

Robust Information Dissemination in Uncooperative Environments*

Seung Jun Mustaque Ahamad Jun (Jim) Xu

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

{jun, mustaq, jx}@cc.gatech.edu

Abstract

The open nature of peer-to-peer systems has played an important role in their growing popularity. The current file-sharing applications, for instance, have been widely used largely because they allow anyone to participate in them. This openness, however, brings up new issues because selfish, malicious, faulty, compromised, or resource-constrained peers may degrade a system. We explore the case for large-scale information dissemination through the design of the Trust-Aware Multicast (TAM) protocol. Nodes in TAM can exhibit uncooperative behavior such as delaying, discarding, modifying, replaying, and fabricating messages. While detecting such behaviors, TAM computes a level of trust for each node and adapts the underlying multicast tree according to trustworthiness of nodes, which leads to performance improvement in the system. The results from our simulation and PlanetLab experiments show that even with a significant portion of nodes being uncooperative, TAM is able to build a stable dissemination tree that provides lower message delay to well-behaved nodes.

1 Introduction

As the Internet becomes more pervasive, new applications will allow users to participate in common activities even when the users do not trust each other completely [10, 7]. This observation is already evident in the peer-to-peer environment. For example, the users of file-sharing systems such as Kazaa and Gnutella exchange files without knowing or caring who they are dealing with. This grassroots openness has brought great success to these applications because users make minimal assumptions about other users in the system. Indeed, we believe that future applications will increasingly require interactions among users who have varying degrees of trustworthiness.

The open nature of a peer-to-peer system, while it can attract users more easily, has generated new issues, one being that the system opens its doors to both good peers and bad peers. Nodes controlled by some users can be resource-constrained and may not be able to execute the agreed-upon protocols, or worse, they can be faulty, selfish, malicious, or compromised. We use the term “uncooperative” to refer to such undesirable behavior. Uncooperative nodes can degrade the system’s ability to meet application needs. In file-sharing systems, some users can inject malware or bogus files disguised as good ones. Peers come and go easily, and many of them are selfish [2]. Even when a certain fraction of user nodes exhibit uncooperative behavior, the system must be designed to work properly.

In this paper, we explore the case for large-scale information dissemination using application-level multicast, which is challenging because end hosts are more liable to be uncooperative than network routers. To address the challenge, we consider a multicast tree in which nodes are mutually suspicious except the root node, which is trusted by all other nodes. In this setting, some nodes will follow the protocol faithfully while others may be uncooperative for aforementioned reasons. As a result of uncooperative behavior, for instance, messages can be illicitly modified or blocked on the way. In this paper, we address the following question: *how can we provide efficient and reliable information dissemination to well-behaved nodes when the messages are relayed via possibly uncooperative nodes?*

To address this question, we propose Trust-Aware Multicast (TAM). TAM is designed to deliver data in a reliable and timely manner, even in the presence of uncooperative nodes in the system. We achieve this goal by detecting uncooperative behavior, evaluating nodes based on their behavior history, and adapting the multicast tree in such a way that more trusted nodes are located closer to the root node. In this way, the system becomes more robust over time. The property of TAM not only improves robustness but also facilitates deployment because the system admits any user

*This paper was supported in part by NSF grant ITR-0121643.

without establishing a trust relationship beforehand. We emphasize the necessity of trust awareness as a design principle for emerging overlay and peer-to-peer applications.

The rest of the paper is organized as follows. Section 2 explains the assumptions we make in this work and its scope. Section 3 proposes our TAM protocol, which enables the system to detect uncooperative behavior. Section 4 discusses how to evaluate the trust levels for nodes based on the behaviors of nodes. Section 5 discusses what constitutes a good multicast tree in our context and how such a tree can be built. Section 6 shows the evaluation results from 95-node PlanetLab experiments and 10,000-node simulation experiments. We show related work in Section 7 and conclude in Section 8.

2 Assumptions

The threat model of TAM does not restrict the behavior of application nodes; they can modify, fabricate, delay, block, replay, or do anything to the messages that they are relaying. In fact, Section 3 shows how to detect such behaviors. On the other hand, TAM does not directly handle impersonation or Sybil attacks [14].

We focus on data integrity and availability while we do not address data confidentiality. After all, TAM is designed to disseminate freely available data as broadly as possible rather than deliver sensitive data to only selected nodes.

As we build an overlay system, we assume that the underlying network is sufficiently robust to support it. We do not attempt to directly address network failures such as packet loss and network partitioning. Nevertheless, TAM is still able to handle such network failures in an indirect manner as they will affect the trustworthiness of the relevant nodes. For example, if a node is unreachable from the rest of the system because of physical link failure, it will eventually be recognized as untrustworthy. We focus on the trustworthiness of application nodes in this work.

3 Trust-Aware Multicast Protocol

TAM is a single-source multicast system working at application level. As each node participating in TAM runs as an application process, links between nodes are virtual (e.g., TCP connections) as opposed to physical. Thus, links can be added or deleted easily. A multicast group consists of one *root node* and many *user nodes*. The root node is the source of disseminated information and takes control of the group. Although the root node can be replicated internally to avoid the single point of failure, it looks as if there exists only one root from the view point of a user node. Later in this section, we discuss how to expand the JOIN protocol so as to have multiple root nodes. All user nodes are function-

ally equivalent. They may cooperate fully or behave in a manner that is inconsistent with the protocol.

3.1 Message Delivery

The unit of delivery in TAM is a *message*. As is the case with the layered architecture, the interpretation of TAM messages depends on the upper layer application that consumes the messages. We deliberately separate the transport mechanism from the message semantics and focus only on the former in this paper. Our concern is to ensure that each user node receives messages from the moment it joins a group until it leaves the group.

