Effective malicious peers identification schemes for overlay multicast streaming

Samarth Shetty B
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Recommended Citation
https://lib.dr.iastate.edu/rtd/19045

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
Effective malicious peers identification schemes for overlay multicast streaming

by

Samarth M Shetty B

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Wallapak Tavanapong, Major Professor
Johnny S. Wong
Manimaran Govindarasu

Iowa State University
Ames, Iowa
2006

Copyright © Samarth M Shetty B, 2006. All rights reserved.
This is to certify that the master's thesis of

Samarth M Shetty B

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy
DEDICATION

To my parents, those values I will forever cherish.
To my dear brother, those childhood days that will never perish.
TABLE OF CONTENTS

LIST OF TABLES ................................................................. vi
LIST OF FIGURES ................................................................. vii
ACKNOWLEDGEMENTS ............................................................ ix
ABSTRACT ............................................................................... x

CHAPTER 1. INTRODUCTION ......................................................... 1

CHAPTER 2. RELATED WORK ..................................................... 5
  2.1 Approach using incentives for collaboration ......................... 5
  2.2 Fault detection approach ................................................. 6
  2.3 Self surveillance approach .............................................. 9
    2.3.1 Trust Aware Multicast (TAM) Scheme ....................... 10
    2.3.2 Signed Acknowledgment (SA) Scheme ..................... 10

CHAPTER 3. PROPOSED TRUST AGENT MONITORING APPROACH 13
  3.1 Overview ....................................................................... 13
  3.2 Monitoring Schemes ..................................................... 14
    3.2.1 Implementation details of monitoring schemes .......... 17
    3.2.2 Protocol for monitoring peers to join and unjoin ....... 18
    3.2.3 Calculation of look ahead buffer size for monitoring peers 20
  3.3 Trust Management ....................................................... 21

CHAPTER 4. PERFORMANCE EVALUATION ................................. 25
  4.1 Analytical model for RM ............................................... 25
  4.2 Analytical model for SM ............................................... 27
4.3 Simulations .................................................. 29
  4.3.1 Simulation Results .................................. 32
4.4 PlanetLab Experiments ................................. 41
  4.4.1 Experiment Results ................................. 43

CHAPTER 5. DISCUSSION ....................................... 45

CHAPTER 6. SUMMARY .......................................... 47

APPENDIX ADDITIONAL MATERIAL ........................ 49

BIBLIOGRAPHY ................................................... 52
;; LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Notation used in buffer space calculation</td>
<td>20</td>
</tr>
<tr>
<td>4.1</td>
<td>Notation for RM analytical model</td>
<td>27</td>
</tr>
<tr>
<td>4.2</td>
<td>Notation for SM analytical model</td>
<td>28</td>
</tr>
<tr>
<td>4.3</td>
<td>Parameters and their values used in simulations</td>
<td>32</td>
</tr>
<tr>
<td>4.4</td>
<td>PlanetLab experiment setup</td>
<td>42</td>
</tr>
<tr>
<td>4.5</td>
<td>Dropped packets measured through PlanetLab experiments</td>
<td>43</td>
</tr>
<tr>
<td>4.6</td>
<td>Effect on viewer experience. Values indicate jitter that was seen without(wh</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>out) monitoring nodes</td>
<td></td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>----------</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1.1</td>
<td>Data flow between peers</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>Leap-Frog technique</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td>RM: Monitoring peer $m_1$ samples random peers.</td>
<td>14</td>
</tr>
<tr>
<td>3.2</td>
<td>SM: Monitoring peers $m_1, m_2$ and $m_3$ sample alternate normal peers of a branch.</td>
<td>15</td>
</tr>
<tr>
<td>3.3</td>
<td>HM: $m_1, \ldots, m_6$ are static monitoring peers and $m_7, m_8$ are dynamic monitoring peers.</td>
<td>15</td>
</tr>
<tr>
<td>3.4</td>
<td>Overlay trees</td>
<td>20</td>
</tr>
<tr>
<td>3.5</td>
<td>Monitoring peer $m_1$ joins between peer $n_1$ and $n_2$.</td>
<td>20</td>
</tr>
<tr>
<td>3.6</td>
<td>Sigmoidal Metric for trust value</td>
<td>22</td>
</tr>
<tr>
<td>3.7</td>
<td>Random Sampling</td>
<td>23</td>
</tr>
<tr>
<td>3.8</td>
<td>Sweep Sampling</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>True and False Positive for 75% Honesty factor</td>
<td>33</td>
</tr>
<tr>
<td>4.2</td>
<td>True and False Positive for 50% Honesty factor</td>
<td>34</td>
</tr>
<tr>
<td>4.3</td>
<td>True and False Positive for 25% Honesty factor</td>
<td>35</td>
</tr>
<tr>
<td>4.4</td>
<td>True and False Positive for Hypocritical-I mode of cheating</td>
<td>36</td>
</tr>
<tr>
<td>4.5</td>
<td>True and False Positive for Hypocritical-II mode of cheating</td>
<td>37</td>
</tr>
<tr>
<td>4.6</td>
<td>True and False Positive for Naive mode of cheating</td>
<td>38</td>
</tr>
<tr>
<td>4.7</td>
<td>True and False Positive of Monitoring schemes v/s Session length</td>
<td>40</td>
</tr>
<tr>
<td>4.8</td>
<td>Average number of messages per iteration</td>
<td>41</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

I shall forever remain indebted to Dr. Wallapak Tavanapong for guiding me through the evolution of this thesis. She has been an inspiration to me and her technical insight, multidisciplinary knowledge and personality have presented me with precious pearls of wisdom. I am also grateful to Dr. Johnny Wong and Dr. Manimaran Govindarasu for serving on my committee and for placing faith in my work. I thank my dear friends Girish Lingappa, Srivatsan Balasubramanian, Souvik Ray, Patricio Galdames, Minh Tran and Muthuprasanna Muthusrinivasan for sharing their knowledge and time towards discussions surrounding the thesis. I am also grateful to all my colleagues at Philips Software Centre, Bangalore who encouraged and supported me to pursue graduate studies. Last, and by no means the least, I wholeheartedly appreciate the Department of Computer Science at Iowa State University for appointing me as a Teaching Assistant which helped me fund my education.
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 present a signed acknowledgement scheme and three monitoring schemes to identify malicious peers in the overlay network. 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 detect a high percentage of malicious peers with different cheating patterns, malicious peer percentages, network size and topologies. Experiments carried on the PlanetLab indicate no degradation in quality after the implementation of the proposed monitoring schemes.
CHAPTER 1. INTRODUCTION

Overlay multicast (also known as Application Layer Multicast) for media streaming has received significant research interest in recent years (1; 2; 16). Peers form an overlay structure to relay data to 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) while 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:

- Type A: Peers stop or delay forwarding data to other peers.
- Type B: Peers corrupt or change data before sending it down the overlay tree.
- Type C: Peers falsely claim that they did not receive data or received corrupted data.
- Type D: Peers flood the overlay tree by sending a large amount of other data along the tree.
- Type E: Peers simulate periodic failure and recovery, thus disturbing the stability of the tree.

In this thesis, our aim is to identify peers exhibiting Type A, B and C forms of malicious behaviors in order to exclude them from an on-going multicast session. Other types of malicious
behavior including collusion would be an extension to the current work. Malicious activities, besides generating extra network traffic, can cause denial of services and have a social impact. For instance, malicious peers can change the multicast content to damage the reputation of the original broadcasting source or can provide wrong evacuation routes in a disaster relief effort.

Detecting the presence of malicious activities using existing fault detection techniques (7; 8; 10; 12; 17; 18; 21; 25) is not sufficient since they do not identify peers that are the source of these malicious activities. Effective techniques are needed to locate malicious peers in order to apply methods that limit the effect of malicious activities. Existing reputation management schemes developed for peer-to-peer (P2P) file sharing systems (22) are not directly suitable for estimating goodness of peers in overlay multicast streaming. In P2P file sharing systems, peers receiving a file assume that the peer sending the file is 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. Furthermore, a malicious peer may cheat in some branches and may not cheat in other branches of the multicast tree. Locating malicious peers in overlay multicast streaming is, therefore, non-trivial as illustrated below.

