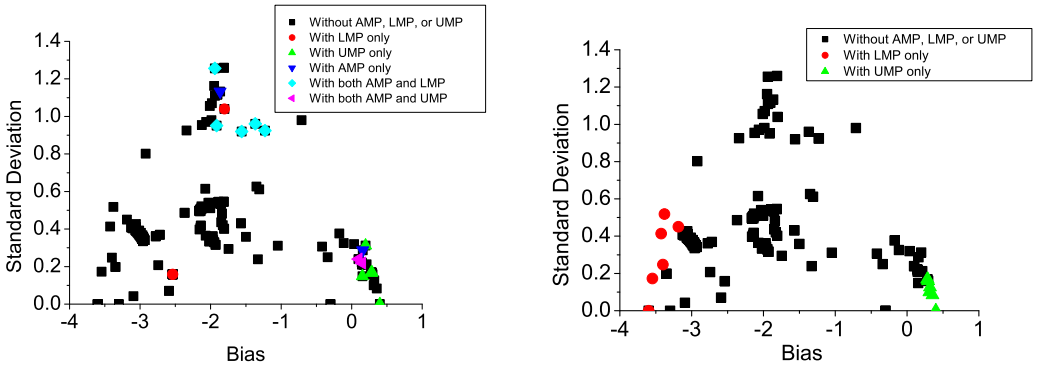

Figure 3.2. (a) P-scheme for product 1 (b) SA-scheme for product 1

- Blue: with AMP mark only
- Red: with LMP mark only
- Green: with UMP mark only
- Cyan: with both AMP and LMP marks
- Pink: with both AMP and UMP marks

Due to space limitation, we only show the results for product 1. The results for other products are similar. In Figure 3.2, the MP values are calculated with P-scheme and BF-scheme. Since the SA-scheme's performance is very similar with BF-scheme, we do not show the performance. By investigating the relationship between bias, variance, and MP values, we made an important observation.

- For each unfair rating detection algorithm, the submissions that generate large MP values mostly locate in a specific region on the variance-bias plot. This is especially obvious when the bias is negative (i.e. downgrading).

For negative bias, we can roughly divide the variance-bias plot into three regions: (R1) large bias, small to medium variance; (R2) medium bias, small to medium variance; and (R3) medium bias, medium to large variance. When the P-scheme is used, the submissions with large MP values are concentrated in region R3. When the other two

schemes are used, the submissions with large MP values are concentrated in region R1. This directly leads to a better understanding on different unfair rating detection schemes.

- The signal-based unfair rating detection scheme can detect unfair ratings with small-to-medium variance, but cannot handle the unfair ratings with large variance very well. That is, to cheat the signal-based scheme, the attacker should insert unfair ratings with large variance and medium bias. The intuitive explanation is that the large variance can weaken the features presented in the unfair ratings and therefore make the detection based on signal features less effective.

- The BF-scheme is not very effective. Comparing Figure 3.2 (a) and (b), we can see that the unfair ratings with large bias make the largest MP. As long as those ratings are not very few, the BF-scheme cannot judge whether they are far from the majority's opinions. This is true for the most of the majority-rule based methods.

The above observations are made for the downgrading attack (i.e. negative bias). In the rating challenge, we have observed that the boosting attack (i.e. positive bias) is not as effective as the downgrading attack in terms of generating large MP values. This is because the mean of the fair ratings is high (which is true for most popular products on commercial websites) and there is no much room to further boost the rating values. As a consequence, the variance-bias plot for positive bias does not have a high 'resolution' such that the different regions can be clearly identified. In this paper, we focus on the downgrading attack, and will examine the details of the boosting attack in the future work.

From above discussion, we can see that *bias and variance are two important features directly related to the strength of the attack*. This enables us to design a heuristic algorithm to find the best region on the variance-bias plot to generate unfair ratings for different algorithms from the attacker's points of view. Identifying these regions is critical for the design and evaluation of defense algorithms.

The heuristic algorithm is described in Procedure 4.

Procedure 4 Heuristic Unfair Rating Value Set Generator

1: Set $interested - area$ as the entire area on the variance-bias plot
2: Set $Flag = true$.
3: **while** $Flag = true$ **do**
4: Divide the $interested - area$ into N subareas
5: **for** each subarea **do**
6: Randomly generate m set of unfair rating data using the bias and variance values represented by the center point of the subarea,
7: Test the rating aggregation systems and determine the maximum MP value resulting from the m set of unfair ratings.
8: **end for**
9: Set the $interested - area$ as the subarea with the largest MP.
10: **if** the $interested - area$ is smaller than a threshold **then**
11: Flag=false
12: **end if**
13: **end while**
14: Output the $interested - area$

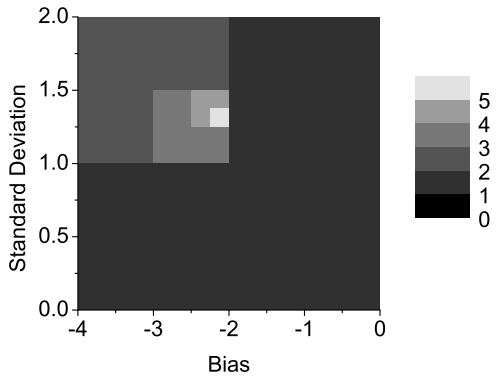

Figure 3.3. Optimum region searching

In Figure 3.3, it demonstrates the process of Procedure 4. The initial $interested - area$ ranges from 0-4(bias) and 0-2(standard deviation), and $N = 4$, $m = 10$. After 4 rounds, it outputs the area with the center point of (-2.325,1.56). The most significant result is that the MP value gained from the generator is larger than any of the submission during the challenge, which demonstrates that this heuristic generator can generate more powerful attack automatically.

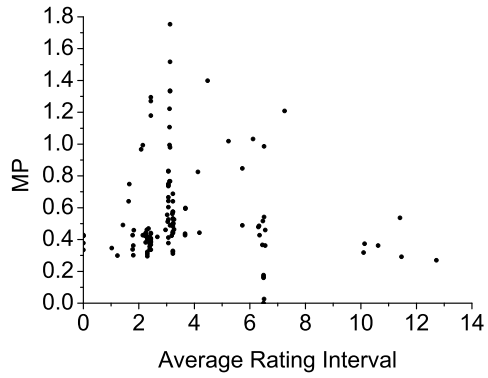

Figure 3.4. Time Analysis for unfair ratings of product 1

3.4.3 Time Domain Analysis

Attack duration, which is defined as the time period when dishonest raters provide their ratings, is an important feature of attack data in the time domain. In the rating challenge, there is a large variation in the attack duration, ranging from 10 days to the entire rating duration. In this subsection, we examine the average *unfair rating interval*, defined as the total number of unfair ratings divided by the attack duration. Given the same number of unfair ratings, longer is the attack duration, larger is the unfair rating inter arrival time.

In Figure 3.4, the horizontal axis is average rating interval, and the vertical axis is the MP value for product 1 when using the P-scheme. Each dot in the plot represents one unfair rating submission. Obviously, with the same inter arrival time, different submissions yield different MP values because the specific attack methods are different. However, it is seen that there exist the best arrival rate, which can produce the largest MP values. In Figure 3.4, this best interval is approximately 3 days.

It is noted that the specific value of the best interval is affected by two factors: the way to calculate the MP value and the unfair rating detection algorithm. In this experiment, the MP value is calculated monthly and the top two MP values are counted. Therefore, without using any unfair rating detection algorithm, the attacker should distribute all 50 unfair ratings within a two month period of time, which makes the best

inter arrival time smaller than 1.2. When using the signal-based detection algorithm, the attack with high arrival rate can be easily detected, and the attack with very large interval does not have much impact on the MP values. This is why the moderate arrival rate leads to the highest MP value.

In this paper, we only conducted the simple analysis along the time domain. Even this simple analysis yields an interesting result. That is, the signal-based defense methods response differently to different unfair rating arrival rate. As a consequence, *in the evaluation of unfair rating detection algorithms, it is necessary to adjust the unfair rating arrival rate in a fine scale.*

3.4.4 Rating correlation

In the current literature, the unfair ratings are always assumed to be independent of each other and also independent of fair ratings. Is this assumption valid?

To answer this question, we examine the real user attack data collected in the rating challenge. We do not find obvious correlation between unfair ratings and fair ratings. We believe that *the assumption of no correlation between unfair ratings and fair ratings is valid for the current attackers.*

However, can the correlation enhance an attack? We conduct the following heuristic experiment to explore this question.

- We choose the real user unfair rating data that generate top 10 MP values.
- We change the order in which the unfair ratings are given to create correlations between unfair ratings and fair ratings with Procedure 5.
- We also change the order randomly 5 times to compare the results.
- With this method, we generate 60 sets of new unfair rating data.
- We compare the generated unfair rating data with the original unfair rating data from real users, in terms of the MP values.

Procedure 5 Heuristic Correlation Algorithm

1: Put all the rating values in rating value set
2: Put all the rating time in rating time set.
3: **while** rating time set is not empty **do**
4: Set $MinT$ as the minimum time in rating time set
5: Set $NearV$ as the fair rating value whose rating time is just before $MinT$
6: Set $MaxV$ as the rating value that has the maximum difference with $NearV$ in rating value set
7: Match $MinT$ and $MaxV$ together
8: Remove $MinT$ out of rating time set
9: Remove $MaxV$ out of rating value set
10: **end while**

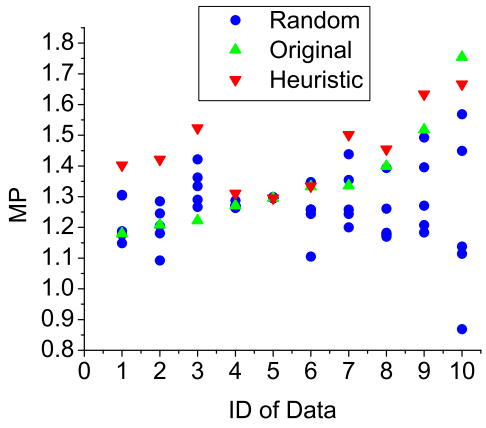

Figure 3.5. Comparison of different order strategies

The comparison results are shown in Figure 3.5. In most of the cases, the generated unfair rating data yield higher MP than the original rating data under the P-scheme, which indicates the *correlation* can improve attack strength. Although the current attackers have not correlated unfair ratings with fair ratings, we believe this is a potential threat in the future. Therefore, it is necessary to evaluate the defense algorithms against unfair ratings in correlation.

3.4.5 Attack Generator

Through the investigation on real user attack data, we get the necessary features to describe attacks against rating systems. These features include bias, variance, arrival rate, and correlation associated with unfair ratings. To assist the evaluation of current

and future rating aggregation systems, we design an attack generator as illustrated in Figure 3.6. It first generates the rating value set and the time set. Then the unfair ratings and the rating time are matched based on the correlation with the fair rating data. The user inputs the range of the parameters and the generator can produce more and more powerful attacks.

We implement this generator in Matlab and give a performance evaluation against the P-scheme. To compare the attacks from real users, we do the experiment like the participants in the rating challenge. Specifically, we downgrade the product 1, 3 and boost product 2, 4, setting the average time interval as 2 , 3 and 4 days, like the winner participants did in the rating challenge. We generate the rating time based on Poisson distribution. Figure 3.6 gives the MP results of the attacks from the generator. Based on the process of Procedure 4, the generator can give more and more powerful attacks by heuristically learning from the previous attack effects. Comparing Figure 3.4 and Figure 3.6, we see that the generator can give the similar strong attack performance as the real users did. However, the attacks from the generator cannot achieve the strongest attack (MP=1.8) from a real user. We investigate this strongest attack and find out that the participate used a complicated strategy to get the best performance. First, its rating time does not follow Poisson distribution; second, not 50 dishonest users all attack the 4 products. Instead, 20 out of dishonest users did not join the boost attack, which made the P-scheme not detect them very well. Although the generator does good jobs in most of the cases, it cannot completely replace the real users' test, which makes our rating challenge more meaningful.