Uncooperative nodes can modify, fabricate, replay, block, and delay messages. To detect such uncooperative behaviors, we construct a TAM message as a tuple of four fields: sequence number, timeout period, payload (service data unit), and cryptographic signature. The first field, the *sequence number*, is used to order messages and detect duplicate or missing messages, which can result when a node has an uncooperative parent or when it is moved to a different position in the tree while message transmission is in progress. The second field, the *timeout period*, specifies a timeout value for the next message; if a node does not receive any message within the specified amount of time, it suspects that a message may be delayed or lost. The root transmits a *null message* if it does not have new data available until the timeout. The third field, the *payload*, contains actual data and optionally meta-data that specifies how the data should be demultiplexed or interpreted. The fourth field is the *cryptographic signature* of the message. The signature is generated by a standard signature algorithm such as the Digital Signature Standard (DSS) [21]. We assume that the public key of the root node is publicly available to the user nodes.

The message, constructed as above, allows nodes to detect misbehavior of other nodes. Illicitly modified or fabricated messages are detected by the message signature. Replayed messages are detected by the sequence number, which is incremented for every message so that every message is unique and not reusable. Note that even null messages increment the sequence number. We assume that the sequence number does not wrap around (by making the field sufficiently large or concatenating it with a timestamp). Delayed or blocked messages are detected by the timeout field.

The timeout value should be chosen carefully. If it is too long, the system may become unresponsive. If it is too short, null messages may waste network bandwidth and other resources. Holbrook et al. discuss a similar issue [16]. In many cases, timeout values can be reasonably estimated. For example, when data are collected from some sensors periodically, the timeout value can be set to that period. As another example, if the root node has large amount of data

that will be sent in several messages (e.g., bulk transmission), it can determine when to send the next message and set the timeout value appropriately. If it is hard to control the timeout value, however, we can use an adaptive scheme in which a timeout value is doubled each time a null message is sent consecutively until the value exceeds a predefined limit.

Since message delivery is subject to jitter, the difference in message delays, due to the fluctuation of network and end host conditions, an additional amount of time, referred to as *timeout slack*, should be allowed before a timeout occurs. The timeout slack can be implemented on either side; either the root node sends messages earlier than the timeout, or the user nodes wait longer than the timeout values. In any case, a straightforward implementation could be vulnerable to a subtle attack. Suppose the timeout slack is set to one second. A malicious node could accumulate these slacks over, say, 1000 messages, and delay the last message by no less than 1000 seconds from the time it was sent from the root without violating the protocol. We address this vulnerability by making each user node keep a variable that accumulates the slacks over the messages while making the root node adhere to the timeout value. A slack may have a negative value when a message arrives earlier than scheduled in the timeout field of the previous message. If the slack-accumulating variable for a node tends to increase over time, it implies that the node undergoes the attack or that the condition of the path is worsening. While TAM does not require clock synchronization among nodes, it assumes that clock skew (first derivative of clock offset with respect to true time [22]) is bounded by a certain value so that any slack larger than the value is considered abnormal.

3.2 Group Operations

A subprotocol that executes a multicast group operation is referred to as a *command*. TAM has four commands: JOIN, LEAVE, REPORT, and RELOCATE. While the first three are initiated by a user node, the last is initiated by the root node. The purposes of JOIN and LEAVE are self-explanatory. The REPORT command is a means of notifying the root of a significant event such as the detection of an uncooperative behavior. The RELOCATE command is used when the root node wants to move a user node to a different place in the tree. For example, a node will receive a RELOCATE when its parent leaves the group.

Figure 1 shows the protocol for JOIN, where C is a joining node, R the root node, and P the parent node chosen by the root. In the first step, C sends R its intention to join along with C_{info} , the information about C , such as its address and maximum number of children it is willing to support. We assume that C knows the address and public key of R . In the second step, R chooses a parent P for C . Sec-

$$\begin{aligned}
 C \rightarrow R & : \text{“join”}, C_{info} \\
 R \rightarrow C & : P_{info}, ticket_P = [\text{timestamp}, C, P]_{K_{R-1}} \\
 C \rightarrow P & : ticket_P \\
 P \rightarrow C & : message_i
 \end{aligned}$$

Figure 1. Join protocol. K_{R-1} denotes the root’s private key, and $[x]_K$ denotes x concatenated with the signature signed by K .

tion 5 discusses how to determine a parent. The root sends back the address of P and the ticket that will be handed to P . This ticket contains the identities of P and C , and a nonce (a random number to avoid replay attack), which is then signed by the root. The timestamp is used as a nonce to guarantee the freshness of the ticket, which avoids a replay attack. Thus, the ticket prevents C from connecting to an unauthorized parent. In the third step, C sends this ticket to P , which accepts C as a child if C submits a legitimate ticket, and is expected not to accept C otherwise. What if P accepts an unqualified node? Since confidentiality is not a concern, P would not be deemed uncooperative even if it did more work than required. In the last step, if C has been accepted as a child, P sends the last-received message to C . After the JOIN command is completed, C becomes part of the tree. Note that, although we omit the detail, a shared session key that is used by subsequent RELOCATES and REPORTS can be established between C and the root, as public-key operations are more costly than shared-key operations.

The LEAVE command simply sends the request to the root before the node leaves the group. Upon the receipt of the request, the root node sends RELOCATE to each child of the node that just left the group.