Consider the chain of peers in Figure 1. The arrows indicate the multicast data flowing through the peers. Assume that peers B1 and B2 are bad (malicious) whereas peers G1, G2 and G3 are good (non-malicious).

Case 1: Peer B1 stops sending data to peer G2. 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 (G2, G3, and B2) down the chain complain that they did not receive the data. Peer G3 accuses an innocent peer G2. Peer G2 accuses peer B1. The malicious peer B1 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 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 1.1 Data flow between peers](image)

To the best of our knowledge, *Trust-Aware-Multicast (TAM)* (11) and *Signed Acknowledgment (SA)* are the only overlay multicast protocols proposed to identify peers that exhibit uncooperative behavior which includes malicious activities. These schemes belong to a self-surveillance approach in which peers report observed behavior to a trust manager. TAM and SA are similar in that they use trust management schemes to assign a trust value to each peer participating in a multicast session. This trust value is calculated based on the complaints sent by peers to the trust manager when encountering undesirable behavior. Peers whose trust values are below a threshold are considered malicious.

Our contributions in this thesis are as follows. 1) We have proposed three novel malicious peer identification schemes. They are a) *Random Monitoring (RM)* b) *Sweep Monitoring (SM)* and c) *Hybrid Monitoring (HM)*. All these monitoring schemes belong to the trust agent monitoring approach. This approach uses trusted peers to monitor (police) normal peers in a multicast session. Trusted peers can be provided as a national infrastructure like distributed caches in NLANR caching project (NLA). Monitoring peers are trusted peers that join a multicast tree at a particular location (i.e., it becomes the child of some normal peer). Normal
peers are neither monitoring peers nor the broadcasting source. Based on the packet it receives from its parent, a monitoring peer determines the presence of malicious activities using existing fault detection techniques. The trust value of its parent and other related peers are updated. The three proposed schemes represent different ways to determine the sampling locations. Peers whose trust value is below a threshold are considered malicious. 2) We have provided a formula to calculate the buffer space requirements for the monitoring peers. This buffer is maintained to avoid jitter that could be introduced due to monitoring peers. 3) We have designed an analytical model for the monitoring schemes. 4) We have evaluated the effectiveness of the monitoring schemes in identifying malicious peers and have compared them with each other and with our SA scheme through simulations. 5) We have performed experiments on the PlanetLab to validate our buffer space calculations.

The rest of the thesis is organized as follows. We present related work in Chapter 2. We present our proposed work in detail in Chapter 3. In Chapter 4, we present our analytical model and show simulation and experimental results to demonstrate the effectiveness of the proposed schemes. We discuss other related issues in Chapter 5. Finally, we offer some concluding remarks in Chapter 6.
CHAPTER 2. RELATED WORK

One of the primary concerns in using an overlay network for media streaming is correctness of data. Current overlays are not secure; even a small fraction of malicious peers can prevent correct message delivery throughout the overlay (4) and hence data delivered over such a system is potentially exposed to a great number of faults. Limited central management to oversee operations makes it difficult to detect and repair the faults and even more challenging to identify the peers causing them. Existing work in this field can be broadly classified into the following categories.

- Approach using incentives for collaboration
- Fault detection approach
- Self surveillance approach

2.1 Approach using incentives for collaboration

These schemes propose a design of P2P systems where participant peers are given incentives for providing service. A peer is punished in the form of bad service for cheating the system. Ngan et al. (15) implement this by having every peer maintain values of the number of objects sent and number of objects received. The difference of these two numbers expresses the debt or credit of that peer. The debt and credit values are used by peers to discriminate freeloaders from other good peers, allowing them to refuse service to freeloaders. Chu and Zhang (5) define altruism of a peer as the ratio (K) of the bandwidth contributed \(f_i\) to the overlay and the bandwidth received \(r_i\).

\[ r_i = \frac{f_i}{K} \]
A content publisher sets the minimum level of altruism for each peer in the network. Thus an altruism policy defines the bandwidth \( f_i \) peer \( i \) should contribute to receive in return a bandwidth of \( r_i \). If a peer contributes more bandwidth, it is entitled to receive more in return.

Habib and Chuang (9) adopt a score-based incentive mechanism that encourages cooperation through indirect reciprocity. Users choose their contribution level in order to maximize their individual utility. The contribution level \( x_i \) of user \( i \) is converted into a score \( S_i \), which in turn is mapped into a percentile rank \( R_i \), determining the rank of the user among all users in the system. Peer selection depends on the rank ordering of the requestors and candidate suppliers. When a peer first joins the system, it begins with a score of zero and receives best-effort service. If a user wishes to receive better-than-best-effort streaming, it must earn a positive score by contributing to the system. User utility \( U_i \) is a function of the streaming session quality \( Q \) and the contribution cost \( C \) is given by

\[
U_i(x_i) = a_i Q(x_i) - b_i C(x_i)
\]

where \( a_i \) and \( b_i \) define the values of streaming quality and contribution cost to user \( i \).

A rational user will determine its optimal contribution level \( x_i \) (or equivalently, optimal score \( S_i \) or optimal rank \( R_i \)) to maximize its utility.

Incentive schemes provide differential service based on the contribution of the participating peers. They discourage free loaders, however they do not detect data corruption or prevent malicious activities.

### 2.2 Fault detection approach

Fault detection methods are directed towards detecting errors in the data stream. Byzantine fault tolerance (12) is a form of majority voting but it is not applicable for large scale distributed systems (17). One common way to verify data integrity is to let the server sign every packet or hash of each packet with its private key (20). A peer caches the packets as well as the signatures and verifies the validity of the data. Methods proposed in (7; 25; 10; 18; 8; 21) are similar in that they rely on encryption to detect the presence of corrupted data. Gennaro and
Rohatgi (7) present two different schemes, one for the off-line case where the entire stream content is known in advance to the signer and the other for the on-line case where the stream content is generated in real-time. For the off-line case, it suggests embedding in each packet $P_i$ the hash of the next packet $P_{i+1}$. After embedding the hash within the packets the signer signs only the first packet. While this method is elegant and provides for a stream signature, it does not tolerate packet loss. The biggest disadvantage, however, is that the entire stream of packets needs to be known in advance. In the on-line scheme, the sender initially sends a signed public key. Then he sends the first packet along with its signed hash based on the public key sent in the previous step. The first packet also contains a new public key to be used to verify the signature on the second packet. This structure is repeated for all the future packets. Wong and Lam (25) propose to use Merkle's signature trees to sign streams. The key idea is to make asymmetric digital signatures more efficient is to amortize one signature generation and verification over multiple messages. Merkle (13) describes how to construct a hash tree over all messages where the signer only digitally signs the root. The limitation of these schemes are that they are not robust against packet loss. In addition, the one-time signature communication overhead is substantial (17). Habib et al. (10) propose the 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. Pappas et al. (17) suggest the use of bloom filters to detect data corruption. Bloom filter is a space efficient data structure that consists of an array of $m$ bits and a set of $k$ independent hash functions, $H_1, H_2, \ldots, H_k$, with range $[0, m-1]$. Hash values $H(A)_1, H(A)_2, \ldots, H(A)_k$ are computed for the recently received packets. This bloom filter is exchanged with other peers in the network receiving the same content. A mismatch of the bloom filter transmitted and the bloom filter stored would identify corruption of data. Goodrich (8) uses encryption techniques and message digests to verify integrity of packets in flooding algorithms. The work in (8) however differs from those in (7; 25; 10; 18) by its ability to identify malicious peers. It is based on a cryptographic hashing strategy called leap-frog linking. The initial setup involves a simple key distribution. Specifically, for each peer $x$ the set $N(x)$ contains the peers
that are neighbors of x (which does not include x itself). That is,

\[ N(x) = \{ y : (x, y) \in \text{Edge and } y \neq x \} \]

