JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 6, No. 3, September 2012, pp. 179-192

Protecting the iTrust Information Retrieval Network against Malicious Attacks

Yung-Ting Chuang*, P. Michael Melliar-Smith, Louise E. Moser, and Isaí Michel Lombera
Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, USA
ytchuang@ece.ucsb.edu, pmms@ece.ucsb.edu, moser@ece.ucsb.edu, imichel@ece.ucsb.edu

Abstract

This paper presents novel statistical algorithms for protecting the iTrust information retrieval network against malicious attacks. In iTrust, metadata describing documents, and requests containing keywords, are randomly distributed to multiple participating nodes. The nodes that receive the requests try to match the keywords in the requests with the metadata they hold. If a node finds a match, the matching node returns the URL of the associated information to the requesting node. The requesting node then uses the URL to retrieve the information from the source node. The novel detection algorithm determines empirically the probabilities of the specific number of matches based on the number of responses that the requesting node receives. It also calculates the analytical probabilities of the specific numbers of matches. It compares the observed and the analytical probabilities to estimate the proportion of subverted or non-operational nodes in the iTrust network using a window-based method and the chi-squared statistic. If the detection algorithm determines that some of the nodes in the iTrust network are subverted or non-operational, then the novel defensive adaptation algorithm increases the number of nodes to which the requests are distributed to maintain the same probability of a match when some of the nodes are subverted or non-operational as compared to when all of the nodes are operational. Experimental results substantiate the effectiveness of the detection and defensive adaptation algorithms for protecting the iTrust information retrieval network against malicious attacks.

Category: Ubiquitous computing

Keywords: Decentralized; Distributed; Search; Retrieval; Malicious attack; iTrust

I. INTRODUCTION

Modern society depends on the retrieval of information over the Internet. For reasons of efficiency, Internet search and retrieval exploits centralized search engines, and assumes that such search engines are unbiased. Unfortunately, it is easy to enable an Internet search engine to conceal or censor information. History and the present have demonstrated that we cannot depend on centralized search engines to remain unbiased. The moment at which society is most dependent on the free flow of information across the Internet might also be the moment at which information is most likely to be censored or suppressed.

The iTrust system [1, 2] is a completely decentralized and distributed information retrieval system, which is designed to defend against the censorship of information on the Internet. In iTrust, metadata describing documents, and requests (queries) containing keywords, are randomly distributed to multiple participating nodes. The nodes that receive the requests attempt to match the key-

Open Access http://dx.doi.org/10.5626/JCSE.2012.6.3.179 http://jcse.kiise.org

This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

Received 31 May 2012, **Accepted** 16 August 2012
*Corresponding Author

†An earlier version of this paper has been published at ICOIN 2012.

Copyright © 2012. The Korean Institute of Information Scientists and Engineers pISSN: 1976-4677 eISSN: 2093-8020

words in the requests with the metadata they hold. If a node has a match, the matching node returns the URL of the associated information to the requesting node. The requesting node then uses the URL to retrieve the information from the source node.

Crucial performance parameters of iTrust are the numbers of nodes to which the metadata and the requests are distributed in order to achieve a high probability of a match. In a network with n participating nodes, distribution of the metadata and the requests to $\lceil 2\sqrt{n} \rceil$ nodes results in a probability of a match that exceeds $1 - e^{-4} \approx 0.9817$ [3]. The communication cost of iTrust is greater than that of a centralized search engine but, if users are concerned about censorship or suppression of information, they should be willing to incur that extra cost.

The decentralized and distributed nature of iTrust makes it very robust against malicious attacks that aim to prevent information retrieval. One specific kind of malicious attack is to insert into the network a large number of nodes that behave normally except that they do not match requests and metadata on certain topics. The appropriate response to such an attack is to increase the number of nodes to which the requests are distributed, thus restoring the probability of a match to the desired level. This paper presents novel statistical algorithms for detecting such malicious attacks, using a window-based method and the chi-squared statistic, and for adaptively defending against them by increasing the number of nodes to which the requests are distributed.

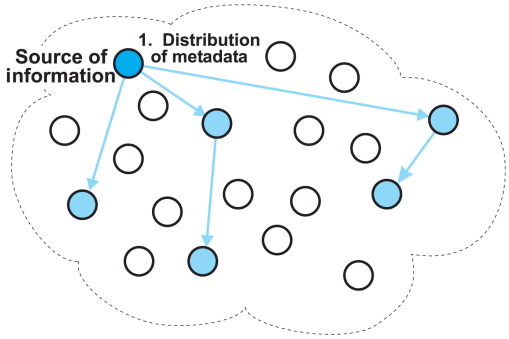


Fig. 1. A source node distributes metadata, describing its information, to randomly selected nodes in the network.

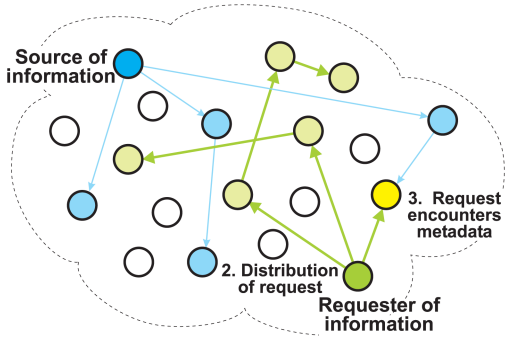


Fig. 2. A requesting node distributes its request to randomly selected nodes in the network. One of the nodes has both the metadata and the request and, thus, an encounter occurs.

II. DESIGN OF ITRUST

The iTrust information retrieval system is completely distributed, and involves no centralized mechanisms and no centralized control. We refer to the nodes that participate in an iTrust network as the *participating nodes* or the *membership*. Multiple iTrust networks may exist at any point in time, and a node may participate in multiple iTrust networks at the same point in time.

Some nodes, the *source nodes*, produce information, and make that information available to other participating nodes. The source nodes produce metadata that describes their information, and distribute that metadata to a subset of the participating nodes chosen at random, as shown in Fig. 1. The metadata are distinct from the information they describe, and include a list of keywords and the URL of the source of the information.

Other nodes, the *requesting nodes*, request and retrieve information. Such nodes generate requests (queries), which refer to metadata for the desired information, and distribute their requests to a subset of the participating nodes chosen at random, as shown in Fig. 2.

The participating nodes compare the metadata in the requests they receive with the metadata they hold. If such a node finds a match, which we call an *encounter*, the matching node returns the URL of the associated information to the requesting node. The requesting node then uses the URL to retrieve the information from the source node, as shown in Fig. 3.

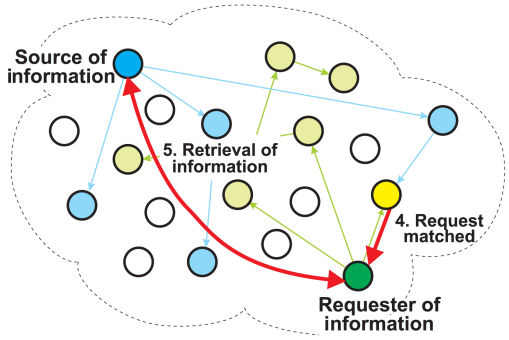


Fig. 3. A node matches the metadata and the request and reports the match to the requester, which then retrieves the information from the source node.

III. IMPLEMENTATION OF iTRUST

The iTrust system runs on laptop, desktop or server

nodes on the Internet. There might be hundreds or thousands of iTrust nodes in a typical iTrust network. The primary goal of each iTrust node is to match a query it receives with metadata it holds and to respond with a URL for that resource, if it has a match.

The iTrust implementation on a node consists of the Web server foundation, the application infrastructure, and the public interface. These three components interact with each other in order to distribute metadata and requests and to retrieve resources from the nodes.

