2014 IEEE 28th International Conference on Advanced Information Networking and Applications

Trust Aware System for P2P Routing Protocols

Haidar Safa, Wassim El-Hajj, and Marwa Moutaweh
Department of Computer Science
American University of Beirut
Beirut, Lebanon
{hs33, we07, mam67}@aub.edu.lb

***Abstract*—A peer-to-peer (P2P) system is known by its scalability and dynamic nature where nodes can join and leave the system easily and anytime. These networks are susceptible to malicious behaviors such as nodes dropping messages and misleading requesting nodes. P2P routing protocols are not immune against these misbehaviors. Therefore, detecting and dealing with malicious nodes will certainly lead to more reliable and secure system. In this paper, we propose a trust aware system for P2P routing protocols. The proposed system analyzes constantly the behaviors of all nodes to determine their trustworthiness then classify them accordingly isolating the ones deemed malicious. It 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 affect other nodes' reputation. We have integrated the proposed approach with several P2P routing protocols and evaluated their performance through simulations measuring parameters such as request delivery ratio, malicious detection, and false negatives. Results show that the proposed approach improves significantly the performance of P2P routing protocols.**

Keywords—Peer-to-peer networks; routing; reputation; trust-awarness;

I. INTRODUCTION

Peer-to-peer (P2P) networks consist of a collection of nodes that can be data consumers and producers at the same time. A distributed traditional P2P network does not contain central servers and can be either unstructured or structured [11, 14]. In unstructured network, each peer typically stores its own data objects and maintains a set of links to neighbors. When a node needs a data, lookup messages are flooded to all its neighbors. In structured networks, data are placed at specific peers and locations using distributed hash tables (DHT) which uses a large space of identifiers to assign uniform keys to data objects and uniform *IDs* to peers. Moreover, peers are organized into a graph that maps each data key to a peer. Each peer maintains a routing table that contains its neighbors' *ID* and *IP* address. Such a table is used to forward messages across overlay paths to peers storing the requested data. This table-driven routing is more elegant than the flooding based routing used in unstructured networks. Chord [1] and Pastry [2] are two examples of structured networks.

P2P networks are scalable and of dynamic nature where nodes can freely join and leave the network. These features make the system susceptible to attacks because malicious nodes may pollute, drop and mislead messages without being detected. In this paper, we propose a system that can be easily incorporated with P2P routing protocols to detect malicious nodes and classify them according to their trust and reputation level. The proposed system monitors nodes' behaviors and isolates the malicious ones. The remainder of this paper is organized as follows. Section 2 surveys P2P routing protocols and other related work. In section 3, we present our proposed system. In Section 4, we evaluate its performance when integrated with known P2P routing protocols such as Chord and Pastry. Conclusion is presented in section 5.

II. BACKGROUND AND RELATED WORK

Many P2P routing protocols have been proposed in literature [1, 2, 3, 10]. Some are not considered realistic as they were designed to work in benevolent environments while others are more realistic taking into consideration malicious attacks and behaviors.

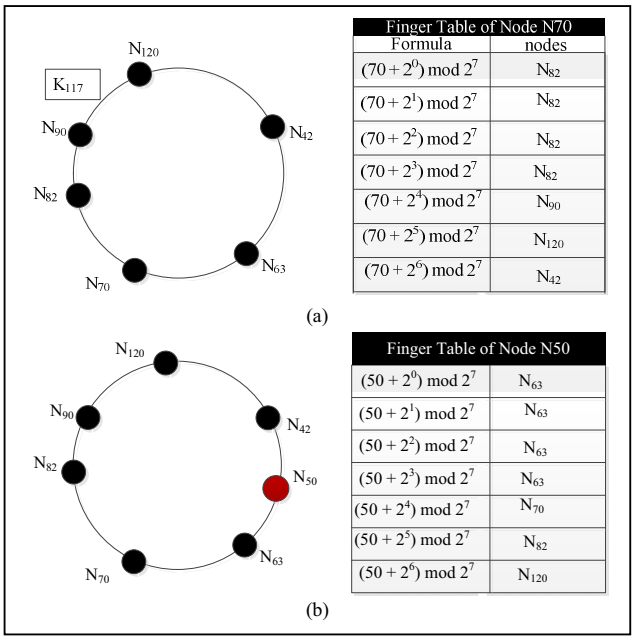
Chord routing protocol [1] uses consistent hashing to give every peer and data item a unique identifier from a circular identifier space, and distributes the data equally on the peers in the system. In consistent hashing, each node is roughly responsible for the same number of data keys. For a set of N nodes and K keys, $O(\frac{K}{N})$ keys need to be remapped, when a new node (N+1) joins/leaves the network. To describe how chord works, suppose that we have a system of *N* nodes and a circular identifier space from 0 till 128 bits. Consistent hashing hashes each node *IP* address to a point on the circle, say N_{ID}, and hashes each data item to the same circle identifier space, say K_{ID}. It then stores the data on the first node whose N_{ID} is equal or clockwisely follows the K_{ID} in the circular space as shown in Fig. 1. In the left side of Fig. 1(a), data item identified by K_{117} will be stored in node N_{120}. In Chord, each node maintains a routing table called finger table that consists of *m* successor nodes (*m* is a preconfigured parameter). Each entry *i* in the finger table contains the *IP address* and N_{ID} of the node whose N_{ID} equal or succeeds $(N_{ID}+2^i) \bmod 2^m$ where $0 \le i \le m-1$. In right side of Fig. 1 (a), for m=7 (i.e. 7 entries in the finger table), the first entry is N_{82} since N_{82} is the first node that is equal or succeeds 71 ($(70+2^0) \bmod 2^7 = 71$). Also the second entry is N_{82} since N_{82} is the first entry that succeeds 72 and so on. Similarly N_{42} is the last entry in this finger table since it is the first node that is equal or succeeds 6 ($(70+2^6) \bmod 2^7 = 6$). *IP* addresses were not shown for simplicity.

