

Detecting Malicious Peers in Overlay Multicast Streaming

Samarth Shetty, Patricio Galdames, Wallapak Tavanapong, Ying Cai
Department of Computer Science
Iowa State University
{shetty,patricio,tavanapo,yingcai}@cs.iastate.edu

Abstract

Overlay multicast streaming is built out of loosely coupled end-hosts (peers) that contribute resources to stream media to other peers. Peers, however, can be malicious. They may intentionally wish to disrupt the multicast service or cause confusions to other peers. We propose two new schemes to detect malicious peers in overlay multicast streaming. These schemes compute a level of trust for each peer in the network. Peers with a trust value below a threshold are considered to be malicious. Results from our simulations indicate that the proposed schemes can detect malicious peers with medium to high accuracy, depending on cheating patterns and malicious peer percentages.

1 Introduction

Overlay multicast (also known as Application Layer Multicast) for media streaming has received significant research interest in recent years [2] [3] [11]. Peers form an overlay structure to relay data for one another. The popular overlay multicast structure is a tree where a peer is either an interior node or a leaf node. An interior peer forwards data to its child peer(s). A leaf peer does not forward any data. Hence, the scalability and reliability of overlay multicast streaming depends on the level of contribution and cooperation of peers that are interior nodes. Peers, however, are less trustworthy and less reliable than routers in performing the data forwarding function. Thus, data streamed over an overlay tree is exposed to variety of faults ranging from intentional to unintentional errors. Intentional errors could be carried out by malicious nodes that strive to decrease the effectiveness and usefulness of media streaming applications. Examples of malicious behavior are as follows.

- Peers stop forwarding data or delay forwarding the data to other peers.
- Peers corrupt data before sending them down the overlay tree.
- Peers falsely claim that they did not receive data or received corrupted data.

In this paper, we aim to identify peers exhibiting the aforementioned malicious behavior. We will investigate other types of malicious behavior such as collusion in our future work. Malicious activities, besides generating extra network traffic, causing denial of services, can have social impact. For instance, malicious peers can change multicast content to damage reputation of the original broadcast source or provide wrong evacuation routes in a disaster relief effort, just to name a few.

Detecting the presence of malicious activities alone using existing fault detection techniques [9] [12] [7] is not sufficient since these techniques do not identify peers that are the source of these malicious activities. Effective techniques that locate which peers are malicious peers are needed. Existing reputation management schemes developed for peer-to-peer (P2P) file sharing systems [16] are not directly suitable for estimating goodness of peers in overlay multicast streaming. In P2P file sharing, peers receiving a file assume that the peer sending the file is also the file owner. The reputation manager penalizes the reputation of the sending peer according to feedbacks from the peers receiving the file from this sender. The reputation manager also considers the reputation of the reporting peers to reduce the effect of faulty feedbacks. However, in overlay multicast streaming, most sending peers (except the tree root) are not the owners of the multicast data; they simply forward the data coming from their parent peer. Therefore, in overlay multicast streaming, if a good peer receives corrupted data, its parent need not always be malicious since the malicious peer may be the ancestor peer of the parent peer. Locating malicious peers in overlay multicast streaming is, therefore, non trivial as illustrated below.

Consider a chain of peers in Fig. 1. The arrows indicate the multicast data flowing through the peers. Assume that peers B_1 and B_2 are bad (malicious) whereas peers G_1 , G_2 and G_3 are good (non-malicious).

Case 1: Peer B_1 stops sending data to peer G_2 . Hereafter, we use the case that peers stop sending data to also represent the case of peers sending corrupted data or delaying data forwarding, since all these cases are similar in

terms of peer interactions. All peers (G_2 , G_3 , and B_2) down the chain complain that they did not receive the data. Peer G_3 accuses an innocent peer G_2 . Peer G_2 accuses peer B_1 . The malicious peer B_1 lies that it has already sent the data to G_2 . Peer G_2 cannot prove its innocence to peer G_3 and cannot substantiate its accusation of peer B_1 .

Case 2: Peer B_1 stops sending data to peer G_2 . All downstream peers (G_2 , G_3 , and B_2) complain similarly as in Case 1. However, the malicious peer B_1 accuses its parent (an innocent peer G_1) of not sending the data to it. Peer G_1 cannot prove its innocence.

Case 3: Peer B_2 issues a false complaint that it has not received data to spoil the reputation of peer G_3 . The parent (peer G_3) in this case has no means to prove that it has forwarded the data.