3.5 Conclusion

In this paper, we exploit the features in unfair ratings against rating aggregation systems. Different from all previous efforts, the exploration is based on attack data collected from real human users through a rating challenge. The rating challenge is carefully designed such that the data collected can represent a broad range of attacks

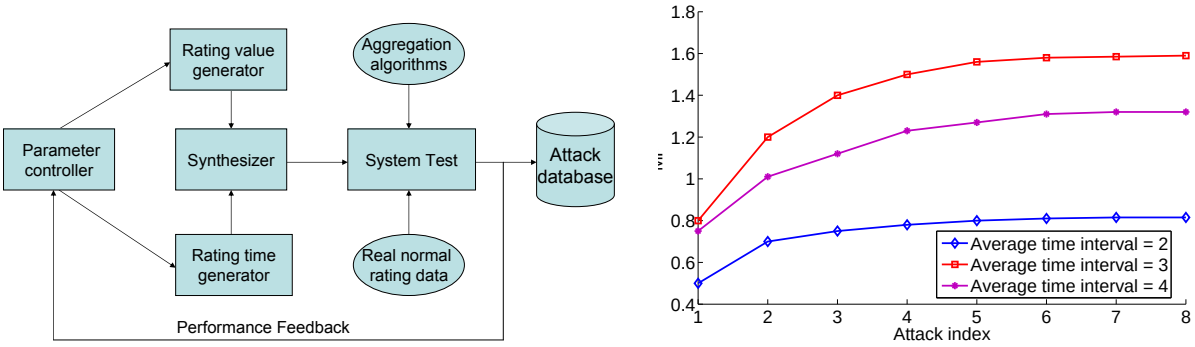

Figure 3.6. Attack generator and performance

against different rating aggregation systems. For the rating challenge, we design a new signal-based unfair rating detection system, which not only outperforms the existing schemes but also encourages creative attacks from the participants.

Based on the analysis of real attack data, we have discovered important features in unfair ratings. For example, the bias and variance greatly affects the strength of attacks; there exists an unfair rating arrival rate that maximizes the attack power; correlation between unfair ratings and fair ratings are not presented in current attacks but can improve the attack power, etc. Furthermore, we build attack models and develop an unfair rating generator. The models and generators developed in this paper can be directly used to test current rating aggregation systems, as well as to assist the design of future rating systems.

List of References

[1] Z. Liang and W. Shi, "Analysis of ratings on trust inference in open environments," *Elsevier Performance Evaluation*, vol. 65, no. 2, pp. 99–128, 2008.

[2] A. Whitby, A. Jøang, and J. Indulska, "Filtering out unfair ratings in Bayesian reputation systems," in *Proc. 7th Int. Workshop on Trust in Agent Societies*, 2004.

[3] J. Zhang and R. Cohen, "Trusting advice from other buyers in e-marketplaces: the problem of unfair ratings," in *Proceedings of the 8th international conference on Electronic commerce*, 2006.

[4] Q. Zhang and T. Yu, "On the modeling of honest players in reputation systems," in *Proceedings of IEEE ICDCS Workshop on Trust and Reputation Management*, 2008.

[5] C. Dellarocas, "Strategic manipulation of internet opinion forums: Implications for consumers and firms," *Management Science*, October 2006.

[6] C. Dellarocas, "Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior," in *Proceedings of the 2nd ACM conference on Electronic commerce*, 2000.

[7] M. Chen and J. Singh, "Computing and using reputations for internet ratings," in *Proceedings of the 3rd ACM conference on Electronic Commerce*, 2001.

[8] J. Weng, C. Miao, and A. Goh, "An entropy-based approach to protecting rating systems from unfair testimonies," *IEICE TRANSACTIONS on Information and Systems*, vol. E89-D, no. 9, pp. 2502–2511, September 2006.

[9] Y. Yang, Y. Sun, J. Ren, and Q. Yang, "Building trust in online rating systems through signal modeling," in *Proceedings of IEEE ICDCS Workshop on Trust and Reputation Management*, 2007.

[10] K. Aberer and Z. Despotovic, "Managing trust in a peer-2-peer information system," *Proceedings of the tenth international conference on Information and knowledge management*, pp. 310–317, 2001.

[11] L.-H. Vu and K. Aberer, "A Probabilistic Framework for Decentralized Management of Trust and Quality," *Eleventh International Workshop on Cooperative Information Agents (CIA 2007)*, 2007.

[12] Z. Despotovic and K. Aberer, "A Probabilistic Approach to Predict Peers' Performance in P2P Networks," *Cooperative Information Agents VIII: 8th International Workshop, CIA 2004, Erfurt, Germany, September 27-29, 2004; Proceedings*, 2004.

[13] A. Whitby, A. Jøsang, and J. Indulska, "Filtering out unfair ratings in bayesian reputation systems," *The Icfain Journal of Management Research*, vol. 4, no. 2, pp. 48–64, 2005.

[14] C. Dellarocas, "Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior," *Proceedings of the 2nd ACM conference on Electronic commerce*, pp. 150–157, 2000.

[15] Z. Despotovic and K. Aberer, "P2P reputation management: Probabilistic estimation vs. social networks," *Computer Networks*, vol. 50, no. 4, pp. 485–500, 2006.

[16] R. Zhou and K. Hwang, "Powertrust: A robust and scalable reputation system for trusted peer-to-peer computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 5, May 2007.

[17] M. Maheswaran and H. Tang, "Towards a gravity-based trust model for social networking systems," in *Proceedings of IEEE ICDCS Workshop on Trust and Reputation Management*, 2007.

[18] Y. Wang and J. Vassileva, "A review on trust and reputation for web service selection," in *Proceedings of IEEE ICDCS Workshop on Trust and Reputation Management*, 2007.

[19] H. Zhao and X. Li, "H-trust: A robust and lightweight group reputation system for p2p desktop grid," in *Proceedings of IEEE ICDCS Workshop on Trust and Reputation Management*, 2008.

[20] A. Gutowska and K. Buckley, "Computing reputation metric in multi-agent e-commerce reputation system," in *Proceedings of IEEE ICDCS Workshop on Trust and Reputation Management*, 2008.

[21] H. Wu, H. Chen, and C. Gao, "A trust management model for p2p file sharing system," in *Proceedings of IEEE ICDCS Workshop on Trust and Reputation Management*, 2008.

[22] L. Xiong and L. Liu, "Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 7, pp. 843–857, July 2004.

[23] K. Fujimura and T. Nishihara, "Reputation rating system based on past behavior of evaluators," in *Proceedings of the 4th ACM conference on Electronic commerce*, 2003.

[24] "Netflix prize dataset," www.netflixprize.com/download.

[25] U. of Rhode Island, "Etan rating challenge," www.etanlab.com/rating.

[26] D. H. McKnight and N. L. Chervany, "The meanings of trust," MISRC Working Paper Series, Technical Report 94-04, arlson School of Management, University of Minnesota, 1996.

[27] Y. Sun and Y. Yang, "Trust establishment in distributed networks: Analysis and modeling," in *Proceedings of IEEE ICC'07*, 2007.

[28] S. Kay, *Fundamentals of Statistical Signal Processing, Volume 2: Detection Theory*. Prentice Hall, 1998.

[29] M. Hayes, *Statistical Digital Signal Processing and Modeling*. John Wiley and Sons, 1996.

[30] A. Jsang and R. Ismail, "The beta reputation system," in *Proceedings of the 15th Bled Electronic Commerce Conference*, June 2002.

MANUSCRIPT 4

Securing Time-synchronization Protocols in Sensor Networks: Attack Detection and Self-healing

Abstract

There have been many time synchronization protocols proposed for sensor networks. However, the security problems related to time synchronization have not been fully solved yet. If malicious entities manipulate time synchronization, failures of many functionalities in the sensor networks would occur. In this paper, we identify various attacks against time synchronization and then develop a detection and self-healing scheme to defeat those attacks. The proposed scheme has three phases: (1) abnormality detection performed by individual sensors, (2) trust-based malicious node detection performed by the base station, and (3) self-healing through changing the topology of synchronization tree. Simulations are performed to demonstrate the effectiveness of the proposed scheme as well as the implementation overhead.

4.1 Introduction

Technology advancement in wireless sensor networks is enabling a wide range of applications, such as crisis prediction for environment protection and mission critical military operations. However, achieving security in sensor networks has long been known as a challenging task [1]. Among many security concerns, securing time synchronization protocols is recently recognized as an important problem [2, 3, 4, 5].

As pointed out in [5, 2], time synchronization is critical to many sensor networks. Precise time is required by many applications and protocols, such as measuring time-of-flight for positioning, forming TDMA radio scheduling, coordinating sensors' sleep-wakeup schedules, preventing replay attacks, and enabling collaborative signal processing. If malicious entities can manipulate the time synchronization protocol, catastrophic failure of many applications and protocols in sensor networks could occur.

There have been many time synchronization protocols proposed for sensor net-

works. Among them, several prototype implementations, such as RBS[6], TPSN[7], FTSP[8], can achieve microsecond accuracy. Majority of those protocols, however, have not taken security into consideration.

This paper contributes to securing time synchronization protocols in sensor networks from two perspectives. First, we identify various attacks that can be launched by compromised sensors. Second, we design a *trust-enhanced detection and self-healing scheme* to defeat major attacks against time synchronization. The defense scheme has three phases: (1) abnormality detection performed by individual sensors, (2) trust-based malicious node detection performed by the base station, and (3) self-healing through changing the topology of the synchronization tree. Simulations show that the proposed scheme can successfully detect several attacks and enable fast recovery from those attacks.

The rest of the paper is organized as follows. Background and related work are introduced in Section 2. Attacks and defense mechanisms are described in Section 3 and Section 4, respectively. Section 5 shows the simulation results, followed by conclusion in Section 6.

4.2 Background and Related Work

4.2.1 Background

Time synchronization protocols in sensor networks can be classified as either *sender-to-receiver* or *receiver-to-receiver*[9]. In the sender-to-receiver based schemes, such as TPSN [7] and FTSP[8], two nodes exchange timing information with each other, and then one node adjusts its clock time according to the other node's clock. In the receiver-to-receiver schemes, such as RBS[6], the sender broadcasts a reference beacon to multiple receivers. The receivers exchange timing information and calculate clock offset among themselves. The receiver-to-receiver schemes often have low efficiency when the number of receivers increases. In this paper, we will focus on the security issues of TPSN scheme.

TPSN, which stands for timing-sync protocol for sensor networks, is a popular sender-receiver-based scheme[7]. It has two phases: level discovery and synchronization.

In the level discovery phase, TPSN creates a spanning tree as follows. The root, which is often the base station, broadcasts a level discovery message to nearby sensors. Upon receiving this message, a sensor sets its level as 1 and broadcasts its level to its neighbors. Continuously, a sensor, which receives broadcast messages from one or several senders, will choose the sender that has the lowest level (say level K) as its parent, and set its own level as $K + 1$. Then, this sensor broadcasts its level to its neighbors. The spanning tree is also referred to as the *sync-tree* in this paper.

In the synchronization phase, each sensor performs pairwise time synchronization with its parent in a top-down fashion. Level 1 nodes first synchronize with the root, then level 2 nodes synchronize with level 1 nodes, and so on. The pairwise time synchronization protocol used in TPSN, as well as in many other popular protocols, is illustrated in Figure 4.1. Assume that node A wants to synchronize its clock with node B.

- A sends a synchronization request message to B at A's local time T_1. T_1 is included in the synchronization request message.

- B receives this message at B's local time T_2, and then send an acknowledgement message, called the synchronization reply message, to A at B's local time T_3. Here, T_2 and T_3 are included in the acknowledgement.