Suppose that node *N* with identifier N_{ID} wants to find the key, K_{ID}. At the beginning, *N* starts by searching its finger table

1550-445X/14 $31.00 © 2014 IEEE
DOI 10.1109/AINA.2014.101

for a node N' with an identifier N'_{ID} that satisfies the condition $N_{ID} < N'_{ID} < K_{ID}$ such that N'_{ID} is the farthest node that precedes K_{ID} in the chord ring. If such a node exists then N will ask N' to find K_{ID} recursively. Otherwise N will ask its immediate successor in the ring to find K_{ID}. The idea of this algorithm is that the closer N_{ID} to K_{ID}, the higher the probability that the node contains the requested data or knows the node holding the data.

Now assume node N_{70} wants to search for data with key K_{117}. N_{70} will send the query to N_{90} since according to it, N_{90} is the farthest node that precedes K_{117}. N_{90} will then search its finger table to find a node with *ID* larger than 90 and smaller than 117. Since such node does not exist, N_{70} will send the query to its immediate successor which is N_{120} which contains the requested data. Hence N_{120} will send its *IP* for the requested node so that direct data download will take place. Note that if N_{120} does not hold K_{117} then this means that the data is not found in the system.



Finger Table of Node N70

Formula	nodes
$(70+2^0) \bmod 2^7$	N_{82}
$(70+2^1) \bmod 2^7$	N_{82}
$(70+2^2) \bmod 2^7$	N_{82}
$(70+2^3) \bmod 2^7$	N_{82}
$(70+2^4) \bmod 2^7$	N_{90}
$(70+2^5) \bmod 2^7$	N_{120}
$(70+2^6) \bmod 2^7$	N_{42}

Finger Table of Node N50

$(50+2^0) \bmod 2^7$	N_{63}
$(50+2^1) \bmod 2^7$	N_{63}
$(50+2^2) \bmod 2^7$	N_{63}
$(50+2^3) \bmod 2^7$	N_{63}
$(50+2^4) \bmod 2^7$	N_{70}
$(50+2^5) \bmod 2^7$	N_{82}
$(50+2^6) \bmod 2^7$	N_{120}

Fig. 1 (a) an example of a Chord overlay network and finger table of node 70; (b) an example of a node (node 50) joining the network and its finger table.

Assume now that a node wants to join the chord system, then it will obtain from the system an *ID*, say N_{50}, and will be provided by an *ID* of an existing node say N_{70}. Then N_{50} sends to N_{70} a lookup query message of keys K_{51}, K_{52}, K_{54}, K_{58}, K_{66}, K_{82}, and K_{114} which are produced as a result of the function $(N_{ID}+2^i) mod\ 2^m$ where $0 \le i \le m-1$. N_{70} will query the lookup messages clockwisely using its finger table (i.e, it will forward K_{51}, K_{52}, K_{54}, K_{58}, K_{66} to N_{42}; K_{82} to N_{82}; K_{114} to N_{90}). Suppose N_{63} has the first requested key, K_{51}, hence N_{50} will receive the following reply: [N_{63} has the key K_{51}] and it will be located between node N_{63} and its predecessor node, N_{42} as shown in Fig. 1 (b). N_{50} fills its routing table entries from responses sent by nodes that have the following keys K_{51}, K_{52}, K_{54}, K_{58}, K_{66}, K_{82}, and K_{114}.

Pastry [2] is one of the most popular structured P2P overlay networks. It is similar to Chord in the overlay network where nodes and keys will be mapped to a circular space using DHT. However, Pastry routing table and algorithm are based on Plaxton, Rajaraman, and Richa trees (PRR Trees) and Plaxton routing [13]. Pastry employs a prefix-based scheme where each node has a unique 128-bit identifier chosen from the circular identifier space which ranges from *0* till $2^{128} - 1$. The node identifiers, N_{ID}, and Keys, K, are sequence of digits with base $B = 2^b$ where b is a system parameter. Each node X has a routing table that consists of three different types of sets*: set of PRR style neighbors, set of leaf neighbors* and *set of neighborhood nodes*. PRR and leaf sets are used in the routing process while neighborhood set is used in the system construction. In the set of leaf neighbors of X, half of the leaf neighbors contain nodes with IDs closer and smaller than *X*'s ID, while the other half have nodes with IDs closer but larger than the *X*'s ID.

In PRR tree, each node and object is assigned an identifier from the identifier space that is independent of node and object location and semantic properties. The main idea is that for each object a rooted tree will be formed. The root node stores a pointer to the server that stores the data object. A peer's PRR table has $log_B N$ rows, where each row holds *B-1* entries. Entries in row *n* contain peers whose N_{ID} shares the first *n* digits with *X*'s ID. In Table I, $B=2^3=8$ shows that entries in row zero does not share digits with node whose *ID* starts with *5642*, while in row *2* all the entries share the first two digits (*56*). It also shows that with $b=3$ and $N=2^{12}$ then the total number of rows is $log_8(2^{12}) = 4$ having *8* entries (*B-1* entry) in each row, while the shaded entry just represents the corresponding digit of the node's ID. The first entry in table I (*row 0*, *column 1*) will be the closest node to node *5642* whose *ID* starts with digit *0* so if the pastry system has two nodes with IDs *0123* and *0234* and node *0123* is closer to node *5642* than node *0234* then the first entry will be *0123*. The same procedure is applied for filling the other entries. If no node was found to satisfy this condition then the entry will be left empty.