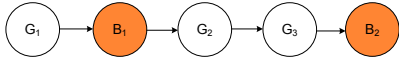


Figure 1. Data flow between peers

To the best of our knowledge, *Trust-Aware-Multicast (TAM)* [8] is the only overlay multicast protocol proposed to detect peers that exhibit uncooperative behavior. TAM assigns a quality value to each peer participating in a multicast session. Peers send negative reports to the root node when encountering undesirable behavior. On receiving a negative report, the reputation manager of TAM penalizes both the reporting peer and its parent since it cannot be sure which peer is malicious. The negative reports and quality values of peers are used to determine the trustworthiness of the peers. The distrust threshold is dynamically adjusted based on system load. In addition, peers with a high quality value are told to move up the tree whereas peers with a lower quality value are relocated down the tree. In TAM, innocent peers (e.g., peers G_1 and G_3) have no mechanism to prove their innocence.

Our contributions in this paper are as follows. First, we introduce two novel malicious peer detection schemes. They are 1) *Signed Acknowledgment (SA)* scheme and 2) *Random Monitoring (RM)* scheme. SA is suitable for TCP-friendly streaming protocols or application-level protocols that use acknowledgment messages to adjust streaming bandwidth based on traffic conditions. The signed acknowledgment messages are used by peers to prove its innocence when they are falsely accused. We have implemented SA with TCP on RedHat Linux. RM, on the contrary, does not rely on acknowledgments, but uses trusted peers to randomly monitor (police) peers in a multicast session. These trusted peers can be provided as a national infrastructure like distributed caches in NLNR caching project [1]. Both proposed schemes rely on an existing fault detection technique. We also propose a new reputation computation

scheme in this paper and compare the effectiveness of SA and RM through simulations.

The rest of the paper is organized as follows. We present our malicious peer detection schemes in detail in Section 2. In Section 3, we show simulation results that demonstrate the effectiveness of the schemes. We present related work in Section 4. Finally, we offer some discussions and concluding remarks in Sections 5 and 6, respectively.

2 Proposed Schemes for Malicious Peer Detection

This section presents the details of the signed acknowledgment (SA) scheme and the random monitoring (RM) scheme. These schemes work under the assumption of no peer collusion. This assumption is also used in [8]. We assume a trust manager that maintains trust-related data structures and computes goodness of each peer participating in the multicast session. The trust manager decides whether a peer is considered malicious or not. Once a malicious peer has been identified, appropriate actions can be taken depending on the policy of the broadcasting source. Note that the trust manager may be same or different from the broadcasting source.

2.1 Signed Acknowledgment (SA) Scheme

Peers detect faults in the data using an existing fault detection scheme and send complaints¹ to the trust manager. Faults refer to cases where (i) peers do not receive data, (ii) peers receive late arriving data, or (iii) peers receive corrupted data². The trust manager updates the trust value of peers based on complaints received from other peers. A single malicious activity will result several complaints to the trust manager. For instance, a malicious peer located near the top of the tree triggers several complaints from all the peers in its subtree. To narrow down from a series of complaints to a suspicious pair of peers such that one peer in this pair is a true malicious peer, SA utilizes the following *fault-node localization* scheme.

Fault-node Localization Scheme: Consider a chain of peers in the same tree branch. The trust manager picks the topmost complaining peer in the chain and the parent of this peer as a suspicious pair.

Applying the above localization scheme to each of the scenarios discussed in the introduction section of the paper results in the following.

Case 1: Peers G_2 , G_3 , and B_2 complain to the trust manager. The trust manager picks the topmost complaining peer (G_2) and its parent (B_1) as a suspicious pair.

¹Peers complain when the number of detected faults exceeds a threshold.

²To detect peers changing content, additional information is needed to indicate that content has been changed.

Case 2: In this case, malicious peer B_1 sends a false complaint about peer G_1 to the trust manager. The trust manager neglects complaints from peers $G_2, G_3,$ and B_2 since peer B_1 is the topmost complaining peer and identifies peers G_1 and B_1 as a suspicious pair.

Case 3: The trust manager identifies peers G_3 and B_2 as a suspicious pair since B_2 issues a false complaint to the trust manager.

In all the three cases, the localization scheme reduces the chain of peers to a suspicious pair, the trust manager can now start analyzing this pair of peers in order to identify the most probable malicious peer, by using (i) signed acknowledgments and (b) calculation of the trust value of peers. The pseudo code for a peer and for a trust manager are shown in Algorithms 1 and 2, respectively.