- A receives the acknowledgement at A's local time T_4. Using T_1, T_2, T_3 and T_4, A can calculate the clock offset (δ) and propagation delay (d) as:

$$\delta = ((T_2 - T_1) - (T_4 - T_3))/2, \tag{4.1}$$

$$d = ((T_2 - T_1) + (T_4 - T_3))/2. \tag{4.2}$$

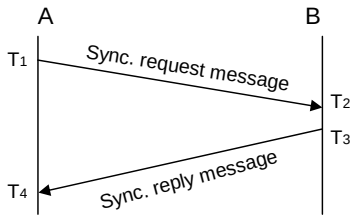

Figure 4.1. Pairwise time synchronization in TPSN.

4.2.2 Related Work

In [5], the transmission delay over a single hop and multiple hops in sensor networks are studied through extensive experiments. In the pairwise time synchronization protocol, if a sensor calculates a delay parameter (d) that is greater than a upper bound, the sensor will believe that it is under attack and therefore abort the synchronization service. This approach has several limitations. First, it can only detect the delay-pause attack, but cannot address other sophisticated attacks. Second, simply aborting the synchronization service cannot defeat the attack. Attackers can continue to damage the network, and even cause the denial of clock synchronization service. In another type of defense methods [2], a sensor tries to calibrate clock with several neighbor nodes and then adopts the medium clock value. It requires that a sensor has a sufficient number of good neighbors, which may not be guaranteed in practice.

4.3 Attacks Against Time Synchronization

Attackers can be either *outsiders* or *insiders*. The outsiders do not possess keys for encryption and authentication. Thus, they can only interfere the transmission of time synchronization messages, but not modify or generate those messages. On the other hand, the compromised sensors, which are insiders, can manipulate the time synchronization protocols in many ways. The outsider attack has been studied in [5]. This work focuses on attacks from insiders.

The primary goal of the attackers is to make other sensors set a wrong clock time. To achieve this goal, an insider can perform the *misleading attack*. The secondary goal of the attackers is to affect more sensors by making themselves locate close to the root of the sync-tree, i.e., having low levels. To achieve this goal, an insider can perform the

wormhole attack. Next, we present these two attacks in detail in the context of attacking TPSN.

4.3.1 Misleading Attack

If a parent node on the sync-tree lies about its local time, the parent node can mislead the clock time of its children as well as the clock time of all the nodes under its children. This attack is referred to as the misleading attack in this paper.

For example, let sensor B be sensor A's parent node. In the procedure described in Figure 4.1, when B is malicious, it can replace T_2 by $T_2 + \Delta$ and replace T_3 by $T_3 + \Delta$. As a consequence, A's clock time becomes $T_A + \Delta$, where T_A denotes the correct time. Furthermore, this wrong time will propagate to all the nodes that are under A on the sync-tree.

4.3.2 Wormhole Attack

Wormhole attack was originally discovered for attacking routing protocols in ad hoc networks [10]. If two malicious nodes that are multi-hop away can communicate directly through a side channel, these two nodes can form a wormhole. The communication through the side channel often has low-latency and is stealthy. The side channel, also referred to as a tunnel, can be created by connecting attackers through wire line or using powerful directional antennas[11].

Wormholes can change the topology of the sync-tree. Let X and Y are two malicious sensors. X locates close to the BS and has level L_x, and Y is far from the BS. If there exists a wormhole between X and Y, X can forward all messages related to time synchronization to Y. Thus, Y will get these messages earlier than its neighbors, and can claim its level to be $L_x + 1$. Then, Y attracts surrounding sensors to be Y's children, and conduct the misleading attack more effectively.

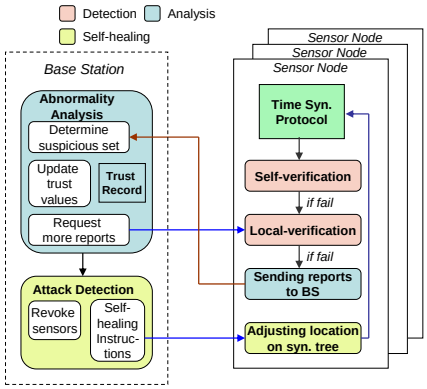

Figure 4.2. The structure of DAS defense mechanisms

4.4 Securing Time Synchronization

In this paper, we investigate moderate-size, multi-hop and static sensor networks. For simplicity, only one base station (BS) is considered. Additionally, the following security assumptions are made. The BS is fully protected and cannot be compromised. Each sensor has pairwise keys with its one-hop neighbors and the BS. That is, a sensor can communicate with its neighbors and the BS securely. All messages that are related with time synchronization are encrypted and authenticated. The malicious entities cannot read or modify others' messages, or impersonate other sensors that are not compromised. The defense mechanisms developed in this paper will be presented in the context of securing TPSN.

4.4.1 Overview of the Defense Mechanisms

We develop a set of defense mechanisms, which are collectively referred to as DAS, an abbreviation of Detection, Analysis and Self-healing. The overall structure of DAS is shown in Figure 4.2. The defense can be divided into four steps.

Step 1: after the level discovery phase, each sensor knows its level and its parent node on the sync-tree. After the first round of time synchronization, a sensor knows its child nodes. To enhance security, DAS requires each sensor to send the information of its level, parent and children to the BS, such that the BS will have a complete view of the sync-tree.

Step 2: sensors detect abnormality through a *self-verification* and a *local-*

verification process. After each round of the time synchronization, a sensor performs self-verification, based on end-to-end delay (d) and the clock offset (δ) calculated in (4.1) and (4.2), as well as the historical data in the previous rounds of time synchronization. The sensor runs a *risk-analysis process* to estimate the risk of being attacked. If the risk is over a threshold, this sensor initiates a local-verification process. In local-verification, the sensor exchanges information with its neighbors. If the local-verification process detects inconsistence among sensors' clock time, warning reports are generated and sent to the BS.

Step 3: the BS analyzes the reports from sensors and determines a set of sensors that are suspected to be responsible for the synchronization abnormality. This set of sensors are called as the *suspicious set*. Then, the BS runs a *trust evaluation module*, which reduces the trust values of the sensors in it.

Step 4: based on the analysis in step 3, the BS instructs the sensors to recover from synchronization errors by changing their locations on the sync-tree such that they are less likely to be affected by malicious nodes. In addition, the BS will isolate the sensors with very low trust values, such that they cannot launch the misleading-attack.

4.4.2 Self-verification

After each round of time synchronization, each sensor performs self-verification to determine the risk of being under attack. Before getting into the details of self-verification, we introduce several notations.

- k denotes the round index of time synchronization.
- t_i^k denotes the clock time of sensor i just before the k^{th} round time synchronization.
- $\hat{t}_i^k$ denotes the clock time of sensor i just after the k^{th} round of time synchronization.

- $d_{i,j}^k$ denotes the delay parameter calculated by (4.2) in the k^{th} round of synchronization, between sensor i and sensor j.

- $\delta_{i,j}^k$ denotes the clock offset between sensor i and sensor j in the k^{th} round of synchronization. In TPSN, if sensor j is the parent node of sensor i, $\hat{t}_i^k = t_i^k - \delta_{i,j}^k$.

Assume that the sensor i synchronizes its clock according to its parent, sensor j, in the k^{th} round of synchronization. Then, sensor i estimates the risk based on three parameters: *delay*, *clock offset*, and *skew*.

- The delay is just $d_{i,j}^k$ defined above. For simplicity, we use d_k to denote $d_{i,j}^k$ in the risk calculation.

- The clock offset is $\delta_{i,j}^k$, denoted by O_k in risk calculation.

- The skew parameter describes how fast the clock drifts. Similar as in [9], the skew, denoted by θ_k, is estimated based on the clock offsets in the past $(m+1)$ rounds, as

$$\begin{aligned} tan(\theta_k) &= \left(\frac{x+z}{x}\right) - \frac{\pi}{4}, \quad \text{where} \\ z &= O_k + O_{k-1} + \cdots + O_{k-m} \\ x &= t_i^k - t_i^{k-m}. \end{aligned} \tag{4.3}$$

  Since sensors only have limited storage and computation resources, m cannot be a large number.

The definition of risk varies greatly in different context. In this work, risk refers to the probability that at least one observable parameter is abnormal, and is calculated as

$$R = 1 - (1 - P_\theta)(1 - P_O)(1 - P_d), \tag{4.4}$$

where P_θ is the probability that the skew is abnormal, P_O is the probability that the clock offset is abnormal, and P_d is the probability that the delay is abnormal.

Estimation of P_θ, P_O and P_d depends on the physical properties of the clock and the statistics of network delay. Although complicated estimation methods can be employed, this work adopts a simple thresholding method in order to reduce the computation complexity at the sensors.

According to the experimental data in [5], the delay parameter resembles a Gaussian distribution. With a 99.97% confidence, the delay falls in the interval $[d_{avg} - 3\sigma, d_{avg} + 3\sigma]$, where d_{avg} is the average delay and σ is the variance. Thus, $d_{max} = d_{avg} + 3\sigma$ can serve as the delay upper bound. One can choose

$$P_d = 1 \text{ when } d \geq d_{max}, \text{ and } P_d = 0 \text{ otherwise.} \tag{4.5}$$

Based on the data provided in [5], a typical value of this upper bound is $d_{max} = d_{avg} + 3\sigma = 762 + 3 * 2.82 \approx 771\mu s$. We set

$$P_\theta = 1 \text{ when } |\theta_k| > \hat{\beta}, \text{ and } P_\theta = 0 \text{ otherwise,} \tag{4.6}$$

where $\hat{\beta}$ is the upper bound for skew, which can be estimated according to the features of the sensor hardware. Furthermore, we can set

$$P_O = 1 \text{ when } |O_k| > s \cdot \hat{\beta} \cdot (t_i^k - \hat{t}_i^{k-1}), \tag{4.7}$$
$$\text{and } P_O = 0 \text{ otherwise.}$$

Here we use a scaling factor, s (≥ 1), because the clock offset in a single round can be affected by estimation error due to variation in round-trip delay, whereas the estimation errors can be averaged out in the calculation of skew.

When the distribution of normal delay, offset and skey are known, the estimation of P_θ, P_O and P_d can be estimated more accurately. This can be done in the future work. In this paper, we adopted the simplest methods to illustrate the proposed algorithms, which surely can be improved.

4.4.3 Local-verification

The basic idea of local-verification is to allow a sensor to verify its clock time with its neighbors that are neither its parent nor its children. If a large time difference is

detected, a warning report will be sent to the BS. In this section, we first introduce the procedure of location verification and then present three activation conditions.

The *local-verification procedure* performed by a sensor, say sensor i, is described as follows.

- The sensor i broadcasts a synchronization request message to its neighbors.
- Upon receiving this message, a sensor, for example sensor j, first checks whether sensor i is its parent or children. If not, sensor j replies to sensor i by sending a synchronization reply message.
- After receiving the reply from sensor j, sensor i calculates the clock offset respect to sensor j. If the previous round of time synchronization is successful, this clock offset should be very small. Sensor i can receive multiple replies. Let b denote the total number of replies, and the clock offsets calculated from those replies are denoted by $\delta^{lv}_{i,j_1}, \delta^{lv}_{i,j_2}, \cdots, \delta^{lv}_{i,j_b}$.
- Finally, if $\max_{a=1}^{b} d^{lv}_{i,j_a} > TH_{lv}$, sensor i will send a warning report to the BS, where TH_{lv} is the *local-verification threshold*. The warning report will contain the IDs of the sensors as well as the clock offsets, i.e. $\{j_1, \delta^{lv}_{i,j_1}, j_2, \delta^{lv}_{i,j_2}, \cdots, j_b, \delta^{lv}_{i,j_b}\}$.

Local-verification will be performed if at least one of the following conditions are satisfied.

Activation Condition 1: When the risk value R, calculated in a sensor's self-verification process, is higher than a threshold, this sensor should perform local-verification. It is noted that the simple method for risk calculation presented in Section 4.4.2 leads to either $R = 1$ or $R = 0$. In this case, local-verification is performed when $R = 1$.