TABLE I. PRR ROUTING TABLE OF A NODE ID= 564X

Row0	0x	1x	2x	3x	4x	5x	6x	7x
Row1	**5**0x	**5**1x	**5**2x	**5**3x	**5**4x	**5**5x	**5**6x	**5**7x
Row2	**56**0x	**56**1x	**56**2x	**56**3x	**56**4x	**56**5x	**56**6x	**56**7x
Row3	**564**0	**564**1	**5642**	**5643**	**5644**	**5645**	**5646**	**5647**

The leaf set is used at the beginning of the query routing where upon receiving a lookup message, a node *X* first checks if the key of the query, *K*, is close to the *ID* of any node in the leaf neighbors set. If it is, then it forwards the query to that leaf node. Otherwise, if there exists a PRR neighbor, which has a common prefix with *K* that is longer than the common prefix between *X* and *K*, the query will be forwarded to that neighbor. If this is not the case, *X* will forward the query to a node which either has the same common prefix length between *X* and *K* or to a node whose identifier is closer to *K* than *X*. For example, when a node *564x* receives a message of search key *2564*, it first checks its leaf set neighbors if it shares common prefixes with *2564*. If it does not, node *564x* will look in its PRR neighbor set for a node that has the longer common prefix with *2564* which appears to be *2x* where *x are* remaining digits and

sends the query to that node which in turn will send it to the nearest node until it reaches the node that has the data key *2564*.

We focused on Chord and Pastry as they are two of the four original distributed hash table protocols for P2P networks. We discuss next our proposed security system that can be incorporated with any P2P protocol. Later we integrate it with Chord and Pastry for evaluation.

III. PROPOSED APPROACH

In this section we present our proposed trust management system (TMS) which takes the opinion of each node about other nodes to calculate the overall node's reputation. It also takes into consideration the credibility of the nodes giving their opinions about other nodes. *Reputation* is defined as an opinion given to an entity. *Credibility* is defined as the believability of an entity. Believability means how much the entity is honest in its actions or evaluations. Initially the TMS considers that 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.

A. Calculating Reputation and Credibility

Let V be the set of all nodes that reported evaluations of node B to TMS. An evaluation is defined as <*belief, disbelief*> where *belief* + *disbelief* = *1*; the opinion will be <1, 0> for a positive evaluation and <0, 1> for a negative one. For instance, when TMS receives an evaluation of *B's* performance by any node say "*v*" where $v \in V$, TMS will first compute *belief* and *disbelief* values of node *v* in node *B* as given in equation (1).

$$belief(v,B) = \frac{totalP_v^B}{totalE_v^B}$$
$$disbelief(v,B) = \frac{totalN_v^B}{totalE_v^B} \quad (1)$$

where $totalP_v^B$ represents the total number of positive evaluations given by *v* about *B*, $totalN_v^B$ represents the total number of negative evaluations given by *v* about *B* and $totalE_v^B$ represents the total number of evaluations given by *v* about *B*.

The TMS will then use eq.(1) to compute the reputation of B as in equation (2).

$$Belief(B) = \frac{\sum_{v \in V} belief(v,B) * Crd(v)}{N}$$
$$Disbelief(B) = \frac{\sum_{v \in V} disbelief(v,B) * Crd(v)}{N} \quad (2)$$

where $Crd(v)$ represents the credibility of node *v,* (i.e. how much node ***v*** is trustworthy), and *N* represents the cardinality of set *V*.

To calculate the credibility of node *x*, TMS uses *trust vector*s [9] which assign weight to the recent node's feedback. Therefore, 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. Let *TV(x)* be the 8 bits trust vector of node *x* and *S(x)* be the number of significant bits in the *TV(x)*. Every node has its own trust vector at the TMS. We normally start with a *TV(x)* where all the bits are set to 0, then after each evaluation, TMS updates the evaluator's *TV(x)* by "1" if it considers its evaluation correct (by correct we mean that the evaluation is not considered "*odd" / "suspicious"* as we will explain later) or by "0" otherwise. This can be formulated as following:

$$TV(x) = (TV(x) \succ\succ 1) \quad OR \quad M$$
$$S(x) = S(x) + 1 \quad (3)$$

Where *M* is a mask address that is equal to 128 (i.e. 10000000 in binary) if the evaluation of *x* is correct or 0 (i.e. 00000000 in binary) if the evaluation of *x* is not correct; $\succ\succ$ is the bitwise right shift operator; *OR* is the bitwise OR that performs logical inclusive OR operation. Every time TV(*x*) is updated, TMS will calculate the credibility of *X* as follows:

$$Crd(x) = CredRating(x) - DiscRating(x) \quad (4)$$

where *CredRating(x)* is defined as the credibility rating of node *x* and *DiscRating(x)* is defined as the Discredibility rating of node *x*. *CredRating* calculates the credibility of a node by giving the weight for the most significant bits in the vector while *DiscRating* calculates the discredibility of nodes by complementing the significant bits in the vector and calculating their weight.

$$\text{CredRating}(x) = \frac{TV(x) \succ\succ (8 - S(x))}{2^{S(x)}}$$
$$\text{DiscRating}(x) = \frac{\overline{TV(x)} \succ\succ (8 - S(x))}{2^{S(x)}} \quad (5)$$

Fig. 2 shows an example on how *Crd(x)* is calculated. Suppose that the evaluation of node *x* is correct then TMS will add a 1 to the most significant bit in *X's* trust vector, *TV(x),* and increment the significant bits number, *S(x),* by 1. Using eq (5), *CredRating(x)* is $\frac{(11101)_{base2}}{2^5}$ = *0.90625* and *DiscrRating(x)* is $\frac{(00010)_{base2}}{2^5}$ = *0.0625* then *Crd(x)* is calculated as per eq (4) *0.90625 – 0.0625 = 0.843.*

TV(x) (before)		TV(x) (after)
11010000	⟶	**11101**000
S(x) = 4		S(x) = S(x)+1=5

Fig. 2 Illustration of How TV(x) is updated after an event is being reported.