Algorithm 1 Client of the SA scheme

```

1: Receive the data from the parent
2: if data is faulty then
3:   /* Data not received or late or corrupted */
4:   if number of faulty packets  $\geq$  threshold then
5:     Complain to the trust manager
6:     number of faulty packets = 0
7:   else
8:     number of faulty packets ++
9:   end if
10: else
11:   Send a signed acknowledgment to the parent
12:   Forward the data to its child peer(s)
13:   if receiving an acknowledgment from its child in a timeout period then
14:     Cache the acknowledgment
15:   else
16:     Resend data until timeout
17:   end if
18: end if

```

Algorithm 2 Trust manager of the SA scheme

```

1: loop
2:   if receiving a complaint then
3:     Perform fault-node localization to identify a suspicious pair
4:     Request signed acknowledgments from the parent of the suspicious pair
5:     Receive the requested acknowledgment from the parent
6:     if correct acknowledgements produced then
7:       Identify the child as malicious and take a necessary action to punish it
8:     else
9:       Update trust values of both peers in the suspicious pair (see Section 2.4)
10:      Identify a peer as malicious for the peer with the trust value below a threshold
11:      Relocate the child peer of the suspicious pair to a new location in the tree
12:    end if
13:  end if
14: end loop

```

A child peer is required to send an acknowledgment signed by its signature to its parent when receiving the data (Algorithm 1 Line 6). The parent peer of this child can present the signed acknowledgment to the trust manager when falsely accused by its malicious child of not sending the data. Each peer caches the signed acknowledgment messages for some time³. We have modified the TCP ac-

³First-in-first-out cache replacement can be used.

knowledge mechanism on RedHat Linux to include the signed acknowledgment to verify that our idea is implementable. When it is not possible for the trust manager to identify which peer of the suspicious pair is malicious based on signed acknowledgments, the trust manager will decrease the trust value of both peers in the suspicious pair (Algorithm 2 Line 9). The details of the trust value calculation is described in Section 2.4. Furthermore, the child along with its subtree will be assigned to a different parent chosen by the trust manager. The relocation of the child is to give a chance for a good child to escape from a bad parent and vice-versa. To ensure jitter free data, peers need to buffer their data in a lookahead buffer. The size of the buffer can be calculated similarly to that of the RM scheme to be discussed shortly.

SA is different from the aforementioned TAM in the following ways. First, unlike TAM, peers in SA have signed acknowledgments to prove their innocence to some degree. Second, SA uses a different method to compute the trust value for each peer. Third, SA moves the child peer (and its subtree) of a suspicious pair to a new parent that is not necessary closer to the leaves of the tree. This new parent chosen at random or to fit a particular requirement of the overlay tree gives an opportunity for a good peer whose trust value has been damaged by a malicious parent to escape.

2.2 Random Monitoring (RM) Scheme

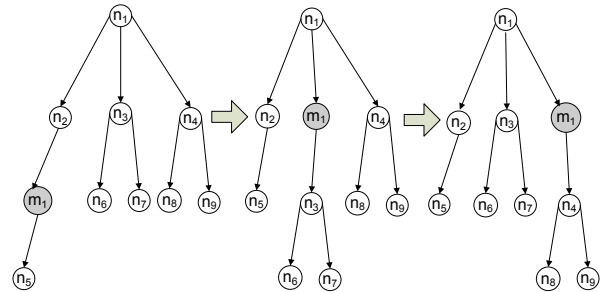


Figure 2. RM: Monitoring peer m_1 samples random peers.

2.2.1 Overview

This scheme uses trusted peers to identify malicious peers in the overlay tree. Each monitoring peer joins the overlay tree at random positions and samples the media forwarded by a peer for a limited period of time. It compares the data forwarded by its parent peer with the original content and based on the result, updates the trust value of the parent. It then moves to another random position in the overlay tree

(Fig. 2) to sample data sent out by another peer. Monitoring peers report to the trust manager results of data sampling. The trust manager collates the reports from all the monitoring peers and updates the trust values of peers.

2.2.2 Protocol for monitoring peers to join and leave

To avoid any jitter that can be introduced due to the periodic join and leave of the monitoring peers, a separate overlay tree consisting of only monitoring peers also referred as alternate overlay tree is constructed (Fig. 3). Both the overlay trees have the same media sent over it. The stream of data being sent over the alternate overlay tree is buffered in a media look-ahead buffer by every monitoring peer. An alternate overlay tree may be an overhead, but in addition of avoiding jitter, it can facilitate fault detection and also minimize the impact of malicious attacks. It could also be used to stream data temporarily to peers whose parents have crashed or have abruptly left the system until that peer finds a new parent. The contents of this media look-ahead buffer is filled with the data sent through the alternate overlay tree. Monitoring peers stream data from this media look-ahead buffer to a child peer whenever it joins the overlay tree.