The RELOCATE command is similar to the JOIN command except that the former is initiated by the root node. That is, RELOCATE starts from the second step of JOIN. By moving a node, we mean moving a subtree that is rooted at the node. The RELOCATE command can also be used when well-behaved nodes are relocated up the tree or when misbehaving nodes are relocated down the tree. In such a non-leave context, RELOCATE requires additional work; the root should disconnect the relocated node r from its current parent. Otherwise, r might keep the link to the old parent as well as the new one in order to take advantage of having two parents or to simply waste system resource by filling up an otherwise free position.

When a node moves in the tree, a synchronization issue arises because the new parent’s last message may not be the first message that the moving node is missing. The problem is likely to occur when a node is moving from a node with high latency to one with low latency. To address this

problem, each node is required to log the last n messages. To estimate n , we consider the latency difference d and the inter-message time t , the time interval between two consecutive messages. To simplify, we define the system-wide latency difference d as the maximum latency from the root to a node in the tree. Then, n can be expressed as d/t .

The REPORT command is used by a user node that reports any type of problem from which it is suffering. This command is also similar to JOIN except that a subtree, instead of a reporting node alone, is moving and that the first-step message specifies one of the predefined problem descriptions (e.g., “message delayed”) in a field. According to the description, if the problem is likely to be persistent, the reporting node is relocated to a new place. Note that a node can report about only its parent. This restriction, together with the fact that a node cannot choose a parent on its own, makes the protocol robust against collusion or selective attacks in which a node or a group of nodes selects a well-behaved node and attempts to make it appear to be a bad one.

If a node that has many descendants delays a message, its descendants will report the problem at the same time, which results in the root node being overwhelmed by reports. Such an avalanche of reports is referred to as a report implosion. The report implosion problem is mitigated in two ways. First, if a node has many descendants, it is likely that the node has been acting reliably for a long time. Thus, the possibility that such a node causes the report implosion is relatively low. Second, when an overwhelming volume of reports occurs, the root can ignore most of them except for a few top-level ones. As a report request contains the sequence number that the reporting node is missing, the root node serves the reporting node only if no ancestor of the node has previously reported the problem for the particular sequence number. The rest of nodes are forced to wait until messages are forwarded from the node that the root just served. To further alleviate the report implosion, we prevent all related reports from occurring simultaneously. Whenever a user node moves, it is informed of a time t_x . If t_m is the timeout value specified in the last message m , and t_s is the timeout slack discussed above, the node sends a report after $t_m + t_s + t_x$ from the receipt of m in the absence of subsequent messages. The value of t_x should be proportional to the level of a node in the tree so that any delaying experienced by an ancestor is likely to be handled during the additional wait. On the other hand, t_x should also be randomized to prevent an adversary from deducing its level in the tree.

4 Trust Assignment

Following the discussion about how to detect and report uncooperative behaviors in the previous section, we discuss

how to evaluate the individual level of trust in this section. A sound trust evaluation scheme must have several properties. First, the evaluation scheme should increase the level of trust slowly, as it reflects how long a user stays in the system so that users have incentive to stay long. Second, it must react to negative feedback quickly to minimize the damage. Third, it should be resistant to collusion. Last, it need not be secret to work properly. That is, revealing the evaluation scheme should not give any advantage to attackers.

According to the behavior a node n manifests in the system, the root node updates n 's quality, or trustworthiness, attribute $q(n)$. Positive behavior is often implicit as the absence of REPORTs from or against a node suggests that the node behaves well. Another positive sign is willingness that a node shows to maintain other nodes as children. To reward the positive behavior, TAM periodically increases $q(n)$ by $\alpha c + \beta$, where α and β are the system parameters, and c the number of children that n maintains at the moment.

In contrast to implicit positive behavior, negative behavior can and should be detected by the nodes that suffer the consequences of such a behavior. Although negative feedback via REPORT indicates a violation of protocol, it is uncertain whom to blame from a single report. That is, it is uncertain whether the event actually happened as the report indicates, or the reporter is falsely accusing its parent. To address this problem, the root node should keep track of *behavior history* of the nodes. For this purpose, the root maintains two per-node attributes (h_s and h_r) in addition to the quality attribute q . Suppose there is a user node n and its parent node p . The attributes $h_s(n)$ and $h_r(n)$ are real numbers that reflect the recent behavior of n as a sender and a receiver, respectively. Lower values of the attributes indicate that the node behaved better. When n REPORTs against p , the root increases $h_s(p)$ by $\gamma_s q(n) + \delta_s$ and $h_r(n)$ by $\gamma_r q(p) + \delta_r$, where γ 's and δ 's are the system parameters. Both p and n pay a penalty because it is uncertain which one, or possibly both, is the cause of the problem. Because of γ 's, nodes are penalized more if they accuse or are accused by more trustworthy nodes. Note that these attributes are maintained internally in the root node and are not publicly available. To prevent adverse effects on the long-residing nodes, h_s and h_r values are decremented periodically.

When a node initially joins the system, it is labeled “trusted,” and each of the attributes q , h_s , and h_r is set to zero. There are two levels of the system thresholds for each of h_s and h_r (hence, a total of four thresholds). If a node's history value $h_s(n)$ or $h_r(n)$ exceeds the corresponding *soft threshold*, th_s^s or th_r^s , the quality attribute $q(n)$ is multiplicatively decreased. When it exceeds the *hard threshold*, th_s^h or th_r^h , the node is labeled “distrusted,” and its children are relocated. Distrusted nodes that do not actively

damage the system may be allowed to stay in the system as leaf nodes if the system has enough capacity. They can be evicted at any time, however, when the system is overloaded and decides not to accommodate less trusted nodes.