The implementation utilizes the Apache Web server along with several hypertext preprocessor (PHP) modules and library extensions. In particular, it uses the standard session and logging modules, and the compiled-in cURL, SQLite, and HTTP PHP extension community library (PECL) modules.

The iTrust system operates over HTTP and, thus, the transmission control protocol/Internet protocol (TCP/IP). As such, it establishes a direct connection between any two nodes that need to communicate; iTrust uses neither flooding nor random walks.

The iTrust system replicates both the metadata and the requests (queries). A source node randomly selects nodes for the distribution of metadata, and a requesting node randomly selects nodes for the distribution of requests. At each node, iTrust maintains a local index of metadata and URLs for the corresponding resources.

When a node receives a request, it compares the metadata (keywords) in the request against a SQLite database consisting of metadata and URLs. If the keywords in the request match locally stored metadata, the node sends a response containing the URL corresponding to the metadata to the requester. When the requester receives the response, it uses the URL in the response to retrieve the information from the source node.

IV. FOUNDATIONS

A. Assumptions and Notation

We assume that all of the participating nodes in an iTrust network have the same membership set. Moreover, we assume that the underlying Internet delivers messages reliably and that the nodes have sufficient memory to store the source files and the metadata, which the nodes generate and receive.

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

- n: The number of participating nodes (i.e., the size of the membership set)
- x: The proportion of the n participating nodes that are operational (i.e., $1 - x$ is the proportion of non-operational nodes)
- m: The number of participating nodes to which the metadata are distributed

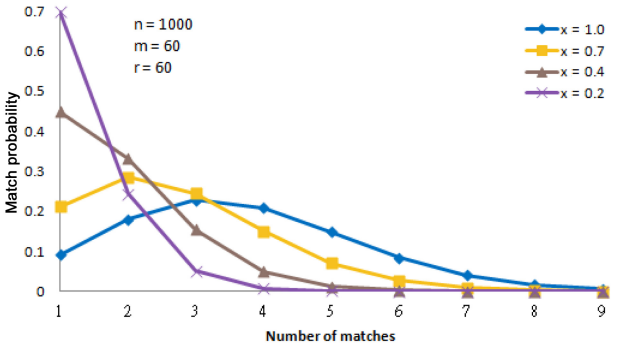


Fig. 4. The analytical probabilities $P(k)$ of specific numbers of matches for $n = 1000$, $m = 60$, $r = 60$, and $x = 1.0, 0.7, 0.4, 0.2$.

- r: The number of participating nodes to which the requests are distributed
- k: The number of participating nodes that report matches to a requesting node.

B. Probabilistic Analysis

Our probabilistic analysis for iTrust is based on the hypergeometric distribution [4], which describes the number of successes in a sequence of random draws from a finite population without replacement. Thus, it differs from the binomial distribution, which describes the number of successes for random draws with replacement. In iTrust, the probability of k matches follows a hypergeometric distribution with parameters n, x, m and r, and is given by:

$$P(k) = \frac{\binom{mx}{k}\binom{n-mx}{r-k}}{\binom{n}{r}} \quad (1)$$

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

Expanding the binomial coefficients in Equation (1), we obtain the following formula for the probability of k matches:

$$P(k) = \frac{\left(\frac{mx}{k}\frac{mx-1}{k-1}\cdots\frac{mx-k+1}{1}\right)\left(\frac{n-mx}{r-k}\frac{n-mx-1}{r-k-1}\cdots\frac{n-mx-r+k+1}{1}\right)}{\left(\frac{n}{r}\frac{n-1}{r-1}\cdots\frac{n-r+1}{1}\right)} \quad (2)$$

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

For an iTrust network with $n = 1000$ participating nodes where the metadata are distributed to $m = 60$ nodes and the requests are distributed to $r = 60$ nodes, Fig. 4 shows the probabilities for the specific numbers of matches, obtained from Equation (2), for $x = 1.0, 0.7, 0.4, 0.2$ as the proportion of operational nodes.

The pseudocode for calculating the analytical probabilities from Equation (2) is given in Algorithm 1.

Our algorithm for detecting malicious attacks uses the analytical probabilities given by Algorithm 1 to determine the proportion of nodes in the iTrust network that

are operational and, thus, the proportion of nodes that are non-operational or subverted.

Algorithm 1 Calculating the analytical probabilities of specific numbers of matches

```
1: procedure ANAL(n, m, r, x, K)
2:     for k = 1 .. K do
3:         P(k) ← ((mx/k) ··· (mx−k+1)/1)((n−mx/r−k) ··· (n−mx−r+k+1)/1) / ((n/r)(n−1/r−1) ··· (n−r+1)/1)
4:     end for
5: return P
6: end procedure
```

From Equation (2), we obtain the probability $P(k \geq 1)$ of one or more matches as follows:

$$P(k \geq 1) = 1 - P(0) = 1 - \frac{\left(\frac{n-mx}{r}\,\frac{n-mx-1}{r-1}\cdots\frac{n-mx-r+1}{1}\right)}{\left(\frac{n}{r}\,\frac{n-1}{r-1}\cdots\frac{n-r+1}{1}\right)} \quad (3)$$

for $mx + r \leq n$.

For an iTrust network with $n = 1000$ nodes and $x = 1.0$, 0.7, 0.4, 0.2 as the proportion of operational nodes, Fig. 5 shows the probabilities of one or more matches, obtained from Equation (3), as the number of nodes to which the metadata and the requests are distributed increases.

Our adaptive algorithm for defending against malicious attacks uses Equation (2) to determine how much to increase the number of nodes to which the requests are distributed to compensate for a decrease in the proportion of operational or non-subverted nodes.

C. Normalization

In our experimental studies and in real-world deployments, we cannot use requests that return $k = 0$ responses, because zero responses can arise not only when the metadata and the requests are distributed to disjoint subsets of nodes, but also when there are no metadata that match a request. Thus, we exclude the requests that return $k = 0$ responses. Moreover, for larger values of k, the probabilities are negligible, so we also exclude such requests. That is, we determine a value K and exclude requests that return k responses for $K < k \leq r$. We then normalize the remaining probabilities, i.e., $P(k)$, $1 \leq k \leq K$.

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

$$Q(k) = \frac{P(k)}{\sum_{k=1}^{K} P(k)} \quad (4)$$

Pseudocode for the normalization method is given in Algorithm 2.

We apply the normalization method given in Algorithm 2 to both the observed probabilities and the analytical probabilities before applying the chi-squared test in the detection algorithm.

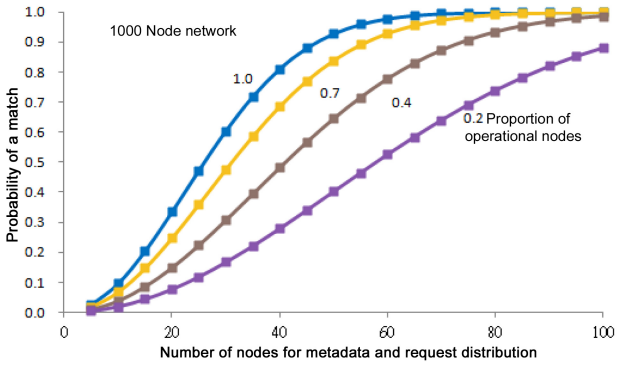


Fig. 5. The probabilities $P(k \geq 1)$ of one or more matches as the number of nodes to which the metadata and the requests are distributed increases, for $x = 1.0$, 0.7, 0.4, 0.2 as the proportion of operational nodes.

