Using K-Nearest Neighbor Algorithm to Reduce False Negatives in P2P Secure Routing Protocols

Haidar Safa
Comp. Sci. Dept.
American Univ. of Beirut
Email: hs33@aub.edu.lb

Wassim El Hajj
Comp. Sci. Dept.
American Univ. of Beirut
Email: we07@aub.edu.lb

Fatima K. Abu Salem
Comp. Sci. Dept.
American Univ. of Beirut
Email: fatima.abusalem@aub.edu.lb

Marwa Moutaweh
Comp. Sci. Dept.
American Univ. of Beirut
Email: mam67@aub.edu.lb

***Abstract*—A peer-to-peer (P2P) system is known for its scalability and dynamic nature, where nodes can join and leave the system easily and at any time. These networks are susceptible to malicious behavior such as nodes dropping messages and misleading other nodes. P2P routing protocols are not immune against such incidents. Additionally, most secure routing protocols in the literature suffer from false negatives. In this paper, we propose to use the K-nearest neighbor (K-nn) algorithm in order to reduce false negatives in P2P secure routing protocols. We incorporate the proposed algorithm in a chord based trust aware P2P routing protocol, and evaluate its performance using the PeerSim simulator. Preliminary simulation results demonstrate that the proposed algorithm reduces the rate of false negatives without impacting the malicious node detection rate.**

***Keywords*—Peer-to-peer networks; machine learning; reputation; trust-awareness;**

I. Introduction

Peer-to-peer (P2P) networks consist of a collection of nodes that can pose to be data consumers and producers simultaneously. A distributed traditional P2P network does not contain central servers [2], [3], [9], [15]. In unstructured P2P systems, each peer typically stores its own data objects and maintains a set of links to its neighbors. When a node needs a data, lookup messages are flooded to all of its neighbors. Gnutella [12] and BitTorrent [13] are examples of such systems. Within the Gnutella framework, and in the instance a node ought to query a file, the query is flooded until either the file is found or the TTL is decremented to zero. Gnutella suffers from free riders problem where most users do not share files, but end up downloading them because sharing a popular file attracts a large volume of traffic. The BitTorrent system addresses this problem by breaking down the service process. Whilst peer serving a popular file, the BiTorrent framework breaks the file down into several sub-files. When a particular file is requested, each client receives a distinct sub-file, and so do the other clients simultaneously. Once the client has received a certain sub-file, it would allow other clients to download this sub-file from its own computer, while it proceeds to download the next sub-file from the original server. The process iterates until the download is complete. In structured P2P systems, data is placed at specific peers and locations using distributed hash tables (DHT), which require a large space of identifiers in order to assign uniform keys to data objects and uniform IDs to peers. Moreover, peers are structured into a graph that maps each data key to a peer. Each peer maintains a routing table that documents its neighbors' IDs and IP address. This table is used to forward messages across overlay paths to peers storing the requested data. The table-driven routing approach has the advantage of being more succinct than the flooding based routing used in unstructured networks. Chord [1], a given example of a structured network, gives every peer and data item a unique identifier chosen from a circular identifier space, and distributes the data across the peers in the system, in such away that each node is assigned roughly the same number of data keys. Given a set of N nodes and K keys, $\Theta(K/N)$ keys need to be remapped when a new node of index $(N+1)$ joins/leaves the network. Each node maintains a routing table called the finger table, consisting of m successor nodes. Each entry i in the finger table contains the IP address and the ID, denoted by N_{ID}, of the node such that $N_{ID} \geq (N_{ID} + 2^i) \bmod 2^m$, for all $0 \leq i \leq (m-1)$. When a node N with identifier N_{ID} wants to find the key, K_{ID}, it starts by searching in its finger table for a node N' with identifier N'_{ID} that satisfies $N_{ID} < N'_{ID} < K_{ID}$ and N'_{ID} is the farthest node that precedes K_{ID} in the chord ring. If such a node exists, node N instructs N' to find K_{ID} recursively. Otherwise, node N instructs its immediate successor in the chord ring to find K_{ID}. The intuition behind this algorithm is that the closer N_{ID} is to K_{ID}, the higher the probability that the node contains the requested data or is able to identify another node holding the data.