The thresholds are adjusted based on the runtime system load. This dynamic adjustment not only helps the system adapt itself to the workload but also prevents malicious nodes from managing to stay “right below” the thresholds. Since the evaluation scheme may be known to anyone, if the thresholds were fixed, malicious nodes might manage to cause trouble without exceeding the thresholds (i.e., being penalized). With the dynamic, nondeterministic thresholds, if they behave right below the thresholds, they become the first to be punished when the thresholds become lowered.

During the trust assignment, false positives may occur. A false positive is the case in which a cooperative node is labeled “distrusted” falsely. A cooperative node may encounter many uncooperative parents or children in a relatively short time. If such children falsely complain about this node, or the parents force it to complain about them, the increased h_r and h_s values of the node may cause it to be labeled “distrusted.” However, the likelihood of false positives is low with a sufficient number of cooperative nodes in the system. Suppose f denotes the fraction of uncooperative nodes in the population. If a node encounters m nodes that are randomly selected, a random variable X , denoting the number of uncooperative nodes among m , follows a binomial distribution with parameters m and f , provided the population is sufficiently large. Plotting $\Pr\{X > a\} = \sum_{i=a+1}^m \binom{m}{i} f^i (1-f)^{m-i}$ against $a = 0, \dots, m-1$ shows that this probability decreases quickly. For example, with $m = 20$ and $f = 0.1$, $\Pr\{X > a\}$ is 0.133, 0.043, 0.011, and 0.002 for $a = 3, 4, 5$, and 6, respectively. With $f = 0.05$, $\Pr\{X > a\}$ is 0.016, 0.003, 0.0003, and 0.00003 for the same a 's, respectively. The assumption that a node encounters randomly chosen nodes is a simplification; in fact, as shown in Section 5, a better node is more likely to encounter good nodes. Thus, the false positive rate is even less than it is in this binomial analysis. Our experiment results in Section 6 also corroborate TAM's low false positive ratio (less than 0.5% for aggressive thresholds and no false positives for conservative ones).

5 Trust-Associated Tree

Since TAM propagates data through a multicast tree, its performance relies largely on the *quality* of the tree. In this section, we rigorously define the notion of quality, which serves two purposes. First, it provides a criterion for evaluating the trustworthiness of a constructed multicast tree. Second, it leads to an algorithm that maintains a high-quality multicast tree despite the dynamic change in trustworthiness of the nodes.

We introduce several notations in order to characterize a good tree. Let n denote a node and $q(n)$ be the *quality* of n . The attribute $q(n)$ captures how well n behaves as expected by the protocol. Specifically, we interpret this attribute as the likelihood that a message goes through this node safely in any given attempt. Therefore, $q(n)$ has a value between zero and one. The assumption behind a single quality attribute for each node is that a node exhibits the same behavior for all other nodes with which it interacts. Although a node may exhibit bad behavior to only selected nodes, the problem is limited because our protocol does not allow nodes to select their parents or children. We believe that the notion of quality we define is reasonable for evaluating our protocol even in the presence of such “smart” attacks.

The quality is associated with nodes rather than with links for three reasons. First, we are more concerned with uncooperative node behavior than network failures, which would be captured by link trust. Second, a link at the application level is a complex entity as it may consist of many physical links and interposed routers. In addition, virtual links can be created and destroyed easily. Third, a link failure can be captured indirectly into the quality of the relevant nodes.

To capture the root-to-node *reliability*, we denote as $r(n)$ the likelihood that the node n receives the packet the root node has sent in the first transmission without retransmission or any reliability mechanism. Thus, it is defined as $r(n) = \prod_{k \in P} q(k)$, where P is the set of all nodes on the path from the root, inclusive, to the node n , exclusive. The node n is excluded from the path P because $q(n)$ represents the forwarding probability as opposed to the delivering (to upper layer) probability. For simplicity, we assume that if the parent of n forwards a message, n is sure to receive it. Note that $r(n)$ is not dependent on $q(n)$; it is position-dependent. As a special case, $r(\text{root})$ is defined to be 1.0. We define the quality of a multicast tree T as

$$q(T) = \sum_{n \in N} r(n),$$

where N is the set of all nodes in T . The *normalized quality* of a tree can be obtained after dividing $q(T)$ by the cardinality of N .

The quality of subtrees can be defined in the same way. We denote $q(T_n)$ as the quality of the subtree T_n , rooted at n . Consistent to the previous definition, the quality of a subtree can also be defined recursively:

$$q(T_n) = 1.0 + q(n) \sum_{m \in C_n} q(T_m),$$

where C_n is the set of n 's children and T_m is the subtree rooted at child m .

The quality of a tree reflects the expected number of nodes that would receive a packet that is sent by the root

and forwarded by nodes in the tree. The higher quality value a tree has, the more nodes receive a message in the initial transmission. Since the nodes that do not receive a message from the initial transmission are relocated in our protocol (see Section 3.2), a tree with low quality value is susceptible to more structural change, which in turn leads to poor stability and latency. Thus, we regard the tree with a higher value of quality as a better one not only from a message delivery perspective but also from a stability perspective.

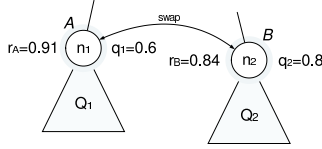


Figure 2. Quality-reliability reversal pair. While the positions in the tree are denoted by A and B , the nodes themselves are denoted by n_1 and n_2 . That is, n_1 is currently located at A , and n_2 at B .