B. TMS Node Evaluation Algorithms

Suppose that *X* evaluates *Y* negatively by sending <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 equations (1) and (2), TMS uses *flag(X)* to keep track of the number of *odd* evaluations submitted by node *X* about any node *z*, providing that TMS did not receive any similar evaluations about *z* from other nodes. If *flag(X)* exceeds certain value, say T_{flag}, TMS will set M to 0 then update *TV(X)* as per equation (3). If this was not the first negative evaluation, then after updating *Y*'s reputation, TMS will set M to *1*28 then update *TV(X)* as per equation (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 then it will be disconnected from the system.

Algorithm 1 shows how TMS behaves upon receiving negative evaluation. Indeed, when *X* reports a negative evaluation about *Y*, the TMS checks if *X* is the first node negatively evaluating *Y*. If it is, TMS will track the *ID* of *X* as the first node that negatively evaluates *Y* (i.e. *NfirstEv(Y)=X*) then increments *flag(X)* and updates *Y's* reputation as per equations (1) and (2). The TMS will check if the number of odd evaluation that *X* submitted, *flag(X)*, is large, then it will decrease the credibility of *X* and set *flag(X)* to *0*. If X is the second node that negatively evaluates *Y* then the TMS will decrement *flag*(*NfirstEv(Y)*) and update *TV*(*NfirstEv(Y)*) (i.e. implicitly makes *NfirstEv(Y)* more credible). Then TMS updates the credibility of *X* and reputation of *Y*. To decide whether *Y* is malicious or not, the TMS does not use only the new disbelief of *Y*, *Disbelief(Y),* but also considers the number of nodes that are evaluating *Y* negatively, $N_{Negative}$ (i.e. this to make sure that several nodes are evaluating *Y* negatively). If $N_{Negative}$ and the *Disbelief(Y)* exceeds certain thresholds, then *Y* will be classified as malicious and the other nodes in the network will be notified.

Algorithm 1: Node X submitted a negative evaluation for nodeY

```
1.  If (X is the first node evaluating Y negatively)
2.      NfirstEv(Y) = X's ID.
3.      Flag(X)++;
4.      Update Y's reputation as per eq (1) and eq (2)
5.      If ( flag(X) >= T_flag )
6.          Update TV(X) by "0" as per eq (3)
7.          Recalculate Crd(X) as per eq (4) and (5).
8.          Flag(X)=0;
9.  Else
10.     If(X is 2nd node evaluating Y negatively && X != NfirstEv(Y)
11.         Flag(NfirstEv(Y))--;
12.         Update TV(NfirstEv(Y)) by "1" as per eq (3)
13.         Recalculate Crd(NfirstEv(Y)) as per eq (4) and (5).
14.     Update TV(X) by "1" as per eq (3)
15.     Recalculate Crd(X) as per eq (4) and (5).
16.     Update Y's reputation as per eq (1) and eq (2)
17.     If (Disbelief(Y) > T_disbelief && N_Negative > T_N )
18.         Y is malicious.
19.         Send a notification message
```

Algorithm 2 shows how TMS will behave upon receiving positive evaluation. When *X* reports a positive evaluation about *Y*, the TMS checks if *X* is the first node positively evaluating *Y*. If it is, TMS will track the *ID* of *X* as the first node that positively evaluates *Y* (i.e. *PfirstEv(Y)=X*) then increments *flag(X)* and updates *Y's* reputation as per eqs (1) and (2). To decide whether *Y* is becoming malicious or not, the TMS does not use only the new *disbelief* of *Y*, *Disbelief*(*Y*), but also considers the number of nodes that are evaluating *Y* negatively ($N_{Negative}$). If $N_{Negative}$ and the *Disbelief(Y)* exceeds certain thresholds, *Y* will be classified as malicious and other nodes in the network will be notified. The TMS will also check if the number of odd evaluation that *X* submitted, *flag(X)*, is large then it will decrease the credibility of *X* and set *flag(X)* to *0*. If the evaluation is the second evaluation that positively evaluates *Y* then TMS will decrement *flag*(*PfirstEv(Y)*) and update *TV*(*PfirstEv(Y))* (i.e. implicitly makes *PfirstEv(Y)* more credible). Then TMS updates the credibility of *X* and reputation of *Y* to check if it's malicious or not.

Algorithm 2: Node X submitted a positive evaluation for node Y

```
1.  If (X is the first node evaluating Y positively)
2.      PfirstEv(Y) = X's ID.
3.      Flag(X)++;
4.      Update Y's reputation as per eq (1) and eq (2)
5.      If (Disbelief(Y) > T_disbelief && N_Negative > T_N )
6.          Y is malicious.
7.          Send a notification message
8.      If ( flag(X) >= T_flag )
9.          Update TV(X) by "0" as per eq (3)
10.         Recalculate Crd(X) as per eq (4) and (5).
11.         Flag(X)=0;
12. Else
13.     If(X is 2nd node evaluating Y positively && X != PfirstEv(Y))
14.         Flag(PfirstEv(Y))--;
15.         Update TV(PfirstEv(Y)) by "1" as per eq (3)
16.         Recalculate Crd(PfirstEv(Y)) as per eq (4) and (5).
17.     Update TV(X) by "1" as per eq (3)
18.     Recalculate Crd(X) as per eq (4) and (5).
19.     Update Y's reputation as per eq (1) and eq (2).
20.     If (Disbelief(Y) > T_disbelief && N_Negative > T_N )
21.         Y is malicious.
22.         Send a notification message
```

C. *Routing Protocol*

The proposed trust management system is integrated inside a routing protocol so when any node misbehaves (sends invalid packets, pollutes, drops and misleads 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. The proposed algorithm ensures that the next hop is trustworthy because when the TMS detects a malicious node, all nodes in the network will be notified. Before describing the flow of messages in secure routing process we will define the following notations and queries:

- *m*: query message originated by the initiator.

- $E_s^-<m>$: Message *m* is encrypted with the private key of node S.
- $E_s^+E_s^-<m>$: decrypting *m* using the public key of S. Normally $E_s^+E_s^-<m>=m$.
- E_s^-*<timestamp>:* the time when the initiator created the message encrypted with the initiator private key (it is used to detect replay attack and other types of attacks such as denial of service).