P2P networks are scalable and of dynamic nature, where nodes can freely join and leave the network. These features make them susceptible to several attacks, a few of which we list below:

1) Detouring routing attacks, in which malicious nodes detour the lookup messages and ship them to a farther node than the valid next hop. This increases the routing path length and disrupts the system performance.
2) Submitting bad requests, where malicious nodes provide corrupted files, wrong requests, or files that may deliver suspicious software to the requesting nodes.
3) Denial of service attacks, where malicious nodes send too many queries in order to overwhelm a victim node and prevent it from serving other nodes.
4) Dropping messages, where malicious nodes drop queries and end up never forwarding them to a valid next hop.
5) Misleading messages, where malicious nodes send queries to wrong nodes (similar to detouring messages).

978-1-4799-5344-8/15/$31.00 ©2015 IEEE

6) Polluting messages, where malicious nodes pollute the messages they receive before forwarding them to the next hop; ...etc.

Several secure Peer-to-Peer routing protocols have been proposed for P2P networks [5], [6], [7], [10]. However, most of these protocols suffer from false alarms and negatives. In this paper, we revisit the K-nearest neighbor (K-nn) algorithm traditionally used in pattern matching for classification and regression. We propose to incorporate the K-nn algorithm in the P2P secure routing protocol, and as a consequence, show that we are able to decrease the percentage of false negatives without affecting the malicious nodes detection rate. The remainder of this paper is organized as follows. In Sec. 2 we present a survey of the literature surrounding P2P secure routing protocols. In Sec. 3, we present our proposed system based on the K-nn algorithm. In Sec. 4, we evaluate the performance of our scheme. We conclude in Sec. 5.

II. Related Work

A Gnutella based approach to provide for P2P security was proposed in [4]. In this approach peers can keep track of the reputation of other peers in the network and may also share with them their own opinion and evaluation of others. This reputation sharing procedure is based on a distributed polling algorithm where the data requester is able to check the reliability of the file's providers before downloading it. Each peer will possess a unique and permanent ID, which will help maintain the history for each servant peer in the system. To determine the source from which to download the requested file, the requester inquires about each node from the pool of nodes that own the file and are willing to provide it, and as such, the requester then chooses the node with the best reputation. The main limitations of this approach are related to the overhead and bandwidth wastage. Additionally, this approach is able to detect only one type of malicious attacks, concerned with supplying bad data.

A peer-based monitoring approach was proposed in [7], in which every node in the system should monitor its neighbors, then periodically report their behavior to some trustworthy tracking nodes called *monitoring nodes*. Monitoring nodes are then organized into a threaded binary tree in order to efficiently store and search messages. To illustrate, suppose node B wishes to report node A's performance. Node B thus needs to send an UPDATE message to A's monitoring node, say, M_A. If this is the very first instance that node B is sending an UPDATE message about A, then B would need to obtain the IP address of M_A. For this, B has to first send A's IP address to a random monitoring node, say R. Node R searches the binary tree in order to identify the monitoring node whose range covers A's IP address. Node M_A then sends a response message to B along with its certificate. Therafter, B will directly send the UPDATE message about A to M_A. Note that both the data sender and the data receiver need to send monitoring messages in order for the monitoring message to detect attacks. In [7], a Linear Prediction sampling (LP sampling) was adopted in order to reduce network overhead, but the number of monitoring message was still deemed high.

A variant of the Chord system that is resilient against detouring routing attacks was proposed in [6]. The proposed solution introduces the concept of reverse edges in the Chord system (R-Chord), so that if the request misses the destination as a result of any malicious attack, the receptor node would return the message to the intially targeted node using the reverse edge. Thus, the path length of the misguided query is minimized, and the resilience of the R-Chord system considerably improved over Chord. Despite that this improved approach manages to forward the misguided message reversely in order to reach the destination node, it still cannot isolate or punish the malicious nodes, nor can it handle other malicious attacks such as dropping and polluting messages.