Algorithm 2 Normalizing the probabilities

```
1: procedure NORM(P, K)
2:     S ← 0
3:     for k = 1 .. K do
4:         S ← S + P(k)
5:         k ← k + 1
6:     end for
7:
8:     for k = 1 .. K do
9:         Q(k) ← P(k)/S
10:        k ← k + 1
11:    end for
12: return Q
13: end procedure
```

D. Chi-Squared Statistic

Our detection algorithm uses Pearson's chi-squared goodness-of-fit test [5] to compare the observed probabilities against the analytical probabilities for different values of x, in order to determine which of the analytical curves best matches the observed data for the probability of a match. The chi-squared statistic is given by:

$$\chi^2 = \sum_{k=1}^{K} \frac{(o_k - e_k)^2}{e_k} \quad (5)$$

where:

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

Pseudocode for the chi-squared method is given in Algorithm 3.

We apply the chi-squared method given in Algorithm 3 in the detection algorithm for estimating x', the proportion of operational nodes in the iTrust network.

Algorithm 3 Computing the chi-squared statistic

```
1: procedure CHISQUARED(O, E_x, K)
2:     χ² ← 0
3:     for k = 1 .. K do
4:         χ² ← χ² + (O(k) − E_x(k))² / E_x(k)
5:         k ← k + 1
6:     end for
7: return χ²
8: end procedure
```

V. DETECTION OF MALICIOUS ATTACKS

Our algorithm for detecting malicious attacks first computes the analytical probabilities for the specific numbers of matches for given values of *n*, *m*, *r* and for different values of *x*. Then, the algorithm collects empirical data on the number of responses that a requesting node has received for a number of requests. Finally, the algorithm compares the observed probabilities against the analytical probabilities for specific numbers of matches in order to estimate the proportion x' of operational nodes in the iTrust network.

For example, consider an iTrust network with $n = 1000$ participating nodes where the metadata are distributed to $m = 60$ nodes and the requests are distributed to $r = 60$ nodes. Fig. 4 shows the probabilities for the specific numbers of matches for $x = 1.0, 0.7, 0.4, 0.2$ as the proportion of operational nodes. If the detection algorithm observes a curve like the $x = 0.2$ shown in Fig. 4, it detects that there is a large number of subverted or non-operational nodes and that the network is under attack.

In Fig. 4, it is easy to distinguish the behavior of the iTrust network with many operational nodes from its behavior with only a few operational nodes. For example, the curves for $x = 1.0$ and 0.2 are quite different. Although the probability of a single match for $x = 0.2$ is greater than the corresponding probability for $x = 1.0$, the probability of k matches, $k \geq 2$, for $x = 1.0$ is greater than the corresponding probability for $x = 0.2$.

The detection algorithm collects statistical data on the number of responses that a requesting node receives for recent requests. Then, it calculates the observed probabilities from the statistical data. The algorithm excludes zero matches (because it cannot distinguish the case in which no node holds both the metadata and the request, and the case in which no such metadata exists). The algorithm then normalizes both the observed probabilities and the analytical probabilities.

Next, it applies the chi-squared statistic in Algorithm 2 to the normalized observed probabilities $O(k)$ and the normalized analytical probabilities $E_x(k)$, $k = 1, 2, ..., m$, for $x = 1.0, 0.7, 0.4, 0.2$. It then chooses the particular value of x' ($x' = 1.0, 0.7, 0.4, 0.2$) for which the chi-squared value is the smallest. Finally, the algorithm chooses that particular value of x' as its best estimate for the observed proportion of operational nodes in the iTrust network.

The pseudocode for the detection algorithm is shown in Algorithm 4. Note that the smallest value of the chi-squared statistic determines the best estimate of x'.

Algorithm 4 Estimating x' using the chi-squared statistic with the normalized observed probabilities and the normalized analytical probabilities

```
1: procedure GUESSX(O, K, E_1.0, E_0.7, E_0.4, E_0.2)
2:     χ²_1.0 ← CHISQUARED(O, E_1.0, K)
3:     χ²_0.7 ← CHISQUARED(O, E_0.7, K)
4:     χ²_0.4 ← CHISQUARED(O, E_0.4, K)
5:     χ²_0.2 ← CHISQUARED(O, E_0.2, K)
6:     x' ← min(χ²_1.0, χ²_0.7, χ²_0.4, χ²_0.2)
7: return x'
8: end procedure
```

For example, suppose that the detection algorithm collects the number of matches for each of 3,000 requests, where the proportion of operational nodes is $x = 1.0$. For these 3,000 requests, the algorithm then counts the number of occurrences for each specific number of matches. Next, the algorithm divides those numbers of matches by 3,000 to obtain the unnormalized observed probabilities. The algorithm then computes the normalized observed probabilities, using Algorithm 2. The detection algorithm computes the analytical probabilities using Algorithm 1. The algorithm then computes the normalized analytical probabilities using Algorithm 2.

Next, the detection algorithm applies the chi-squared statistic to calculate the chi-squared value using the normalized observed probabilities and the normalized analytical probabilities for $x = 1.0$. Similarly, it calculates the chi-squared values for $x = 0.7$, 0.4, and 0.2. Finally, it estimates the value of x' based on the smallest chi-squared value obtained.

As illustrated in Fig. 6, based on the 3,000 requests collected, the detection algorithm compares the observed

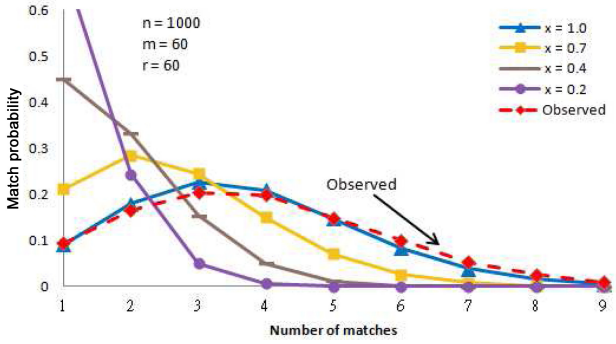


Fig. 6. The analytical probabilities of specific numbers of matches for $n = 1000$, $m = 60$, $r = 60$ and $x = 1.0, 0.7, 0.4, 0.2$ vs. the observed probabilities.

probabilities against the analytical probabilities for x = 1.0, 0.7, 0.4, and 0.2 for the specific number of matches, and estimates that the proportion x' of operational nodes in the current iTrust network is $x' = 1.0$. From Fig. 6, we see that the observed curve (shown in red) is close to the analytical curve for x = 1.0 (shown in blue). Thus, the chi-squared test achieves good discrimination between the curves by considering the probabilities for the various numbers of matches.

VI. DEFENSE AGAINST MALICIOUS ATTACKS

If the detection algorithm determines empirically that the proportion of operational nodes is $x' < 1.0$, then the defensive adaptation algorithm increases the number of nodes to which the requests are distributed, to restore the probability of a match to the desired level. That is, the defensive adaptation algorithm increases the number of nodes to which the requests are distributed to r', so that the probability of a match for x' and r' is the same as the probability of a match for x and r. The pseudocode for the defensive adaptation algorithm is shown in Algorithm 5.

Algorithm 5 Iteratively finding the value of r' to maintain a high probability of a match even if some of the nodes are subverted

```
1:  procedure FINDR(n, x', m, r, prevX)
2:      y_0 ← 1 − ((n−m)/(n)) ((n−m−1)/(n−1)) ... ((n−m−r+1)/(n−r+1))
3:      r' ← r
4:      if prevX > x' then
5:          repeat
6:              r' ← r' + 1
7:              y' ← 1 − ((n−m'x')/(n)) ... ((n−m'x'−r'+1)/(n−r'+1))
8:          until y' > y_0
9:      else
10:         repeat
11:             r' ← r' − 1
12:             y' ← 1 − ((n−m'x')/(n)) ... ((n−m'x'−r'+1)/(n−r'+1))
13:         until y' < y_0
14:     end if
15: return r'
16: end procedure
```

For example, if the detection algorithm determines empirically that $x' = 0.7$, the defensive adaptation algorithm determines that $r' = 86$. To better understand how it makes that determination, consider the curve for n = 1000, x = 1.0, and $m = r = 60$ shown in Fig. 7. The defensive adaptation algorithm first uses Equation (3) to compute $y_0 = P(k \geq 1)$ on the curve for x = 1.0 corresponding to $m = r = 60$. It determines that $y_0 = P(k \geq 1) = 0.978298$. From the calculated value of y_0, the algorithm then finds the value of r' corresponding to this value of y_0 on the curve for $x' = 0.7$ using Algorithm 5. It iteratively, solves the equation $y_0 = P'(k \geq 1)$ with n = 1000, $x' = 0.7$, m = 60 and r' to determine that $r' = 86$.