Figure 2.1 Leap-Frog technique

A secret key \( k(x) \) is shared by all peers in \( N(x) \), but not by x itself. The message that is broadcasted to a peer x, is of the form \((s, M, h_1, h_2)\), where s is the sender, M is the message, \( h_1 \) and \( h_2 \) are cryptographic hash values. \( h_1 \) and \( h_2 \) are encrypted with keys shared by \( N(s) \) and \( N(x) \) respectively. \( h_1 \) enables x to authenticate the message from s and \( h_2 \) enables \( N(x) \) to authenticate the message sent by x. Thus any peer s that sends a modified message will be discovered by one of its neighbors \( x \in N(s) \). This technique has a loophole. A malicious peer operating under this scheme can escape identification and can cheat the system by changing the hash of the message. In the above example, malicious peer s can change \( h_2 \) and escape detection from x. This is possible since \( h_2 \) is encrypted with a key shared by \( N(x) \) and because x cannot check the validity of \( h_2 \), it forwards \( h_2 \) to peers in \( N(x) \). However a modified/incorrect \( h_2 \) which is forwarded by x will mismatch with the hash generated from message M causing peers in \( N(x) \) to falsely assume that the message M has been modified. Thus, peers in \( N(x) \) will accuse an innocent peer x.
2.3 Self surveillance approach

In the self surveillance approach, peers collaborate with one another or with a centralized node to identify malicious peers. The centralized node which is commonly referred to as Trust manager or Reputation manager uses peer feedback and reputation management schemes to identify malicious peers.

Peers complain to the trust manager whenever they detect faults in the media. Faults are detected using existing fault detection schemes (see Section 2.2). Faults here include cases where (i) peers do not receive data (ii) peers receive late arriving data or (iii) peers receive corrupted data. The trust manager updates the trust value of peers based on complaints received. In a given tree branch, the trust manager could receive complaints from multiple peers. For instance, a malicious peer located near the top of the tree can trigger several complaints from all the peers in its subtree. The trust manager cannot judge goodness of the peers based on a single positive feedback or a complaint received from peers. Given a pair of peers with a parent-child relationship, either the parent or the child or both the peers could be good as illustrated below:

- **Bad-parent-good-child**: The sender (B₁ in Figure 1) is bad, but the receiver (G₂ in Figure 1) is good.

- **Good-parent-bad-child**: The sender (G₃ in Figure 1) is good, but the receiver (B₂ in Figure 1) is bad.

- **Bad-ancestor**: Both the sender and the receiver (G₂ to G₃ in Figure 1) are good, but at least one of the ancestors (B₁ in Figure 1) is bad.

Currently there are two self surveillance schemes which identify malicious peers in an overlay multicast streaming system.

---

1To detect peers changing content, additional information is needed to indicate that content has been changed.
2.3.1 Trust Aware Multicast (TAM) Scheme

TAM (11) computes a level of trust for each peer in the network and adapts the multicast tree according the trustworthiness of peers. The multicast tree is is adapted in such a way that more trusted nodes are located closer to the root node. According to the behavior a peer manifests in the system, the trust manager updates that peers' quality or trustworthiness. When the trustworthiness falls below a distrust threshold the children of such a peer is relocated. Peers that report any type of problems are also relocated to a new place. The distrust threshold is dynamically adjusted based on system load. However, the paper does not give any information on how the distrust threshold is adjusted based on the load. To improve the quality of the tree, TAM finds pairs of nodes in the multicast tree such that a lower quality peer is located at a position that is suitable for a node with higher quality. Such pairs of nodes are swapped regularly along with their subtree to improve the quality of the tree.

2.3.2 Signed Acknowledgment (SA) Scheme

SA is related work proposed by the author and is suitable for TCP-friendly streaming protocols (24) or application-level protocols that use acknowledgment messages to adjust streaming bandwidth based on traffic conditions. SA uses trust management and signed acknowledgments to identify malicious peers. Signed acknowledgment messages are used by peers to prove their innocence when they are falsely accused of not sending the data or for sending incorrect data. We have implemented a prototype of SA with TCP on RedHat Linux, an overview of the implementation is provided in Section 6.

A malicious peer located at a higher position in the multicast tree will result in complaints from all the peers in the subtree. To narrow down from a series of complaints to a 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: From a chain of complaining peers belonging to the same tree branch, the trust manager picks the topmost complaining peer and its parent 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 G2, G3, and B2 complain to the trust manager. The trust manager picks the topmost complaining peer (G2) and its parent (B1) as a suspicious pair.

- **Case 2:** In this case, malicious peer B1 sends a false complaint about peer G1 to the trust manager. The trust manager neglects complaints from peers G2, G3, and B2 since peer B1 is the topmost complaining peer and identifies peers G1 and B1 as a suspicious pair.

- **Case 3:** The trust manager identifies peers G2 and B2 as a suspicious pair since B2 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 (ii) 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

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Receive the data from the parent</td>
</tr>
<tr>
<td>2</td>
<td>if data is faulty then</td>
</tr>
<tr>
<td>3</td>
<td>/* Data not received or late or corrupted */</td>
</tr>
<tr>
<td>4</td>
<td>Complain to the trust manager</td>
</tr>
<tr>
<td>5</td>
<td>else</td>
</tr>
<tr>
<td>6</td>
<td>Send an signed acknowledgment to the parent</td>
</tr>
<tr>
<td>7</td>
<td>Forward the data to its child peer(s)</td>
</tr>
<tr>
<td>8</td>
<td>if receiving an acknowledgment from its child in a timeout period then</td>
</tr>
<tr>
<td>9</td>
<td>Cache the acknowledgment</td>
</tr>
<tr>
<td>10</td>
<td>else</td>
</tr>
<tr>
<td>11</td>
<td>Resend data until timeout</td>
</tr>
<tr>
<td>12</td>
<td>end if</td>
</tr>
<tr>
<td>13</td>
<td>end if</td>
</tr>
</tbody>
</table>

A child peer is required to send an acknowledgment signed by it 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 time. We have modified the

---

2First-in-first-out cache replacement can be used.
TCP acknowledgment 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 are described in Section 3.3. 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 its data in a look-ahead buffer. The size of the buffer can be calculated similarly to that of the monitoring scheme which is discussed in later sections.
CHAPTER 3. PROPOSED TRUST AGENT MONITORING APPROACH

3.1 Overview

We propose the use of monitoring schemes and trust management to identify malicious peers in an overlay multicast tree. Monitoring schemes use trusted peers to monitor peers during a multicast session and assume the presence of a trust manager to maintain trust related data structures. Trusted peers rely on existing fault detection techniques to detect faulty data and can be provided as a national infrastructure like distributed caches of the NLANR caching project (NLA).

A trusted peer, also referred to as monitoring peer, joins the overlay tree at different positions, samples the media forwarded by its parent peer for a limited period of time and checks the data for correctness. This can be done efficiently by comparing the hash of the data being streamed with the hash obtained from the original content. Depending on the result, the monitoring peer updates the trust value of the parent peer. After sampling the peer for a limited period of time, the monitoring peer moves to another position in the overlay tree to sample data sent out by another peer. Normal peers have limited bandwidth. This fixed bandwidth limits the number of children a peer can support in the overlay tree. Monitoring peers thus have to join the tree by replacing an existing child of a peer. This replaced peer is then made the child of the monitoring peer.

A trust manager is assigned with the job of trust management. Monitoring peers report the results of data sampling to the trust manager. The trust manager collates reports from all monitoring peers and calculates the trust value/goodness of peers participating in the multicast session. The trust manager also decides if a peer is to be considered as malicious or not. In
this thesis, a peer is considered malicious if its trust value is below a specified trust threshold. 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 as or different from the broadcasting source.