The tracer algorithm proposed in [5] enables the initiator to monitor the query routing in a Chord system. This algorithm solves three types of malicious attacks: dropping, misleading, and polluting messages. Its routing algorithm was designed to control the routing path. Indeed, when the initiator issues a query message, it sends both the request q and another encrypted copy of q using its private key, in order to ensure message integrity and authentication. An intermediate node X checks if it has the needed data, in the event of which it replies by sending the requested data to the initiator. Otherwise, node X resends the query to the next hop and replies with an ACK message to the initiator. The ACK message contains the ID, N_{ID}, of the next hop, encrypted with its own private key. Although the tracer algorithm can detect several types of malicious attacks, it fails when another intermediate node Y is a liar node which keeps sending a fake warning message to the initiator that reports the pollution of the message it has received from the previous node X. The tracer algorithm also does not detect the cases where the same query is subjected to several attacks at the same time. Additionally, a node's reputation is locally stored; thus, any malicious node is only known to some nodes, while the others will continue to use it.

A trust aware system for P2P routing protocols was proposed in [10]. It constantly analyzes the behavior of all nodes in order to determine their trust-worthiness and then to classify them accordingly, thus isolating the ones deemed malicious. This system tracks the nodes' reputation based on evaluation reports from the nodes themselves. The credibility of nodes that are inaccurately evaluating other nodes is also monitored. Thus, malicious evaluations would not be able to affect other nodes' reputation. Our proposed approach will be incorporated into this algorithm, the details of which we reveal in the next section.

III. The K-nearest neighbor algorithm for reducing the rate of false negatives

In this section we illustrate how to use the K-nearest neighbor (K-nn) algorithm, a machine learning algorithm of [14], in order to decrease the percentage of false negatives in secure P2P systems. The proposed algorithm will be incorporated in the trust aware routing protocol (TARP) developed in [10]. The TARP protocol employs a trust management system (TMS) that filters the opinion of each node about other nodes in order to calculate a given node's overall reputation. The TMS depends upon an initial premise that stipulates all peers have a good reputation but zero credibility. After each transaction, the source node will update its opinion about the nodes that have participated in routing the query. We integrated TARP with Chord, referred to as TARP-Chord. In this section we

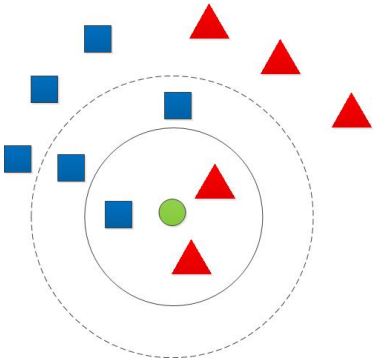

Fig. 1. K-nn classification

demonstrate how to use the K-nn algorithm at the TMS level to assist in classifying nodes as either malicious or non-malicious, after feeding the algorithm with supervised data.

A. The K-nearest neighbor algorithm

Unlike humans who are able to learn from past experiences, a machine can only learn from data streamed into it. A machine learning algorithm is able to predict the class to which a new data item belongs after feeding it with (supervised or un-supervised) data – i.e. in the pre-processing phase. The supervised data is the training data fed to an algorithm when it is labeled with pre-defined classes. Then, the test data, i.e. the newly unclassified data recently fed to the algorithm, will be classified into one of these pre-defined classes. The K-nn algorithm is used to classify objects according to the characteristics of the majority of its neighbors. It incorporates a form of "lazy learning" because the outcome of the algorithm is approximated locally. A given object of interest is assigned a class, which is common among its neighbors. For example, the ball in Fig. 1 will be classified as a triangle when applying the 3-nn algorithm because after considering the three closest neighbors to the circle, the majority of the objects are triangles (i.e. two triangles but one square). Inside the $k = 5$ neighborhood, however, the ball will be classified as a square, because the majority of neighbors in this case are squares. Alg. 1 illustrates the detailed workings of the k-nn algorithm. Given a new data item d, the algorithm will compute the Euclidean distance between d and each point in the supervised data set, D. It then chooses the k closest points to d (forming a set P) and determines the class of d to be that of the majority points in P.