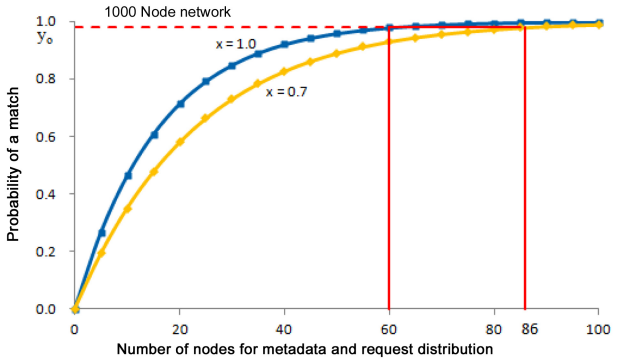


Fig. 7. Increase in the number r of nodes to which the requests are distributed when x = 0.7 to achieve the same probability of a match as when x = 1.0 and $m = r$ = 60 for n = 1000 nodes.

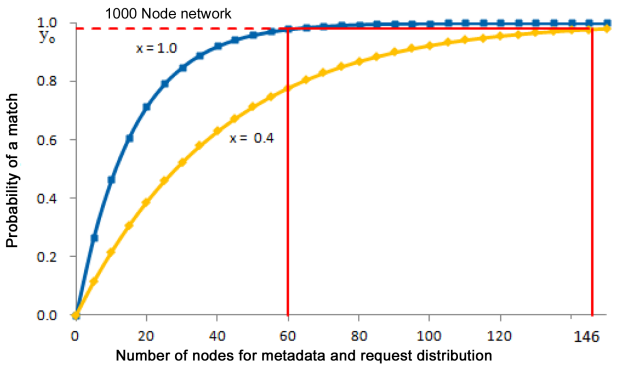


Fig. 8. Increase in the number r of nodes to which the requests are distributed when x = 0.4 to achieve the same probability of a match as when x = 1.0 and $m = r$ = 60 for n = 1000 nodes.

Similarly, in Fig. 8, if the detection algorithm determines empirically that $x' = 0.4$, the defensive adaptation algorithm determines that $r' = 146$. To accomplish this, again it first uses Equation (3) to compute $y_0 = P(k \geq 1)$ on the curve for x = 1.0 corresponding to $m = r = 60$ and determines that $y_0 = P(k \geq 1) = 0.978298$. From the calculated value of y_0, the defensive adaptation algorithm then iteratively finds the value of r' corresponding to this value of y_0 on the $x' = 0.4$ curve using Algorithm 5. That is, it iteratively solves the equation $y_0 = P'(k \geq 1)$ with n = 1000, $x' = 0.4$, m = 60 and r' to determine that $r' = 146$.

VII. EXPERIMENTAL METHOD

In order to evaluate the ability of the detection algorithm to estimate the proportion x' of operational nodes, we performed experiments using an emulation of iTrust. In the emulation, we have multiple virtual hosts installed on a single Apache Web server, where each virtual host represents a node in the iTrust network. Each node has a

separate SQLite database file that resides on the Web server, where it stores queries and resource information. The emulation allows us to evaluate the accuracy of the detection algorithm, whereas a real-world deployment of iTrust would not allow us to do so, because in the simulation we can control the value of x. We determine the number of times the detection algorithm's estimate of x' is correct by comparing the value of x' with the actual value of x.

We use libCURL (which is a free client-side URL transfer library for transferring data using various protocols) to collect the number of responses (matches). We randomly select nodes without repetition from the membership set for distribution of the metadata and the requests.

Before we start the emulation program, we initialize the values of n, m, r, and x. The program clears the node's resources and databases, and then adds the nodes to the membership. For each source node, iTrust creates the metadata for the document that the node wishes to share, and distributes the metadata to m randomly selected nodes in the membership set. Then, iTrust distributes requests to r randomly selected nodes in the membership set. After five seconds, the emulation program records the number of nodes that found a match and returned a response.

A. Parameters of the Detection Algorithm

The primary parameters that we investigate for the detection algorithm are defined below:

- K: An upper bound on the number k of responses to requests, i.e., an upper bound on the number of buckets used for the responses. Note that, from Equation (2), $K \leq \min\{mx, r\}$.
- t: The number of chi-squared evaluations that indicate the same change in the value of x' before action is taken to change the value of r'.
- s: The number of requests for which responses are accumulated before the chi-squared statistic is calculated
- w: The number of recent requests that are used in the calculation of chi-squared. For example, for $w = 10$, we use the current request i, the previous request $i - 1$, ..., back to the 9th request $i - 9$ in the calculation of chi-squared.

B. Ranges for the Experimental Evaluation

Initially, we collected 10,000 requests from the emulation program with $x = 1.0$. The emulation program uses a counter to keep track of the number of correct estimates of x' in these 10,000 requests. The program ends when it reaches 10,000 requests, and then reports the number of correct estimates. We repeat these steps 100 times and average the results at the end of the 100th iteration. The same process is performed again, but now with $x = 0.7$ and then with $x = 0.4$.

VIII. PERFORMANCE METRICS

A. Accuracy

By definition, accuracy is the number of correct estimates of x' divided by the total number of estimates of x' within an interval of requests.

To determine the accuracy of the detection algorithm, for each value of $x = 1.0, 0.7, 0.4, 0.2$, we consider a window of w requests. For each request in the window, the requester sends r requests to randomly selected nodes in the iTrust network. If a node holds metadata that match the keywords in a request it receives, it sends a response to the requester.

For each request in the window, the algorithm determines the number of responses the requester received for that request. To determine the observed probability $P'(k)$ for the window, $1 \leq k \leq r$, the algorithm determines the number $N(k)$ of requests in the window for which it received k responses and then divides that number by w, i.e.,

$$P'(k) = \frac{N(k)}{w} \tag{6}$$

and then normalizes $P'(k)$ by dividing by $\sum_{k=1}^{K} P'(k)$.

From the analytical formula (2), the algorithm calculates $P(k)$ using the values of n, m, r and x and then normalizes $P(k)$, by dividing by $\sum_{k=1}^{K} P'(k)$.

The algorithm then computes the chi-squared statistic using Algorithm 3 with the normalized $P'(k)$ and the normalized $P(k)$ for $k = 1, 2, 3,..., K$ and chooses the particular value of x' ($x' = 1.0, 0.7, 0.4, 0.2$) for which the chi-squared value is the smallest.

In our experiments, we consider windows of sizes $w = 10, 20,..., 100$. For each value of w, we repeat the experiment $S = 10{,}000$ times, and determine the number C out of S times for which the algorithm's estimate of x' is correct. The accuracy $A(w)$ is computed as:

$$A(w) = \frac{C}{S} \tag{7}$$

We repeat the same process 100 times and average the results to obtain the final accuracy $A(w)$.

B. Response Time

The response time represents the number of responses before the detection algorithm determines a change in the estimated value of x' after a change in the actual value of x. Using the accuracy results, we determine the mean response time of the detection algorithm. The mean response time $R(w)$, represented as the number w of requests in a window, is given by:

$$R(w) = \sum_{i=1}^{T} iwA(w)(1-A(w))^{i-1} \tag{8}$$

In Equation (8), the probability that the value of x is estimated correctly after the first window is $A(w)$, contributing $1wA(w)$ requests to the sum. The probability that the value of x is estimated incorrectly after the first window, and correctly after the second window is $A(w)(1 - A(w))$, contributing $2wA(w)(1 - A(w))$ requests to the sum, etc. Here, T is a large integer such that $TwA(w)(1 - A(w))^{T-1}$ makes a negligible contribution to the sum.