Activation Condition 2: A sensor should perform the local-verification occasionally even if the self-verification always generates low risk values. This is to prevent

sophisticated attacks that may possibly pass the self-verification check. In addition, the sensors that are closer to the root on the sync-tree should perform the local-verification more frequently. This is because the synchronization error at a lower level can propagate to higher levels. Let p_l denote the probability with which a sensor at level l should perform the local-verification.

Condition 2 can lead to better security, but it also introduces additional overhead even when there are no attackers. To control this overhead, we set p_l as follows.

Assume that there are roughly N sensors in the network and the maximum level of the sync-tree is L. Then, the degree of the sync-tree can be roughly estimated as $D = \sqrt[L]{N}$. The network resource consumed by local-verification is approximately proportional to the total number of verifications, which can be expressed as

$$C_{overhead} \approx D \cdot p_1 + D^2 \cdot p_2 + \cdots + D^L \cdot p_L. \tag{4.8}$$

If we set $p_{l-1} = Dp_l$, then $C_{overhead} \approx D \cdot L \cdot p_1$. Therefore, given the $C_{overhead}$, we can choose

$$p_1 = \frac{C_{overhead}}{L \cdot D} \text{ and } p_l = \frac{p_{l-1}}{D}. \tag{4.9}$$

Activation Condition 3: In this work, we do not assume that sensors are synchronized at the beginning. The self-verification mechanism does not work well in round 1, but the attackers can cause large synchronization errors if they attack in round 1. Therefore, it is important to perform local-verification after round 1. In particular, the sensors at level l should perform local-verification with probability p_l^* in round 1. Here, p_l^* is calculated using (4.9) with $C^*_{overhead}$ replacing $C_{overhead}$, and $C^*_{overhead}$ is much larger than $C_{overhead}$.

4.4.4 Abnormality Analysis and Self-Healing

In DAS, we assign the task of malicious sensor detection to the BS. Upon receiving the reports from sensors, the BS will analyze those reports and perform malicious node detection in three steps.

Step 1: Calculation of Negative Marks. The BS assigns negative marks to the sensors that are considered to be responsible for the time inconsistence detected in the local-verification process. Let Neg_j, which is between -1 and 0, denote the negative mark assigned to sensor j.

Step 2: Calculation of Trust Values. The BS maintains a trust value for each sensor. Let r_j^k denote the trust value of sensor j before the k^{th} round of synchronization, and $\hat{r}_j^k$ denote the trust value of sensor j after the k^{th} synchronization. In step 2, the new trust value, $\hat{r}_j^k$, is calculated as a function of r_j^k, Neg_j, and k.

Step 3: Self-Healing. In this step, the topology of the sync-tree is adjusted based on the negative marks and the trust values, in order to reduce the damage caused by suspicious sensors. In addition, the sensors with very low trust values will be isolated or removed from the sync-tree.

Next, we describe the above steps in detail.

Calculation of Negative Marks

The negative mark, a value between -1 and 0, describes how much responsibility a senor should take for the time inconsistence detected by local-verification. For a concise presentation, some terminologies are introduced.

- *Inconsistence Pair:* If the clock offset between sensor i and sensor j is larger than the local-verification threshold (TH_{lv}), these two sensors form an inconsistence pair, denoted as $IC(i, j)$.

- *Path:* $Path_{i,j}$ denotes the set of nodes that are on the path from sensor i to sensor j on the sync-tree, including sensor i and sensor j.

- *Responsible Set:* Among the nodes that are on both $Path_{i,root}$ and $Path_{j,root}$, one can identify the node with the lowest level. Assume that this node is sensor q. (Recall that the root of the sync-tree has level 0). Then, the responsible set (RS) of $IC(i, j)$ contains all nodes on $Path_{i,q}$ and $Path_{j,q}$ excluding the root node.

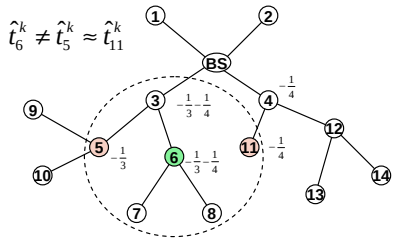

Figure 4.3. A simple sync-tree for demonstrating negative marks

Figure 4.3 shows a simple sync-tree. Assume that sensor 6 sends a report to the BS, saying $\delta_{6,11}^{lv} > TH_{lv}$ and $\delta_{6,5}^{lv} > TH_{lv}$. In this example, there are two inconsistence pairs: $IC(6, 11)$ and $IC(6, 5)$. The RS of $IC(6, 5)$ contains sensor 5, 6 and 3, and the RS of $IC(6, 11)$ contains sensor 6, 3, 4, 11.

From the procedures in TPSN, it is easily seen that at least one sensor in the responsible set of $IC(i, j)$ must be misbehaving in order to cause $IC(i, j)$. A simple strategy is to equally blame the sensors in the responsible set. The Procedure 1 shows how to generate the negative marks using this simple strategy.

Procedure 6 BS Generates Negative Marks

1: **for** each received report **do**
2: Find all sensors that can form inconsistent pairs with sensor i, where sensor i is the sender of the report.
3: **for** each inconsistent pair **do**
4: Identify the responsible set (RS).
5: Assign a mark with value $-\frac{1}{T}$ to each node in the RS, where T is the number of nodes in the RS.
6: **end for**
7: **for** each node on the sync-tree with marks **do**
8: Summarize the values of all marks associated with this node.
9: **end for**
10: Normalize all negative marks such that the summation of all negative marks is -1.
11: **end for**

We continue to study the example shown in Figure 4.3. Assume that the BS only receives the report from sensor 6. Because of $IC(6, 11)$, sensor 6, 3, 4, 11 each receive a mark $-\frac{1}{4}$. Because of $IC(6, 5)$, sensor 6, 3, 5 each receive a mark $-\frac{1}{3}$. After normalization, $Neg_3 = -\frac{7}{24}$, $Neg_4 = -\frac{1}{8}$, $Neg_5 = -\frac{1}{6}$, $Neg_6 = -\frac{7}{24}$, and $Neg_{11} = -\frac{1}{8}$.

Update Trust Values

Trust is an overloaded term in the current literature of network security. There are many trust definitions and metrics. This paper adopts the Beta-function-based trust metric proposed in [12] because it has a clear theoretical meaning and has demonstrated its advantage in some sensor network applications[13].

Briefly speaking, if a node has performed well for $(\alpha - 1)$ times and performed badly for $(\beta - 1)$ times in the past, the Beta-function-based trust value of this node is calculated as $\frac{\alpha}{\alpha+\beta}$.

Particularly, the trust values associated with time synchronization are updated as follows. Before the k^{th} round of time synchronization, sensor j has trust value r_j^k. Assume that this trust value is calculated as $r_j^k = \alpha_j^k/(\alpha_j^k + \beta_j^k)$. In round k, if sensor j receives a negative mark Neg_j, then the trust value is updated as $\hat{r}_j^k = \alpha_j^k/(\alpha_j^k + \hat{\beta}_j^k)$, where $\hat{\beta}_j^k = \beta_j^k - Neg_j$. The initial trust value can be chosen as $r_j^1 = 1/2$ with $\alpha_j^1 = \beta_j^1 = 1$.

Self-Healing

The BS sets two thresholds: TH_{change} and $TH_{isolate}$. Here, $0 < TH_{isolate} < TH_{change} < 1$. The BS will pick the sensors whose trust values are lower than TH_{change} but higher than $TH_{isolate}$. These sensors are highly suspected to be malicious or be affected by malicious sensors. The BS will instruct these sensors to change their parent nodes on the sync-tree. By doing so, if a sensor is good, it will less likely be affected by malicious sensors.

If a sensor's trust value drops below $TH_{isolate}$, this sensor is identified as misbehaving. The BS can instruct all the child nodes of the misbehaving sensor to choose a different parent node. Thus, this sensor will be isolated on the sync-tree and cannot launch the misleading attack in the future.

The instructions about changing sync-tree topology are generated by the BS and sent to the sensors. With more rounds of time synchronization, the topology of the sync-tree will evolve such that misbehaving sensors will be isolated. In other words,

with the distributed abnormality detection performed by the sensors and the centralized misbehaving detection performed by the BS, the sensor network can heal itself from attacks against time-synchronization.

4.5 Performance Evaluations

4.5.1 Simulation Setup

The proposed DAS scheme has been implemented in TOSSIM, the simulator of TinyOS [14]. There are total 100 sensors, with 10m transmission range, randomly deployed in a rectangular area with size 50m $\times$ 50m or 100m $\times$ 100m. The smaller network size represents a dense network where the average number of neighbors of a sensor is 12. The larger network size represents a sparse network where the average number of neighbors is 4. The level discovery is achieved through a simple TinyOS beacon protocol. A sensor sends the information about its level, parent and children to the BS, or sends warning reports to the BS through 36 bytes long *report packets*. The BS sends self-healing instructions to sensors through 10 bytes long *instruction packets*. The synchronization request and reply messages are also 10 bytes long.

Although TPSN can achieve μs clock accuracy between two real sensors[7], TOSSIM cannot perform μs level simulations. Therefore, we choose the interval between time synchronization to be 1000 seconds. Thus, the clock drift will be at the ms level. Time synchronization is performed for every 1000 seconds. Sensor i's clock time follows $t_i^k = (1 + \rho_i)\hat{t}_i^{k-1}$, where ρ_i is sensor i's clock drift rate and is randomly generated by a uniform distribution between -5 and 5ms/1000s. Here, 5ms/1000s means that a sensor's clock can drift up to 5ms during 1000s.

Finally, the important thresholds in DAS are set up as follows. In self-verification, the upper bound of O_k in (4.7) is chosen as 5ms. As shown in (4.3), after the synchronization interval is determined, the skew value θ_k is determined solely by the z value. Setting a threshold for θ_k is equivalent to setting a threshold for z. Since z approximately follows a Gaussian distribution, $N(\mu, \sigma)$, with $\mu = 0$ and $\sigma^2 = m(5 + 5)^2/12$. We set $m = 6$ and get $\sigma = 7$. Thus, with a 99.97% confidence, the z value will fall in

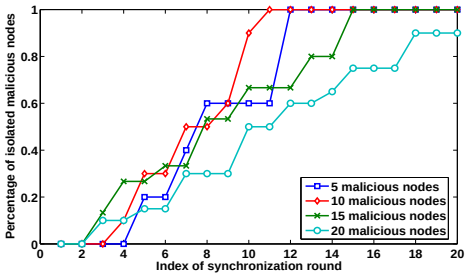

Figure 4.4. Detection of misleading attacks (N=100, 50m by 50m)

the interval $[\mu - 3\sigma, \mu + 3\sigma]$. Thus, the upper bound of z is set to be 21ms. In addition, since detecting abnormal delay, as shown in (4.5), is straightforward for any defense mechanisms. Smart attackers will not cause abnormal delay. In the simulation, we set $P_d = 0$. In local verification, the threshold TH_{lv} is chosen to be 2ms. For the detection of misbehaving sensors, TH_{change} and $TH_{isolate}$ are chosen dynamically. Particularly, $TH_{change} = 0.6 \cdot T_{avg}$ and $TH_{isolate} = 0.4 \cdot T_{avg}$, where T_{avg} is the average of the current trust values of all sensors.

4.5.2 Performance under Misleading Attack

Malicious sensors can perform the misleading attack in different ways. They can start to attack at the beginning (in round 1), or behave well for sometime and then start to attack. The later case can be easily captured by the self-verification process, and therefore is easier to be detected than the former case. In addition, attackers can mislead arbitrarily in each round. In this work, we test DAS against strong attacking behaviors. That is, all malicious sensors collude to mislead others toward the same wrong clock (10 ms faster than the correct time) in the first round, and then try to make others believe this wrong clock time in the following rounds.

