

Faithful Distributed Shapley Mechanisms for Sharing the Cost of Multicast Transmissions

Nandan Garg
Dept. of Computer Science
Wayne State University
5143 Cass Avenue, Detroit, MI 48202.
nandang@wayne.edu

Daniel Grosu
Dept. of Computer Science
Wayne State University
5143 Cass Avenue, Detroit, MI 48202.
dgrosu@cs.wayne.edu

Abstract

Sharing the cost of multicast transmissions was studied in the past and two mechanisms, Marginal Cost and Shapley Value, were proposed. Compared to the Marginal Cost Mechanism the Shapley Value mechanism has the advantage of being budget balanced. Although both of them are strategyproof mechanisms, the distributed protocols implementing them are susceptible to manipulation by autonomous nodes. We propose two protocols that implement the Shapley Value mechanism in which the nodes do not have incentives to deviate from the protocol specifications. We show that the proposed protocols are faithful implementations of the Shapley Value mechanism. We deploy these protocols on PlanetLab and analyze their performance.

1. Introduction

Multicast transmissions have widespread use for information and content distribution (e.g., audio and video). They are efficient as they minimize the number of messages traversing a single link by creating a multicast tree. A multicast transmission incurs a cost and it is justifiable that the users who receive the transmission pay for it. Most applications such as audio and video multicast transmissions involve a content provider who charges money from the receivers for providing the content. However the amount a user is willing to pay is determined by the benefit it receives from the transmission.

The benefit for each user e is quantified by a private single valued parameter u_e known as the user's *utility*. User e will like to receive the transmission if her cost share x_e is less than her utility, i.e., if her welfare $w_e = u_e - x_e$ is positive. Cost sharing mechanisms determine who receives the multicast and how much they have to pay for the service. However, the calculation of cost shares is not a trivial task,

as the users may cheat by lying about their utility.

Algorithmic Mechanism Design (AMD) [12] and Distributed Algorithmic Mechanism Design (DAMD) [7] study mechanisms in which the outcome and the payment depend on the input provided by the participants. These participants (called agents) are assumed to be rational (i.e., they want to maximize their profit). They have strong motivation to lie about their private values in order to get extra benefit. The task of the system designer is to design mechanisms that achieve system-wide goals. These goals may be hampered if the agents lie about their private values. To prevent manipulation and motivate users to participate honestly, Mechanism Design theory (a subfield of Microeconomics) is used to model the behavior of agents. Of importance are the *strategyproof mechanisms* [9], in which the users obtain maximum profit when they declare their private values truthfully. In standard AMD [12], a centralized trusted entity implements the mechanism, i.e., collects the input from agents, calculates the outcome and distributes the payment. When the mechanism is implemented in a distributed system [6], the agents themselves execute the algorithm and collectively calculate the outcome and the payments. Distributed implementations of mechanisms have been proposed for various problems such as multicast cost sharing [6] and scheduling [3].

The underlying assumption in DAMD is that the agents may lie about their private values but they do not deviate from the specified distributed algorithm. This model of distributed implementation is known as the *Tamper-Proof Model (TPM)*. This assumption is weak because agents can easily manipulate the distributed algorithm in their favor, since they control it. The model in which the agents can deviate from the specified distributed algorithm, is called the *Autonomous Nodes Model (ANM)*.

Feigenbaum *et al.* [6] proposed two distributed mechanisms to calculate the cost shares of participants in a multicast transmission: the Marginal Cost (MC) mechanism

and the Shapley Value (SH) mechanism. They presented the MC and SH mechanisms assuming the TPM. Mitchell and Teague [10] proposed a MC mechanism assuming the ANM. They augmented the original MC mechanism for the TPM proposed in [6] with asymmetric key cryptographic primitives to prevent cheating. Specifically they used digital signatures to authenticate the sender of the messages and use auditing to verify that the agents executed the protocol correctly.