Several messages are exchanged between the initiator and intermediate nodes. The *query* shown in Fig. 3 includes the message *m*, E_s^-*<m>*, the *timestamp* and E_s^-*<srcID>* encrypted with the forwarding node's private key, and the ID of the next hop (*NextHop*) encrypted with the forwarding node's private key (i.e $E^-_{forwardingNode}<carried>$). The set of fields grouped as *carried* are sent in every query acknowledgment (ACK). The whole query should be then encrypted with the forwarding node's private key.

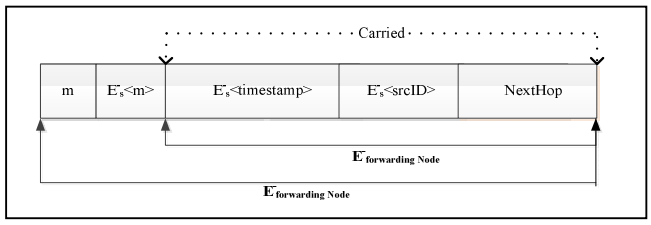


Fig. 3 Format and Content of Query .

Fig. 4 shows a flowchart that describes how the intermediate node processes a query. The initiator of the query tracks the routing procedure. It maintains information about the previous node from which it has received an ACK, (i.e. it will store it as *PreviousAckSource*), and the next node from which it should expect the next ACK, (i.e. *ExpectedACKSource*). Initially, the initiator sets *PreviousAckSource* to its own address and *ExpectedACKSource* to the node to which it sends the message. Then upon receiving the first ACK from the *ExpectedACKSource*, it will update its value using the *forwardto* field in ACK message shown in Fig. 5. The ACK message contains: *forwardto* field which represents the ID of the next hop that the ACK source is going to send the query to, the *PreviousHop* field which represents the ID of the previous node that forwarded the query to this current node, and some encrypted fields that are *carried* from the query received from previous hop which are used by the initiator to check validity of the query forwarded through the network and to check if the *PreviousHop* is correct by decrypting these fields using *PreviousHop's* public key. The carried *NextHop* field in ACK represents the node which was supposed to receive the forwarded query (i.e. it should be equal to the ACK source in case of non-malicious behavior).

The routing protocol 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 initiator, S, public key and compare it to the un-encrypted copy *m* to check if message is polluted. X compares *m* and the decrypted on $(E_s^+(E_s^-<m>)=m)$. In case the decryption fails or the message was polluted, *X* will send a WARN message (Fig. 6) to S. Otherwise, it will decrypt the *carried* fields using the forwarding node's public key and checks if the encrypted source ID is equal to S' ID (to ensure that the query is really issued by S). After that, it will check if *NextHop* is equal to its own ID to prevent any malicious attack that may appear as a result of a combination attack. In case any check fails, *X* will send a WARN to *S,* else it will forward the query to the next hop and send an ACK to S.

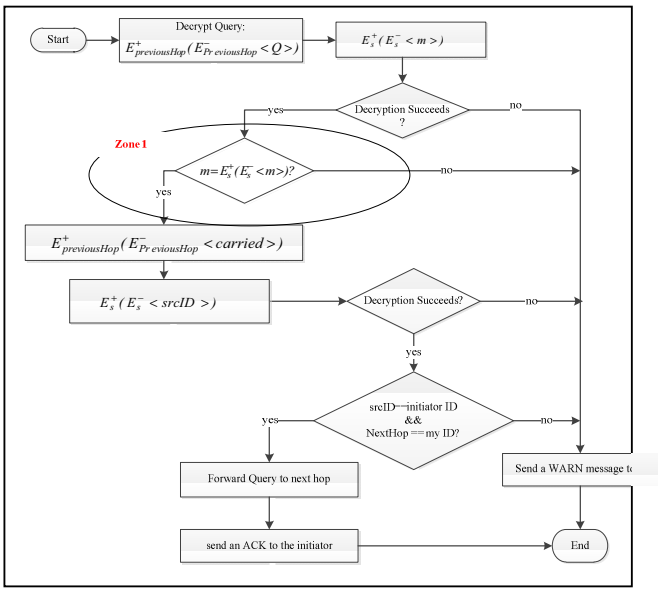


Fig. 4 Routing at Intermediate node.

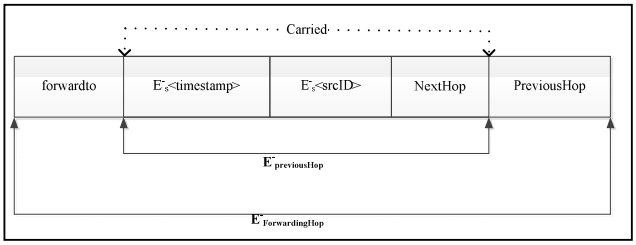


Fig. 5 Ack Format.

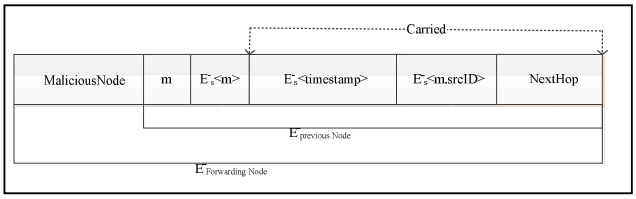


Fig. 6 WARN Format.