Algorithm 1 K-nn algorithm (D, d, K)

1: Compute the Euclidean distance between the new data item d and each supervised data in D.
2: Choose the k data items in D that are nearest to D. Denote the set of such k points by P.
3: Associate with d the class which is the most frequent class in P (the majority class).

B. Reputation, credibility, and routing in TARP

To calculate a given node's overall reputation in TARP, the TMS filters in the opinion of all other nodes given about it. The TMS also takes into consideration the credibility of the nodes providing their opinion. Reputation is defined as an opinion surrounding an entity. Credibility is defined as the believability of an entity. Believability indicates to what extent thee entity is honest in delivering actions or evaluations. Initially, the TMS considers that all peers possess a good reputation, but zero credibility. After each transaction, the source node will update its opinion about the nodes that have participated in routing the query.

To describe how reputation and credibility are computed, let V denote the set of all nodes that have reported evaluations of node B to TMS, and let $Crd(v)$ denote the credibility of node v – that is, a measure of how much node v is trustworthy. The belief/disbelief factors are treated as probabilities and an evaluation is defined as the tuple $< belief, disbelief >$, where $belief + disbelief = 1$. An opinion will be the tuple $< 1, 0 >$ for a positive evaluation and $< 0, 1 >$ for a negative one. To further describe how the reputation of a given node B is computed using those probabilities, let $\Gamma(B)$, and $\Delta(B)$ denote the belief and disbelief values of node B respectively. Those values will in turn depend on the evaluation given by each node $v \in V$. Particularly, let $\gamma(v, B)$, $\delta(v, B)$ denote the belief and disbelief values produced by v surrounding B. Let E_v^B denote the set of all evaluations given by v about B, and P_v^B, N_v^B denote the subsets of E_v^B consisting of positive and negative evaluations respectively, such that $E_v^B = P_v^B \cup N_v^B$. Now, when TMS receives an evaluation of B's performance by any node $v \in V$, it will first compute belief and disbelief values of node v surrounding node B as given in the equation below:

$$\begin{array}{rcl} \gamma(v,B) & = & \frac{|P_v^B|}{|E_v^B|}, \\ \delta(v,B) & = & \frac{|N_v^B|}{|E_v^B|} \end{array} \quad (1)$$

TMS will then proceed to compute the overall reputation of B as in the equation below:

$$\begin{array}{l} \Gamma(B) = \frac{\sum_{v \in V} \gamma(v,B) \cdot Crd(v)}{|V|}, \\ \Delta(B) = \frac{\sum_{v \in V} \delta(v,B) \cdot Crd(v)}{|V|} \end{array} \quad (2)$$

To calculate the credibility of a node x, TMS uses trust vectors ([8]) which assign a weight to the node's recent feedback. For instance, if any node behaves well for a long time and then misbehaves, its credentials will decrease quickly. Hence, malicious nodes cannot fool other nodes by behaving well for a short period time only. Let $TV(X)$ denote the 8-bit trust vector of node X and $S(X)$ the number of significant bits in $TV(X)$. Each node possesses its own trust vector at the TMS. We normally start with a trust vector $TV(X)$ where all the bits are set to 0. Thereafter following each evaluation, TMS updates the evaluator's trust vector $TV(X)$ by "1" if it considers the evaluation *correct* (by correct we imply that the evaluation is not considered "odd" or "suspicious" as we explain later) or by "0", otherwise. This can be formulated using bitwise and bit shift operators as follows:

$$\begin{array}{rcl} TV(X) & = & (TV(X) >> 1) \vee \mathcal{M} \\ S(X) & = & S(X) + 1 \end{array} \quad (3)$$

where $\mathcal{M}$ is a mask address that is equal to 128 (i.e. 10000000 in binary) if the evaluation of X is correct, or is equal to 0 (i.e. 00000000 in binary) if the evaluation of X is not correct.

Also, $>>$ is the bitwise right shift operator and $\vee$ is the logical inclusive disjunctive operator.