Figure 4.4 shows the DAS's capability of detecting malicious sensors, when there are 5, 10, 15 or 20 malicious nodes in a 50m by 50m network. The horizontal axis is the index of rounds of time synchronization, and the vertical axis is the percentage of malicious sensors that are isolated on the sync-tree. It is observed that all malicious nodes are detected after 12 rounds when the percentage of malicious nodes is less than

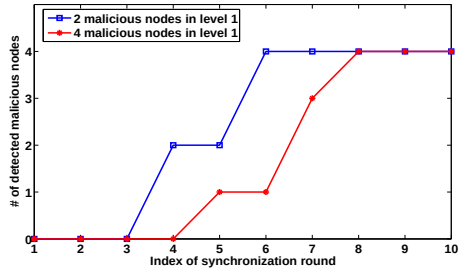

Figure 4.5. Malicious nodes detection under wormhole attack (N=100, 100m by 100m) 15%. When the number of malicious nodes reaches 20, DAS can still detect 18 of them after round 18.

4.5.3 Performance under Wormhole Attack

The wormhole attack is simulated in a sparse network (100m by 100m) because the wormhole attack is stronger if there are more levels on the sync-tree. In the simulation, two wormholes are created. Each wormhole links an attacker at level 1 and another attacker that locates far away from the BS. Two cases are simulated. In the first case, the wormholes brings the far away sensors to level 2, as described in Section 4.3.2. In this case, there are two attackers at level 1 and two attackers at level 2. In the second case, the wormholes bring the far away sensors to level 1. To achieve this, the attacker at level 1 forwards messages (i.e. serves as relay) between the far away attacker and the BS, and makes the BS think that the far away attacker is its neighbor. From the attackers' point of view, case 2 is more difficult to launch than case 1, but can cause bigger damage. In case 2, there will be four attackers at level 1 on the sync-tree. In addition, the attackers launch the misleading attack described in Section 4.5.2.

Figure 4.5 and Figure 4.6 shows the number of victims and the detection rate of DAS under the wormhole attack. Two major observations are made. First, the wormhole attack can significantly increase the number of victims. As shown in Figure 4.6, two wormholes can lead to 23 victims in case 1 and 36 victims in case 2, when no defense mechanisms are used. Second, DAS works well under the wormhole attack. As shown in Figure 4.5, all four attackers are detected in 6 rounds in case 1, and in 8 rounds in

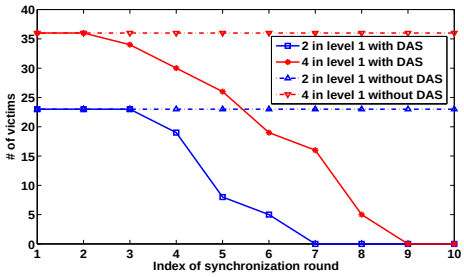

Figure 4.6. # of victims under wormhole attack (N=100, 100m by 100m)

case 2. This is due to the negative mark and trust value calculation methods. Based on our methods, the wormhole attackers are always included in the responsible set. More victims are there, more local verifications are performed, and lower trust values will be assigned to the attackers.

4.6 Conclusion

This paper identified several attacks against time synchronization protocols in sensor networks. Among them, the most interesting one was the combination of the wormhole attack and the misleading attack. It was a powerful attack but had not been discovered previously. More importantly, this paper presented a set of approaches, called DAS, included abnormality detection through self-verification and local-verification, misbehaving detection based on negative marks and trust values, and self-healing through sync-tree topology change. Simulations demonstrated that DAS can quickly detect and defeat the misleading attack and the wormhole attack, with reasonable implementation overhead.

List of References

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

[2] K. Sun, P. Ning, and C. Wang, "Secure and resilient clock synchronization in wireless sensor networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 2, Feb. 2006.

[3] H. Song, S. Zhu, and G. Cao, "Attack-resilient time synchronization for wireless sensor networks." *Elsevier Journal of Ad Hoc Networks*, vol. 5, no. 1, 2007.

[4] M. Manzo, T. Roosta, and S. Sastry, "Time synchronization attacks in sensor networks." in *Proc. the 3rd ACM workshop on Security of ad hoc and sensor networks SASN*, Alexandria, VA, Nov. 2005.

[5] S. Ganeriwal, S. Capkun, C. Han, and M. Srivastava, "Secure time synchronization service for sensor networks," in *Proc. Wireless Security Workshop (WiSe)*, New York City, USA, Nov. 2005.

[6] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," in *Proc. Fifth Symposium on Operating Systems Design and Implementation (ONDI 2002)*, Boston, MA, Dec. 2002.

[7] S. Ganeriwal, R. Kumar, and M. Srivastava, "Timing-sync protocol for sensor networks," in *Proc. ACM SenSys'03*, Los Angeles, CA, Nov. 2003.

[8] M. Maroti, B. Kusy, G. Simon, and A. Ledezi, "The flooding synchronization protocol," in *Proc. the second ACM Conference on Embedded Networked Sensor Systems(SenSys)*, Baltimore, MD, Nov. 2004.

[9] B. Sundararaman, U. Buy, and A. Kshemkalyani, "Clock synchronization for wireless sensor networks: A survey," *Ad-hoc Networks*, vol. 3, no. 3, pp. 281–323, May 2005.

[10] Y.-C. Hu, A. Perrig, and D.B.Johnson, "Packet leashes: A defense against wormhole attacks in wireless networks." in *Proc. IEEE Inforcom*, Apr. 2003.

[11] W. Wang and B. Bhargava, "Visualization of wormholes in sensor networks." in *Proc. the ACM workshop on wireless security 2004*, Oct. 2004.

[12] A. Josang and R. Ismail, "The beta reputation system." in *Proc. the 15th Bled Electronic Commerce Conference*, Bled, Slovenia.

[13] S. Ganeriwal and M. Srivastava, "Reputation-based framework for high integrity sensor networks," in *Proceedings of ACM Security for Ad-hoc and Sensor Networks (SASN)*, Washington, D.C., USA, Oct. 2004.

[14] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: Accurate and scalable simulation of entire tinyos applications." in *Proc. the First ACM Conference on Embedded Networked Sensor Systems (SenSys03)*, Los Angeles, CA.

MANUSCRIPT 5

Trust Establishment in Distributed Networks: Analysis and Modeling

Abstract

Recently, trust establishment is recognized as an important approach to defend distributed networks, such as mobile ad hoc networks and sensor networks, against malicious attacks. Trust establishment mechanisms can stimulate collaboration among distributed computing and communication entities, facilitate the detection of untrustworthy entities, and assist decision-making in various protocols. In the current literature, the methods proposed for trust establishment are always evaluated through simulation, but theoretical analysis is extremely rare. In this paper, we present a suite of approaches to analyze trust establishment process. These analysis approaches are used to provide in-depth understanding of trust establishment process and quantitative comparison among trust establishment methods. The proposed analysis methods are validated through simulations.

5.1 Introduction

Recently, *building trust* among distributed network entities has been recognized as a new security approach to simulate cooperation and improve security in distributed networks [**?**, 1]. Briefly speaking, with trust establishment mechanisms, network entities will know whether and how much they can trust other network entities. This trust information guides network entities not to take highly risky actions such as asking the nodes with low trust value to forward packets or perform data aggregation. As a consequence, network performance and robustness will be improved. Furthermore, trust establishment mechanisms provide an incentive for cooperation. The network entities will behavior more responsibly with the expectation that their trust value or reputation will be hurt by non-cooperative or destructive behaviors.

The research on the subject of trust in computer networks has been extensively

performed for a wide range of application scenarios, including authorization and access control[2, 3, 4, 5, 6, 7, 8, 9, 10], electronics commerce[**?**, 11, 12], peer-to-peer networks[13, 14], ad hoc and sensor networks[15, 16, 17, 18, 19, 20, 21], and pervasive computing[22, 1, 23].

Currently, the methods for trust establishment in computer networks have always been evaluated through simulations for specific applications[1]. Theoretical analysis is extremely rare in this field. Without performing extensive simulations, researchers can hardly compare different trust establishment methods, which are often proposed for different applications. Even with simulations, simulation results cannot be easily decoupled from specific simulation implementations. In this paper, we present a set of methods that can analyze the trust establishment process in different application scenarios. The proposed methods lead to insightful understanding and comparison among existing trust establishment approaches. The analysis methods are developed in five steps.

- *Step 1*: Understanding basic components in trust establishment/reputation systems;
- *Step 2*: Developing the architecture for performing theoretical analysis, including specifying inputs, outputs and basic modules;
- *Step 3*: Solving problems in individual modules;
- *Step 4*: Validating the analysis methods;
- *Step 5*: Utilizing the analysis methods to *quantitatively* compare different trust models, and provide in-depth understanding of trust establishment process.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 presents step 1, and Section 4 presents step 2 and 3. Section 5 describes step 4 and step 5. The conclusion is drawn in Section 6.

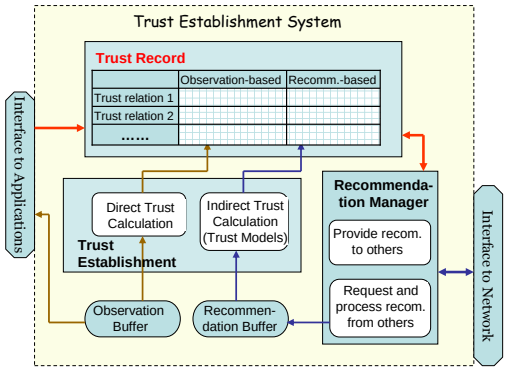

Figure 5.1. Basic Elements in Trust Establishment System

5.2 Related Work

As discussed in Section 5.1, there have been a lot of work on trust establishment for various applications. Simulation has been the dominating evaluation method. To our best knowledge, the analytical study on trust establishment was only presented in [24] previously. In this work, a simple trust evaluation method for autonomous networks is analyzed. Close form solutions are obtained and interesting results are observed. However, the theories in this work cannot be directly applied to more complicated trust evaluation methods. For examples, only simple trust models are supported. In fact, close form solutions are only possible for very simple systems. Thus, the method in [24] is not sophisticated enough to provide theoretical comparison among different trust establishment methods.

5.3 Core Design of Trust Establishment Methods

Trust can be established in a centralized or distributed manner. The later is more suitable for distributed networks, such as MANET and sensor networks. This work focuses on distributed trust establishment where network participants have similar responsibilities. The basic elements in distributed trust-establishment systems are shown in Figure 5.1 and described as follows.

Trust Record stores information about trust relationship and associated trust values. A *trust relationship* is always established between two parties for a specific action. That is, one party trusts the other party to perform an action. In this work, the first

party is referred to as the *subject* and the second party as the *agent*. A notation $\{subject : agent, action\}$ is introduced to represent a trust relationship. For each trust relationship, one or multiple numerical values, referred to as *trust values*, describe the level of trustworthiness.

There are two common ways to establish trust in computer networks. First, when the subject can directly observe the agent's behavior, *direct trust* can be established. Second, when the subject receives recommendations from other entities about the agent, *indirect trust* can be established.

Direct Trust is established upon observations on whether the previous interactions between the subject and the agent are successful. The observation is often described by two variables: s denoting the number of successful interactions and f denoting the number of failed interactions. That is, the direct trust value can be calculated as $f_{DT}(s, f)$. The notation T^d_{AB} denotes the direct trust values between node A and B. Thus, $T^d_{AB} = f_{DT}(s, f)$, and T^d_{AB} can be a vector.

Recommendation trust is a very special direct trust. It is for trust relationship $\{subject$: $agent$, making correct recommendations$\}$. Assume that the subject can judge whether a recommendation is correct or not, and this subject receives s_r good recommendations and f_r bad recommendation from the agent. Then, the recommendation trust value can be calculated as $f_{DT}(s_r, f_r)$. Recommendation trust is important for establishing indirect trust. The recommendation trust between node A and B is denoted by T^r_{AB}.

Indirect Trust: Trust can transit through third parties. For example, if A and B have established a recommendation trust relationship and B and Y have established a direct trust relationship, then A can trust Y to a certain degree if B tells A its trust opinion (i.e. recommendation) about Y. This phenomenon is called *trust propagation*. Indirect trust is established through trust propagations.