To improve the quality of the tree, we should find points that can be improved. Suppose there exist two nodes n_1 and n_2 in the multicast tree such that $r(n_1) > r(n_2)$ and $q(n_1) < q(n_2)$, which means that despite its lower quality, n_1 is located at a more reliable position that is suitable for a node with higher quality. Such a pair of nodes is referred to as a *quality-reliability reversal pair*. The following theorem gives an idea of how to improve the quality of the tree.

Theorem 5.1 (reversal pair). *Given a tree T and a quality-reliability reversal pair of nodes, swapping either these two nodes or two subtrees rooted at them always increases $q(T)$. Furthermore, we can always decide whether to swap nodes or subtrees.*

Proof. From the recursive definition of $q(T)$, we can derive the equation for change of quality $\Delta_{q(\cdot)}$:

$$\Delta_{q(T)} = \sum_{n \in N'} r(n) \cdot \Delta_{q(T_n)},$$

where N' is the set of nodes that satisfies two conditions: (1) T_n for all $n \in N'$ covers all the changes in T , and (2) for any two nodes in N' , one is not an ancestor of the other.

In Figure 2, q_1 is short for $q(n_1)$ and q_2 for $q(n_2)$; and $Q_1 = \sum_{m \in C_{n_1}} q(T_m)$ and $Q_2 = \sum_{m \in C_{n_2}} q(T_m)$. Recall that the values of r_A and r_B are unchanged, no matter what node substitutes the positions of A and B , as long as their ancestors remain the same. We divide the problem into two cases.

For Case 1, where $Q_1 \geq Q_2$, swapping the two nodes increases the quality because

$$\begin{aligned} \Delta_{q(T)} &= r_A(q_2Q_1 - q_1Q_1) + r_B(q_1Q_2 - q_2Q_2) \\ &= (r_AQ_1 - r_BQ_2)(q_2 - q_1) > 0. \end{aligned}$$

For Case 2, where $Q_1 < Q_2$, swapping the two subtrees increases the quality because

$$\begin{aligned} \Delta_{q(T)} &= r_A(q_2Q_2 - q_1Q_1) + r_B(q_1Q_1 - q_2Q_2) \\ &= (r_A - r_B)(q_2Q_2 - q_1Q_1) > 0. \end{aligned}$$

□

Noteworthy are two special cases for the quality-reliability reversal pair. First, for a reversal pair, one is the parent of the other. In this case, swapping these two nodes always increases the quality. Second, in the case that A in Figure 2 is empty, moving the node at B to A increases the quality.

The optimal tree is one that maximizes the quality, given a set of nodes and their quality values. For simplicity, we assume that the quality values of the nodes are all distinct.¹ Without loss of generality, the siblings are ordered from left to right in the decreasing order of their quality values so that we can avoid isomorphic trees. We denote as n_i the i -th visited node by the left-to-right breadth-first traversal from the root, i.e., n_1 . Recall the constraint that the maximum degree of a node is d .

Theorem 5.2 (optimal tree). *The optimal tree T , in which every node has a unique quality value and the maximum out-degree d , must have the following two properties. First, the quality value of the nodes is monotonically decreasing. That is, for any i and j ($i < j$), $q(n_i) > q(n_j)$. Second, T is a complete d -ary tree. That is, there exists an index k such that $\text{degree}(n_i) = d$ for all $i < k$, $0 \leq \text{degree}(n_k) \leq d$, and $\text{degree}(n_j) = 0$ for all $j > k$.*

Proof. It suffices to show that a tree violating either property is suboptimal. Suppose a tree violates the first property. Then, there must exist the smallest index i such that $q(n_{i-1}) < q(n_i)$. Note that the parents of n_{i-1} and n_i must differ according to our convention for the order of siblings. Since the quality values are decreasing before n_i , $r(n_{i-1})$ must be greater than $r(n_i)$. Thus, the two nodes are a reversal pair, and the tree is suboptimal by Theorem 5.1.

If a tree satisfies the first property but violates the second property, there must exist two indexes i and j ($i < j$) such that $\text{degree}(n_i) < d$ and $\text{degree}(n_j) > 0$. Since moving one of n_j 's children under n_i increases the quality of the tree (the second special case of the reversal pair), the tree is suboptimal. □

From Theorems 5.1 and 5.2, we can conclude the following:

¹If quality values of nodes can be equal, more than one optimal tree that have the same $q(T)$ can exist. Our theorem can be easily extended to find all such optimal trees.

Corollary 1. *Nonexistence of quality-reliability reversal pairs is the sufficient and necessary condition for a tree being optimal.*

Proof. Nonexistence of quality-reliability reversal pairs is a necessary condition because if there exists such a pair in a tree, the tree cannot be optimal according to Theorem 5.1. It is also a sufficient condition because any suboptimal tree must be different from the optimal tree that is defined in Theorem 5.2 and therefore must have at least one reversal pair (contrapositive). \square

When nodes constantly join and leave, and their quality values frequently change, it is difficult to remove all reversal pairs. Nevertheless, we can approximately achieve the goal of maximizing the quality of tree. We adapt the tree incrementally by moving subtrees, each of which is rooted at the reporting node, one at a time. Thus, selecting appropriate new parents is the only means that achieves the goal of keeping the tree in a good shape. When the root node picks up a new parent for a node n , it retrieves the rank $r(n)$ by the descending order of the quality values q 's for all nodes. Its appropriate level l is computed as $\log_d r$, where d is the average out-degree. Then, the root node chooses randomly a node that is at or near level $l-1$. To support this operation, we periodically sort the list of nodes in the descending order of quality. The sorted list may be inconsistent because nodes join and leave, and the quality of nodes can change at any time. To mitigate this inconsistency, TAM records a major event such as a node being labeled "distrusted" in a temporary storage until the next sort.