The MC mechanism is strategyproof and efficient (maximizes the overall welfare), but not budget balanced. In fact it is known that it generally runs budget deficit and in many cases does not generate any revenue at all [11]. No strategyproof mechanism can be both efficient and budget balanced at the same time [8]. The MC mechanism is recommended when the multicast delivery may be subsidized, if the mechanism runs a budget deficit [7]. Since the MC mechanism does not generate sufficient revenue, it is not a suitable mechanism from the content provider's point of view. The content provider will quickly go out of business, especially when there exists competing content providers. Also the MC mechanism is susceptible to collusion [4].

The SH mechanism is a better choice from these considerations because it is budget balanced and group strategyproof. The SH mechanism possesses Consumer Sovereignty (CS), in addition to No-Positive Transfers (NPT) and Voluntary Participation (VP) properties. The MC mechanism satisfies NPT and VP, but does not satisfy CS [11]. Although the SH mechanism is not efficient, for large user populations it approaches perfect efficiency. Also from the class of group-strategyproof mechanisms which are budget-balanced, the SH mechanism minimizes the worst-case welfare loss [11]. The only drawback of the SH mechanism is that it has a higher network complexity [5]. However, we believe that the cost-share calculation will induce a relatively small overhead to the overall multicast transmission. Thus it is justifiable to prefer the SH mechanism, given its excellent properties.

In this paper our objective is to develop distributed protocols that implement the Shapley Value mechanism assuming the Autonomous Nodes Model. Similar to [10], we use digital signatures to authenticate the messages sent by the nodes. We use auditing and verification to detect cheating by the nodes. We propose two variants of distributed implementations. The second variant is an extension of the first one, in which more information is sent in the messages. This enables the receiver nodes to verify that the values they have received are computed according to the protocol. Thus the extended protocol has stronger cheating detection capabilities. Since the two protocols differ in their resource requirements and cheating detection capabilities, the administrator can choose one of them depending on the requirements of the content provider. The limitation

in [10] is that it assumes that there is only one user per node, however, in our protocols we allow more than one user per node. Since the SH mechanism involves more than one iteration of bottom-up and top-down traversals (unlike the MC mechanism which has only one iteration), we propose methods to prevent deviations in any iteration.

Shneidman *et al.* [14] proposed the concept of *faithful implementation* of a mechanism. When the distributed mechanisms are implemented by the agents themselves, they may deviate from the specified algorithm if it is beneficial to do so. These deviations are categorized in three groups, information revelation, message passing and computation. A *faithful implementation* is a specification of a mechanism where the agents cannot gain any benefit by deviating from it. We show that our proposed distributed protocols are faithful implementations of the SH mechanism.

The organization of the rest of the paper is as follows. In section 2, we present the background including the network model and the SH mechanism for TPM. Section 3 begins with the description of how nodes can cheat in the original implementation of the SH mechanism for TPM. We then present our proposed protocols implementing the SH mechanism for ANM. In section 4 we show that our protocols are faithful implementations of the SH mechanism. We also implemented our proposed mechanisms and deployed them on PlanetLab. In section 5 we describe the experimental setup and the results we obtained from the experiments. Section 6 concludes the paper with a summary and future research directions.

2. Background

2.1. Model

We assume the following network model. The user population P resides at N nodes. Each user $e \in P$ resides at some node $i \in N$. The nodes are connected by bidirectional links. $R \subseteq P$ is the set of users who receive the multicast transmission. The transmission starts from a node $root \in N$ and flows through a static multicast tree $T(R) \subseteq T(P)$, where $T(R)$ and $T(P)$ denote the multicast tree connecting the nodes in R and P respectively. The techniques used to create these trees are described in [6, 15]. We denote the subtree rooted at node i as T_i . Each link connecting node i to its parent node p has a cost c_i associated with it. C_i denotes the set of all the t children $k_1, \dots, k_t$ of node i and r_i denotes the set of all the users at node i .