Upon receiving an ACK, the initiator S should decrypt the ACK message using the ACK source's public key and then decrypt the *carried* fields using the public key of the *PreviousHop* node (i.e. *PreviousHop* field in the ACK packet). After successful decryption, *S* will check if the ACK source

and the *ExpectedACKSource* (that S maintains) are equal. If they are, then S will set its *ExpectedACKSource* value to *forwardto* value from the ACK packet and will wait for another ACK from *forwardto* after checking the validity of the encrypted *srcID*, *timestamp* and *NextHop*. If the ACK source is not equal to the *ExpectedACKSource* (i.e. malicious behavior) then S will check if the *PreviousHop* (i.e. the node which forwarded the query) is equal to the *PreviousAckSource.* If it is, it will check if the *NextHop* (i.e. the node to which the query was supposed to be forwarded) is equal to the ACK source. If they are equal, then *PreviousHop* (i.e. in this case equal to *PreviousAckSource*) is malicious since it is misleading queries either by reporting a wrong *forwardto* node or by not forwarding the node to the expected ACK source. So S will send a negative evaluation about the *PreviousHop* to the TMS else it will check if *NextHop* is equal to *ExpectedACKSource*. In case *NextHop* and *ExpectedACKSource* are equal, S will send a negative evaluation about the ACK source and *ExpectedACKSource* to the TMS because the ACK source is faking the *PreviousHop* value and the *ExpectedACKSource* received the query and forwarded it without sending any ACK. Otherwise, it will send a negative evaluation about the ACK source to the TMS because the ACK source did not send a WARN message to the initiator telling that the *NextHop* is not equal to its ID. If the *PreviousHop* is not equal to *PreviousACKSource,* S will check if the *PreviousHop* is equal to the *ExpectedACKSource.* If it is the case then S will send a negative evaluation about the *PreviousHop* to the TMS since the *PreviousHop* did not send an ACK to S. Otherwise, it will check if the *NextHop* is equal to the ACK Source ID. If it is not the case then it will send a negative evaluation about the *PreviousHop* (since it didn't send an ACK) and *ACK Source* (since it didn't send a WARN) to the TMS else it will send a negative evaluation about *PreviousHop* only. At the end of each query transmission, the initiator will positively evaluate each participating node in the routing process except the malicious ones which will be evaluated negatively when detected.

D. Potential attack scenarios

To explain how the malicious attacks are detected, we will give an illustration example. Consider Fig. 7 in which Node1 is the initiator and Node3 is the malicious node. When Node1 wants to request some data, it routes the query to Node2 (as shown in Fig. 8). Node2 first decodes the whole query using Node1's public key. It also makes sure that all the following are valid: $E^{+}_{Node1}(E^{-}_{Node1} < m >) = m$, $E^{+}_{Node1}(E^{-}_{Node1} < Node1 >) = Node1$, and $NextHop = Node2$, Node2 will then forward the query to Node3 as shown in Fig. 9 and send back the ACK as shown in Fig. 10 to the initiator. Let's suppose that Node3 is malicious that pollutes the message by changing its content before forwarding it to Node4 and sending an ACK to Node1 as shown in the diagram in Fig. 11. Node4 will receive the message and detect that it is polluted since $E^{+}_{Node1}(E^{-}_{Node1} < m >) != m$; hence, Node4 will send a warning message to Node1. The WARN packet includes the malicious node ID (Node3) and the polluted query so that the initiator makes sure the WARN is correct. Node1 will resend the query and alarm Node2 to exclude Node3 from the routing procedure. It will also send a negative evaluation to the TMS about Node3.

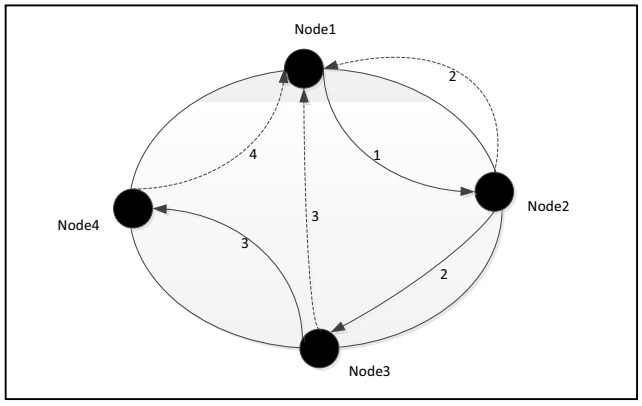


Fig. 7. Routing in our Secured Protocol.

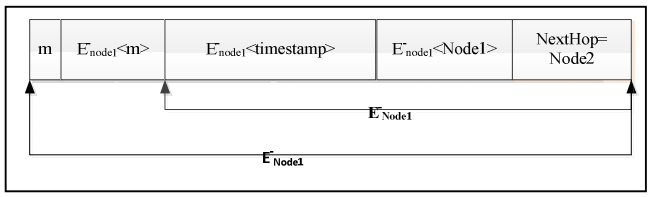


Fig. 8. Query Sent by Node1.

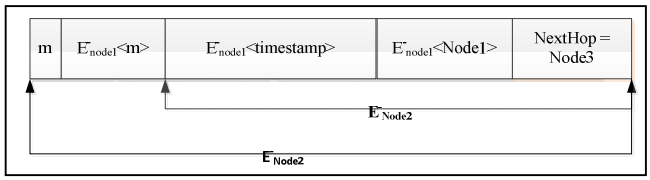


Fig. 9. Query forwarded byNode2.

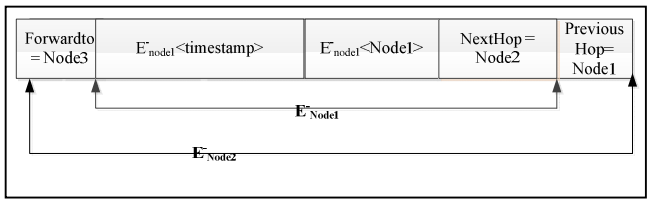


Fig. 10. ACK sent by Node 2.

Similarly, there are variations of many other scenarios such as dropping message scenario where a node receives the message, drops it, and never sends an ACK to source; misleading message scenario where a node receives the query and forwards it to another node (say node 5) but in the ACK message sent to source, it sets the *forwardto* field to a different node (say node 4). As a result, the source will wait for an ACK from node 4. We next evaluate the performance of proposed security management system.