6 Evaluation

We evaluated how TAM performed in both 95-node PlanetLab experiments and 10,000-node simulation experiments. The parameters for the experiments are listed in Table 1, in which N is the total number of nodes, f is the fraction of uncooperative nodes, and d is the maximum out-degree of each node. The rest are used for assigning trust values to nodes and are explained in Section 4. Updating trust for implicit positive feedback occurred every ten seconds for the PlanetLab experiments or every round, explained later, for the simulation experiments. The parameters were set to the values in the table during the experiments unless otherwise stated.

We model the behavior of nodes by assigning each node two behavior parameters. The *forward parameter*, ranging $[0,1]$, is the probability that a node forwards a message each time it receives one. The *report parameter*, also ranging $[0,1]$, is the probability that a node does *not* generate a false report each time it receives a message. Thus, a lower value for both parameters results in uncooperative behavior more frequently. When a node is instantiated, if a node's

Table 1. Experiment parameters. The numbers in parentheses are the default values for the simulation experiments.

Param.	Description	Value
N	Total number of nodes	95 (10,000)
f	Fraction of uncooperative nodes	0.1
d	Maximum capacity of children	2 (5)
α	Weight factor for children	0.01
β	Trust increment parameter	0.01
$\gamma_{\{s,r\}}$	Multiplicative parameters	4.0
$\delta_{\{s,r\}}$	Additive parameters	1.0
$th_{\{s,r\}}^h$	Hard thresholds	40.0
$th_{\{s,r\}}^s$	Soft thresholds	20.0

forwarding behavior is chosen to be uncooperative, the forward parameter is chosen uniformly randomly from $[0,0.5]$; otherwise, it is set to one in the PlanetLab experiments or uniformly randomly chosen from $[0.995,1]$ in the simulation experiments to simulate natural failure. The report parameter is chosen in the same way as, but independently of, the forward parameter. We classify a node as uncooperative if either or both of the parameters are chosen from the lower range. Since a node is instantiated as uncooperative with the probability f , the fraction of uncooperative nodes in the population is probabilistically equal to f . The actual fractions were very close to f in all the experiments.

6.1 PlanetLab Experiments

We implemented the protocol in C++ with the cryptlib toolkit [15] for cryptographic operations. We ran the root node on a Linux machine equipped with an Intel Pentium 4 2.4 GHz processor and 512 MB memory. For the user nodes, we selected one machine from each of 95 PlanetLab [23] sites. Although the machines of PlanetLab tend to have higher network capacity than those that comprise the typical peer-to-peer environments [5], the testbed enabled us to subject the experiment to the real Internet characteristics. We controlled user nodes through SSH remote execution.

To compensate for the relatively small number of nodes, we set the value of the maximum degree d to 2 so that the tree could be sufficiently deep. We tested three cases, each of which represented a different value of the uncooperative node ratio f (0.1, 0.2, and 0.4). We randomized the order of three runs to reduce the impact of uncontrollable effects such as node loads and background traffic. Then, we replicated a set of three cases three times (hence, a total of nine

runs). The plots show the results averaged over the three replicated runs. We looked into the individual results, and the variations within the same case were sufficiently small for us to be confident in the consistency of the results.

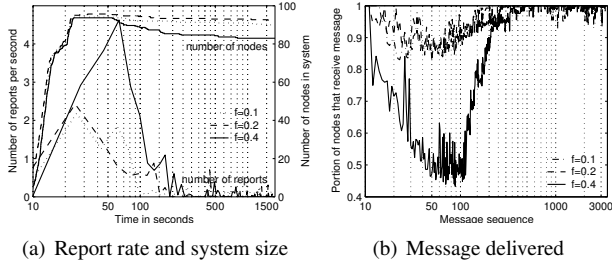


Figure 3. PlanetLab experiment. Note that the x-axes of both graphs are in log scale. In (a), The three Γ -shape lines represent the individual number of nodes in the system, corresponding to the right y-axis.

The three Γ -shape lines in Figure 3(a) represent the number of nodes in the system plotted against time, corresponding to the right y-axis. The lines show sharp increase in the beginning and slight decrease afterwards. Note that the x-axis is in log scale. While the sharp increase occurs because we invoked all user nodes remotely at the same time, the slight decrease occurs because some nodes were identified as uncooperative and eliminated. Although we reserved 95 nodes on PlanetLab, some nodes did not respond occasionally. That is why the $f=0.2$ line stays on top of the $f=0.1$ line. The former case had more nodes to begin with on average than the latter case. The remaining lines in Figure 3(a) show the report rates plotted against time. All cases show that the system stabilized quickly and stayed stable. The difference between the $f=0.4$ case and the $f=0.2$ case in the report rates is noticeably larger than between the $f=0.2$ case and the $f=0.1$ case. In a balanced tree of out-degree two, roughly half of the nodes are non-leaf. As the uncooperative factor f approaches this extreme, the cost to stabilize the system becomes even larger than the increase of f .

The PlanetLab experiments ran in an unreliable delivery mode, which means that missing messages were not retransmitted. Figure 3(b) shows that as time progressed, messages were delivered to more nodes. Between times 10 and 100, which correspond to message sequences 20 and 200, many nodes missed messages, which corresponds to high report rate in Figure 3(a). After the system became stable, the success rate of delivery stayed high.