2.2. Shapley Value Mechanism for the Tamper-Proof Model

The Shapley Value [13] mechanism is implemented using an iterative algorithm. In the bottom-up traversal each

node i determines the number of users α_i in T_i who choose to receive the transmission. β_i is the cost share of each of the resident users at node i who receive the transmission, i.e., $x_e = \beta_i, \forall e \in r_i \cap R$. n_i represents the number of users at node i who choose to receive the transmission. The bottom-up traversal starts from the leaf nodes. A leaf node k reports its α_k value to its parent (for the leaf nodes, $\alpha_k = n_k$). Nodes (other than leaf nodes) calculate $\alpha_i = \sum_{k \in C_i} \alpha_k + n_i$ and send it to their parent node. After $root$ receives $\alpha_i, i \in C_{root}$, it initiates the top-down traversal, where it sends $\beta_{root} = 0$ to each of its children. Each node receives β_p from its parent and computes β_i as follows:

$$\beta_i = \left(\frac{c_i}{\alpha_i} \right) + \beta_p \quad (1)$$

β_i is then sent to all the children of node i (i.e., all nodes $k \in C_i$). All users $e \in r_i$ are assigned the cost share $x_e = \beta_i$. If the cost share x_e of any user e is greater than its utility u_e then user e declines to receive the transmission. In that case α_i decreases and it needs to be updated in the next bottom-up traversal. This increases the cost shares of the other users sharing the links with e . Thus in each iteration of the bottom-up and the top-down traversal, users may be removed from the receiver set R and the cost shares are updated. These iterations are repeated until no more users are dropped and until the cost share of any user does not change in two subsequent iterations. Initially $R = P$ and in the worst case one user is dropped in each iteration. If we assume that the algorithm converges in m iterations, the number of messages required in this case is $\Omega(n \times m)$. The detailed analysis of computational and communication complexity is presented in [6]. An example of the execution of one iteration of the SH mechanism is shown in Figure 1.

3. Shapley Value Mechanism for Autonomous Nodes Model

The distributed implementation of the SH mechanism in [6], is vulnerable to deviations by the nodes. In this section we present our proposed protocols that implement the SH mechanism and prevent such deviations. We first present the notation used in describing the proposed protocols. Then we describe the ways in which a node can cheat in the original protocol. Finally we describe our proposed protocols.

3.1. Notation

As described in section 2.2, SH is an iterative mechanism. It performs more than one iteration of the bottom-up and top-down traversals. The number of users at node i , in iteration j , who choose to receive the multicast transmission is denoted by n_i^j . For iteration j the number of users in

the subtree rooted at i which receive the transmission is α_i^j . The cost share of users at node i , calculated in iteration j , is denoted by β_i^j . During the top-down phase of iteration j , node i receives cost share β_p^j from p . The cost share β_i^j is calculated using the formula $\beta_i^j = (c_i/\alpha_i^j) + \beta_p^j$ (from (1)). The message M signed by node i using its private key K_i is denoted by $E_{K_i}[M]$.

3.2. Cheating in the Tamper-Proof Model

In the following we show how a node can cheat by manipulating the values sent to other nodes. The scenario in which no cheating occurs is shown in Figure 1. For simplicity we assume that every node has only one user. The values of α_i^j and β_i^j are shown for iteration j . In Figure 1, the user at node 3 has to pay 4 and users at node 4 and 5, each pays 6.

By modifying β_i^j sent to its children, node i ensures that the users at node i receive the transmission but pay nothing. To maintain the budget balance, node i makes its children (and all the nodes in its subtree) pay an extra amount, to compensate for the cost of transmission received by the users at node i . Node i sends $\beta_i^{j'}$ to its children instead of β_i^j . $\beta_i^{j'}$ is calculated using the formula $\beta_i^{j'} = (c_i + \beta_p^j * n_i^j) / (\alpha_i^j - n_i^j) + \beta_p^j$. Essentially node i divides the cost of link c_i among the users in its subtree, excluding users residing at i . The users at node i have also to share the costs of links from p to $root$, which is β_p^j . Node i distributes the share of its users to its descendants by adding $\beta_p^j * n_i^j$ to c_i and dividing only by the number of descendants $(\alpha_i^j - n_i^j)$. Thus node i assigns zero as the cost share to its resident users. The cost share $\beta_i^{j'}$ is calculated in such a way that the budget remains balanced and the root cannot detect the cheating. Figure 2 shows how node 3 cheats. The user residing at node 3 pays nothing and the users at node 4 and 5 each pays 8. Thus each of them pays an extra amount of 2. The total amount overpaid collectively by the two users at node 4 and 5 is 4, compensating for the payment of the user at node 3.