Consider the following scenario in Fig. 4. Peers n_1 and n_2 are normal peers that are part of the media streaming multicast session. Normal peers may or may not be malicious. Peer n_1 sends the data packet to peer n_2 . Let t_1 be the time needed for a packet to reach peer n_2 from peer n_1 . Let m_1 be a monitoring peer that is introduced between n_1 and n_2 to sample data sent by n_1 such that the time needed for a packet to reach peer m_1 from peer n_1 is t_2 seconds and the time needed for a packet to reach peer n_2 from peer m_1 is t_3 seconds (Fig. 4).

Case 1: $(t_2 + t_3) > t_1$

Peer m_1 introduces a delay that may result in a jitter. To avoid this, the following is proposed.

- Peer m_1 receives data from the server through a separate stream which it uses to check for correctness of data. This stream can be used to prevent the jitter.
- Peer m_1 sends the data that it receives from the server directly to peer n_2 as soon as it joins the network.
- Peer m_1 will sample the contents sent by peer n_1 and drop those packets.
- In order to avoid jitters, the overlay tree consisting of monitoring peers must receive data earlier than the overlay tree consisting of normal peers.
- The upper bound on the buffer that is to be maintained by each monitoring peer m_i is equal to the worst possible delay that can be introduced by adding a monitoring peer. Later in section 2.3 we introduce a model to calculate the size of the buffer at the monitoring peers.

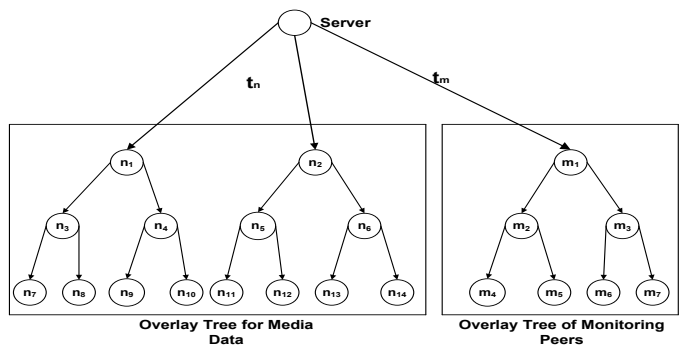


Figure 3. Overlay trees.

Case 2: $(t_2 + t_3) \leq t_1$ Monitoring peer m_1 reduces the time taken for a packet to reach peer n_2 . The monitoring peer forwards the contents it receives from the normal peer or uses the buffered content. When the monitoring peer leaves the network, it is a graceful leave. Because of this, it is possible to ensure that it stops streaming data to peer n_2 only after data from n_1 reaches it.

2.3 Look ahead buffer size for monitoring peers

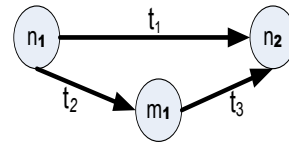


Figure 4. Monitoring peer m_1 joins between peer n_1 and n_2 .

The size of the buffer to be maintained at the monitoring peers can be calculated using the notations as follows:

N :	Set of normal peers n_i where $i = 1, \dots, n$
M :	Set of monitoring peers m_j where $j = 1, \dots, m$
t_n :	Time the first data packet was sent over the overlay tree of normal peers (i.e., the start of the multicast streaming session)
t_m :	Time at which the first data packet was sent over the overlay tree of monitoring peers
d_{m_i} :	Propagation delay from the root of the overlay tree of the monitoring peers to reach m_i
d_{n_j} :	Propagation delay from the root of the overlay tree of the normal peers to reach the n_j
b_{m_i} :	Buffer size at monitoring peer m_i
$D_{(i,j)}$:	Propagation delay between peer i and peer j

At time t , the monitoring peer m_i receives the data

packet with id $[t - t_m - d_{m_i}]$. With a buffer of b_{m_i} , it has data packets with id in the interval

$$[t - t_m - d_{m_i} - b_{m_i}, t - t_m - d_{m_i}] \quad (1)$$

and normal peer n_j has the data packet with id $[t - t_n - d_{n_j}]$. Suppose the monitoring peer m_i samples data from the parent of peer n_j , then to prevent jitter we want peer n_j to receive data from monitoring peer m_i starting from time t . In other words, the first packet from m_i must reach n_j at time t . Given the propagation delay of $D_{(m_i, n_j)}$, monitoring peer m_i must send the first packet at least by time $t - D_{(m_i, n_j)}$. The packet id of this packet can be calculated as

$$id = t - (t_n + d_{n_j}) \quad (2)$$

At time $t - D_{(m_i, n_j)}$, m_i must have the packet with $id = t - (t_n + d_{n_j})$ in its buffer.