6.2 Simulation Experiments

To complement the PlanetLab experiments, we set up simulation to evaluate our system in terms of three met-

rics. First, the stability, measured by the number of relocated subtrees, should increase over time in the presence of uncooperative nodes. Second, the quality of the multicast tree, defined in Section 5, should also increase over time. Third, the quality-depth correlation indicates how TAM rewards cooperative nodes by locating them closer to the root than uncooperative ones.

The time unit in simulation is a *round*, which represents a delivery of a single message. Unlike the PlanetLab experiments, in which several messages can be pipelined in transit, only one message is delivered at a time. Thus, each message corresponds to a round, and vice versa. In each of the first ten rounds, 10% of the N nodes joined the system. After that, no more nodes joined. Unlike the PlanetLab experiments, the nodes that were identified as distrusted were not eliminated from the system.

Two tree management schemes, the random and the tam schemes, have been tested. In the *random* scheme, whenever a new parent has to be selected, it is chosen randomly from the nodes whose out-degree is not saturated. The random scheme serves as a baseline case. The *tam* scheme is implemented as we describe TAM in this paper.

As each experiment was replicated three times, each line in the graphs shows the average of the results. We also inspected individual results and confirmed that the variance between replicated runs was sufficiently small.

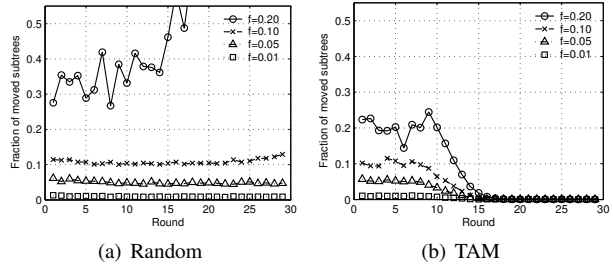


Figure 4. Stability change. The y-value can exceed 1.0 because a node is allowed to move more than once in a round.

Figure 4 plots against rounds the number of relocated subtrees divided by the total number of nodes, as each line represents a particular fraction of uncooperative nodes in the population. We use this fraction of relocated subtrees as a metric for stability of the system. Since a node can move multiple times in a round until it receives, the y-value can exceed 1.0. The random case shows no improvement over time. Interestingly, the $f=0.20$ line continued to go up to 2.4 at the end of simulation, although clipped out in the graph. This instability resulted from the nodes that continued to cause the victims to move around without being punished properly. In contrast, since TAM responded quickly, the system became and stayed stable in all cases.

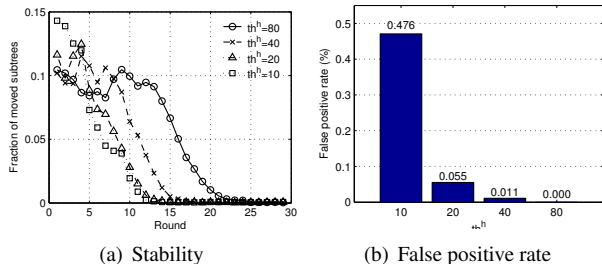


Figure 5. Trade-off between the transient period and the false positive rate. Note that the unit of (b) is percent.

Figure 5 shows the effect of the threshold value on the transient period and the false positive rate. Recall that TAM labels the nodes that misbehave consistently as distrusted. A false positive occurs when a node with behavior parameters chosen from the higher range (explained in the beginning of this section) is identified as distrusted by TAM. In every line in the graphs, both th_s^h and th_r^h were set to the same value as indicated by th^h . The soft thresholds were set to half of the hard threshold values. Lower threshold values stabilize the system more quickly, but also tend to result in more false positives. In the $th^h=10$ case, which corresponds to the first bar in Figure 5(b), 4 nodes, on the average of three runs, were labeled as distrusted among 8995 nodes whose parameters were chosen from the higher range (hence, supposed to be cooperative) at the end of the simulation. A threshold set to 40 led to no false positives.

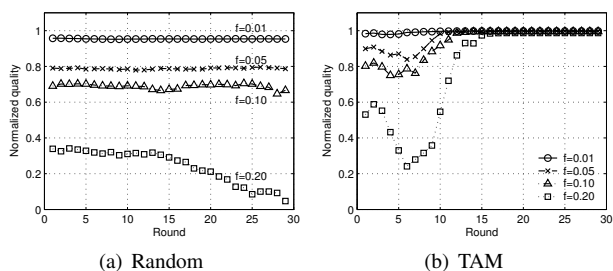


Figure 6. Quality change.

Figure 6 shows how the quality of trees, as defined in Section 5, changes for the two different schemes. The $f=0.20$ case of the random scheme shows degrading quality that matches increasing instability in Figure 4. While the random scheme shows decent performance when f is small, the tam scheme shows good performance close to the limit after round 10 even if f is large. The tam scheme performed poorly in the beginning rounds in which a large number of nodes were joining the system.

We analyzed the simulation data to see how many cooperative and uncooperative nodes occupied the bottom positions of the tree. The notation “bottom $x\%$ ” refers to $x\%$ of nodes that are the farthest from the root node. A tie was bro-

ken by random selection. In the beginning, we observe that uncooperative nodes were evenly distributed throughout the tree. At the end of the simulation, half of the uncooperative nodes were located at the bottom 10%. In particular, 91% of nodes among the bottom 5% were uncooperative. Some uncooperative nodes, especially those that joined late, were not identified as such because they remained leaf nodes. If more nodes had joined, these uncaught nodes would have been selected as parents and eventually identified as distrusted. We believe that if the system works on an ongoing basis, most uncooperative nodes will be identified correctly.