### 3.2 Monitoring Schemes

Ideally we would like every normal peer to be monitored by a monitoring peer at all times during the session. This assumption is impractical and also not scalable for large overlay trees. Thus we assume only few monitoring peers to be available for monitoring during the session. Let \( \{m_1, m_2, m_3, \ldots, m_k\} \) denote the monitoring peers and \( \{n_1, n_2, n_3, \ldots, n_l\} \) denote the normal peers in the overlay multicast tree. We propose the following three monitoring schemes to identify the malicious peers in the multicast tree.

**Random Monitoring (RM):** Monitoring peers are loosely coupled and join the network at random positions (Figure 3.2). Monitoring peers sample the data for a certain period of time before moving to another randomly picked position in the tree. When a monitoring peer comes across bad data it reports it to the trust manager.
Figure 3.2  SM: Monitoring peers $m_1, m_2$ and $m_3$ sample alternate normal peers of a branch.

Figure 3.3  HM: $m_1, \ldots, m_6$ are static monitoring peers and $m_7, m_8$ are dynamic monitoring peers.
• **Sweep Monitoring (SM):** A monitoring peer operating under the RM scheme on receiving bad data cannot accuse its parent peer as malicious. This is because the bad data could have originated from any ancestor of its parent. Ex: In Figure 3.2, peer m₁ on receiving bad data from n₂ cannot be sure if the source of bad data is n₂ or n₁. SM is designed to overcome this drawback. In SM, monitoring peers on receiving bad data can conclusively blame its immediate parent to be malicious. This is accomplished by making the monitoring peers alternate with normal peers to form a chain of peers. This chain of peers starts with the root of the tree or with a monitoring peer and ends with a normal peer (Figure 3.2). This chain corresponds to a branch in the overlay tree. The monitoring peers in the chain sample the data for some time before shifting to a different branch in the tree. The chain of monitoring peers sweeps through all the branches of the tree by moving in a way such that peers at the lower levels move first followed by the peers above it. Algorithms 3 and 4 describe the way in which monitors in the SM scheme sweep through all the branches of the tree.

• **Hybrid Monitoring (HM):** This is a combination of random and sweep monitoring schemes. The motivation behind the HM scheme is to minimize network instability that can be caused due to frequent join and unjoin of monitoring peers. Monitoring peers in HM are of two types: 1) Static: These peers do not change their positions during media streaming. They join the network at fixed positions and occupy the upper levels of the overlay tree. 2) Dynamic: These peers implement random sampling and sweep sampling interchangeably and they change their positions to sample different peers at different intervals. Dynamic peers sample peers that are at the lower levels of the tree. Malicious peer identification involves two stages. In the first stage, dynamic peers perform random sampling and in the second stage they perform sweep sampling. In the first stage, sampling of data results in identifying regions of the tree that are more probable to have malicious peers. This localization is done by picking branches of the tree between two monitoring peers that show malicious activities beyond a malicious activity threshold. In the second stage, dynamic monitoring peers regroup and sweep the regions that have
been identified as malicious in the earlier stage (Figure 3.2). After the sweep operation, dynamic peers continue to implement random sampling at the lower levels of the tree till another malicious region is identified. The advantage of this method is that static peers at the upper levels of the tree provide more stability since they do not change positions. Also since the dynamic peers are at the lower levels of the tree only few peers are affected during the join and unjoin operations.

Algorithm 3 Sweep Monitoring::ChangePosition()

1: if nextMonitor->ChangePosition() then
2: /*nextMonitor represents the downstream monitor*/
3: /* Downstream Monitor has changed its position*/
4: return true
5: else
6: if Connected to the lastChild of parent then
7: /* Cannot change position of this monitor. Upstream monitor needs to change position */
8: return false
9: else
10: /* Change position of this monitor*/
11: newChild = NextChildofParent(); /*Gets the next child of this monitors parent */
12: currentChild = this->child; /*Gets the current child of the monitor */
13: SetChild(newChild,this) /*Set the monitor as parent of newChild*/
14: SetChild(currentChild,this->parent) /*Set currentchild as the child of the monitor's parent*/
15: /* Set new parents on all monitors down the tree*/
16: nextMonitor->Join(this)
17: return true
18: end if
19: end if

Algorithm 4 Sweep Monitoring::Join()

1: /*Gets the first child of the upstream monitor */
2: newParent = parentMonitor->firstChild;
3: /*Set this monitor as child of newParent*/
4: SetChild(this,newParent)
5: nextMonitor->Join(this)

3.2.1 Implementation details of monitoring schemes

In this section we look into the implementation details of the proposed monitoring schemes. In RM, the server assigns to each monitoring peer, a set consisting of normal peers that needs to be monitored. Monitoring peers select peers from this set for sampling. These sets are non-overlapping and are updated periodically by the server whenever a peer joins or leaves the network. Each update will require a message to be sent by the server to the monitoring peer. In SM, the number of monitoring peers in each chain of peers is determined by the
height of the overlay tree. Monitoring peers in the system are divided into different groups by the server. Each group has enough number of members to sweep even the longest branch of the tree. For example, if there are 9 monitoring peers and the height of the tree is 4, then the monitoring peers are arranged into 2 groups of 4 peers each. The remaining 1 monitoring peer remains unused. Each group is then randomly assigned a branch of the tree. After sampling the assigned branch, the group of monitoring peers picks the next branch using the Algorithms 3 and 4. Monitoring peers communicate with other members of its group during a sweep operation. After sampling a branch, if a monitoring peer cannot change its position/parent (ChangePosition() returns false), it informs the monitoring peer above it in the chain to execute the ChangePosition() function. If the monitoring peer changes its position then it informs other monitoring peers down the chain (step 4 of the Join()) to select new parents. In HM, dynamic peers are associated with a set of normal peers and also are divided into groups. Dynamic peers select peers from the set for sampling. When a branch crosses a malicious activity threshold, the server instructs the dynamic monitoring peers of a group to sweep the malicious branch.

3.2.2 Protocol for monitoring peers to join and unjoin

Jitter can be introduced due to the periodic join and leave of the monitoring peers. To avoid any jitter, a separate overlay tree consisting of only monitoring peers also referred as alternate overlay tree is constructed (Figure 3.2.3). Both overlay trees have the same media sent over them. 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 to avoiding jitter, it can facilitate fault detection and also minimize the impact of malicious attacks. It can 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 content 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: 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 frame 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 frame to reach peer $m_1$ from peer $n_1$ is $t_2$ seconds and the time needed for a frame to reach peer $n_2$ from peer $m_1$ is $t_3$ seconds (Figure 3.2.3).

**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 be sent 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 $m_1$ is equal to the worst possible delay that can be introduced by adding a monitoring peer. Later, in section 3.2.3 we introduce a model to calculate the size of the buffer at the monitoring peers.

