Repeated-Game Modeling of Multicast Overlays

Mike Afergan
MIT CSAIL
Email: afergan@alum.mit.edu

Rahul Sami †
School of Information
University of Michigan, Ann Arbor
Email: rsami@umich.edu

ABSTRACT

***Abstract*— This paper studies multicast application overlay networks in a repeated-game framework. In these overlays, users have both the motivation and the means to alter their position in the overlay tree. We introduce a repeated-game model of user behavior that captures the practical tradeoff between a user's short-term desire for quality and long-term desire for the network's continued existence. We simulate overlay tree-formation protocols with this model to study their robustness to selfish users. We show that this model can explain user cooperation and provide insight into how overlay systems scale in the absence of heavyweight mechanisms or identity systems. We also use the model to derive practical guidance on how to make multicast overlay protocols more robust to selfish users.**

I. INTRODUCTION

Many applications, such as live streaming video or stock tickers, require efficient distribution of real-time data. IP multicast is a router-based solution designed to provide a scalable infrastructure for such applications. By contrast, in an *application-layer multicast overlay* network, end-user nodes use IP unicast to create a virtual multicast tree. Such overlays have been examined extensively in the literature and are now beginning to be deployed. Application overlays can approximate many of the same design goals of IP multicast, in particular sizeable gains in network efficiency. Further, they are easier to deploy than IP multicast. Application overlays typically have a much smaller scale and further, ISPs do not need to be involved in the deployment, thereby eliminating many complex cost and pricing issues.

However, because these overlay networks rely on end-user systems, they encounter the issue of user incentives just like other peer-to-peer applications. Nodes deeper in the tree may suffer from increased latency, jitter, or loss. Further, nodes supporting many children may suffer from increased bandwidth or CPU utilization. This may cause nodes to attempt to place themselves higher in the tree or to support fewer children. Even this simple selfish behavior can wreak havoc on the system and its efficiency.

One approach to these incentive issues is to design mechanisms to directly detect or prevent cheating. Proposed tools here include identities, monitoring systems, micro-payments, and reputation schemes. Indeed, a study of the system using standard one-shot game theory suggests that we cannot engender user cooperation without either financial incentives or tight monitoring. However, it is often impossible to implement either payment schemes or tight controls; even if possible, they likely require a heavy infrastructure investment. Further, systems (e.g., [5]) *without* this additional infrastructure often perform reasonably well in practice!

In this paper, we demonstrate that this apparent paradox can be explained by the key observation that *even selfish users want the system to exist in the future*. We use the tools of *repeated game theory* to develop a model of user behavior that captures this positive aspect of user incentives. In our model, a user's benefit depends on her position in the overlay tree. A strategic user evaluates each potential action based on its total expected benefit: the sum of her immediate benefit and a (discounted) future benefit over the expected lifetime of the system. An action to deviate from the protocol might improve her immediate benefit, but have the side-effect of degrading the system performance and hence shortening its expected lifespan. Thus, she will deviate from the protocol only if the immediate gains from cheating exceed the future expected loss.

In contrast to most earlier work on mechanism design for networks, we assume a *minimal infrastructure*. We explicitly assume that there is no central trusted entity, nor any form of a payment system. Further, we do not attempt to devise an effective monitoring, identity, or reputation system. These assumptions reflect the reality of the current Internet. Further, even if such machinery could exist, it would likely come at a significant (monetary or performance) cost. It is therefore important to consider alternative design approaches when possible. With this model, we study the performance of overlay multicast formation protocols in the face of selfish user behavior. We aim to evaluate the protocol performance, and to identify protocol parameters or features that improve robustness to selfish users.

Another key difference of our model is that the repeated-game analysis makes the motivation for cooperation *endogenous* to the model. Whether a user deviates from the protocol is determined by her current and future payoffs, and is thus directly dependent on the system design. In contrast, earlier explanations of cooperation in overlay networks assume that users have an exogenous fixed *type* that determines their propensity to cheat. For example, users are modeled as cheaters or non-cheaters [17], or assigned an altruism parameter [11] [8]. Real user behavior is probably influenced

Portions of this work were reported in an author's PhD thesis [1].
†Author supported in part by NSF grant ITR-0219018. Most of this work was done at the MIT Computer Science and Artificial Intelligence Laboratory.

by exogenous type differences as well as other factors such as bounded rationality. An ideal model would include all these factors. However, we believe that an endogenous model of cooperation is useful for two reasons. (1) It has fewer degrees of freedom than models requiring a distribution of altruism types; this makes it easier to make sharp, testable predictions. For example, our model naturally predicts that, as the number of users grows, the fraction of users who deviate will tend to increase. Such a trend was earlier observed in various systems by Huang *et al.* [12]. (2) Even if real user behavior involves both a system-independent altruism type and a system-dependent endogenous incentive to cooperate, the latter aspect is likely to be more useful to system designers.