Thus, from Equations (1) and (2), we have

$$t - D_{(m_i, n_j)} - t_m - d_{m_i} - b_{m_i} \leq t - (t_n + d_{n_j}) \quad (3)$$

$$t - (t_n + d_{n_j}) \leq t - D_{(m_i, n_j)} - t_m - d_{m_i} \quad (4)$$

Finally,

$$b_{m_i} \geq t_n - t_m + d_{n_j} - d_{m_i} - D_{(m_i, n_j)} \quad (5)$$

To summarize, the buffer size of the monitoring peer is dependent on the propagation delay between the server and monitoring peer, server and normal peer and monitoring peer and the normal peer. This buffer size can be tuned by varying the time at which the media is sent to the normal and the monitoring overlay trees. We have evaluated the buffer size calculation via implementation of a monitoring scheme and experiments on PlanetLab.

2.4 Trust Management

In both SA and RM, the trust manager maintains a trust vector and a trust value for each peer.

- *Trust vector* is a vector of n floating point numbers that captures the n most recent transactions. We say that a transaction occurs when a peer receives a data packet. A negative number represents an unsuccessful transaction whereas a positive number represents a successful transaction.
- *Trust value* is a numeric value that indicates the goodness of a peer.

The trust manager uses a sigmoidal function to map the trust vector to the trust value as follows.

$$Trust\ value = \frac{1}{1 + e^{-x}},$$

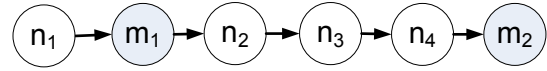


Figure 5. Random Sampling.

where x is the sum of the individual values in the trust vector. With the sigmoidal function, a good peer will not experience a significant drop in its trust value when it is identified as part of a suspicious pair. Similarly, a malicious peer will have to perform several successful transactions to improve its trust value.

SA and RM use the same trust-related data structures and the same sigmoid function to calculate the trust values of peers from the trust vectors. However, these schemes differ in the way they update the trust vectors. For each transaction, an update of a trust vector is done by shifting the values in the trust vector by one position and by assigning a new value that represents the result of the most recent transaction into the trust vector at the most significant position.

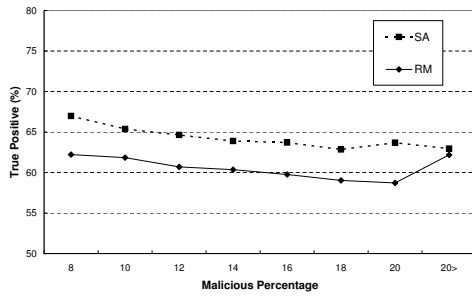
- *Updating a Trust Vector for SA:* On receiving a complaint, the trust manager updates the trust vector with a value which is the negative of the other peer's current trust value. On not receiving a complaint from a peer, the trust manager updates the trust vector with a positive constant.
- *Updating a Trust Vector for RM:* Monitoring peers join the overlay tree at random positions and periodically updates the trust vector of all the peers between itself and another monitoring peer above it in the same tree branch. For instance, in Fig. 5, peers m_1 and m_2 are monitoring peers. Peer m_2 samples the data sent by peer n_4 and periodically updates the trust vector of all peers above it namely n_2 , n_3 and n_4 . On detecting malicious behavior, monitoring peers update the trust vectors of peers with a value of -1. Whereas, it updates the trust vector of peers with a value of 1 on receiving correct data.

3 Performance Evaluation

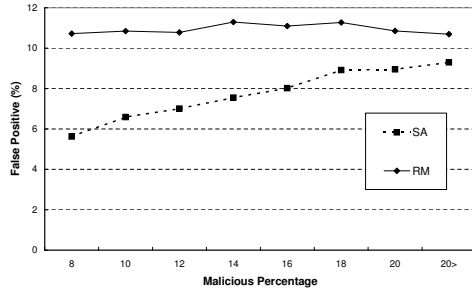
Table 1. Simulation parameters and values.