**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 can use the buffered content.

When the monitoring peer leaves the network, it is a graceful exit. As a result of this, it is possible to ensure that it stops streaming data to peer $n_2$ only after data from $n_1$ reaches it.
Table 3.1 Notation used in buffer space calculation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N = {n_1, n_2, n_3, \ldots, n_l}$</td>
<td>Set of normal nodes</td>
</tr>
<tr>
<td>$M = {m_1, m_2, m_3, \ldots, m_k}$</td>
<td>Set of monitoring nodes</td>
</tr>
<tr>
<td>$t_n$</td>
<td>Time the first data packet was sent over the overlay tree of normal nodes (i.e., the start of the multicast streaming session).</td>
</tr>
<tr>
<td>$t_m$</td>
<td>Time at which the first data packet was sent over the overlay tree of monitoring nodes</td>
</tr>
<tr>
<td>$d_{mi}$</td>
<td>Propagation delay from the root of the overlay tree of the monitoring nodes to reach $m_i$</td>
</tr>
<tr>
<td>$d_{nj}$</td>
<td>Propagation delay from the root of the overlay tree of the normal peers to reach the $n_j$</td>
</tr>
<tr>
<td>$b_{mi}$</td>
<td>Buffer size at $m_i$</td>
</tr>
<tr>
<td>$PD_{(i,j)}$</td>
<td>Propagation delay between node $i$ and $j$</td>
</tr>
</tbody>
</table>

3.2.3 Calculation of look ahead buffer size for monitoring peers

The size of the buffer to be maintained at the monitoring nodes for comparison and to prevent jitter can be calculated as follows:

At time $t$, the monitoring peer $m_i$ has data packets with id's in the interval:

- $m_i = [t - t_m - d_{mi} - b_{mi}, t - t_m - d_{mi}]$
and normal peer \( n_j \) has the data packet with id

\[ n_j = [t - t_n - d_{n_j}] \]

Suppose the monitoring node \( m_i \) samples data from the parent of node \( n_j \), then to prevent jitter we want node \( n_j \) to receive data from monitoring node \( m_i \) 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 \( PD(m_i, n_j) \), monitoring node \( m_i \) must send the first packet at least by time \( t - PD(m_i, n_j) \).

- The packet id of this packet can be calculated as
  \[ id = t - (t_n + d_{n_j}) \]

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

Thus,

\[ t - PD(m_i, n_j) - t_m - d_{m_i} - b_{m_i} \leq t - (t_n + d_{n_j}) \leq t - PD(m_i, n_j) - t_m - d_{m_i} \]

Finally,

\[ b_{m_i} \geq t_n - t_m + d_{n_j} - d_{m_i} - PD(m_i, n_j) \]

To summarize, the buffer size of the monitoring node is dependent on the propagation delay between the server and monitoring node, server and normal node and monitoring node and the normal node. This buffer size can be tuned by varying the time at which the media is sent to the normal and the monitoring overlay trees.

### 3.3 Trust Management

The trust management system comprises of a trust manager that calculates the trust value/goodness of the peers in the multicast session. It maintains the following trust related data structures:
- **Trust vector**: It 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**: It is a numeric value that indicates the goodness of a peer.

The trust manager uses a *sigmoidal function* (see Figure 3.3) to map the trust vector to the trust value as follows:

\[
\text{Trust value} = \frac{1}{1 + e^{-x}}
\]

where "\( x \)" is the sum of 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 receives a bad feedback. Similarly, a malicious peer will have to perform several successful transactions to improve its trust value.

![Figure 3.6 Sigmoidal Metric for trust value](image)
All monitoring schemes and our previous signed acknowledgement (SA) based technique (Section 2.3.2) use the same trust-related data structures and the same sigmoidal function to calculate the trust values 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 to the right and by inserting the value representing the result of the most recent transaction at the left most position.

![Random Sampling](image1)

Figure 3.7 Random Sampling.

![Sweep Sampling](image2)

Figure 3.8 Sweep Sampling.

- **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 Monitoring schemes**: Monitoring peers join the overlay tree at different positions and periodically update the trust vector of all the peers between itself and another monitoring peer above it along the same tree branch. For instance, in Figure 3.3, peers \( m_1 \) and \( m_2 \) are monitoring peers operating under the random sampling scheme. 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 \( n_4 \). In sweep monitoring (Figure 3.3), since monitoring peers alternate with normal peers, monitoring peers update the trust
vector of only its immediate parent. Monitoring peers update the trust vectors of peers with a value of -1 on detecting malicious behavior. Whereas, it updates the trust vector of peers with a value of +1 on receiving correct data.
CHAPTER 4. PERFORMANCE EVALUATION

In this section, we propose an analytical model for RM and SM schemes. The RM analytical model provides an upper bound for the speed of detecting malicious peers in the system. The SM analytical model provides an upper and lower bound on the true positive values. We also discuss about our simulations, experiments, their configuration and the results gathered. We compare the proposed schemes using a simulator. Through experiments run over the PlanetLab, we look into the effects of the monitoring schemes on the quality of the streamed media.

4.1 Analytical model for RM

Due to the forwarding nature of streaming systems, monitoring peers on receiving corrupted data cannot blame their immediate parent of being malicious (except in the case of sweep monitoring). Because of this limitation we use trust management schemes along with monitoring schemes to evaluate the goodness of peers. An ideal scenario would be one, where a random monitoring peer can judge the goodness of its parent based only on the data it receives from its parent. In other words, if a monitoring peer receives corrupted data, then the parent peer is considered to be bad otherwise the parent peer is good. Under such an ideal scenario we would like to analytically evaluate the performance of the random monitoring scheme. Since in reality such a scheme cannot exist we can consider this analytical model to be give an upper bound for the random monitoring scheme.

The monitoring scheme can be modeled as a Queuing system as follows:

- Monitoring peers are assumed not to sample peers that have already been sampled.
- Each monitor can be considered as a server of the queuing system.
• Each data sampling corresponds to a job.

• In queuing theory, a server operates on a job for a specific amount of time. In the monitoring scheme, a monitoring node samples a peer for specific amount of time. Hence the correspondence between the monitoring scheme and a queuing system. A M/D/1 model represents exponential distributed inter-arrival times, constant service times and a single server. Peers joining the overlay tree are assumed to follow a Poisson process. A monitoring node samples data for constant time. Thus, a monitoring node along with a queue of peers can be viewed as a M/D/1 queue. A monitoring scheme with "n" monitoring nodes can be viewed as a system consisting of n copies of the M/D/1 queue. With "n" M/D/1 queues in the system, an incoming peer is assigned to one of the M/D/1 queues with a probability of \( \frac{1}{n} \). Monitoring nodes (servers) pick a peer from their respective queues to sample data.\(^1\)

• Given the arrival rate of peers (\( \lambda \)), service time (\( \mu \)) and the number of monitors (\( n \)), we can find the mean waiting time of the peers in the system.

• Waiting time signifies the amount of time a peer can be in the system without being sampled.

• Peers arriving at each M/D/1 queue follow a Poisson process with an arrival rate that is \( \frac{1}{n} \) times the arrival rate of the peers in the system.

For a M/D/1 system with arrival rate of \( \lambda \) and service time of \( \mu \), queuing theory specifies the average waiting time as shown below:

\[
\rho = \frac{\lambda}{\mu}
\]

\[
R = \frac{\lambda}{2\mu^2}
\]

\(^1\)The monitoring scheme can also be reduced to a M/D/N queue. In this case, all the incoming peers enter a single queue and monitoring nodes pick peers for sampling from one common queue. The waiting time for such a model has been calculated in (23).
Table 4.1 Notation for RM analytical model

| $\rho$ | System utilization |
| $\lambda$ | Average number of jobs arriving in one unit of time |
| $\mu$ | Number of jobs the facility is capable of servicing in one unit of time |
| $R$ | Mean residual service time |
| $W_q$ | Expected time a job must wait in the queue before being serviced |
| $W$ | Expected time of a job in the system |

\[
W_q = \frac{R}{1 - \rho} = \frac{\rho}{2\mu(1 - \rho)}
\]

\[
W = W_q + \frac{1}{\mu}
\]

Expected service time “$W$” for the monitoring scheme corresponds to the average time a malicious peer can be in the system before being detected by a monitoring peer. If “$W$” is small, a peer is quickly sampled. Hence, this metric indicates the speed at which a malicious peer can be detected using a monitoring scheme. Given the expected arrival rate and the service time, this model can also be used to determine the number of monitoring peers needed to guarantee sampling of a peer within a specified time.

4.2 Analytical model for SM

To calculate the true positive value for SM we assume a binary overlay tree with uniformly distributed malicious peers.

Each time the data is sampled, a chain of monitoring peers covers at least one node of the tree. When $nsampled = t$, the chain of monitoring peers samples a subtree with $t$ leaf nodes. Thus a lower bound on the number of nodes sampled during a sweep ($n_{low}$) is equal to $t$ and the upper bound ($n_{up}$) on the number of nodes sampled is equal to the maximum number of nodes present in the subtree.