Two key factors determine the indirect trust establishment. The first is to determine

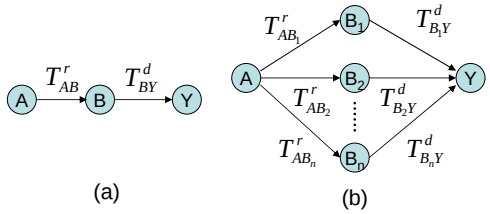

Figure 5.2. Trust Propagation for Indirect Trust Establishment

when and from whom the subject can collect recommendations. For example, in an ad hoc network, a node may collect recommendations only from its one-hop neighbors, or from all nodes in the network. This affects the number of recommendations can be collected and the overhead of collecting recommendations. This first factor is referred to as the *recommendation mechanism*.

The second is to determine how to calculate indirect trust value based on recommendations. This calculation is governed by *trust models*. A trust model usually contains two parts: concatenation model and multipath model, as illustrated in Figure 5.2(a) and 5.2(b), respectively. In Figure 5.2(a), B observes the behavior of node Y and establishes direct trust in Y with trust value T^d_{BY}. Node A has recommendation trust in B with trust value T^r_{AB}. Node B provides recommendation about Y by telling A the value of T^d_{BY}. The concatenation model is a function that calculates the indirect trust values between A and Y, denoted by T^{ind}_{AY} from T^d_{BY} and T^r_{AB}. This function is denoted by $f_{ctp}(.)$, and

$$T^{ind}_{AY} = f_{ctp}(T^d_{BY}, T^r_{AB}) \tag{5.1}$$

In Figure 5.2(b), A receives recommendations from multiple nodes. The multipath model is a function that combines trust established through multiple paths. This function is denoted by $f_{mtp}(.)$, and

$$T^{ind}_{AY} = f_{mtp}\left(\{f_{ctp}(T^d_{B_iY}, T^r_{AB_i})\}_{i=1,2,\cdots}\right) \tag{5.2}$$

As a summary, a trust establishment system needs to specify at least three key elements: (1) direct trust calculation, (2) recommendation mechanism, and (3) trust models.

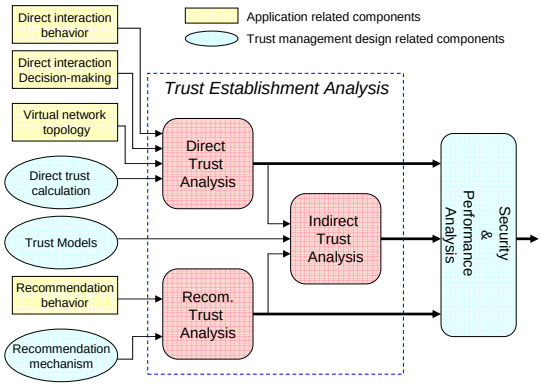

Figure 5.3. Structure of Trust Establishment Analysis

5.4 Trust Establishment Analysis

5.4.1 Overview

The overall structure of the proposed analysis method is illustrated in Figure 5.3. It contains four types of components.

*T*ype 1 components describe specific design of trust evaluation methods. They include direct trust calculation, recommendation mechanism, and trust models.

*T*ype 2 components are abstractions that describe the interaction between applications and trust establishment. They allow us to take the application context into consideration but not restrict the analysis to detailed network setup. Rectangles in Figure 5.3 represent type 2 components.

*T*ype 3 components are the core of trust establishment analysis. They include direct trust analysis, indirect trust analysis and recommendation trust analysis. The design and implementation of type 3 elements is one of the main contributions of this work.

*T*ype 4 component generates a set of metrics describing the performance of trust evaluation methods. The performance analysis module is type 4.

The details of these components are presented in the following subsections.

5.4.2 Inputs

In this section, we introduce type 1 and type 2 components, i.e the inputs to the trust analysis module. These inputs describe the application scenarios and the design of trust establishment methods.

Direct Interaction Decision-making Model and Virtual Network Topology

The probability that one entity will interact with another entity is an important aspect that will influence trust establishment. This aspect is closely related with application scenarios. This probability is described by the Direct Interaction Decision-making (DID) Model. To simplify the analysis, we assume that this probability is determined by two factors: closeness and trust among the entities.

We introduce the concept *virtual distance*. The virtual distance between entity A and entity B, denoted by d_{AB}, represents the closeness between A and B in terms of having direct interactions. Generally speaking, smaller is d_{AB}, more likely A and B will directly interact with each other, when no trust information is available. In an ad hoc network or a sensor network, virtual distance might be directly related with the real physical distance. The *virtual network topology model* specifies the virtual distance among all network participants.

For the DID model, we define $f_I(d_{AB}, T_{AB})$ as the probability that the node A will interact with node B in some time interval, where d_{AB} denotes the virtual distance and T_{AB} denotes the trust value between A and B. Furthermore, it is assumed that $f_I(d_{AB}, T_{AB}) = g(d_{AB}) \cdot h(T_{AB})$. That is, the influence of virtual distance and trust values can be considered separately. The DID model should specify $g(.)$ and $h(.)$.

Direct Interaction Model and Recommendation Behavior Model

These two models describe the behavior of good and bad entities when they interact with other entities and when they provide recommendations.

In the simplest model, there are only two types of nodes: good and bad, and only two outcomes after an interaction: success and failure. When a node interacts with a good node, the interaction will succeed with probability θ_a. When a node interacts with a bad node, the interaction will succeed with probability β_a. This simple direction interaction behavior model can be described by two parameters: θ_a and β_a.

Similarly, in the simplest recommendation behavior model, there are only two types

of recommendations: correct and incorrect. If A's recommendation about X agrees with X's behavior, the recommendation is correct. Otherwise, the recommendation is incorrect. This recommendation behavior model is also described by two parameters: θ_r, which is the probability that the recommendation is correct when it is from a good node, and β_r, which is the probability that the recommendation is correct when it is from a bad node.

Direct Trust Calculation and Trust Models

As discussed in Section 5.3, the function $f_{DT}(.)$, $f_{ctp}(.)$ and $f_{mtp}(.)$ represent the core design of trust establishment methods. As the inputs to our analysis, these three functions need to be specified.

Recommendation Mechanism

A brief procedure of collecting and using recommendations is shown in Procedure 7.

Procedure 7 A wants to establish indirect trust in B

1: A requests the nodes in A's *recommendation range* for recommendations about B. (If node X is in A's recommendation range and has established direct trust in B, node X will send its recommendation to A.)
2: A puts all received recommendations in a buffer, and calculates indirect trust in B using the trust model.
3: If A observes B's behavior after establishing indirect trust in B, A can compare B's behavior with the recommendations in the buffer, and then update the recommendation trust in the nodes that have provided recommendations about B.

From this procedure, one can see that the recommendation mechanism needs to specify two things: recommendation range and how to update recommendation trust.

Recommendation range can be described by the recommendation distance d_{rec}. Node X is in node A's recommendation range if $d_{AX} \leq d_{rec}$. A node can only collect recommendations from the nodes in its recommendation range.

To establish and update recommendation trust, the first step is to estimate whether a particular recommendation is correct or incorrect. For example, if the difference be-

Inputs Blocks	Parameters or functions
Virtual network topology	Virtual distance among entities
Direct interaction decision-making	$f_I(d_{AB}, T_{AB})$
Direct interaction behavior	θ_a, β_a
Recommendation behavior	θ_r, β_r
Direct trust calculation	$f_{DT}(s, f)$
Trust Models	$f_{ctp}(.)$, $f_{mtp}(.)$
Recommendation mechanisms	d_{rec}, way to determine s_r and f_r

Table 5.1. Modules that provide inputs to trust establishment analysis

tween the recommendation about B and A's observation about B's behavior is smaller than a threshold, this recommendation is considered to be correct. Then, the recommendation trust can be calculated based on s_r and f_r values as described in Section 5.3.

In the process of modeling application scenarios and trust establishment methods, there are many simplifications. Even with simplifications, the most important features of trust establishment methods are maintained. The input parameters of the analysis modules are summarized in Table 5.1.

5.4.3 Direct Trust Establishment

A node may continuously observe other nodes' behavior. However, it is not necessary to update trust record whenever new observations are made. Thus, we introduce the concept of *round*. Assume that there are total N nodes in the network and let k denote the index of the round. The trust establishment process in round k is as follows.

It is noted that direct trust, as well as indirect trust and recommendation trust, can be updated at the end of round k. When studying the direct trust, the possible update in indirect trust and recommendation trust is not considered. Therefore, Procedure 8 does not include the establishment of indirect and recommendation trust.

Figure 5.4 shows the process of establishing direct trust between two nodes: say A and B. In this figure, each rectangle represents a state. Each state is described by two

Procedure 8 Establishing direct trust in round k

1: **for** A=1:N **do**
2: Node A determines a set of nodes with which A will interact with in round k. This set of nodes, denoted by $\mathbf{B}$, is determined by the direct interaction decision-making model. If the model uses trust values, those trust values are from the trust record established at the end of round $k-1$.
3: The interactions between A and the nodes in $\mathbf{B}$ may succeed or fail, depending on the direct-interaction-behavior model.
4: Based on the outcome of the interactions, node A updates its direct trust record about the nodes in $\mathbf{B}$.
5: **end for**

parameters (s, f). As discussed earlier, s is the number of successful interactions, f is the number of failed interactions, and the direct trust value is calculated from (s, f). Thus, the direct trust value between A and B, i.e. T_{AB}^d, depends only on which state the system is at. In other words, if we can determine the probability of the states at round k, we can obtain the probability mass function (pmf) of the direct trust values at round k.

The transition between the states is uniquely determined by three inputs to trust analysis: direct interaction decision-making model, direct interaction behavior model, and the calculation of direct trust. As shown in Table 5.1, these three models are represented by $f_I(d_{AB}, T_{AB})$, $f_{DT}(s, f)$, θ_a and β_a. In particular, if the current state is (s, f) at round k, the system can possibly transfer to the following states in round $k+1$.

- to state $(s+1, f)$ with probability $\theta \cdot f_I(d_{AB}, f_{DT}(s, f))$;
- to state $(s, f+1)$ with probability $(1-\theta) \cdot f_I(d_{AB}, f_{DT}(s, f))$;
- remain at state (s, f) with probability $1 - f_I(d_{AB}, f_{DT}(s, f)))$;

where $\theta = \theta_a$ if B is a good node and $\theta = \beta_a$ if B is a bad node.

Based on Figure 5.4, one can calculate the pmf of T_{AB}^d for any given round $\hat{k}$ using a simple program shown in Procedure 9. Here, let $p_{s,f,k}$ denote the probability that $T_{AB}^d = f_{DT}(s, f)$ at round k.

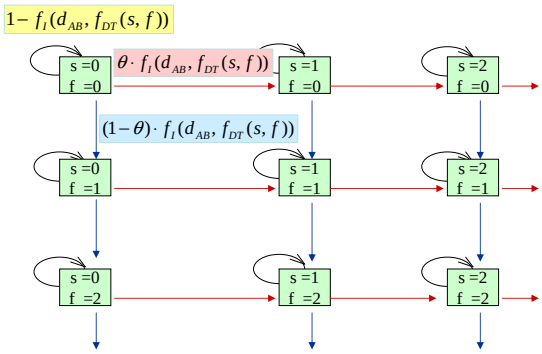

Figure 5.4. The model for direct trust establishment

Procedure 9 Calculating the pmf of direct trust values

1: Set $p_{0,0,0} = 1$ and all other $p_{s,f,k}$ values as 0.
2: **for** $k = 1 : \hat{k}$ **do**
3: **for** all possible (s, f) values **do**
4: update $p_{s,f,k+1}$ using (5.3).
5: **end for**
6: **end for**

The equations used in Procedure 9 is

$$\begin{aligned} p_{s,f,k+1} &= p_{s,f,k}(1 - f_I(d_{AB}, f_{DT}(s,f))) \\ &\quad + p_{s-1,f,k} \cdot \theta \cdot f_I(d_{AB}, f_{DT}(s-1,f)) \\ &\quad + p_{s,f-1,k} \cdot (1-\theta) \cdot f_I(d_{AB}, f_{DT}(s,f-1)). \end{aligned} \tag{5.3}$$

With $p_{s,f,k}$, it is easy to generate the pmf of T^d_{AB} at any given round. It is noted that if the function $f_{DT}(s, f)$ yields the same value for some different (s, f) values, $p_{s,f,k}$ and the pmf of T^d_{AB} are not exactly same. Some simple conversion is needed.