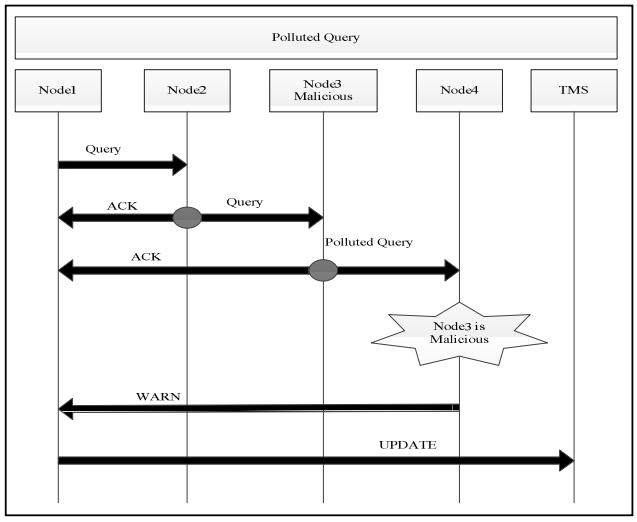


Fig. 11. Exchanged messages in case of Polluted Query.

IV. PERFORMANCE ANALYSIS

In this section we study the performance of our approach. To do so, we have integrated it with both Pastry and Chord (discussed in section II), referred to as trust aware routing protocol (TARP)-Pastry and TARP-Chord respectively. These protocols were implemented using PeerSim simulator [12]. Hence, we refer to Evaluation metrics include:

- Request delivery ratio (RDR) which is defined as the number of successful requests delivered to the destination nodes under the malicious attacks, divided by the total number of queries sent through the network.
- Percentage of malicious node detected by the Trust systems which is defined as the total number of malicious nodes detected by the TMS, divided by the total number of malicious nodes in the system.
- Percentage of false negatives defined as the number of good nodes that were considered malicious by the trust system.

These metrics were measured while varying other parameters such as network size, malicious node percentage and request rates. The simulation parameters are shown in Table 2 below.

TABLE II. SIMULATION PARAMETERS

Parameters	Value
Network Size	100, 600, 1000, 1500
Malicious node percentage	10%, 20%, 30%, 40%
Request rate	1packet/2 seconds, 1 packet/5 seconds
Time	Up to 300000 seconds
T_{flag}	3
threshold Combination for Chord and Pastry	*((Disbelief(x) > 0.5) || (Disbelief(x) > 0.2 && $N_{Negative}$ > 10) || (Disbelief(x) > Belief(x) && $N_{Negative}$ < 10))*

We have used the central limit theorem to calculate the number of runs while achieving 90% confidence level with a precision value of 3%. The number of runs varied from 2 to 6 depending on the measured metric and the used protocol. Therefore, we have performed 6 runs for all scenarios.

The left side of Fig. 12 shows the request delivery ratio with respect to the number of malicious nodes. The figure shows that the proposed approach improves significantly the RDR. We observe that, TARP under pastry (TARP- Pastry) shows approximately an RDR from 99 to 95% while Pastry shows an RDR from 90 to 62%. TARP-Chord shows RDR from 96 to 84%, while Chord shows RDR from 90 to 64%. As a result, when the number of malicious nodes increases, using TARP-Pastry may yield more than 25% improvement in RDR over Pastry while TARP-Chord may yield more than 25% improvement in RDR over Chord. The right side of Fig. 12 shows similar results but with respect to network size. We can easily observe that the proposed TARP improves significantly the RDR (to above 90%) when integrated with Pastry and Chord. Pastry and Chord alone have an RDR that varies from 78% to 68% depending on the network size.

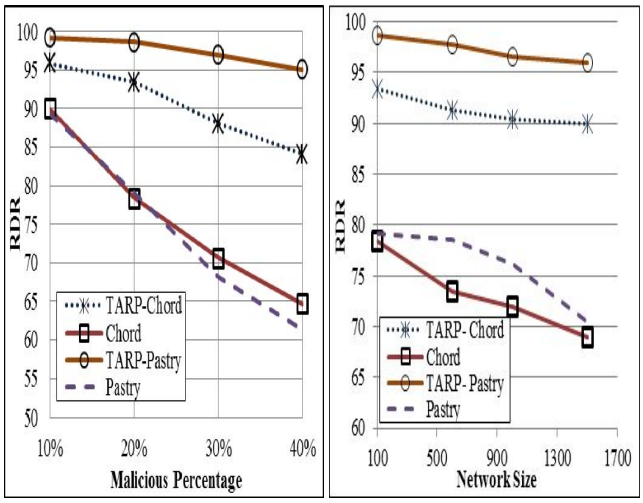


Fig. 12. (left) RDR with respect to the percentage of Malicious nodes in the network, request Rate= 1packet/5seconds, Network Size =100; (Right) RDR with respect to Network Size, request rate = 1 packet/ 5 seconds, Malicious Percentage = 20%

Left side of Fig. 13 shows the percentage of malicious nodes detected in the network. We can observe that our approach was able to detect up to 50% of malicious nodes under Chord regardless of the number of malicious nodes in the network. TARP-Pastry was able to detect 30% of those malicious nodes when 10% of the nodes are malicious and then the percentage of detected nodes increased as the number of malicious nodes increases. It is worth mentioning that the malicious nodes we are testing with do not act maliciously all the time, thus lowering the percentage of malicious detection. The right side of Fig. 13 shows the behavior of the proposed approach when the size of the network increases. We observe that that TARP integrated with Chord maintains a high level of malicious node detection even when the network size increases.