When $nsampled = t$, the minimum and maximum number of malicious peers sampled is given
Table 4.2 Notation for SM analytical model

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n)</td>
<td>Number of normal peers in the network</td>
</tr>
<tr>
<td>(n_{\text{mon}})</td>
<td>Number of monitoring peers</td>
</tr>
<tr>
<td>(\text{perm}_{\text{al}})</td>
<td>Percentage of malicious peers</td>
</tr>
<tr>
<td>(\text{min}_{\text{Smpl}}(t))</td>
<td>Minimum number of malicious peers sampled in time (t).</td>
</tr>
<tr>
<td>(\text{max}_{\text{Smpl}}(t))</td>
<td>Maximum number of malicious peers sampled in time (t).</td>
</tr>
<tr>
<td>(p_{\text{cheat}})</td>
<td>Probability of cheat (1-Honesty Factor)</td>
</tr>
<tr>
<td>(TP_{\text{low}}(TP_{\text{up}}))</td>
<td>Lower(Upper) bound on the percentage of malicious peers detected</td>
</tr>
<tr>
<td>(n_{\text{samplered}})</td>
<td>Number of times the normal peer is sampled</td>
</tr>
<tr>
<td>(n_{\text{sweep}})</td>
<td>Number of parallel sweeps</td>
</tr>
<tr>
<td>(h_t)</td>
<td>Height of the overlay tree</td>
</tr>
<tr>
<td>(\epsilon_{\text{max}}(\epsilon_{\text{min}}))</td>
<td>Maximum(Minimum) number of children per peer.</td>
</tr>
</tbody>
</table>

by,

\[
\text{min}_{\text{Smpl}}(t) = \frac{\text{n}_{\text{low}} \times \text{perm}_{\text{al}}}{100} \\
\text{max}_{\text{Smpl}}(t) = \frac{\text{n}_{\text{up}} \times \text{perm}_{\text{al}}}{100}
\]

where \(\text{n}_{\text{low}} = t\), and \(\text{n}_{\text{up}} = \frac{1 - \left(\frac{\text{min}_{\text{Smpl}}}{\text{perm}_{\text{al}} \times n}\right) \log(t)}{1 - \left(\frac{\text{max}_{\text{Smpl}}}{\text{perm}_{\text{al}} \times n}\right)}\).

We now look into the upper and lower bounds of true positive values under different cheating behaviors.

- **Naive cheating behavior**: Since malicious peers cheat all the time, a monitoring peer can identify the malicious peer every time it samples it. Hence the percentage of malicious peers sampled is equivalent to the true positive value.

  When \(n_{\text{mon}} = h_t - 1\),

  For \(0 \leq t \leq \frac{n}{2}\),

  \[
  TP_{\text{low}} \geq \frac{\text{min}_{\text{Smpl}}(t)}{\text{perm}_{\text{al}} \times n} \times 100 \\
  \geq \frac{\text{n}_{\text{low}} \times 100}{n}
  \]

  \[
  TP_{\text{up}} \leq 100 \leq \frac{\text{max}_{\text{Smpl}}(t)}{\text{perm}_{\text{al}} \times n} \times 100 \\
  \leq \frac{\text{n}_{\text{up}} \times 100}{n}
  \]
For \( t > \frac{n}{2} \), \( TP_{low} = 100 \).

When \( n_{mon} > ht - 1 \), we can have multiple chains of peers sweeping the overlay tree.

\[ nsweep = \left\lfloor \frac{n_{mon}}{ht - 1} \right\rfloor \]

Thus, for \( 0 < t < \frac{n}{2} \),

\[ TP_{low} \geq \frac{n_{low} \times 100}{n} \times nsweep \quad (4.1) \]
\[ TP_{up} \leq 100 \leq \frac{n_{up} \times 100}{n} \times nsweep \quad (4.2) \]

- **Hypocritical-I cheating behavior:** Consider a malicious peer with \( e \) children and a cheat probability of \( p_{cheat} \). On sampling this peer, the probability of detecting it as a malicious peer is \( e \times p_{cheat} \). We use this concept to calculate lower and upper bounds of the true positive values.

When \( n_{mon} = ht - 1 \),

\[ TP_{low} \geq \frac{n_{low} \times 100}{n} \times e_{min} \times p_{cheat} \]
\[ TP_{up} \leq 100 \leq \frac{n_{up} \times 100}{n} \times e_{max} \times p_{cheat} \]

When \( n_{mon} > ht - 1 \),

\[ TP_{low} \geq \frac{n_{low} \times 100}{n} \times e_{min} \times p_{cheat} \times nsweep \quad (4.3) \]
\[ TP_{up} \leq 100 \leq \frac{n_{up} \times 100}{n} \times e_{max} \times p_{cheat} \times nsweep \quad (4.4) \]

- **Hypocritical-II cheating behavior:** Malicious peers of this type do not cheat initially and hence in the beginning are not identified by the sweep monitors. However when they start cheating they have a naive cheating behavior and hence Equation (4.1) and (4.2) can be used to calculate the true positive values.

### 4.3 Simulations

The performance of the monitoring schemes and SA were evaluated using a network simulator written in C++. The simulator is a discrete event simulator and was designed to generate
overlay networks with different tree structures, nodes of different behaviors, multicast sessions of different lengths and monitors operating under different schemes.

The aim of the simulations was to calculate and compare the success of the SA scheme and the proposed monitoring schemes (RM, SM, HM) under different scenarios. Recall that SA is the Signed Acknowledgment scheme (Section 2.3.2), RM, SM and HM are Random Monitoring, Sweep Monitoring and Hybrid Monitoring respectively (Section 3.2). The success of the schemes was measured based on 1) True Positive: Percentage of malicious peers that are correctly detected as malicious. 2) False Positive: Percentage of good peers that are falsely accused as malicious. 3) False Negative: Percentage of malicious peers that have escaped detection. False Negative = 100 - True Positive and 4) Message Overhead: Number of messages exchanged. An ideal scheme is one that has the highest true positive value, lowest false negative value and least message overhead.

Success of the different schemes was measured by varying the different parameters of the network simulator namely,

- Number of nodes: It indicates the number of normal peers (malicious and non-malicious) in the network. Its value was varied from 500 to 1500.

- Malicious peer types: Three types of malicious peers are simulated in the streaming system.
  - Naive: Peers that always cheat.
  - Hypocritical-I: Peers who cheat with certain probability, governed by Honesty factor.
  - Hypocritical-II: Peers who cheat in the beginning of the session and stop cheating towards the end. The Honesty factor controls the time at which the peers start cheating the system.

- Honesty factor: The cheating pattern of Hypocritical peers is governed by this metric. 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 Honesty factor is 25%, then a Hypocritical-I peer sends correct packets with a probability
of 0.25. On the other hand, with a 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 a Hypocritical-II peer cheats for the rest of the session.

- Malicious percentage: It indicates the percentage of malicious peers in the network. Its value was varied from 8% to 24%.

- Session length: It specifies the length of the media session. The value of the Session length parameter was varied from 25 units to 75 units. For SA, a unit is defined as the time after which a peer sends a feedback to the trust manager. For monitoring schemes, a unit is defined as the time after which a monitoring peer needs to change its position in the network. In the simulator, “unit” refers to one data packet and is same for both SA and the monitoring schemes.

- Malicious peer behavior: Malicious peers were simulated to cheat the system by not sending data or by sending corrupted data, or by issuing false complaints. When a malicious peer decides to cheat it randomly picks one of these three behaviors.

For all results in this section each data point was computed by averaging the results over 25 independent simulation experiments. We used networks of different size and topologies in our simulations. 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. For the HM scheme, the ratio of the static to the dynamic monitoring peers was set to 6:4.