As a summary, the direct trust establishment module can output the statistical distribution of direct trust value between arbitrary pair of nodes in any given round.

5.4.4 Recommendation Trust Establishment

In this section, the goal is to calculate the pmf of recommendation trust values. From Procedure 7, one can see that node B can make recommendations to node A about node Y in round k if and only if the following three conditions are satisfied.

(C1) A requests recommendations about node Y in round k.

(C2) B has established direct trust in node Y in the previous $k-1$ rounds.

(C3) B is in A's recommendation range, i.e. $d_{AB} \leq d_{rec}$.

Through B's recommendation about Y, A can establish recommendation trust in B in round k if one additional condition is satisfied:

(C4) A can judge whether B's recommendation about Y is correct or not.

To make the analysis manageable, the following assumptions are made.

(A1) A requests recommendations about node Y in round k if A selects node Y to interact with in round k.

(A2) A makes accurate judgment on whether B's recommendation about Y provided in round k is honest or not, if A establishes direct trust with Y in round k or in the previous rounds.

(A3) The recommendation mechanism follows Procedure 7.

(A4) One direct interaction is sufficient to establish direct trust. For example, if B has one direct interaction with node Y, then B can provide recommendations about node Y.

Here, the first assumption (A1) is made because we have to know when condition 1 will be satisfied. This assumption can greatly simplify the analysis. The second assumption (A2) can simplify the handling of condition 4. In particular, with (A1) (A2) and (A3), condition 4 is satisfied whenever condition 1 is satisfied. For condition 3, we only need to multiply whatever probability results we obtained by $\mathbf{1}(d_{AB} \leq d_{rec})$, where $\mathbf{1}(statement)$ equals to 1 when the statement is true and equals to 0 otherwise. To make a concise presentation, we omit the term $\mathbf{1}(d_{AB} \leq d_{rec})$ in the rest of this section.

From above discussion, one can see that

$$Pr\{B \text{ makes recommendation to } A \text{ about } Y \text{ in round } k\}$$

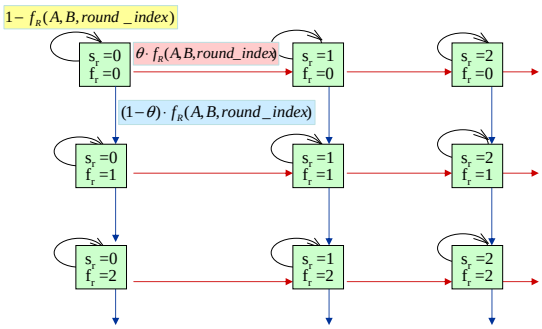

Figure 5.5. The model for recommendation trust establishment

$$= Pr\{\text{condition 1 is satisfied}\} \cdot Pr\{\text{condition 2 is satisfied}\} \tag{5.4}$$

Let $T^d_{AY,k}$ denote the value of T^d_{AY} in round k. From the direct interaction model, one can see that

$$Pr(\text{condition 1 is satisfied}) = f_I(d_{AY}, T^d_{AY,k-1}), \tag{5.5}$$

$$Pr(\text{condition 2 is satisfied}) = 1 - \prod_{i=1}^{k-1}(1 - f_I(d_{BY}, T^d_{BY,i-1})), \tag{5.6}$$

From (5.4) (5.5) and (5.6), we get

$$Pr(B \text{ makes recommendation to } A \text{ in round } k)$$

$$= \sum_Y f_I(d_{AY}, T^d_{AY,k-1}) \cdot \left(1 - \prod_{i=1}^{k-1}(1 - f_I(d_{BY}, T^d_{BY,i-1}))\right) \tag{5.7}$$

We assume that $\sum_Y f_I(d_{AY}, T^d_{AY,k-1}) < 1$. That is, a node is unlikely to interact with more than one node in one round. Due to this assumption and (A1)-(A4), the probability calculated in equation (5.7) equals to the probability that A observes whether B makes one honest or dishonest recommendation in round k. We denote this probability as $f_R(A, B, k)$, where $f_R(.)$ is a function of node index (A and B), round index (k), and trust values among many nodes in round $k-1$, as indicated in (5.7).

With $f_R(A, B, k)$ and the recommendation behavior model, we are ready to calculate the pmf of the recommendation trust.

Figure 5.5 shows the model for calculating recommendation trust. Similar as in Figure 5.4, the rectangles represent states, which are described by (s_r, f_r). If the current state is (s_r, f_r) at round k, the systems can transfer to the following states in round $k+1$:

- to state $(s_r + 1, f_r)$ with probability $\theta \cdot f_R(A, B, k)$;
- to state $(s_r, f_r + 1)$ with probability $(1 - \theta) f_R(A, B, k)$;
- remain at state (s_r, f_r) with probability $1 - f_R(A, B, k)$;

where $\theta = \theta_r$ if B is a good node and $\theta = \beta_r$ when B is a bad node.

Let T^r_{AB} represent the recommendation trust between A and B. Let $p^r_{s_r,f_r,k}$ denote the probability that $T^r_{AB} = f_{DT}(s_r, f_r)$ at round k. Based on Figure 5.5, it is not difficult to calculate $p^r_{s_r,f_r,k}$, using a procedure similar to Procedure 9.

5.4.5 Indirect Trust Establishment

The last module of trust establishment analysis is to obtain the distribution of indirect trust values. As discussed in Section 5.3, indirect trust is calculated from direct trust and recommendation trust. The calculation is governed by the trust models.

Figure 5.2(b) shows the scenario of trust propagation that will be analyzed in this section. It is noted that this analysis only allows one-hop concatenation trust propagation. This limitation is due to the following reasons. As the length propagation chain increases, the cost of collecting recommendations increases rapidly (e.g. exponentially in [19]) and the trust between the subject and the agent degrades quickly. Therefore, many trust establishment methods only allow one-hop trust propagation, such as in [21]. Additionally, the analysis for multi-hop trust propagation is extremely difficult.

As discussed in Section 5.3, the indirect trust between A and Y is calculated from direct trust and recommendation trust using trust models. When the pmf of direct trust ($T^d_{B_iY}$) and the pmf of recommendation trust $T^r_{AB_i}$ are obtained using the procedure in Section 5.4.3 and 5.4.4, the pmf of T^{ind}_{AY} can be calculated.

It is noted that direct trust values are calculated from (s, f), where s and f are integers. Therefore, the values of T^d_{AY} are discrete. Similarly, the recommendation trust values are also discrete. The fact that T^d_{AY} and $T^r_{AB_i}$ are discrete random variables enables simple numerical calculation of the pmf of T^{ind}_{AY}. It is also noted that the complexity

of the calculation increases rapidly with the number of rounds.

5.4.6 Outputs

As shown in Figure 5.3, for any given pair of nodes, our analysis can generate the pmfs of the direct trust, recommendation trust and indirect trust. Of course, these pmfs are for specific trust establishment methods under certain application scenarios.

Based on these pmfs, more metrics can be calculated by the performance analysis module. For any pair of nodes (A, Y), one can compute the mean and variance of direct/indirect/recommendation trust between A and Y as a function of round index. In addition, when the trust values are used in malicious node detection algorithms, the detection rate and false alarm rate can be computed.

5.5 Development, Testing, and Results

We implement an analysis tool in Matlab based on the approaches presented in Section 5.4. This tool takes the inputs described in Table 5.1, and outputs the statistical distribution of direct, recommendation and indirect trust between two arbitrary nodes in the network.

To validate the analysis methods proposed in the paper, we build a simulation testbed that takes similar inputs as the the analysis tool. This testbed simulates the trust establishment process and estimates the pmf of trust values based on a large number of tests.

It is important to note that this testbed can be used to study more complicated scenarios in which theoretical analysis is impractical. For example, when the attackers launch the sophisticated attacks described in [25], analysis becomes very difficult to manage. In this paper, the testbed is used to validate the analysis methods. In the future work, the testbed can be used to study more sophisticated scenarios. The importance of analysis model is not undermined by the testbed. The critical advantage of analysis is that it can describe the system behavior as parameters change continuously. In addition, since simulation is reliable only if a large number of tests are performed, simulation

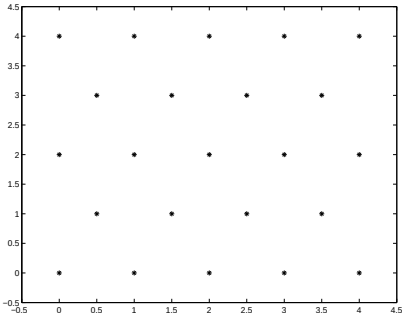

Figure 5.6. An example of 2D virtual topology

requires much higher computation than analysis.

In this section, we first compare the analysis and simulation results, and then utilize the analysis tool to derive important research results.

5.5.1 Validating Analysis Through Simulations

The first experiment is to show the process of establishing direct trust values. The inputs are chosen as follows. The 2D virtual network topology is shown in Figure 5.6. For direct trust calculation, $f_{DT}(s, f) = (s+1)/(s+f+2)$, where s and f are defined in Section 5.3. This has been used in many trust establishment methods. In the DID model, $g(d_{AB}) = 1/N$ and $h(T_{AB}) = t(T_{AB})$, where N is the total number of nodes in the network, and function $t(T_{AB}) = f_{DT}(s, f)$. In the direct interaction model, $\theta_a = 0.9$ and $\beta_a = 0.1$.

Figure 5.7 shows the pmf of the direct trust values between two nodes at round 5, 20, 40 and 120. The left four plots are for $T^d_{AY_1}$ and the right four plots are for $T^d_{AY_2}$. A and Y_1 are good node, and Y_2 is a bad node. Y_1 and Y_2 are two hops away from A. In all plots, the lines marked with $\diamond$ are for analysis results, and the lines marked with $*$ are for simulation results. One can see that the analysis and simulation match well. When two nodes do not have direct interaction, (i.e. $s = 0$ and $f = 0$), the trust value is 0.5. This is why some plots have a peak at 0.5.

Similar to the first experiment, we verify the analysis of recommendation trust and indirect trust through simulations. For example, Figure 5.8 shows the mean and

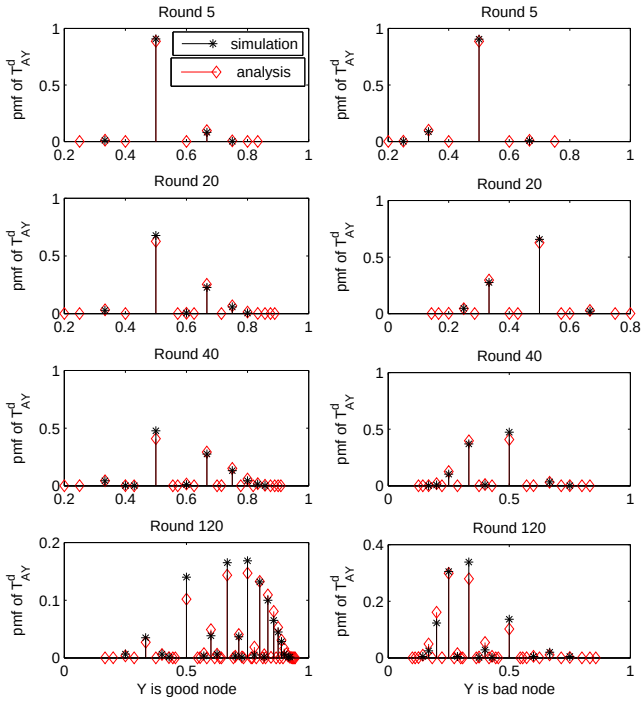

Figure 5.7. Statistical distribution of direct trust values

variance of the recommendation trust between A and Y_1 as a function of round index. In this experiment, $\theta_r = 0.8$, $\beta_r = 0.5$, and $g(d_{AB})h(T_{AB}) = 1/N$. The trust models are chosen to be the simple model described in Section 5.5.3.