Now, each time $TV(X)$ is updated, TMS will calculate the credibility of X as follows:

$$Crd(X) = Crd^{+}(X) - Crd^{-}(X) \tag{4}$$

where $Crd^{+}(X)$ is the credibility rating and $Crd^{-}(X)$ is the discredibility rating of node X. The credibility rating $Crd^{+}(X)$ computes the credibility of a node by assigning the weight for the most significant bits in the vector. Also, $Crd^{-}(X)$ computes the discredibility of a node by complementing the significant bits in the vector and calculating their weight:

$$\begin{array}{rcl} Crd^{+}(X) & = & \frac{TV(X)>>(8-S(X))}{2^{S(X)}} \\ Crd^{-}(X) & = & \frac{\overline{TV(X)}>>(8-S(X))}{2^{S(X)}} \end{array} \tag{5}$$

In TARP, when any node misbehaves by sending invalid packets, or polluting, dropping, or misleading packets, other nodes will detect and report this misbehavior to the TMS. If the TMS evaluates a node as malicious, it will notify other nodes, which will isolate the malicious node from their routing tables if it exists. TARP ensures that the next hop is trustworthy because when the TMS detects a malicious node, all nodes in the network will be notified.

The routing in TARP is described in [10] but can be briefly summarized as follows. First, let m denote the query message originated by the initiator, $E_S^- < m >$ denote the message m when encrypted with the private key of node S, and $E_S^+ E_S^- < m >$ denote the decrypted version of m using the public key of S. Normally, $E_S^+ E_S^- < m >= m$. Furthermore, let $E_s^- < timestamp >$ denote the time when the initiator created the message encrypted with the initiator's private key. Then, the routing starts when the initiator S creates a query Q and forwards it to the next hop. Upon receiving the query, each intermediate node X will decrypt it using the forwarding node's public key (forwarded messages are decrypted to preserve message integrity and authentication). Then, it will decrypt the message $E_S^- < m >$ using the public key of the initiator S, and compare it to the un-encrypted copy m to check if the message has been polluted. Node X compares m to the decrypted one in $E_S^+ E_S^- < m >$. In case the decryption fails or the message is polluted, X will send a warning message to S. Otherwise, it will decrypt the carried fields using the public key of the forwarding node and check if the encrypted source ID is equal to the ID of S (in order to ensure that the query is really issued by S). Afterwards, it will check whether NextHop is equal to its own ID in order to prevent any malicious attack that may appear as a result of a combination attack. In case any check fails, X will issue a warning to S, else, it will forward the query to the next hop and send an ACK to S.

IV. Incorporating the K-nn algorithm into TARP

The supervised data that we collected from our pre-processing phase is a vector of two dimensions: $\left(\frac{|P^X|}{|N^X|}, \frac{\Gamma(X)}{\Delta(X)}\right)$, where P^X and N^X denote the set of total positive and negative evaluations given about X respectively, and $\Gamma(X)$ and $\Delta(X)$ as defined in Sec. III, denoting the belief and disbelief values respectively. The related class of the supervised data can be distinguished as malicious versus non-malicious. We use the 1-nn algorithm because it is recommended to choose $K = \left\lfloor \sqrt{d} \right\rfloor$, where d is the dimension of the data. In a 1-nn algorithm, a node would be labeled malicious if the first closest node to it is malicious, else, it would be labeled non-malicious. To describe how the 1-nn algorithm can be incorporated in our approach, we further define the following notations:

- S_i is the set of nodes participating in the routing process in run i, and classified as malicious by TARP-Chord – i.e. these may include false negatives.
- D_i is the set of supervised data for all nodes in S_i.
- $d(X)$ is the supervised data for each node X in S_i.

In the preprocessing phase, we performed several runs on TARP-Chord. For every run, we collected for each node x that participated in the routing process during run i, the following supervised data:

$$d(x) = \begin{cases} \left(\frac{|P^X|}{|N^X|}, \frac{\Gamma(X)}{\Delta(X)}\right) & \text{if } X \in S_i \\ C(x) & \text{otherwise.} \end{cases} \tag{6}$$

where $C(X)$ is the class of node X. This node is designated as either malicious or non-malicious as follows: after each run, we consider all nodes in S_i, and filter them against the set of initial malicious nodes. If node X belongs to S_i as well as to the set of initial malicious nodes at the same time, we assign to $C(X)$ the value M (for "malicious"). If X is in S_i but is not in the set of initial malicious nodes, we assign $C(X)$ to be the value N (for "non-malicious").