Figures 4.3.1.1 to 4.3.1.1 represent true and false positive values when Hypocritical-I and II peers were present in the system with equal probability. Figures 4.3.1.1, 4.3.1.1 and 4.3.1.1 represent true and false positive values when only Hypocritical-I, Hypocritical-II and Naive peers respectively were present in the system. To compare the schemes, we selected the best threshold for each scheme. The best threshold is the one that yields the lowest value for the threshold metric which is defined as the weighted sum of equally weighted false negative and
false positive values. In other words, the results shown here are the best case results wherein false negative and false positive values are equally important.

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

4.3.1 Simulation Results

In the following sections, we analyze the effect of cheating behaviors, multicast session lengths and percentage of malicious peers on the success of the proposed schemes.

4.3.1.1 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 4.3. We varied the percentage of malicious peers and the results are shown in Figures 4.3.1.1 to 4.3.1.1. Simulations were repeated for different values of the parameters. Column “Variation” in Table 4.3 indicate the range of values used in the simulations and the column “Steps” indicate the incremental value at each iteration.

Results indicate that SM has higher values of true positives and significantly lower values of false negatives than any other schemes. SA performs better than RM, however HM performs better than SA. True Positives for SA, SM and RM schemes remain constant for increasing values of malicious peers. However HM has lower success with increasing values of malicious peers. This is because an increase in the number of malicious peers will cause more frequent sweep operations in a suspicious subtree resulting in the decrease in the number of randomly monitoring peers in other subtrees.
A comparison of success of different schemes with respect to the Honesty factor indicate that all schemes detect more malicious peers for lesser values of Honesty factor. Hypocritical peers that are less honest cheat more frequently, thus the probability of a monitoring peer sampling it while it is cheating increases.

Figure 4.1 True and False Positive for 75% Honesty factor

SM has the least false positive value. SA outperforms RM. HM has lesser false positive values than RM and SA. The false positive values are constant in monitoring schemes for increasing percentages of malicious peers. False positives value increases with the increase in malicious peers for SA.
Figure 4.2  True and False Positive for 50% Honesty factor
Figure 4.3 True and False Positive for 25% Honesty factor
Figure 4.4 True and False Positive for Hypocritical-I mode of cheating
Figure 4.5 True and False Positive for Hypocritical-II mode of cheating
Figure 4.6 True and False Positive for Naive mode of cheating
A comparison of different monitoring schemes indicate that SM is a better scheme than HM and RM in terms of higher true positive and lower false positive values.

4.3.1.2 Effect of Multicast Session Length

The performance of the proposed schemes was checked for different session lengths. For these experiments, we simulated a network of 750 nodes and introduced 16% of malicious nodes. The media session length was set to 25, 50 and 75 to represent short, intermediate and long media sessions respectively. As shown in Figure 4.3.1.2, SM has lower success for shorter sessions. This is because for short sessions the sweep would not have covered sampling most of the peers. It can also be seen that HM has similar performance to RM for very short sessions. This is because the malicious regions would not have crossed the threshold, thus a sweep of malicious regions would not have been triggered. As the session length increases, the performance of the HM and SM increases, whereas the performance of RM remains constant. For short and intermediate length sessions, HM is a better scheme than SM and RM. It has a higher true positive value than SM. Also, HM has fewer monitoring peers changing positions than RM and hence it is preferred over RM.

4.3.1.3 Message Overhead

The trust manager updates the trust vector when it receives a feedback from a peer. Normal peers in SA and monitoring peers in RM, SM, and HM send feedback to the trust manager whenever they receive bad data. On not receiving a feedback, the trust manager updates the trust vector with a positive value. Monitoring peers in the SM and HM schemes also need to exchange messages when they change parents during the sweep operation.

The message overhead of the proposed schemes was calculated as the sum of number of messages exchanged between monitoring peers and the trust value update messages sent to the trust manager. Figure 4.3.1.3 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 nodes in the network. This
Figure 4.7 True and False Positive of Monitoring schemes v/s Session length
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.

4.4 PlanetLab Experiments

Monitoring peers join and leave the network at frequent intervals. To avoid any jitter that can be introduced due to this periodic join and leave we proposed in Section 3.2.3, a formula to calculate the buffer size for the monitoring peers. To verify the effectiveness of a buffer in avoiding jitter, we conducted experiments over PlanetLab (pla) (19).

The setup involved PlanetLab nodes as well as two local machines that act as server and receiver for the data. Data is streamed in the server using a VLC player (vlc) which sends out data to a PlanetLab node. PlanetLab nodes correspond to interior nodes of the overlay tree and are configured to multicast data. They route data to other PlanetLab nodes. The second local machine acts as a receiver and corresponds to the leaf node of the overlay tree. It has a VLC player listening at a specific port which plays the media being streamed.

Monitoring nodes are PlanetLab nodes that are configured to join the overlay network at random intervals and at random positions. They maintain a buffer to prevent jitter and
To calculate the buffer size, it is required that the network latencies between the nodes of the overlay tree be known. Network latency is calculated by executing a "ping" before every experiment. The maximum value of time delay returned by the "ping" program is used in the buffer size calculations.

Experiments were conducted under three different configurations. Each configuration has five machines that represent the interior nodes of a branch in the overlay multicast tree (Table 4.4):

- **Local Network**: All the machines that take part in the experiment are within the LAN. This configuration does not use PlanetLab nodes.
- **Intra-US**: The PlanetLab nodes are distributed across United States. The selected nodes are spread along the east and west coasts of the country.
- **Inter-Continent**: The PlanetLab nodes are distributed across the world. Selected nodes are spread out in Asia, Europe and America.

The motivation of the experiment was to validate the buffer space calculations. A single
monitoring node joining the overlay network at different positions and at different instants of
time will enable us to check for the correctness of the calculations.

4.4.1 Experiment Results

Table 4.5 Dropped packets measured through PlanetLab experiments

<table>
<thead>
<tr>
<th>Media Name</th>
<th>Bit Rate(kbps)</th>
<th>Maximum Buffer Size</th>
<th>Number of Dropped Packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVSEQ14.mpg</td>
<td>1421</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>Carnatic.mp3</td>
<td>128</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>AVSEQ14.mpg</td>
<td>1421</td>
<td>61</td>
<td>4</td>
</tr>
<tr>
<td>Carnatic.mp3</td>
<td>128</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.6 Effect on viewer experience. Values indicate jitter that was seen without(with) monitoring nodes

<table>
<thead>
<tr>
<th>Media Name</th>
<th>Media Type</th>
<th>Media Length(secs)</th>
<th>Local Network</th>
<th>Intra-US</th>
<th>Inter-Continent</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVSEQ14.mpg</td>
<td>Music Video</td>
<td>118</td>
<td>0(0)</td>
<td>8(8)</td>
<td>21(24)</td>
</tr>
<tr>
<td>Carnatic.mp3</td>
<td>Audio File</td>
<td>252</td>
<td>0(0)</td>
<td>2(2)</td>
<td>3(3)</td>
</tr>
</tbody>
</table>

We streamed a video file (AVSEQ14.mpg) and an audio file (Carnatic.mp3) in our experi-
ments. From the experiments it was found that each node in the overlay tree adds a delay of
0.029 secs. This processing delay was calculated using the Local Network configuration and it
corresponds to a delay of 3.17(0.5) packets for the video(audio) stream. The calculated values
of buffer size had to be updated to accommodate this processing delay. The new value for
the size of the buffer was calculated by taking into consideration both the processing and the
propagation delays as shown below.

\[ b_m \geq t_n - t_m + d_{n_j} - d_m - PD_{(m_i,m_j)} + \text{Difference in Number of Hops} \times 0.029 \]

where, "Difference in Number of Hops" is the difference in the number of nodes from the
server to node \( n_j \) along the normal overlay tree and along the monitoring overlay tree respec-
tively.
In the Intra-US experiment setup, the maximum buffer size according to our calculations was found to be 15(2) packets for the video(audio) stream. Similarly, in the Inter-Continent setup the maximum buffer size was found to be 61(8) packets for the video(audio) stream. To check the validity of buffer space calculations, monitoring nodes during the experiment maintained a buffer whose size was set according the calculated values. Monitoring nodes were simulated to join each interior node twice during the experiments. The experiment was repeated 5 times for each setup. The results displayed in Table 4.5 and Table 4.6 is the average value of all the trials.