In addition to the cheating described in Figure 2, there are other ways in which a node can cheat. One way is that node i can increase the value of α_i^j sent in the bottom-up phase. Thus the cost of the links connecting i to $root$ will be shared among a greater number of users, thus reducing the per-user share at nodes in the tree $T(P) - T_i$ (nodes sharing the links connecting i to $root$). Then node i can manipulate β_i^j sent to the children and make them pay that amount. In this case the benefit is received not only by node i but by all the nodes in subtree $T(P) - T_i$.

Another way a node i can manipulate the protocol is by sending a very high value of β_i^j to its children. If β_i^j is sufficiently high, the utility of the users in subtree T_i will be

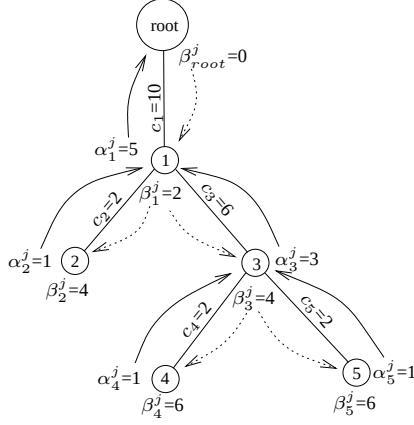


Figure 1. No cheating

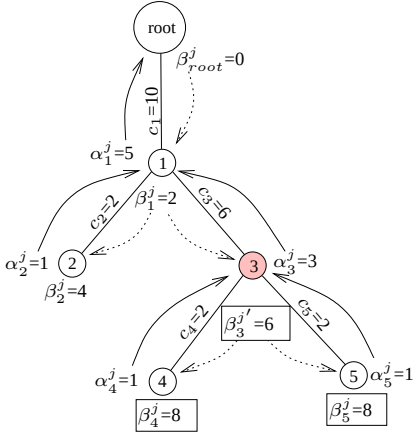


Figure 2. Node 3 cheats by sending modified values to its children

less than their cost share and thus they will have to remove themselves from the set of receivers R of the transmission. Node i still receives the transmission, but it does not have to send the transmission further in the multicast tree. Thus node i can save the upload bandwidth. Here the node is not acting maliciously, but rationally.

We want to detect such cheating and prevent it by penalizing the nodes who cheat. For this we require the nodes to sign the messages they sent using digital signatures. We then use auditing/verification procedures to detect cheating. Since SH mechanism is essentially an iterative mechanism, a node can cheat in any of the iterations. To efficiently detect cheating, signing and auditing should be done in each iteration.

Node i executes

```

j = 1;
do
{
Phase 1 (Bottom-up)
  for each child  $k \in C_i$ 
    recv( $E_{K_k}[\alpha_k^j], k$ );
    Calculate  $\alpha_i^j = \sum_{a=1}^t \alpha_{k_a}^j + n_i^j$ ;
    send( $E_{K_i}[\alpha_i^j], p$ );

Phase 2 (Top-down)
  if (node is root)
    for each child  $k \in C_{root}$ 
      send( $E_{K_{root}}[0 || \alpha_k^j], k$ );
  else
    recv( $E_{K_p}[\beta_p^j || \alpha_i^j], p$ );
    Calculate  $\beta_i^j = (c_i / \alpha_i^j) + \beta_p^j$ ;
    for each child  $k \in C_i$ 
      send( $E_{K_i}[\beta_i^j || \alpha_k^j], k$ );
    j = j + 1;
}
while ( $\beta_p^j \neq \beta_p^{j-1}$ );
m = j - 1;
```