The supervised data D is collected using all the runs and is illustrated as follows: $D = \cup_{i=1}^{n} D_i$, where n is the total number of simulation runs. Fig. 2 represents the two dimensional supervised data D, where the x-axis represents $\frac{|P^X|}{|N^X|}$ and the y-axis represents $\frac{\Gamma(X)}{\Delta(X)}$. Each point in Fig. 2 represents either an actual malicious node that was detected by TARP, or a false negative node (i.e. a node classified as malicious by TARP but one that is actually not).

To use the 1-nn algorithm, we first have to feed it with the supervised data D, then run TARP-Chord with some modifications in the TMS as follows. When both the disbelief value and the total number of nodes evaluating a given node X negatively exceed certain specified thresholds (T_δ and T_N respectivel), TMS will call the 1-nn algorithm in order to compute the Euclidean distance between node X and all the points in D using Eq. 1. The 1-nn algorithm will choose the nearest node to X and assign the class of X to be that of the nearest node.

The TMS evaluation algorithm can be briefly described as follows. Suppose that node X evaluates node Y negatively by sending a $< 0, 1 >$ update message to TMS. If X is the first node evaluating Y negatively, then these evaluations are considered by the TMS as "odd" evaluations (i.e. suspicious). After updating the reputation of Y according to Eqs. 1 and 2, TMS uses $flag(X)$ to keep track of the number of odd

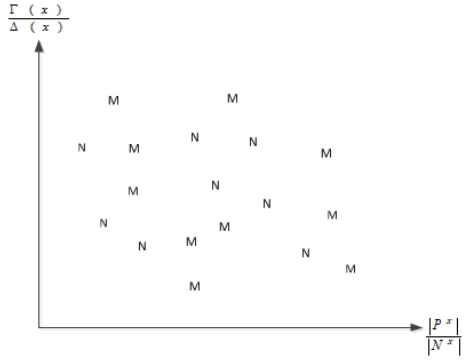

Fig. 2. Two-dimensional supervised data

evaluations submitted by node X about any node Z, provided that TMS had not received any similar evaluations about Z from other nodes. If $flag(X)$ exceeds a certain value, say T_{flag}, TMS will set the mask $\mathcal{M}$ to 0 then update $TV(X)$ as per Eq. 3. If this was not the first negative evaluation, then after updating Ys reputation, TMS will set $\mathcal{M}$ to 128, then update $TV(X)$ as per Eq. 3. Similarly, if X is the first node evaluating Y positively, then these evaluations are also considered by the TMS as "odd" evaluations. Upon recalculating the reputation of a node, TMS will broadcast a notification (UPDATE) message if the node appears to be malicious. Nodes receiving this message will isolate malicious nodes for a period of time, then give them a chance to repent. If a node is isolated several times, it will be disconnected from the system.

Alg. 2 below illustrates how the TMS uses the 1-nn algorithm to classify nodes. It shows how TMS behaves upon receiving a negative evaluation. Indeed, when X reports a negative evaluation about Y, the TMS checks whether X happens to be the first node provinding such an evaluation. If it is the case, TMS will track the ID of X as being the first node that negatively evaluated Y (indicated below by the assignment $NfirstEv(Y) = X$), then increment $flag(X)$ and update the reputation of Y as per Eqs. 1 and 2. The TMS will check whether the number of odd evaluations that X submitted given by $flag(X)$ is larger than a certain threshold T_{flag}. If so, it would decrease the credibility of X and set $flag(X)$ to 0. If X is the second node that negatively evaluates Y, TMS would decrement $flag(NfirstEv(Y))$ and update $TV(NfirstEv(Y))$ (thus, implicitly making $NfirstEv(Y)$ more credible). TMS then updates the credibility of X and reputation of Y. To decide whether Y is malicious or not, the TMS not only uses the new disbelief value of Y, $\Delta(Y)$, but also considers the number N^- of nodes that are evaluating Y negatively (this is to insure whether several nodes are evaluating Y negatively). If N^- and $\Delta(Y)$ exceed certain thresholds, the TMS will not directly consider Y to be malicious. Instead, it will call the 1-nn algorithm and check the return value of the latter. If the return is "M" (for "malicious"), then Y will be classified as malicious, and the remaining nodes in the network will be notified. Else, node Y will not be considered malicious.