As seen in Table 4.5, introducing monitoring nodes with a buffer did not result in any lost packets in the Intra-US setup. Number of dropped packets in the Inter-Continent setup was 0 for the audio file and 4 for the video file. We presume the cause of dropped packets to be the increase in network delays. During the experiments, we observed that certain nodes have a bandwidth cap that would result in dropped packets. In our experiment setup, we have selected only those nodes that did not have this bandwidth cap. Also the network delays varied depending on the time of the day. The results correspond to the experiments that were conducted from 21:00 to 23:00 hours.

We have also studied the end user experience under this experimental setup. We counted the number of times a user experiences a jitter while listening or watching the streaming media. What exactly constitutes a jitter may vary for different users, but in our experiments, we considered discontinuity, noise or garbled images as jitter. The high jitter value seen in Table 4.6 is a limitation of the application. The application does not control or manage the inter-packet delays. For the video file, the inter-packet delay is very small, and any variance in inter-packet arrival rates causes jitter.

As seen in Table 4.6, introducing monitoring nodes does not degrade the viewing experience for the Intra-US setup. However in the Inter-Continent setup, we see an increase in the jitter for the video file while monitoring nodes are present. This jitter could be because of the dropped packets as explained earlier.
CHAPTER 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 a common assumption and can also be seen in similar works (11). In our schemes, we can think of collusion attacks as a sequence of independent malicious activities. For example, malicious peers can cheat in a sequential manner such that at a given time only one malicious peer is active. To a monitoring scheme this attack is equivalent to Hypocritical-I peers cheating with a probability of $\frac{1}{n}$. Malicious peers can also collude by exchanging lists of IP-addresses of peers that have joined and left them in the current session. The intersection of these lists could point to the IP-addresses of the monitoring peers. This attack is mitigated by the Trust Agent Infrastructure where each monitoring peer has a dynamic IP-address picked from a large address space of uncorrelated entries. Effect of Sybil Attacks: Forging of multiple identities constitute Sybil attack (6). Reputation systems are vulnerable to these attacks. A malicious peer can assume multiple identities. It can act malicious with one identity and continue to receive service by acting non-malicious with another identity. 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 provide 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* (14). 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 (3).
Identifying malicious behavior and 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 thesis, we have identified different types of possible malicious behaviors. We have proposed and implemented 1) A Signed Acknowledgement (SA) scheme and 2) Three monitoring schemes belonging to the trust agent monitoring approach. These schemes were evaluated, compared and analyzed using a discrete event simulator written in C++. Monitoring nodes in the trust agent monitoring approach need to maintain a buffer to avoid jitter. We have proposed a formula to calculate the size of this buffer. This model was validated by conducting experiments using the PlanetLab infrastructure.

Simulation results show that SM and HM schemes are more successful than the SA scheme in detecting malicious peers. Success is quantified in terms of high true positive and low false positive values. For long sessions, SM has a higher true positive and lower false positive value than the HM scheme. HM on the other hand has higher true positive values for sessions of short and intermediate lengths. HM scheme also has fewer monitoring peers changing positions at the higher levels of the tree than in the SM scheme. This results in a more stable overlay tree. In general, monitoring schemes have an advantage over SA scheme in that the structure of the overlay tree does not change drastically with every trust value update. On the other hand, in the SA scheme there is a constant movement of peers after every negative feedback. This difference is significant because the construction of the overlay tree may be optimized on several parameters such as locality of nodes, network traffic etc. Constant changes in the overlay structure may result in an overlay tree that is not optimal.
Experiments conducted on PlanetLab validate the buffer calculation model. Results show that packets are not lost whenever monitoring nodes join the tree. Analysis of the end user experience indicate that there is no degradation in the video and audio quality because of monitoring nodes.

This is the first work that concentrates on identifying malicious peers. We hope that this work triggers many more interesting problems and novel solutions, e.g., locality aware trusted peers. As future work, effects of colluding peers and schemes that are able to identify them will be considered.
TCP Implementation in Linux

Flow of Input TCP Packet

When a packet is received by the NIC, it is put into kernel memory by the card DMA engine. The available data area must be large enough to hold the maximum size of packet that a particular interface can receive. The interrupt handler creates the packet descriptor (struct sk_buff). During subsequent processing, the packet data remains at the same location. The packet is manipulated through the pointer struct sk_buff.

General Data Structures

- sk_buff: TCP needs to store a large set of variables. It maintains data and status of both incoming and outgoing packets. In the networking code, virtually every function is invoked with a sk_buff.

![Figure A.1 sk_buff structure](image-url)
Description of sk_buff structure

- **list** points to the queue where the socket buffer is currently located.
- **sk** points to the socket that created the packet.
- **stamp** specifies the time when the packet arrived in the Linux (in jiffies)
- **dev** states the current network device on which the socket buffer operates. Once the routing decision is made, dev points to the network adapter on which the packet leaves
- **len** specifies the length of a packet.
- **data**, **tail** point to currently valid packet data.
- **head**, **end** point to the total location that can be used for packet data.
- The space between head and data is called **headroom**, and the space between tail and end is called **tailroom**.

Description of the procedures of the sk_buff structure

- **Alloc_skb**: Allocate a network buffer. The returned buffer has no headroom and a tail room of size bytes. The object has a reference count of one. The return is the buffer. On a failure the return is NULL. Buffers may only be allocated from interrupts using a gfp_mask of GFP_ATOMIC.
- **Skb_reserve**: Increase the headroom of an empty &sk_buff by reducing the tail room. This is only allowed for an empty buffer.
- **Skb_put**: This function extends the used data area of the buffer. If this would exceed the total buffer size the kernel will panic. A pointer to the first byte of the extra data is returned.

- **sock**: It maintains the state of a connection. It keeps data about a specific TCP connection (e.g., TCP state) or virtual UDP connection.
- **tcp_opt**: It is part of the sock structure and is used to maintain the TCP connection state. Both IP and UDP are stateless protocols with a minimum need to store information about their connections.

### Prototype Implementation of SA

#### Overview

We developed a prototype\(^1\) to demonstrate the feasibility and complexity of implementing the SA scheme. The implementation was done in Linux 2.6.13.4 kernel and was tested by implementing a simple TCP socket program. We will give an overview of the important data structures related to the TCP/IP stack and details of code changes needed to implement SA.

#### SA Implementation

The files that handle most of the processing of tcp packets are `tcp_input.c` and `tcp_output.c`. The functions that we were relevant for SA were `tcp_ack()` and `tcp_sendack()`. The former creates an ack packet and prepares it for transmission while the latter receives and processes the ack packet. The following code was added to the `tcp_ack()` function of the `tcp_input.c` file. An analogous change was done in the `tcp_output.c` file.

#### Algorithm 5 Code snippet from the SA prototype

```c
1: struct sk_buff *skb;
2: tmp = 2 * TCP_MAX_HEADER;
3: skb = alloc_skb(tmp, GFP_KERNEL);
4: skb_reserve(skb, TCP_MAX_HEADER);
5: data = skb_put(skb, TCP_MAX_HEADER);
6: skb->csum = csum_and_copy_from_user(from, data, copy, len, &err);
7: tcp_send_skb(skb, skb, queue_it);
```

\(^1\)The prototype was implemented along with Patricio Galdames.
BIBLIOGRAPHY


Symposium on Operating System Design and Implementation (OSDI '02), pages 299–314, Boston, Massachusetts, USA. ACM Press.