Phase 3 (Payment and Auditing)

```

Calculate  $payment_i = \beta_i^m * n_i^m$ ;
send( $payment_i, root$ );
if (node is root)
  request proof from node  $i$  with probability  $Prob_p$ ;
  recv( $proof_i, i$ );
  verify( $proof_i$ );
else
  if proof is requested from root
    Prepare  $proof_i =$ 
       $\|_{s=1}^m (n_i^s || E_{K_p}[\beta_p^s || \alpha_i^s] || E_{K_{k_1}}[\alpha_{k_1}^s] || \dots$ 
       $|| E_{K_{k_t}}[\alpha_{k_t}^s]);$ 
    send( $proof_i, root$ );
```

Figure 3. SH-ANM Protocol

3.3. SH-ANM Protocol

The following notation and primitives will be used in the protocol description: m denotes the final iteration in which the mechanism stabilizes; $||$ is the concatenation operator; **send**(M, R) denotes sending message M to a node R ; **recv**(M, R) denotes receiving message M from a node R ; and **verify**($proof_i$) represents the procedure used to verify the proof received from node i .

The proposed protocol is presented in Figure 3. The protocol is executed in three phases. Phase 1 (bottom-up) and

Phase 2 (top-down) are executed iteratively until the mechanism stabilizes. In each iteration j , in Phase 1, node i receives $E_{K_k}[\alpha_k^j]$ from each children $k \in \mathcal{C}_i$. Node i calculates

$$\alpha_i^j = \sum_{a=1}^t \alpha_{k_a}^j + n_i^j \quad (2)$$

and sends $E_{K_i}[\alpha_i^j]$ to its parent p .

Phase 2 (top-down) is initiated by *root* sending $\beta_{root}^j = 0$ to its children. Node i receives $E_{K_p}[\beta_p^j \parallel \alpha_i^j]$ from its parent p . It calculates

$$\beta_i^j = \left(\frac{c_i}{\alpha_i^j} \right) + \beta_p^j \quad (3)$$

and sends $E_{K_i}[\beta_i^j \parallel \alpha_k^j]$ to all its children. The cost share of the users at node i who receive the transmission is β_i^j .

Phase 3 starts after the mechanism stabilizes. In this phase payments are sent by the nodes to *root* and auditing is done by *root*. Assume that the mechanism stabilizes in m iterations. The payment sent by node i to *root* is calculated using $payment_i = \beta_i^m * n_i^m$. The root node audits node i with probability $Prob_p$ by asking for a proof of payment: $proof_i = \parallel_{s=1}^m (n_i^s \parallel E_{K_p}[\beta_p^s \parallel \alpha_i^s] \parallel E_{K_{k_1}}[\alpha_{k_1}^s] \parallel \dots \parallel E_{K_{k_t}}[\alpha_{k_t}^s])$. It is assumed that the root knows the public key of all the nodes who want to participate in the multicast transmission. The root calculates α_i^j using (2) and β_i^j using (3) for each iteration j . It then verifies that the payment received from users residing at node i is $\beta_i^m * n_i^m$. If the payment received is not equal to $\beta_i^m * n_i^m$, node i has to pay a penalty $\mathcal{P}$ which is higher than any possible gain by cheating. If a node a can present two different messages signed by node b then node b has to pay the penalty $\mathcal{P}$ and node a gets a reward (may be equal to $\mathcal{P}$).

When the mechanism stabilizes in m rounds, the number of messages required in SH-ANM is $\Omega(nm + n)$, which is a linear increase from that of SH. This increase is due to the extra messages required for auditing by the root node (requesting and sending the proofs).

3.4. Extended Protocol - SH-ANM-E

SH-ANM-E has more information in each message sent in the top-down phase, to make it stronger and thus difficult to cheat. The bottom-up phase remains the same as in SH-ANM. The top-down phase is modified as follows. Assume that the grandparent of i , (i.e., the parent of p) is denoted by g . The top-down phase is initiated by *root* sending $\beta_{root}^j = 0$ to its children. Node i receives $E_{K_p}[\alpha_i^j \parallel \alpha_p^j \parallel \beta_p^j \parallel \beta_g^j]$ from its parent p . Node i checks that the parent p has calculated its values correctly using equation $\beta_p^j = (c_p/\alpha_p^j) + \beta_g^j$. It then calculates β_i^j using (3) and sends $E_{K_i}[\alpha_{k_a}^j \parallel \alpha_i^j \parallel \beta_i^j \parallel \beta_p^j]$ to all its children ($a = 1, \dots, t$).