Algorithm 2 Node X submitting a negative evaluation of node Y

if X is the first node evaluating Y negatively **then**
  Set $NfirstEv(Y)$ to be the ID of X
  $flag(X)++$
  Update the reputation of Y as per Eq. (1) and Eq. (2)
  if $flag(X) \geq T_{flag}$ **then**
    {incorrect evaluation}
    Update $TV(X)$ by "0" as per Eq. (3)
    Recalculate $Crd(X)$ as per Eq. (4) and Eq. (5)
    Set $flag(X) \leftarrow 0$
  end if
else
  if X is the second node evaluating Y negatively and $X \neq NfirstEv(Y)$ **then**
    $flag(NfirstEv(Y))--$
    Update TV(NfirstEv(Y)) by "1" as per Eq. (3)
    Recalculate $Crd(NfirstEv(Y))$ as per Eq. (4) and Eq. (5)
  end if
  Update $TV(X)$ by "1" as per Eq. (3)
  Recalculate $Crd(X)$ as per Eq. (4) and Eq. (5)
  Update Ys reputation as per Eq. (1) and Eq. (2)
  if $\Delta(Y) > T_\delta$ and $N^- > T_N$ **then**
    Call the 1-nn algorithm on $\left(\frac{|P^Y|}{|N^Y|}, \frac{\Gamma(Y)}{\Delta(Y)}\right)$. Return $value$.
    if $value \equiv$ "M" **then**
      Label Y as malicious
      Send a notification message
    end if
  end if
end if

V. Performance Analysis

In this section we evaluate the performance of TARP-Chord employing the K-nn algorithm for reducing the rate of false negatives against its performance in the absence of the K-nn algorithm. The protocol was implemented using the PeerSim simulator [12]. We have evaluated the percentage of false negatives, defined as the number of good nodes that were considered malicious by the trust system. We have also evaluated the percentage of malicious node detected by the trust system, defined as the total number of malicious nodes detected by the TMS, divided by the total number of malicious nodes in the system.

These metrics were measured in a network of a hundred nodes, and request rate of 1 packet / 2 seconds. Percentages of malicious nodes varied from 10% to 40%. Simulation time lasted for up to $300,000$ seconds, and $Tflag$ was set to 3. We used the following combination as a threshold:

$$(\Delta(X) > 0.5) \vee (\Delta(X) > 0.2 \wedge N^- > 10) \\ \vee (\Delta(X) > \Gamma(X) \wedge N^- < 10)$$

We also used the central limit theorem to calculate the number of runs while achieving a 90% confidence level with a precision value of 3%. The number of runs varied from two to six depending on the measured metric and the used protocol.

We performed the pre-processing phase after which we fed the algorithm with the supervised data, using the 1-nn algorithm to calculate the resulting percentages of the false

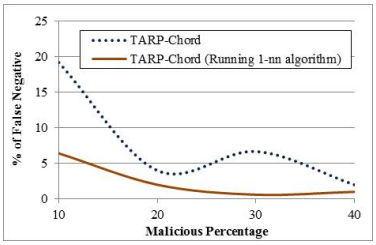

Fig. 3. Percentage of false negatives in relation to the percentage of malicious nodes in the network. Requested rate = 1 packet/2 sec. and network size = 100

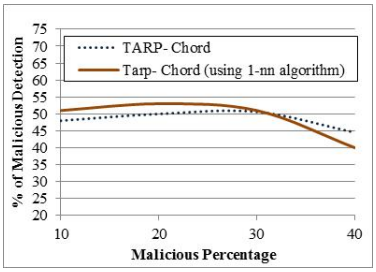

Fig. 4. Percentage of malicious detection. Traffic rate = 1 packet/ 2 sec. and network size = 100

negatives and the malicious detection. Fig. 3 shows that after using the 1-nn algorithm, the false negative percentages is reduced from 19% to 6% when the percentage of malicious nodes is at 10%, and from 7% to 0.6% when the percentage of malicious nodes is at 30%, while it decreases slightly when the percentage of malicious ndoes is at 20% and 40%. To check whether using the 1-nn algorithm bears any impact on the malicious detection percentage, we performed some experiments and measured this particular rate. Fig. 4 shows that the malicious detection percentage is not affected by applying the machine learning algorithm and the loss is minor.