Simplifying Equation (8), we obtain:

$$\begin{aligned} R(w) &= wA(w)\sum_{i=1}^{T} i(1-A(w))^{i-1} \\ &= wA(w)\frac{1-(T+1)(1-A(w))^{T}+T(1-A(w))^{T+1}}{A(w)^2} \\ &= w\frac{1-(1+TA(w))(1-A(w))^{T}}{A(w)} \\ &\approx \frac{w}{A(w)} \end{aligned} \tag{9}$$

The response time, in seconds, depends not only on the number of requests in the window but also on the time interval between requests, i.e., the time between issuing a request and obtaining the responses for that request plus the time to create and issue the request (think time). If it takes s seconds between each request in the window, then the response time between requests is $sR(w)$ seconds. More frequent requests yield a lower response time. However, frequent requests might overload the iTrust network, if the number of participating nodes is large.

IX. WINDOW-BASED ALGORITHM

In our window-based algorithm, the requester issues requests, collects responses, and determines the observed probabilities based on the responses to the w most recent requests, stored in the queue W. The pseudocode for the window-based algorithm is given in Algorithm 6.

A. Determining an Appropriate Value of K

For the window-based algorithm and the chi-squared statistic, first we consider the number K of buckets used by the algorithm. If we use only a few buckets, say $K < 5$, the program guesses the incorrect x' quite often and, therefore, the accuracy is low. On the other hand, when we use a lot of buckets, say $K > 9$, the match probabilities

Algorithm 6 Window-based algorithm

```
1: procedure WINDOW(numResp, W, w)
2:     if size(W) > w then
3:         remove_front(W)
4:     end if
5:     insert_rear(numResp, W)
6: return W
7: end procedure
```

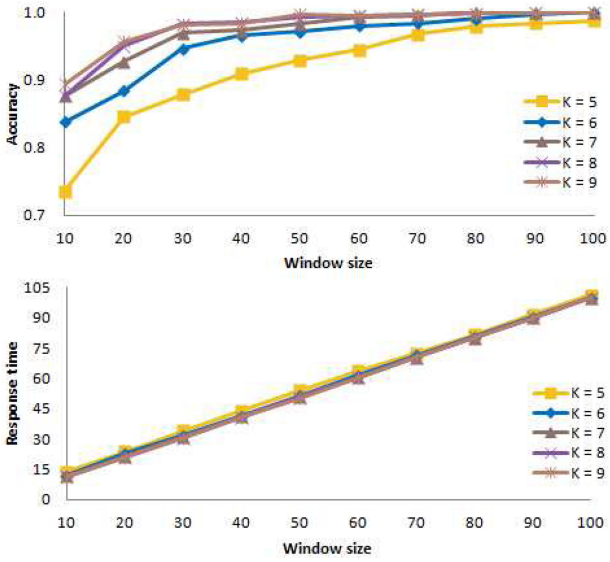


Fig. 9. Mean accuracy and mean response time for $K = 5, 6, 7, 8, 9$ with $n = 1000$, $x = 1.0$, $m = 60$, $r = 60$, $s = 1$, and $t = 1$.

$P'(k)$, $9 < k \le K$, are negligible. Thus, in our experiments, we investigated five cases $K = 5, 6, 7, 8, 9$ for $n = 1000$, $x = 1.0$, $m = 60$, $r = 60$, $s = 1$, and $t = 1$.

Fig. 9 presents the mean accuracy at the top of the figure, and the mean response time at the bottom of the figure, as a function of the window size. In the figure, we see that the accuracy for $K = 5$ is less than that for the other values of K. Moreover, we observe that the accuracy increases when the value of K is increased to $K = 6$. However, we don't see much improvement in the accuracy for $K = 8$ or 9; in both cases, the accuracy is nearly the same for all window sizes. We also observe that the response times are almost the same for all five values of K. In addition to $x = 1.0$, we performed the experiments for $x = 0.7$ and 0.4; for those values of x, the accuracy values and the response times were almost the same for corresponding window sizes.

Based on these results, for $n = 1000$, $m = 60$, $r = 60$, $s = 1$, and $t = 1$, we conclude that $K = 7$ buckets is enough to obtain the mean accuracy and mean response time.

B. Determining an Appropriate Value of t

Next, for the window-based algorithm, we consider the number t of successive estimates by the chi-squared test that indicate the same change in the value of x before the algorithm accepts that change and takes action to change the value of r'. A smaller value of t results in a "hair trigger" algorithm that responds more quickly to changes in the value of x, but that makes more mistakes. A larger value of t results in a more conservative algorithm that responds more slowly to changes in the value of x.

In these experiments, we consider three different values of t ($t = 1, 2, 3$). For $t = 1$, the algorithm detects a

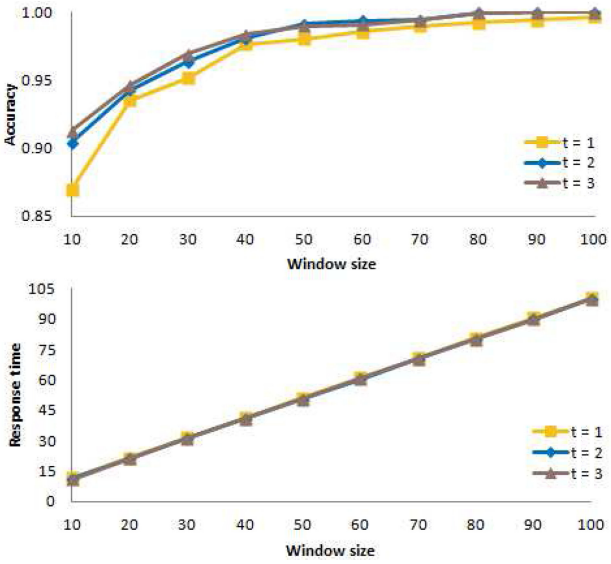


Fig. 10. Mean accuracy and mean response time for t = 1, 2, 3 with $n = 1000$, $x = 1.0$, $m = 60$, $r = 60$, $K = 7$, and $s = 1$.

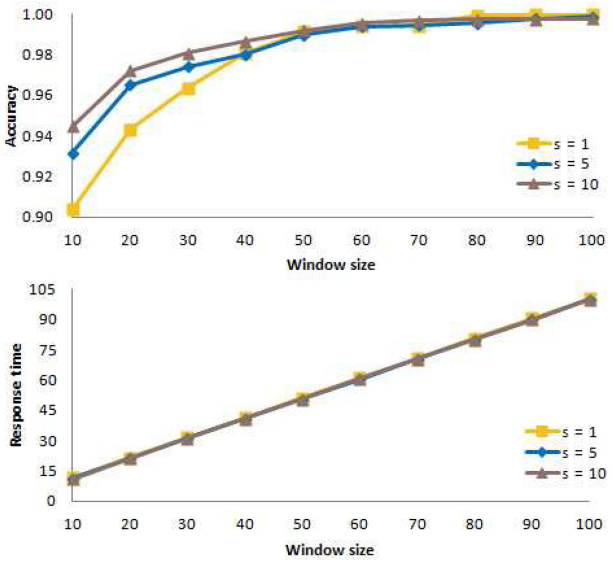


Fig. 11. Mean accuracy and mean response time for $s = 1, 5, 10$ with $n = 1000$, $x = 1.0$, $m = 60$, $r = 60$, $K = 7$, and $t = 2$.

change in the value of x and takes action immediately. For $t = 2$, the algorithm takes action when it detects the same change in the value of x in two consecutive estimates. For $t = 3$, the algorithm takes action when it detects the same change in the value of x in three consecutive estimates. Here, we set $n = 1000$, $x = 1.0$, $m = 60$, $r = 60$, $K = 7$, and $s = 1$.