After the complete execution of all the iterations, the root audits node i with probability $Prob_p$ by asking for a proof of payment ($proof_i$), which is constructed as follows:

$proof_i = \parallel_{s=1}^m (E_{K_p}[\alpha_i^s \parallel \alpha_p^s \parallel \beta_p^s \parallel \beta_g^s] \parallel E_{K_{k_1}}[\alpha_{k_1}^s] \parallel \dots \parallel E_{K_{k_t}}[\alpha_{k_t}^s])$ The root calculates α_i^j using (2) and β_i^j using (3) for each iteration j . It then verifies that the payment received from each user residing at node i is β_i^m .

Both protocols, SH-ANM and SH-ANM-E, prevent node deviations, however they differ in their detection capabilities and resource requirements. In SH-ANM, the nodes are audited in Phase 3 with probability $Prob_p$, so there is a chance that a cheating node may remain undetected. In SH-ANM-E, node i verifies the computation of β_p^j by parent p and so there is no possibility of remaining undetected after cheating. The auditing is still required at the end in SH-ANM-E to prevent any collusion between a node and its children. SH-ANM-E requires more network resources because the message size is bigger as compared to SH-ANM. Also more computational resources will be required because of the extra step of verification of β_p^j . Since there is a trade-off between the cheating detection capability and resource requirements, the administrator of the system has to choose one of them according to the requirements of the content provider.

4. Properties

In this section we characterize the properties of the protocols proposed in section 3. We first present the definitions of strong-communication compatibility, strong-algorithm compatibility and faithful implementation and then show that the proposed protocols are faithful implementations of the SH mechanism.

Definition 4.1 (Strong-Communication Compatibility) [14] A distributed mechanism is Strong-Communication Compatible (Strong-CC), if a participating node cannot obtain higher utility by deviating from the suggested message-passing strategy (independent of its information-revelation and computational actions), when other nodes follow the suggested specification.

Definition 4.2 (Strong-Algorithm Compatibility) [14] A distributed mechanism is Strong-Algorithm Compatible (Strong-AC), if a node cannot obtain higher utility by deviating from the suggested computational strategy (independent of its information-revelation and message-passing actions), when other nodes follow the suggested specification.

Definition 4.3 (Faithful implementation) [14] A distributed mechanism specification is a faithful implementation when the corresponding centralized mechanism is strategyproof and when the specification is Strong-CC and Strong-AC.

Theorem 4.1 *SH-ANM and SH-ANM-E are faithful distributed implementations of the SH mechanism.*

Proof. (sketch) In order to prove this we show that both SH-ANM and SH-ANM-E satisfy the three properties in Definition 4.3.

(i) The corresponding centralized mechanism is strategyproof: SH-ANM and SH-ANM-E are distributed implementations of the SH mechanism. Since the SH mechanism is group-strategyproof [11], which is a stronger property than strategy-proofness, it is also strategy-proof.

(ii) Strong-CC: In all the three phases of both SH-ANM and SH-ANM-E no explicit message-passing takes place, *i.e.*, there is no message M which is received by node i and the same message M is sent by node i . Thus, the Strong-CC property is satisfied.