7 Related Work

Peer trust can provide robust service against uncooperative users in several different contexts: anonymous search network [19], file sharing system [12], e-commerce [24], and general settings [1]. In addition to peer trust, other approaches consider redundancy (e.g., secure routing for structured overlay [8]) and game theory (e.g., unicast ad-hoc routing among selfish nodes [25]).

Diot et al. explain the deployment issues in IP multicast [13]. To ease the deployment, recent research has moved a step towards the application level [17, 18, 4]. Banerjee and Bhattacharjee explain and compare some of them [3]. More recent application-level multicast systems put emphasis on better utilizing available resources from all participating nodes for both performance and fairness. SplitStream [9] achieves this goal by using multiple trees. Each peer serves as an internal node in only one of the trees while being a leaf node in the rest of them. Bullet [20] takes a hybrid approach; it relies on the tree for initial delivery, and then data are simultaneously disseminated via a mesh of nodes. TAM has a potential weakness in that some distrusted nodes, when becoming leaf nodes, may not contribute as much as they should. However, the problem diminishes because such nodes are evicted first whenever the system resource becomes limited.

Several reliable multicast systems have been proposed. Holbrook et al. [16] propose the receiver-side (NAK-based) reliability, which presumes that no problem exists until receivers complain. TAM shares the same reliability semantics. Pbcast [6] provides the “almost all or almost none” semantics, relaxed from total reliability, in favor of scalability while it relies on a native multicast facility for initial transmission. These *reliable* multicast systems primarily deal with moderate packet loss or network failure. A distinct feature of TAM is its ability to deal with uncooperative nodes.

BitTorrent [11] has complementing features as it is a pull-based model while TAM is a push-based model.

8 Conclusion

This paper discusses a large-scale overlay multicast system that works without having a priori knowledge about trustworthiness of the participating nodes. Working without such knowledge is becoming more important as it becomes common for a system to work beyond a single administration boundary. TAM, using the reports from the nodes that detect uncooperative behavior, evaluates the trustworthiness of participating nodes. According to the evaluation, well-behaved nodes are rewarded by moving closer to the source while misbehaving nodes are punished by moving down in the tree, or even being evicted, so that they impact fewer nodes and contribute less to instability.

Peer-to-peer systems are gaining popularity largely because they appeal directly to a huge number of end users. The direct appeal requires the systems to be robust against uncooperative peers because individual peers have autonomy to act as they wish; their choice often leads to many problems if the systems are not designed properly. Thus, we argue that trust awareness must be inherently built in these systems. We believe that our work demonstrates a concrete instance of trust-aware computing.

References

- [1] K. Aberer and Z. Despotovic. Managing trust in a peer-2-peer information system. In *Proc. the International Conference on Information and Knowledge Management*, 2001.
- [2] E. Adar and B. A. Huberman. Free riding on Gnutella. *First Monday*, 5(10), Oct. 2000.
- [3] S. Banerjee and B. Bhattacharjee. A comparative study of application layer multicast protocols. <http://www.cs.umd.edu/projects/nice/papers/compare.ps.gz>.
- [4] S. Banerjee and B. Bhattacharjee. Scalable application layer multicast. In *Proceedings of ACM SIGCOMM*, 2002.
- [5] S. Banerjee, T. G. Griffin, and M. Pias. The interdomain connectivity of PlanetLab nodes. In *Passive and Active Measurements (PAM) Workshop*, Apr. 2004.
- [6] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Transactions on Computer Science*, 17(2), May 1999.
- [7] M. S. Blumenthal and D. D. Clark. Rethinking the design of the internet: The end-to-end arguments vs. the brave new world. *ACM Trans. Internet Technology*, 1(1), 2001.
- [8] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proc. the Symposium on Operating Systems Design and Implementation*, 2002.
- [9] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth multicast in a cooperative environment. In *Proceedings of ACM Symposium on Operating Systems and Principles*, 2003.
- [10] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in cyberspace: Defining tomorrow's internet. In *Proceedings of ACM SIGCOMM*, Aug. 2002.
- [11] B. Cohen. Incentives build robustness in BitTorrent. <http://bittorrent.com/bittorrentecon.pdf>, May 2003.
- [12] F. Cornelli, E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Choosing reputable servents in a P2P network. In *Proc. the international World Wide Web Conference*, 2002.
- [13] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, January/February 2000.
- [14] J. R. Douceur. The sybil attack. In *Proc. the 1st International Workshop on Peer-to-Peer Systems*, 2002.
- [15] P. Gutmann. cryptlib encryption toolkit. <http://www.cs.auckland.ac.nz/~pgut001/cryptlib>.
- [16] H. W. Holbrook, S. K. Singhal, and D. R. Cheriton. Log-based receiver-reliable multicast for distributed interactive simulation. In *Proceedings of ACM SIGCOMM*, 1995.
- [17] Y. hua Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proc. SIGMETRICS*, 2000.
- [18] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. J. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proc. OSDI*, 2000.
- [19] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The EigenTrust algorithm for reputation management in p2p networks. In *Proc. the International World Wide Web Conference*, 2003.
- [20] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proceedings of ACM Symposium on Operating Systems and Principles*, Oct. 2003.
- [21] National Institute of Standards and Technology. Digital signature standard (DSS). FIPS PUB 186, May 1994.
- [22] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis. RFC 2330. framework for ip performance metrics, May 1998.
- [23] PlanetLab. <http://www.planet-lab.org>.
- [24] L. Xiong and L. Liu. A reputation-based trust model for peer-to-peer eCommerce communities. In *Proc. IEEE Conference on E-Commerce*, 2003.
- [25] S. Zhong, J. Chen, and Y. R. Yang. Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks. In *Proc. INFOCOM*, 2003.