Fig. 10 shows the mean accuracy at the top of the figure, and the mean response time at the bottom of the figure when action is taken immediately ($t = 1$) and when action is delayed ($t = 2$ or 3). The figure shows a useful increase in the mean accuracy as t is increased from $t = 1$ to 2, but a minimal increase in the mean accuracy as t is further increased to t = 3. The figure also shows that, for all three values of t, the mean response time increases linearly as the window size is increased and that the mean response times are almost the same. In addition to $x = 1.0$, we performed the experiments for $x = 0.7$ and 0.4 and obtained nearly the same results.

Thus, from our experiments, we conclude that $t = 2$ is an appropriate choice for our subsequent experiments.

C. Determining an Appropriate Value of s

Now, for the window-based algorithm, we consider the number s of requests for which responses are accumulated before the detection algorithm calculates chi-squared and estimates the value of x using the chi-squared test. If the algorithm reevaluates chi-squared immediately after obtaining a new sample, the computational and memory costs are greater. However, if the algorithm waits too long to reevaluate chi-squared, the response time for detecting a change in x is greater. Therefore, the goal of these experiments is to find a value of s that balances these two extremes.

In these experiments, we consider three different values of s ($s = 1, 5, 10$). For $s = 1$, the algorithm reevaluates chi-squared each time it obtains a new sample. For $s = 5$, it reevaluates chi-squared after the 5th new sample, rather than immediately. For $s = 10$, it reevaluates chi-squared after the 10th new sample, rather than immediately. Here, $n = 1000$, $x = 1.0$, $m = 60$, $r = 60$, $K = 7$, and $t = 2$.

Fig. 11 shows the mean accuracy at the top of the figure and the mean response time at the bottom of the figure when action is taken immediately ($s = 1$) and when action is delayed ($s = 5$ or 10). As we can see from the figure, the improvements in the accuracy are more substantial when $s = 5$ than when $s = 1$. However, as we further increase s from $s = 5$ to 10, there is only a small increase in the accuracy. For the mean response time, all three values of s appear to produce the same results, and they all grow linearly as the window size is increased. In addition to $x = 1.0$, we performed the experiments for $x = 0.7$ and 0.4, and obtained similar results.

From these experiments, we conclude that $s = 5$ is an appropriate choice for our subsequent experiments.

D. Determining an Appropriate Value of w

Fig. 12 shows the mean accuracy at the top of the figure and the mean response time at the bottom of the figure, as a function of the window size w for $x = 1.0, 0.7, 0.4$. These experiments were performed for an iTrust network with $n = 1000$, $m = 60$, $r = 60$, $K = 7$, $t = 2$, and $s = 5$.

The figure illustrates that the accuracy values for $x = 1.0$ are greater than the corresponding values for $x = 0.7$ and 0.4, and that the accuracy values for $x = 0.7$ and 0.4 are very similar. Furthermore, for the window size $w >$

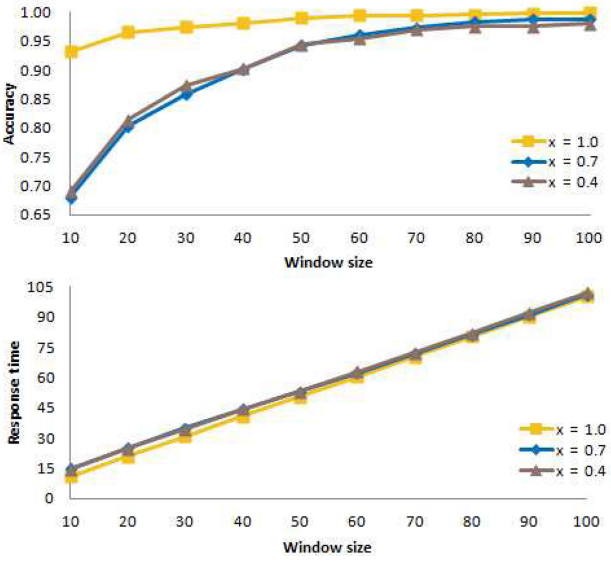


Fig. 12. Mean accuracy and mean response time for x = 0.4, 0.7, 1.0 with n = 1000, m = 60, r = 60, K = 7, t = 2, and s = 5, as a function of the window size w.

50, the accuracy values for all values of x are greater than 0.9. The figure shows that the mean response time increases linearly with the window size w for each value of x. Moreover, for different values of x and the same value of w, the mean response times are very close.

Ideally, the accuracy should be high, and the response time should be low. However, when we compare the top curve and the bottom curve for specific values of w, we see that there is a tradeoff between the accuracy and the response time. A small value of w results in a low value of the response time but also a low accuracy value. A large value of w results in a high value of the accuracy but also a high response time value. As we see from the figure, there is not much gain in the accuracy with $w > 50$ for all values of x. Therefore, we determine that a compromise value of w, say $w = 50$, is an appropriate value for our subsequent experiments.

X. COMBINING DETECTION AND DEFENSE

A. The Combined Algorithms

The pseudocode for the combined algorithms is shown in Algorithm 7.

In lines 2 and 3, the variables are initialized. Line 4 begins a loop that repeats indefinitely until there are no more requests. Within the loop, in line 5, the requester issues a request based on r' (where, initially, $r' = r$).

In line 6, the algorithm calculates the current observed P array using the window-based algorithm and, in line 7, it normalizes P to obtain the normalized observed probabilities O. In lines 8 to 15, the algorithm calculates the

Algorithm 7 Combined detection and defensive adaptation algorithms

```
1:  procedure DETECTDEFEND(n, m, r, K, w)
2:      r' ← r
3:      x' ← 1
4:      loop
5:          numResp ← NEXTREQUEST(r')
6:          P ← WINDOW(numResp, P, w)
7:          O ← NORM(P, K)
8:          P ← ANAL(n, m, r', 1.0, K)
9:          E_1.0 ← NORM(P, K)
10:         P ← ANAL(n, m, r', 0.7, K)
11:         E_0.7 ← NORM(P, K)
12:         P ← ANAL(n, m, r', 0.4, K)
13:         E_0.4 ← NORM(P, K)
14:         P ← ANAL(n, m, r', 0.2, K)
15:         E_0.2 ← NORM(P, K)
16:         prevX ← x'
17:         x' ← GUESSX(O, K, E_1.0, E_0.7, E_0.4, E_0.2)
18:         if prevX ≠ x' then
19:             r' ← FINDR(n, x', m, r', prevX)
20:         end if
21:     end loop
22: end procedure
```

analytical probabilities and normalizes them to obtain the normalized analytical probabilities $E_{1.0}$, $E_{0.7}$, $E_{0.4}$, $E_{0.2}$.

In line 17, the algorithm estimates the value of x' from the normalized observed probabilities O and the normalized analytical probabilities $E_{1.0}$, $E_{0.7}$, $E_{0.4}$, $E_{0.2}$, based on the smallest chi-squared value it obtained using those values.

In line 18, the algorithm compares x' against $prevX$. If they are not equal, then in line 19 it changes the value of r' based on the value of x' it determined in line 17, so that the probability of a match remains the same as it was with the prior value $prevX$ as determined by Algorithm 5.

Then, the algorithm goes back to line 5 to acquire another request based on the value of r' obtained in line 19. These steps are repeated indefinitely.

B. Experimental Results

Now, we compare the mean accuracy and the mean response time for the detection algorithm against the mean accuracy and the mean response time for the combined detection and defensive adaptation algorithms for x = 0.2, 0.4, 0.7, and 1.0.