We believe these two differences are significant and use our model to derive novel practical insight into multicast overlay formation techniques. Because our model explains a user's willingness to cheat based on her location in the tree, we can compare the robustness of different topologies and tree formation techniques. In simulation, we compare the tree-formation protocol NICE [2] to a naive tree-formation algorithm that maximizes the network efficiency. The results suggest that NICE is more robust to selfish users, and can actually lead to greater efficiency under fairly general conditions. We then consider the impact of a basic NICE parameter, the cluster size. We find that, under reasonable conditions, increasing the cluster size can create more robust, and thus potentially more efficient, trees.

We believe that our assumptions and style of analysis may be applicable to a variety of problems, including other peer-to-peer applications, traditional wireless networks, as well as ad-hoc networking problems.

The rest of the paper is structured as follows. In Section II, we present some background information on overlay trees, focusing on the NICE protocol. In Section III, we explain and motivate the problem. In Section IV, we examine several natural candidate solutions, and argue that they are not useful in this setting. Given this motivation, we present our model in Section V and our simulation methodology in Section VI. In Section VII we present our core results. Because we use a simple and specific model in our main analysis, we then consider a wide range of relaxing assumptions in Section VIII, and see that our qualitative results are insensitive to these assumptions. We summarize our conclusions in Section IX.

II. NICE and Multicast Tree Formation Protocols

The purpose of a tree formation protocol is to connect the user-nodes to the source in an overlay tree that is efficient in multiple dimensions. Desirable properties include scalability, low communication overhead, and fast convergence in the face of nodes churn. Traditional performance metrics include *stress*, the maximum load induced on any router in the network; and *stretch* or relative-delay-penalty (RDP), the ratio of the latency of receiving a packet through the overlay versus directly from the source. Due to differing approaches, requirements, and tradeoffs between these metrics, there are a variety of proposed protocols for this particular overlay problem.

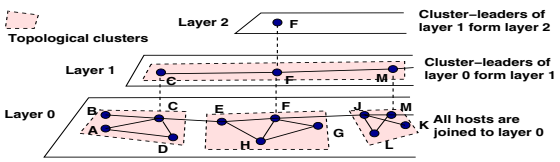


Fig. 1. A three-dimensional depiction of the NICE protocol. Leaders from the lower layers become the members of the higher layers. (Figure taken from [2])

For most of this paper, our analysis focuses on a particular protocol, NICE, which we select for several reasons. For the single-source application we consider, so called tree-first protocols which create singly rooted trees are most appropriate. NICE is a tree-first protocol which specifies several constraints on the tree (discussed below). Those tree-first protocols that do not specify constraints easily permit strategic manipulation. (For example, if a node can select its number of children, as in [5], it can easily select 0.) Finally, relative to other multicast-overlay protocols, NICE has been shown to have good performance across a variety of metrics [2] including communication overhead. While some of these metrics are orthogonal to our study, it makes NICE a good candidate for exploration.

NICE structures the user-nodes into a hierarchical tree of clusters. Roughly, each cluster contains nodes that are close to each other, according to some cost metric. These nodes in turn select the centroid of the cluster to be their leader. Clusters are arranged in a hierarchy of layers, such that the members of a cluster at layer ℓ are leaders of clusters at layer $\ell - 1$, as depicted in Figure 1. As such, all nodes belong to a layer 0 cluster.

Tree maintenance is a completely distributed process. To join, a node descends through the tree, selecting the closest node in a given cluster and then querying it for its children. To balance the tradeoff between stretch and stress, the tree maintains a cluster size parameter, k. If a cluster becomes smaller than k, it merges with a nearby cluster that shares a common parent. Similarly, if a cluster becomes larger than $3k$, the cluster is split, and the two new cluster leaders become children of the old cluster leader's parent. The cluster leaders also periodically attempt to improve the tree, either by transferring leadership to a new centroid or by finding a better parent.