VI. Conclusion

In this paper, we proposed to use the K-nearest neighbor algorithm to decrease the percentage of false negatives in secure P2P systems. The proposed algorithm was incorporated in the trust aware routing protocol (TARP) of [10] which employs a trust management system (TMS) that filters the opinion of each node about other nodes in order to calculate a given node's overall reputation. We implemented TARP over Chord, a scheme we refer to as TARP-Chord, then used the K-nearest neighbor algorithm at the TMS level in order to classify nodes as either malicious or non-malicious, after feeding the algorithm with the supervised data. We evaluated the performance of the proposed system using PeerSim simulator. Preliminary simulation results have shown that the proposed algorithm reduces the rate of false negatives without impacting the malicious node detection rate.

Acknowledgment

This work was supported in part by a grant from the University Research Board of the American University of Beirut, Lebanon.

References

[1] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications", *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 1732, 2003.

[2] D. Korzun, A. Gurtov, "Hierarchical architectures in structured peer-to-peer overlay networks", *Peer-to-Peer Networking and Applications*, vol. 7, no. 4, pp. 1-37, 2013.

[3] Q. H. Vu, M. Lupu, B. C. Ooi. Springer-Verlag Berlin Heidelberg, *Peer-to-Peer Computing: Principles and Applications*, Springer, 2010.

[4] F. Cornelli, E. Damiani, S. De Capitani di Vimercati, "Choosing reputable servants in a P2P network", in Proc. WWW '02, 2002, pp. 376-386.

[5] X. Xu, T. Jin, "Efficient secure message routing for structured peer-to-peer systems", *Journal of Convergence Information Technology*, vol. 5, no. 4, pp. 75-83 , 2010.

[6] D. Xuan, S. Chellappan, M. Krishnamoorthy, "RChord: An enhanced chord system resilient to routing attacks", in Proc. MobiCom'03, 2003.

[7] X. Jin, S. H. Gary Chan, "Detecting malicious nodes in peer-to-peer streaming by peer-based monitoring", *ACM Transactions on Multimedia Computing, Communications and Applications*, vol. 6, no. 2, 2010.

[8] S. A. Aydn, U. Ersin, and R. P. Mark, "A reputation-based trust management system for P2P networks", *International Journal of Network Security*, vol. 6, no. 3, pp. 235-245, 2008.

[9] M. El Dick, E. Pacitti, R. Akbarinia, B. Kemme,"Building a peer-to-peer content distribution network with high performance, scalability and robustness, *Information Systems*, vol. 36, no. 2, pp. 222-247, 2011.

[10] H. Safa, W. El-Hajj, and M. Moutaweh, "Trust aware system for P2P routing protocols", in Proc. of AINA'14, pp. 829-836, 2014.

[11] (2014) PeerSim: A Peer-to-Peer Simulator. [Online]. Available: http://peersim.sourceforge.net/

[12] (2002) Ultrapeers: Another Step Towards Gnutella Scalability. [Online]. Available: http://rfc-gnutella.sourceforge.net/Proposals/Ultrapeer/Ultrapeers.htm

[13] (2003) BitTorrent. [Online]. Available: http://www.bittorrent.com/

[14] T. M. Cover, P. E. Hart, "Nearest neighbor pattern classification", *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21-27, 1967.

[15] A. Iamnitchi, P. Trunfio, J. Ledlie, F. Schintke, Eds., *Peer-to-Peer Computing*, ser. Lecture Notes in Computer Science. Berlin Heidelberg: Springer-Verlag, 2010, vol. 6271.