(iii) Strong-AC: In Phase 1, a node i may send a manipulated value of α_i^j to its parent p . Similarly in Phase 2, it may send a modified value of β_i^j to its children. However in Phase 3, *root* requests $proof_i$ from node i . $proof_i$ is composed of the messages received by node i in all iterations. For example, in SH-ANM, $proof_i = \parallel_{s=1}^m (n_i^s \parallel E_{K_p}[\beta_p^s \parallel \alpha_i^s] \parallel E_{K_{k_1}}[\alpha_{k_1}^s] \parallel \dots \parallel E_{K_{k_t}}[\alpha_{k_t}^s])$. In both SH-ANM and SH-ANM-E, if node i cheated during Phase 1 or Phase 2, such cheating will be detected by the *root* in Phase 3 with probability $Prob_p$. This will cause node i to pay a penalty $\mathcal{P}$. Since penalty $\mathcal{P}$, is strictly greater than any gain node i can get by deviating from the computational strategy, it has no incentive to cheat in Phase 1 or Phase 2. Thus the mechanisms' specifications are Strong-AC. Additionally in SH-ANM-E, node i can detect cheating by parent node p , as described in section 3.4. Node i presents the message $E_{K_p}[\alpha_i^j \parallel \alpha_p^j \parallel \beta_p^j \parallel \beta_g^j]$ to *root* and if the cheating is detected, node p will be penalized. A node cannot increase its utility by deviating and thus the mechanism specification is Strong-AC.

Since all three properties are satisfied, according to definition 4.3, SH-ANM and SH-ANM-E are faithful implementations of the SH mechanism. $\square$

5. Experimental Results

We implemented the existing SH protocol and the newly proposed SH-ANM and SH-ANM-E protocols in a distributed environment. The program implementing the protocol runs on all the nodes participating in the multicast (including the root node). For encryption and decryption of messages (using public key cryptography, specifically RSA), Openssl library [1] is used. It is assumed that the public keys of the child and parent nodes are already available on each node, so no key exchange mechanism is employed. The implementation can be easily integrated within a multicast application where the SH-ANM (or SH-ANM-

E) protocol is executed first and then the multicast data (*e.g.*, on-demand video) is transmitted.

For our experiments we ran the protocols on the PlanetLab [2] distributed environment. We selected eight PlanetLab nodes and created a multicast tree. The number of users was generated randomly using the discrete uniform distribution over the interval [1,5]. The utilities of the users were also generated randomly using the discrete uniform distribution over the interval [1,100]. We call a complete execution of a mechanism an experiment. Nodes 2, 3 and 4 cheated with a probability $Prob_c = 0.2$. We ran the experiments with different values of the probability of auditing $Prob_p$. For each value of $Prob_p$, we ran 100 experiments and took the average over those experiments. This was particularly done because PlanetLab is a very dynamic environment and only one experiment would be insufficient to draw conclusions. The main data collected for analysis was the time required for each node to execute the mechanism, the number of rounds required to stabilize, the number of times a node cheated and the number of times the cheating was detected by the root node.

In Figure 4 we observe that the original SH mechanism takes less time as compared to the SH-ANM mechanism. SH-ANM mechanism requires more time because it involves encryption and decryption of the messages sent to all the nodes as well as collecting and checking the proofs at the end. However we note that the increase in the overall execution time is less than 50% in the case of SH-ANM with $Prob_p = 0.2$. Thus SH-ANM does not induce a significant overhead. The average time required for SH-ANM is between 4 to 7 seconds which is negligible as compared to the actual multicast transmission (typically an audio or video of average length of several minutes). We also observe that the time required by the nodes to complete the protocol is directly dependent on the probability of auditing ($Prob_p$). This is expected because the higher the probability, the higher the time spent in sending and checking the proofs.

Figure 5 shows that increasing the auditing probability $Prob_p$ the protocol is able to detect more cheating. As expected at $Prob_p = 1$, the protocol detects all the cheating done by the nodes.

The experimental results show that SH-ANM protocol is very effective in detecting the cheating by the nodes and it does not induce a significant overhead to the multicast transmission. The SH-ANM-E protocol requires less time for execution on average because cheating (if any) will be detected early by the children reporting it to the root, who stops the protocol. When the nodes do not cheat, the execution time for SH-ANM-E will be higher as compared to SH-ANM because of the overhead of checking the values by children nodes at each step. Due to space limitations we are not able to present here the experimental results for SH-

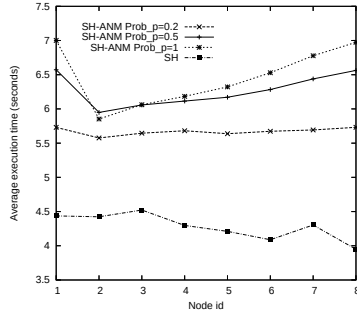


Figure 4. Average execution time of the mechanisms on different nodes

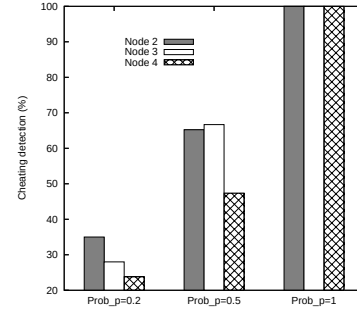


Figure 5. Percentage of time node cheating is detected

ANM-E. They will be presented in an extended version of this paper.