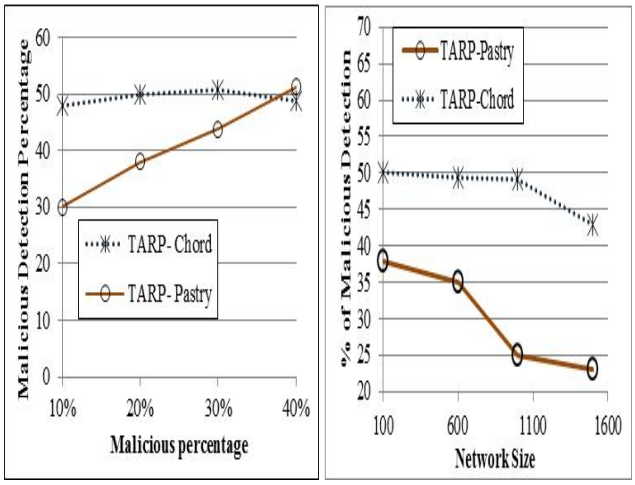


Fig. 13. (left) Malicious nodes detected with respect to the percentage of Malicious nodes in the network. Network size= 100, request rate= 1 packet/ 2seconds, (right) Percentage of Malicious Detection wrt Network Size. Malicious %= 20

Fig. 14 (left) shows the false negative percentage under TARP-Chord and TARP-Pastry with respect to the malicious percentage in the network. It shows that TARP-Pastry has less false negative percentage than TARP-Chord most of the time but when the network size increases TARP-Pastry results in more false negatives than TARP-Chord (right side of the figure). The Figure shows a tradeoff between the percentage of malicious detection and that of false negatives. When the percentage of malicious detection goes down the percentage of false negatives goes up.

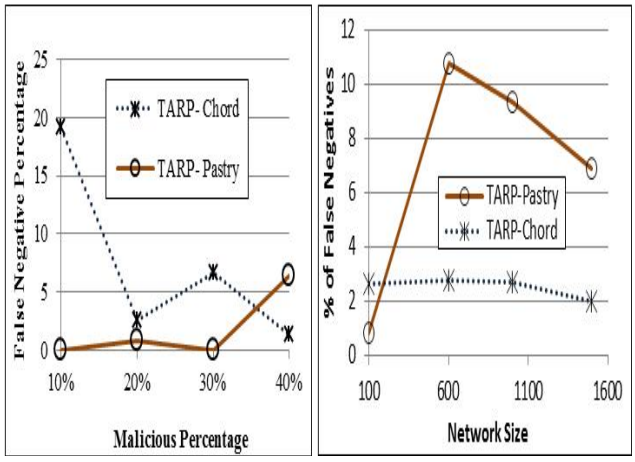


Fig. 14. (left) False negative percentage with respect to the percentage of malicious nodes in the network. Request Rate = 1packet/ 2 seconds and Network Size= 100; (right)Percentage of False Negatives wrt Network Size. Malicious %=20

V. CONCLUSION

In this paper, we have proposed a trust aware routing protocol (TARP) that is able to detect malicious nodes and classify them according to their trust and reputation level. When malicious nodes pollute, drop, redirect messages and launch a denial of service attacks, the node that initiated the request will detect this malicious behavior and report it to its reputation manager system which keeps track of the trust level of all the nodes. The reputation management system also makes sure that if any node provides incorrect feedback then its credibility will be reduced and thus this node's evaluation will not affect other's nodes reputation. We have integrated our proposed approach with both Pastry and Chord and implemented it using the Peersim simulator. To evaluate the performance of the proposed approach we performed several simulation experiments while measuring the request delivery ratio, malicious detection, false negatives and scalability. The proposed approach proved to have better performance than the other protocols.

REFERENCES

[1] 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. 17–32, 2003.

[2] Antony Rowstron and Peter Druschel. "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer system" Appears in Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001). Heidelberg, Germany, November 2001.

[3] Xiang Xu, and Tan Jin. "Efficient Secure Message Routing for Structured Peer-to-Peer Systems." Journal of Convergence Information Technology Volume 5, Number 4, *June 2010*.

[4] C. Harvesf and D. M. Blough. "The Effect of Replica Placement on Routing Robustness in Distributed Hashed Tables". In Peer-to-Peer computing, pages 57-66, 2006.

[5] Salma ktari, Mathieu Zoubert, Artur Hecker, and Houda Labiod. "Performance Evaluation of Replication Strategies in Dhts under Churn". In proceedings of the 6th international conference on Mobile and ubiquitous multimedia, pages 90-97, 2007.

[6] Marc Sanchez Artigas, Pedro Garcia Lopez and Antonio F.Gomez skarmeta. "A Novel Methodology for Constructing Secure Multipath Overlays". Internet Computing, IEEE. Volume 9, issue 6. Pages 50-57. 2005.

[7] Kristen Hildrum and John Kubiatowicz. "Asymptotically efficient approaches to fault tolerance in peer to peer networks", In Distributed Computing, volume 2848of lecture notes in computer science, pages 321-336. 2003

[8] C. Harvesf and D. M. Blough. "The Effect of Replica Placement on Routing Robustness in Distributed Hashed Tables". In Peer-to-Peer computing, pages 57-66, 2006.

[9] Selcuk Ali Aydın, Uzun Ersin, and Resat Pariente Mark "A Reputation-based Trust Management System for P2P Networks". International Journal of Network Security, Vol.6, No.3, PP.235-245, May 2008.

[10] 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, 2011, pp. 222-247

[11] QH Vu, M Lupu, BC Ooi. Springer-Verlag Berlin Heidelberg, Peer-to-Peer Computing: Principles and Applications, ISBN 9783642035135, 2010.

[12] http://peersim.sourceforge.net, last updated October 2013.

[13] C. Plaxton, R. Rajaraman, and A. Richa, "Accessing nearby copies of replicated objects in a distributed environment," in Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures, 1997.

[14] D. Korzun, A. Gurtov, "Hierarchical architectures in structured peer-to-peer overlay networks". *Peer-to-Peer Networking and Applications*, Springer, 2013, 1-37.