For these experiments, we set n = 1000, m = 60, r = 60, K = 7, t = 2, s = 5, and w = 50, as determined in Section IX.

Fig. 13 shows the mean accuracy at the top of the figure and the mean response time at the bottom of the figure, for the detection algorithm and the combined detection and defensive adaptation algorithms as a function of x = 0.2, 0.4, 0.7, 1.0.

At the top of the figure, the light (yellow) curve shows the accuracy of the results obtained from the detection algorithm, and the dark (blue) curve shows the accuracy

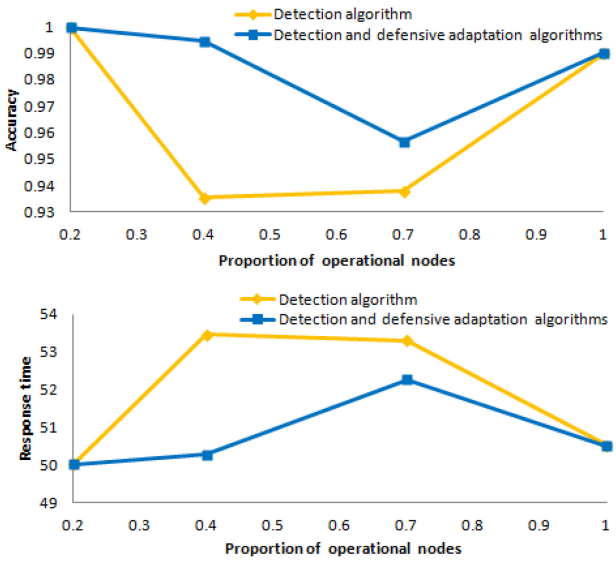


Fig. 13. Mean accuracy and mean response time of the detection algorithm vs. mean accuracy and mean response time of the combined detection and defensive adaptation algorithms. The points on the curves correspond to $x = 0.2, 0.4, 0.7, 1.0$.

of the results obtained from the combined detection and defensive adaptation algorithms. We see that the accuracy for $x = 0.7$ and 0.4 increases substantially when we use the combined algorithms. There is not much accuracy improvement for $x = 0.2$ and 1.0 because, in both cases, the accuracy is already high when we use only the detection algorithm. Moreover, we note that the chance of incorrectly estimating x' is quite low. When we use only the detection algorithm, the probability of estimating x' correctly is greater than 0.93 for all four values of x. When we use the combined detection and defensive adaptation algorithms, the probability of estimating x' correctly is even higher, and is greater than 0.955 for all four values of x.

Similarly, at the bottom of the figure, the light (yellow) curve shows the response time for the detection algorithm, and the dark (blue) curve shows the response time for the combined detection and defensive adaptation algorithms. We observe that, when we use the combined detection and defensive adaptation algorithms, the response time decreases substantially for both $x = 0.7$ and 0.4.

Overall, these experiments demonstrate that the detection algorithm is effective in discovering that the iTrust network contains non-operational or subverted nodes, and that false alarms rarely occur. Furthermore, the defensive adaptation algorithm is effective in defending against malicious attacks, because it appropriately adjusts the number of nodes to which the requests are distributed in order to maintain a high probability of a match. By selecting appropriate values of the parameters, and by using the combined algorithms, we obtain high accuracy and reasonable response time.

XI. RELATED WORK

Peer-to-peer networks have been characterized as structured or unstructured [6]. The structured approach requires the nodes to be organized in an overlay network, based on a distributed hash table, tree, ring, etc. The unstructured approach uses randomization and/or gossiping to distribute the metadata (or data) and the requests to nodes in the network. The iTrust system uses the unstructured approach.

Gnutella [7], one of the first unstructured peer-to-peer networks, uses flooding of requests. A node makes a copy of a file when it receives the file it requested. If the query rate is high, nodes quickly become overloaded and the system ceases to function satisfactorily. To address this problem, Ferreira et al. [8] use random walks, instead of flooding, to replicate objects and propagate queries. Gia [9] employs biased random walks to direct queries towards high-capacity nodes. iTrust does not use flooding or random walks, but rather distributes the metadata and the requests to nodes chosen uniformly at random.

Freenet [10] replicates an object at a node, even though the node did not request it. Nodes that successfully respond to requests receive more metadata and more requests, making it easy for a group of untrustworthy nodes to gather most of the searches into their group. Likewise, the adaptive probabilistic search (APS) method [11] uses feedback from previous searches to direct future searches, instead of distributing the requests uniformly at random, like iTrust does.

Cohen and Shenker [12] show that square root replication is theoretically optimal for minimizing the search traffic. They replicate objects based on the access frequencies (popularities) of the objects. Lv et al. [13] also use square root replication, and adaptively adjust the replication degree based on the query rate. iTrust likewise exploits square root replication but distributes the metadata uniformly at random, so that popular nodes are not more vulnerable to attack.

BubbleStorm [14] replicates both queries and data, hybridizes random walks and flooding, and considers churn, joins, leaves and crashes. Leng et al. [15] present mechanisms for maintaining the desired degree of replication in BubbleStorm, when each object has a maintainer node. In iTrust, we use different techniques to maintain the desired degree of replication of the metadata and the requests.

PlanetP [16] maintains a local index that contains metadata for documents published locally by a peer. Similarly, iTrust maintains a local index of metadata and corresponding URLs for the data. PlanetP also uses a global index that describes all peers and their metadata in a Bloom filter, which it replicates throughout the network using gossiping. GALANX [17] directs user queries to relevant nodes by consulting a local peer index. Like iTrust, GALANX is based on the Apache HTTP Web server.

Morselli et al. [18] describe an adaptive replication protocol with a feedback mechanism that adjusts the number of replicas according to the mean search length, so that an object is replicated sufficiently. Their adaptive replication protocol is quite different from the defensive adaptation algorithm of iTrust.

Sarshar et al. [19] use random walks and bond percolation in power law networks with high-degree nodes. The high-degree nodes in such networks are subject to overloading and are vulnerable to targeted malicious attacks. They note that "protocols for identifying or compensating for such attacks, or even recovering after such an attack has disrupted the network are yet to be designed." We present such protocols in this paper.

Jesi et al. [20] identify malicious nodes in a random overlay network based on gossiping and put such nodes on a blacklist. They focus, in particular, on hub attacks in which colluding malicious nodes partition the network by spreading false rumors. iTrust does not use gossiping but, rather, uses random distribution of the metadata and the requests and, thus, is less subject to hub attacks.

Condie et al. [21] present a protocol for finding adaptive peer-to-peer topologies that protect against malicious peers that upload corrupt, inauthentic or misnamed content. Peers improve the trustworthiness of the network by forming connections, based on local trust scores defined by past interactions. Effectively, their protocol disconnects malicious peers and moves them to the edge of the network.

Several network security researchers use the chi-squared statistic in their work. In particular, Goonatilake et al. [22] apply the chi-squared statistic to intrusion detection. We also use the chi-squared statistic in the detection algorithm of iTrust.

XII. CONCLUSIONS AND FUTURE WORK

We have presented novel statistical algorithms for protecting the iTrust information retrieval network against malicious attacks. The detection algorithm determines empirically the probabilities of specific numbers of matches based on the number of responses that a requesting node receives. Then, it compares the normalized observed probabilities with the normalized analytical probabilities for specific numbers of matches, to estimate the proportion of nodes that are operational, using a window-based method and the chi-squared statistic. If some of the nodes are subverted or non-operational, the defensive adaptation algorithm increases the number of nodes to which the requests are distributed to maintain the same probability of a match as when all of the nodes are operational. By selecting appropriate values of the parameters, and by using the combined detection and defensive adaptation algorithms, we obtain high accuracy and reasonable response time. In the future, we plan to investigate other kinds of malicious attacks on iTrust, as well as algorithms for protecting iTrust against those kinds of malicious attacks.