6. Summary and Future Work

The Tamper-Proof Model (TPM) assumes that the agents participating in the mechanism will not deviate from the distributed implementation. In the Autonomous Nodes Model (ANM), the agents may deviate and thus the existing TPM mechanisms are vulnerable to manipulations. We proposed two distributed SH mechanisms for ANM (SH-ANM and SH-ANM-E). We used digital signatures for authentication of messages and auditing by the root node to penalize the cheating nodes. Thus no node has incentives to cheat. The second protocol SH-ANM-E includes verification done by the children thus detecting the cheating early in the execution and saving time. We proved that both protocols are faithful implementations of the SH mechanism.

We implemented these mechanisms in a real-world environment to analyze the overhead induced by the additional computation and communication resulting from the authentication and auditing procedures. From the experimental results we conclude that SH-ANM protocol does not induce a significant overhead to the multicast transmission.

In the future we plan to study the convergence of the SH mechanism and also develop faithful mechanisms for other distributed computing problems.

References

- [1] Openssl. www.openssl.org.
- [2] Planet-lab. www.planet-lab.org.
- [3] T. E. Carroll and D. Grosu. A strategyproof mechanism for scheduling divisible loads in tree networks. In *Proc. of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2006)*, April 2006.
- [4] J. Feigenbaum, A. Krishnamurthy, R. Sami, and S. Shenker. Approximation and collusion in multicast cost sharing. In *EC '03: Proceedings of the 4th ACM Conference on Electronic Commerce*, pages 280–280, June 2003.
- [5] J. Feigenbaum, A. Krishnamurthy, R. Sami, and S. Shenker. Hardness results for multicast cost sharing. *Theoretical Computer Science*, 304(1-3):215–236, 2003.
- [6] J. Feigenbaum, C. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. *Journal of Computer and System Sciences*, 63(1):21–41, August 2001.
- [7] J. Feigenbaum and S. Shenker. Distributed algorithmic mechanism design: Recent results and future directions. In *Proc. of the 6th ACM Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 1–13, September 2002.
- [8] J. Green, E. Kohlberg, and J. Laffont. Partial equilibrium approach to the free-rider problem. *Journal of Public Economics*, 6(4):375–394, 1976.
- [9] A. Mas-Colell, M. D. Whinston, and J. R. Green. *Microeconomic Theory*. Oxford University Press, New York, 1995.
- [10] J. Mitchell and V. Teague. Autonomous nodes and distributed mechanisms. In *Software Security - Theories and Systems. Mext-NSF-JSPS International Symposium (ISSS 2002)*, pages 58–83. Springer LNCS, 2003.
- [11] H. Moulin and S. Shenker. Strategyproof sharing of sub-modular costs: budget balance versus efficiency. *Economic Theory*, 18(3):511–533, 2001.
- [12] N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Econ. Behaviour*, 35(1/2):166–196, April 2001.
- [13] L. S. Shapley. A value for n-person games. In *Contribution to the Theory of Games*, volume 2, pages 31–40, Princeton, NJ, 1953. Princeton University Press.
- [14] J. Shneidman and D. C. Parkes. Specification faithfulness in networks with rational nodes. In *PODC '04: Proceedings of the 23rd annual ACM Symposium on Principles of Distributed Computing*, pages 88–97, July 2004.
- [15] S. Siachalou and L. Georgiadis. Algorithms for precomputing constrained widest paths and multicast trees. *IEEE/ACM Transactions on Networking*, 13(5):1174–1187, 2005.