Parameter	Default	Variation
Number of normal nodes	1000	500-1500
Multicast session length	75	25-75
Malicious percentage	16	8-24
Monitoring percentage	10	N/A

In this section, we discuss the performance comparison results between the SA and RM using a network simulator written in C++. The simulator was designed to generate

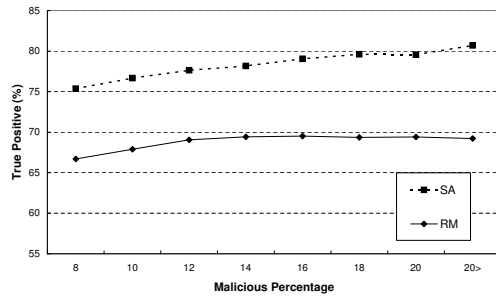


(a) True Positive

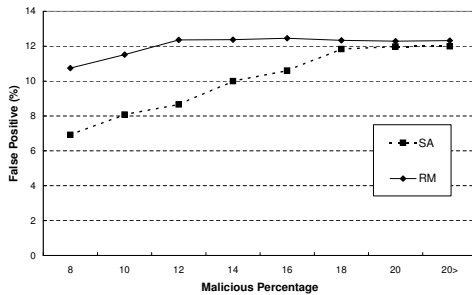


(b) False Positive

Figure 6. True and False Positive for 75% Honesty factor



(a) True Positive



(b) False Positive

Figure 7. True and False Positive for 25% Honesty factor

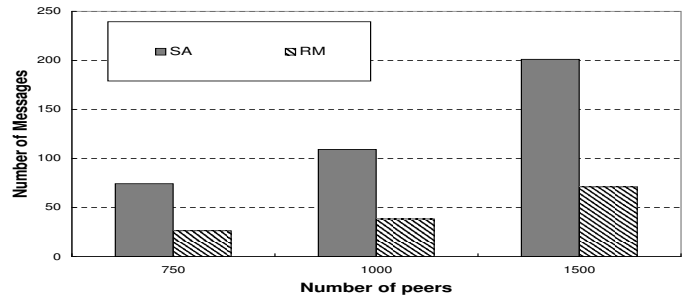


Figure 8. Average number of trust value updates per iteration

overlay networks of different size and topologies, nodes of different behaviors, multicast sessions of different lengths. We compare the success of SA and RM under different scenarios. The success of the schemes was measured based on 1) True Positive: Percentage of malicious peers that are identified as malicious. Given true positive, we can also compute the percentage of malicious peers that escape the detection. 2) False Positive: Percentage of good peers that are falsely accused as malicious. 3) False Negative: $100 - \text{True Positive}$ 4) Message Overhead: Number of messages exchanged. An ideal scheme is one that has the highest true positive value, lowest false positive value and least message overhead. Success of the different schemes was measured by varying the different parameters of the network simulator. We also developed a prototype to demonstrate the feasibility and complexity of implementing digitally signed acknowledgements.

Table 1 summarizes the parameters used in the simulations. The malicious percentage is the percentage of malicious peers to the total number of normal nodes. The monitoring percentage is the percentage of monitoring peers to the total number of normal nodes. Three types of malicious peers were simulated. 1) Naive peers are peers that always cheat. 2) Hypocritical-I peers cheat with a probability governed by Honesty factor. 3) Hypocritical-II peers cheat in the beginning of the session and stop cheating towards the end. For these peers, the Honesty factor controls the time at which the peers start cheating the system. In our simulations, the Honesty factor was set to be either 25%, 50%, or 75%. Honesty factor indicates the percentage of time a malicious peer is non-malicious. For example, if the Honesty factor is 25%, then a Hypocritical-I peer sends packets with correct data with a probability of 0.25. In other words, the peer does not cheat 25% of the time. On the other hand, with the Honesty factor of 25% a Hypocritical-II peer does not cheat during the first 25% of the total number of the packets it sends to its children. After that it cheats for the rest of the session. Malicious peers were simulated to cheat the system by not sending data or by sending corrupted data,

or by issuing false complaints.

For all results, each data point was computed by averaging the results over 25 independent simulation experiments. Simulations were repeated for different values of the parameters as shown in Table 1. The trust value of the peers was scaled to be within the range of [0-2]; peers with their trust value below a threshold were classified to be malicious. Simulations were carried out for values of thresholds ranging from 0.9 to 1.1 with intervals of size 0.01. To compare the schemes, the best threshold for each scheme was selected. The best threshold is the one that yields the lowest value for the *threshold metric*.

$$\text{threshold metric} = \frac{\text{false negative} + \text{false positive}}{2}$$

Note that, we have not compared our two schemes with TAM. This is because TAM adjusts thresholds dynamically based on system load and the description of dynamic threshold adjustment is not clearly discussed in the original paper [8]. Furthermore, true positives were not presented in the paper. Due to space constraints, we present plots of only selected experiments.