As expected, the mean of the recommendation trust increases as the round index increase. Initially, one would expect that the variance value decreases with the round index. However, it is not completely true. The indirect trust values are discrete. In the first few rounds, there are only a few possible $T^r_{AY_1}$ values, which yields a relatively small variance. As the round index increases, $T^r_{AY_1}$ can take more values, which yields a larger variance. As the round index further increases, the node A collects more observations about Y_1, and more observations lead to a smaller variance in $T^r_{AY_1}$.

5.5.2 Probability of Trust Establishment

Trust values are used to assist decision-making in distributed systems. From the application points of view, the earlier trust can be established the better. Therefore, it is

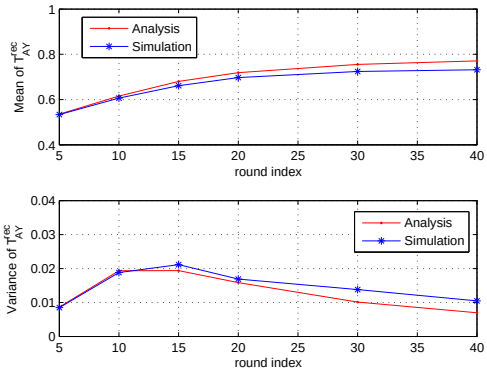

Figure 5.8. Mean and variance of recommendation trust

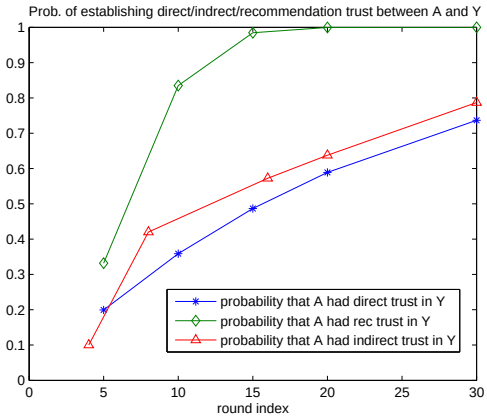

Figure 5.9. Probability of establishing direct/indirect/recommendation trust

important to evaluate how likely trust can be established as the round index increases.

In Figure 5.9, the probability that direct, recommendation, and indirect trust can be established between node A and Y_1 are shown as a function of the round index. The simulation setup is similar to that in Section 5.5.1. Several observations are made. First, compared with direct trust, indirect trust is more likely to be established. Thus, with recommendation mechanism, the subject can use indirect trust information before direct trust can be established. Second, it is highly likely that recommendation trust is established long before the direct trust. Thus, when A tries to determine whether Y is trustworthy or not, recommendation trust between A and Y should be used, especially at the beginning of trust establishment (i.e. when the round index is small).

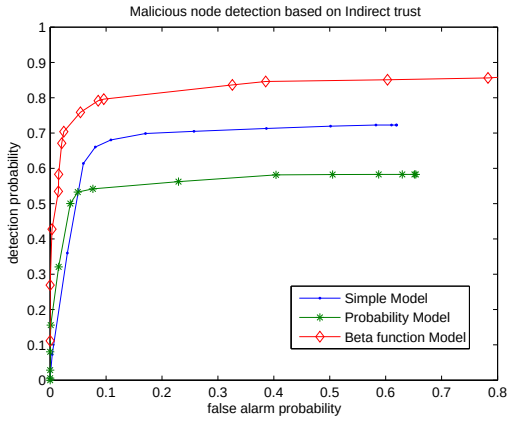

Figure 5.10. Detection probability v.s. false alarm rate (detection are based on indirect trust, round index=30).

5.5.3 Comparison Among Trust models

An important application of the proposed trust analysis tool is to facilitate comparison among different trust establishment methods. As discussed earlier, the design of trust models is critical for indirect trust establishment. Thus, we compare different trust models by examining the pmf of $T_{AY_1}^{ind}$ and $T_{AY_2}^{ind}$. Here, $T_{AY_1}^{ind}$ and $T_{AY_2}^{ind}$ represent the indirect trust of a good and a bad node, respectively.

Assume that we are going to use the indirect trust to determine whether a node is good or bad. Given a detection threshold D_{th}, if $T_{AY}^{ind} < D_{th}$, node Y is detected as a suspicious node.

Based on the pmf of $T_{AY_1}^{ind}$ and $T_{AY_2}^{ind}$, for a given D_{th}, we calculate the *detection probability* as the probability that Y_2 is marked as suspicious, and the *false alarm probability* as the probability that Y_1 is marked as suspicious. Furthermore, by changing D_{th}, we can obtain the curve: detection probability v.s. false alarm probability. Such curves for several different trust models are shown in Figure 5.10. The three models evaluated in this experiment are

- The *simple model* used in many trust evaluation schemes. In this model, T_{AB} is a scalar and $T_{AB} = (s+1)/(s+f+2)$, $f_{ctp}(x, y) = xy$, and $f_{mtp}(\{x_i\}_i)$ is just the average of $\{x_i\}_i$.

- The *probability model* proposed in [19]. The trust value T_{AB} is a scalar, $f_{ctp}(x, y) = xy + (1 - x)(1 - y)$, and $f_{mtp}(.)$ is based on a data fusion model.

- The *Beta function model* proposed in [26]. In this model, T_{AB} is a 2 by 1 vector. One element represents trust, and the other element represents confidence. The detailed equations can be found in [26] and [25].

In Figure 5.10, we first examine the region when false alarm probability is smaller than 0.05. The malicious node detection algorithms should always work in this region. The probability model performs better than the simple model. This is because the probability model can better handle incorrect recommendations from malicious nodes. The beta function model performs much better than the other two models. This is because the beta function model considers the possible estimation error when calculating the trust values. This observation agrees with the qualitative arguments in the current literature. More importantly, The proposed analysis tool provides a *quantitative comparison* among trust models. The quantitative comparison is not currently available in the literature, but it is important for the future research in this field.

When the false alarm probability is higher than 0.05, the detection probability starts to saturate. For a given round index, the detection probability cannot be higher than the probability that A can establish indirect trust in Y_2. In Figure 5.10, the probability model has the lowest detection probability when the false alarm probability is large. Compared with two other models, the probability model results in the least likelihood of establishing indirect trust. In both regions, the beta function model has the best performance.

5.6 Conclusion

This paper proposed the methods to analyze the process of trust establishment in distributed networks. The tools for performing the analysis were implemented, and validated by simulations. The proposed analysis methods were utilized to solve an important research problem: quantitative comparison among different trust models. In the

future, we will exploit more applications of the analysis tool, such as understanding the effects that application context has upon trust establishment, and guiding the design of better trust establishment methods.

List of References

[1] M. Langheinrich, "When trust does not compute - the role of trust in ubiquitous computing," in *Proceedings of the 5th International Conference on Ubiquitous Computing*, Seattle, Washington, October 2003.

[2] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pp. 164-173, May 1996.

[3] M. Blaze, J. Feigenbaum, and A. D. Keromytis, "KeyNote: Trust management for public-key infrastructures," *Lecture Notes in Computer Science*, vol. 1550, pp. 59–63, 1999.

[4] U. Maurer, "Modelling a public-key infrastructure," in *Proceedings 1996 European Symposium on Research in Computer Security(ESORICS' 96), volume 1146 of Lecture Notes in Computer Science*, pp. 325-350, 1996.

[5] M. K. Reiter and S. G. Stubblebine, "Toward acceptable metrics of authentication," in *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, 1997.

[6] A. Jøsang, "An algebra for assessing trust in certification chains," in *Proceedings of the Network and Distributed Systems Security (NDSS'99) Symposium*, 1999.

[7] W. Stallings, *Protect Your Privacy, A Guide for PGP Users*. Prentice Hall, 1995.

[8] R. Levien and A. Aiken, "Attack-resistant trust metrics for public key certification," in *Proceedings of the 7th USENIX Security Symposium*, pp. 229-242, January 1998.

[9] D. Clarke, J.-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R. L. Rivest, "Certificate chain discovery in SPKI/SDSI," *Journal of Computer Security*, vol. 9, no. 4, pp. 285–322, 2001.

[10] A. Abdul-Rahman and S. Hailes, "A distributed trust model," in *Proceedings of 1997 New Security Paradigms Workshop, ACM Press*, pp. 48-60, 1998.

[11] P. Resnick and R. Zeckhauser, "Trust among strangers in internet transactions: Empirical analysis of ebay's reputation system," in *Proceedings of NBER workshop on empirical studies of electronic commerce*, 2000.

[12] D. Manchala, "Trust metrics, models and protocols for electronic commerce transactions," in *Proceedings of the 18th IEEE International Conference on Distributed Computing Systems*, pp. 312 - 321, May 1998.

[13] S. Kamvar, M. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *Proceedings of 12th International World Wide Web Conferences*, May 2003.

[14] R. Guha, R. Kumar, P. Raghavan, and A. Propagation, "Propagation of trust and distrust," in *Proceedings of International World Wide Web Conference*, 2004.

[15] S. Buchegger and J. L. Boudec, "Performance analysis of the CONFIDANT protocol," in *Proceedings of ACM Mobihoc*, 2002.

[16] P. Michiardi and R. Molva, "Core: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks," in *Proceedings of the IFIP TC6/TC11 Sixth Joint Working Conference on Communications and Multimedia Security*, 2002, pp. 107–121.

[17] S. Buchegger and J. L. Boudec, "Coping with false accusations in misbehavior reputation systems for mobile ad-hoc networks," EPFL Technical Report IC/2003/31, EPFL-DI-ICA, 2003.

[18] G. Theodorakopoulos and J. S. Baras, "Trust evaluation in ad-hoc networks," in *Proceedings of the ACM Workshop on Wireless Security (WiSE'04)*, Oct. 2004.

[19] Y. Sun, W. Yu, Z. Han, and K. J. R. Liu, "Information theoretic framework of trust modeling and evaluation for ad hoc networks," *IEEE JSAC Special Issue on Security in Wireless Ad Hoc Networks*, vol. 24, pp. 305– 317, February 2006.

[20] L. Eschenauer, V. Gligor, and J. S. Baras, "On trust establishment in mobile ad-hoc networks," in *Security Protocols, Proc. of 10th International Workshop, Springer Lecture Notes in Computer Science (LNCS)*, April 2002.

[21] S. Ganeriwal and M. Srivastava, "Reputation-based framework for high integrity sensor networks," in *Proceedings of ACM Security for Ad-hoc and Sensor Networks (SASN)*, Washington, D.C., USA, Oct. 2004.

[22] N. Shankar and W. Arbaugh, "On trust for ubiquitous computing," in *Proceedings of Workshop on Security in Ubiquitous Computing, UBICOMP'02*, 2002.

[23] T. Kindberg, A. Sellen, and E. Geelhoed, "Security and trust in mobile interactions: A study of users' perceptions and reasoning," in *Proceedings of UBICOMP'04*, 2004, pp. 196–213.

[24] T. Jiang and J. S. Baras, "Trust evaluation in anarchy: A case study on autonomous networks," in *Proceedings of Infocom'06*, 2006.

[25] Y. Sun, Z. Han, W. Yu, and K. J. R. Liu, "A trust evaluation framework in distributed networks: Vulnerability analysis and defense against attacks," in *Proceeding of the 27th Conference on Computer Communications (INFOCOM'06)*, Barcelona, Spain, April 2006.

[26] A. Jsang and R. Ismail, "The beta reputation system," in *Proceedings of the 15th Bled Electronic Commerce Conference*, June 2002.