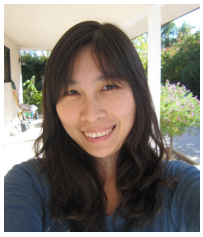
ACKNOWLEDGMENTS

This research was supported in part by a U.S. National Science Foundation grant (NSF CNS 10-16193). A shorter version of this paper appeared in ICOIN 2012 [23], where it won the Best Paper Award.

REFERENCES

1. Y. T. Chuang, I. Michel Lombera, L. E. Moser, and P. M. Melliar-Smith, "Trustworthy distributed search and retrieval over the Internet," *Proceedings of the International Conference on Internet Computing*, Las Vegas, NV, 2011, pp. 169-175.
2. I. Michel Lombera, Y. T. Chuang, P. M. Melliar-Smith, and L. E. Moser, "Trustworthy distribution and retrieval of information over HTTP and the Internet," *Proceedings of the 3rd International Conference on Evolving Internet*, Luxembourg City, Luxembourg, 2011, pp. 7-13.
3. P. M. Melliar-Smith, L. E. Moser, I. Michel Lombera, and Y. T. Chuang, "iTrust: trustworthy information publication, search and retrieval," *Proceedings of the 13th International Conference on Distributed Computing and Networking*, Hong Kong, China, 2012, pp. 351-366.
4. W. Feller, *An Introduction to Probability Theory and Its Applications*, New York: Wiley, 1968.
5. P. E. Greenwood and M. S. Nikulin, *A Guide to Chi-Squared Testing*, New York: Wiley, 1996.
6. J. Mischke and B. Stiller, "A methodology for the design of distributed search in P2P middleware," *IEEE Network*, vol. 18, no. 1, pp. 30-37, 2004.
7. Gnutella, http://en.wikipedia.org/wiki/Gnutella.
8. R. A. Ferreira, M. K. Ramanathan, A. Awan, A. Grama, and S. Jagannathan, "Search with probabilistic guarantees in unstructured peer-to-peer networks," *Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing*, Konstanz, Germany, 2005, pp. 165-172.
9. Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-like P2P systems scalable," *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Karlsruhe, Germany, 2003, pp. 407-418.
10. I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: a distributed anonymous information storage and retrieval system," *Proceedings of the International Workshop on Designing Privacy Enhancing Technologies*, Berkeley, CA, 2001, pp. 46-66.
11. D. Tsoumakos and N. Roussopoulos, "Adaptive probabilistic search for peer-to-peer networks," *Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, Linkoping, Sweden, 2003, pp. 102-109.
12. E. Cohen and S. Shenker, "Replication strategies in unstruc-

tured peer-to-peer networks," *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Pittsburgh, PA, 2002, pp. 177-190.
13. Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," *Proceedings of the 16th International Conference on Supercomputing*, New York, NY, 2002, pp. 84-95.
14. W. W. Terpstra, J. Kangasharju, C. Leng, and A. P. Buchmann, "Bubblestorm: resilient, probabilistic, and exhaustive peer-to-peer search," *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computing Communications*, Kyoto, Japan, 2007, pp. 49-60.
15. C. Leng, W. W. Terpstra, B. Kemme, W. Stannat, and A. P. Buchmann, "Maintaining replicas in unstructured P2P systems," *Proceedings of the ACM Conference on Emerging Network Experiments and Technology*, Madrid, Spain, 2008.
16. F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen, "PlanetP: using gossiping to build content addressable peer-to-peer information sharing communities," *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, Seattle, WA, 2003, pp. 236-246.
17. Y. Wang, L. Galanis, and D. J. DeWitt, "GALANX: an efficient peer-to-peer search engine system," University of Wisconsin, Madison, WI, 2003.
18. R. Morselli, B. Bhattacharjee, A. Srinivasan, and M. A. Marsh, "Efficient lookup on unstructured topologies," *Proceedings of the 24th Annual ACM Symposium on Principles of Distributed Computing*, Las Vegas, NV, 2005, pp. 77-86.
19. N. Sarshar, P. O. Boykin, and V. P. Roychowdhury, "Percolation search in power law networks: making unstructured peer-to-peer networks scalable," *Proceedings of the 4th International Conference on Peer-to-Peer Computing*, Zurich, Switzerland, 2004, pp. 2-9.
20. G. P. Jesi, D. Hales, and M. van Steen, "Identifying malicious peers before it's too late: a decentralized secure peer sampling service," *Proceedings of the 1st International Conference on Self-Adaptive and Self-Organizing Systems*, Boston, MA, 2007, pp. 237-246.
21. T. Condie, S. D. Kamvar, and H. Garcia-Molina, "Adaptive peer-to-peer topologies," *Proceedings of the 4th International Conference on Peer-to-Peer Computing*, Zurich, Switzerland, 2004, pp. 53-62.
22. R. Goonatilake, A. Herath, S. Herath, S. Herath, and J. Herath, "Intrusion detection using the chi-square goodness-of-fit test for information assurance, network, forensics and software security," *Journal of Computing Science in Colleges*, vol. 23, no. 1, pp. 255-263, 2007.
23. Y. T. Chuang, I. Michel Lombera, P. M. Melliar-Smith, and L. E. Moser, "Detecting and defending against malicious attacks in the iTrust information retrieval network," *Proceedings of the International Conference on Information Networking*, Bali, Indonesia, 2012, pp. 263-268.



Yung-Ting Chuang

Yung-Ting Chuang is a Ph.D. candidate in the Department of Electrical and Computer Engineering at the University of California, Santa Barbara. Currently, she is working in the distributed systems and computer networks group, led by Professor Michael Melliar-Smith and Professor Louise Moser. Yung-Ting received the B.S. degree in Computer Engineering in 2006, and the M.S. degree in Electrical and Computer Engineering in 2008, from the University of California, Santa Barbara. Her research interests include peer-to-peer networks, mobile networks, distributed systems and, in particular, information retrieval, membership/churn, and network attacks.

Michael Melliar-Smith

Dr. Michael Melliar-Smith is a Professor in the Department of Electrical and Computer Engineering at the University of California, Santa Barbara. Previously, he worked as a research scientist at SRI International in Menlo Park. His research interests encompass the fields of distributed systems and applications, and network architectures and protocols. He has authored or coauthored more than 285 publications in computer science and engineering. He has served on many technical program committees, and as a reviewer for many conferences and journals. Dr. Melliar-Smith is a pioneer in the field of fault-tolerant distributed computing. He received a Ph.D. in Computer Science from the University of Cambridge, England.

Louise Moser

Dr. Louise Moser is a Professor in the Department of Electrical and Computer Engineering at the University of California, Santa Barbara. Her research interests span the fields of computer networks, distributed systems, and software engineering. Dr. Moser has authored or coauthored more than 275 conference and journal publications. She has served as an Associate Editor for the IEEE Transactions on Services Computing and the IEEE Transactions on Computers and as an Area Editor for IEEE Computer magazine in the area of networks. She has served as Technical Chair of the 2012 IEEE International Conference on Services Computing and Technical Chair of the 2011 IEEE International Conference on Web Services, and on many technical program committees. She received a Ph.D. in Mathematics from the University of Wisconsin, Madison.

Isaí Michel Lombera

Isaí Michel Lombera is a Ph.D. candidate in the Department of Electrical and Computer Engineering at the University of California, Santa Barbara. He obtained the B.S. degree in Computer Engineering in 2007, and the M.S. degree in Electrical Engineering in 2009, from San Diego State University. He has worked in industry as a Web software developer and a custom software-hardware designer for embedded systems with an emphasis on video processing using field programmable gate arrays. Currently, he is working in the distributed systems and computer networks group, led by Professor Louise Moser and Professor Michael Melliar-Smith. His research interests include mobile ad-hoc networks, social networks, and information search and retrieval.