3.0.1 Effect of Malicious Percentage

In this study, we fixed the number of normal peers, the number of monitoring peers, and the session length to their default values shown in Table 1 and varied the percentage of malicious peers. Fig. 6 and Fig. 7 show true and false positive values. Malicious peers are only of type Hypocritical-I and Hypocritical-II with equal probability.

Across the range of parameters, the results indicate that SA has higher true positives than RM for different percentages of malicious peers (Fig. 6 and 7). The true positives do not vary significantly for different values of malicious peer percentages. SA also has fewer false positives than RM. However, in SA, the false positives increase as the number of malicious peers in the system increases. A comparison of success with respect to the different values of the Honesty factor indicate both the schemes detect more malicious peers for lesser values of Honesty factor. This is because for lesser values of Honesty factor, peers indulge in more frequent malicious activities and hence can be detected more easily.

3.0.2 Message Overhead

The trust manager updates the trust vector when it receives a feedback from a peer. Figure 8 shows the results obtained for a media session of length 75 with 16% malicious and 10% monitoring peers. In SA, the average number of trust value updates increases significantly with the increase in the number of peers in the multicast session. This is because in the SA scheme all peers starting from the malicious peer

to the peer at the leaf, send negative feedback to the trust manager. As the number of nodes increases the height of the tree increases, causing more peers to send their feedback to the trust manager.

4 Related Work

Several fault detection methods have been proposed to identify faulty data. Byzantine fault tolerance [9] is a form of majority voting but, it is not applicable for large scale distributed systems [12]. One other common way to verify data integrity is to let the server sign every packet or hash of each packet with its private key [14]. A peer caches the packets as well as the signatures and verifies the validity of the data. Reference [6] proposes use of message digests for fault detection. To reduce the overhead incurred in the transmission of digests, the scheme uses a probabilistic approach and generates digests only for a subset of segments. Several similar schemes [13] [15] rely on encryption to detect the presence of corrupted data in the network. Reference [12] suggests the use of bloom filters to detect data corruption. Bloom filter is space efficient data structure that is exchanged with other peers in the network. A mismatch of the bloom filters would identify corruption of data. TAM [8] uses peer feedback and a reputation management scheme to detect malicious behaviors and to identify malicious peers. It associates a level of trust for each peer and adapts the underlying multicast tree according to trustworthiness of peers.

5 Discussion

Trust Agent Infrastructure: We envision that trusted machines can be provided as part of a public infrastructure. It is desirable for a trusted machine to be able to use different uncorrelated entities (IP-addresses) at different times to avoid being detected by malicious peers. This infrastructure can be used in several other applications. For example, trusted peers can be used in P2P file sharing applications to identify peers sharing copyrighted material. *Collusion:* The proposed schemes do not address collusion. This is an assumption that is also used in similar work [8]. In our schemes, we think of collusion attacks as a sequence of independent malicious activities. *Effect of Sybil Attacks:* Forging of multiple identities constitute *Sybil attack* [5]. A malicious peer can assume multiple identities. On being detected as malicious, it can leave the system and rejoin later with a different identity. *Guard against Sybil Attacks:* One way to counter this attack is through the use of free but irreplaceable pseudonyms, e.g., through the assignment of strong identities by a central trusted authority. In the absence of such mechanisms, a penalty can be imposed on all newcomers. The penalty may be in the form of a monetary fee or could be in terms of requiring the peer to pro-

vide computation resources to the network. *Punishment on detection*: The malicious peer identification scheme should be independent of the punishment policy. Malicious peers for example could be 1) Removed from the system 2) Punished monetarily or 3) Provided bad service. Malicious peer identification schemes should be run multiple times in one session. The entire media session can be divided into small sub-sessions and malicious peers can be identified and punished at the end of each sub-session. *Technique for distance measurements*: The buffer size of a monitoring peer is dependent on propagation delays. These delays can be calculated using well known nodes called *landmarks* [10]. Distances between any two peers is the sum of the distance from each peer to its nearest *landmark* and the distance between the two *landmarks*. In monitoring schemes, normal peers inform the server its distance from the *landmarks*. Monitoring peers also calculate their distance from the *landmark* nodes. Using this distance information, delay calculations needed for the buffer space formula can be implemented. Bazzi and Konjevod show that malicious peers cannot cheat the system by pretending to be at different positions [4].

6 Conclusion

Identifying malicious behavior and the malicious peers is important for the success of P2P media streaming systems. Existing approaches concentrate on detecting malicious behavior and not on the problem of identifying malicious peers. In this paper, we have identified different types of possible malicious behaviors and have also proposed, implemented and evaluated two schemes that identifies malicious peers. Results show that SA and RM have similar and high success rates in detecting malicious peers. However, in the RM scheme the number of messages sent to the trust manager to update the corresponding trust vectors is less for large overlay trees. RM also has the advantage that the structure of the overlay tree does not change drastically with every trust value update. This property is significant because the construction of the overlay tree may be optimized on several parameters such as locality of nodes, network traffic, etc. With the SA scheme, the constant movement of subtrees after every negative feedback may result in an overlay tree that is not optimized. This drawback could make SA incompatible with several overlay tree construction protocols. As future work we would like to investigate into the effects of colluding peers and look into schemes that identify them.

References

[1] The National Laboratory for Applied Network Research. <http://www.nlanr.net/>.
 [2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable Application Layer Multicast. In *Proc. of the 2002 Conference on Applications, Technologies, Architectures, and*

Protocols for Computer Communications (SIGCOMM '02), pages 205–217, Pittsburgh, PA, USA, 2002.
 [3] S. Banerjee, S. Lee, R. Braud, B. Bhattacharjee, and A. Srinivasan. Scalable Resilient Media Streaming. In *Proc. of the 14th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '04)*, pages 4–9, Kinsale, County Cork, Ireland, 2004.
 [4] R. A. Bazzi and G. Konjevod. On the Establishment of Distinct Identities in Overlay Networks. In *Proc. of ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC '05)*, pages 312–320, Ottawa, ON, Canada, 2005.
 [5] J. Douceur. The Sybil Attack. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, pages 251–260, Cambridge, MA, USA, 2002.
 [6] R. Gennaro and P. Rohatgi. How to Sign Digital Streams. In *Proc. of the 17th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '97)*, pages 180–197, Santa Barbara, CA, USA, 1997.
 [7] A. Habib, D. Xu, M. Atallah, B. Bhargava, and J. Chuang. Verifying Data Integrity in Peer-to-Peer Media Streaming. In *Proc. of the 12th Annual Multimedia Computing and Networking (MMCN '05)*, pages 1–12, San Jose, CA, USA, 2005.
 [8] S. Jun, M. Ahamad, and J. J. Xu. Robust Information Dissemination in Uncooperative Environments. In *Proc. of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS '05)*, pages 293–302, Columbus, OH, USA, 2005.
 [9] Lamport, Shostak, and Pease. The Byzantine Generals Problem. In *Advances in Ultra-Dependable Distributed Systems, N. Suri, C. J. Walter, and M. M. Hugue (Eds.)*. Washington DC, VA, USA, 1995.
 [10] T. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-based Approaches. In *Proc. of IEEE Conference on Computer Communications (INFOCOM '02)*, pages 170–179, New York, NY, USA, 2002.
 [11] V. N. Padmanabhan, H. J. Wang, and P. A. Chou. Resilient Peer-to-Peer Streaming. In *Proc. of the 11th IEEE International Conference on Network Protocols (ICNP '03)*, page 16, Atlanta, GA, USA, 2003.
 [12] V. Pappas, B. Zhang, A. Terzis, and L. Zhang. Fault-Tolerant Data Delivery for Multicast Overlay Networks. In *Proc. of the 24th International Conference on Distributed Computing Systems (ICDCS '04)*, pages 670–679, Tokyo, Japan, 2004.
 [13] A. Perrig. The BiBa One-Time Signature and Broadcast Authentication Protocol. In *Proc. of 8th ACM Conference on Computer and Communication Security (CCS '01)*, pages 28–37, Philadelphia, PA, USA, 2001.
 [14] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 26(1):96–99, 1983.
 [15] P. Rohatgi. A Compact and Fast Hybrid Signature Scheme for Multicast Packet Authentication. In *Proc. of the 6th ACM Conference on Computer and Communications Security (CCS '99)*, pages 93–100, Singapore, 1999.
 [16] A. A. Selcuk, E. Uzun, and M. R. Pariente. Reputation-Based Trust Management for P2P Networks. In *Proc. of the 2004 IEEE International Symposium on Cluster Computing and the Grid (CCGrid '04)*, pages 251–258, Chicago, IL, USA, 2004.