Further details and results on NICE can be found in [2].

III. The Problem

A crucial problem with many tree-formation protocols is that they assume that users will be faithful to the protocol. Unfortunately, users have *both the motive and the means* to alter their location in the tree. The location in the tree can significantly impact the stream quality and the load on a node. Further, in all of these protocols, there are a number of ways to lie and thus alter one's position in the tree. In this section we discuss these motives and means; and examine

some alternative approaches to the problem. Throughout this discussion, we refer to this tree-altering behavior as *cheating*.

A. Why Nodes Want to Cheat

The utility of an end-user is a function of 1) the content (data, streaming video, etc) obtained from the tree; and 2) the responsibilities incurred from participating in the tree. These are in turn defined by the user's location in the tree and the tree structure. Consequently, the tree structure is very important.

For a large class of applications, *a node's proximity to the root significantly impacts the user utility*. In any networked application, processing by intermediate nodes creates the potential for decreased quality, such as increased latency, loss, or jitter. This is especially relevant in the case of overlay networks where the intermediate nodes are assumed to be end-user machines, with potentially limited resources.

Another practical consideration is that *supporting children has a negative impact on a user's utility*. Each additional child supported requires overhead state and resources. While this overhead need not scale linearly with the number of children, its impact can be non-trivial. This can be problematic for several reasons: 1) the user may also care about other tasks also running at that time, 2) the underlying application using the overlay may itself be resource intensive (e.g., streaming), and 3) the hardware and operating system of the machine is likely not optimized for it to function as a server. These concerns are validated in practice, for example, in that many peer-to-peer applications allow users to disable serving as a parent node [5] [15].

The result of these two considerations is significant. In the absence of any enforcement mechanism or concern for the future, these two factors can transform efficient overlay trees into degenerate, unicast trees.

B. How Nodes Can Cheat in NICE

Given the incentive to cheat, it is important to understand how readily a node could eliminate children and/or move higher in the tree. Unfortunately, there are a variety of ways to do one and/or both in most protocols. In some protocols, such as those where the user selects the number of children she wishes to support (e.g., the CMU system [5]) this is trivial. In others, such as NICE, the process is more subtle but still quite easy.

For example, to move higher up the tree:[1]

- A node desiring to be part of a layer ℓ cluster can just claim to be the leader of a layer $\ell - 1$ cluster, even if such a cluster did not exist.
- A node can manipulate the heartbeat messages so as to become leader of a given cluster, and then continue to manipulate while refusing to relinquish leadership. This could be repeated multiple times to move up the tree. (This is discussed in further detail in [17].)

To support fewer children:

- A node can simply refuse to serve its children, or partially-serve them (e.g.,transmit a fraction of the packets).
- A node can delay or drop probes from new children
- When joining the tree, a node can select a cluster where performance is marginally worse but where it is very unlikely to become the leader.

The existence of these tricks should not be viewed as a flaw of NICE. It is designed to operate with faithful users and thus has no protection whatsoever from any of these cheats. Further, some of these cheats (such as some of the RTT manipulations) can potentially be addressed.

However, some manipulation techniques seem to be very difficult, if not impossible, to address. One particularly potent technique is a *Sybil attack*. Sybil attacks involve the creation of multiple identities to benefit the true user. Sybil attacks are particularly potent and particularly difficult to prevent. Amongst other significant practical obstacles for detecting Sybil attacks is the prevalence of Network Address Translators (NATs) – which cause many legitimate users to share the same IP address.

A Sybil attack can easily manipulate a NICE tree. For example, in the case of NICE, a user could simply claim that she is a layer $\ell - 1$ leader and thus should join a layer ℓ cluster. One simple defense would be to require the user to present the IP addresses of the user's children and descendants. However, a Sybil attack here would allow the user to create fake descendants and thwart that defense.

There are several possible, but ultimately incomplete, defenses to Sybil attacks. One simple approach to a Sybil attack is computational puzzles; for example, each parent could require each child to solve one puzzle per descendant. However, in this application, this is unlikely to be an effective solution. The puzzle is forced to be mild because the system must support a wide range of end-user machines, and just a few fake children might be sufficient for a node to cheat. Other approaches that are discussed in the next section include identities and payment schemes.

IV. Why This Problem is Hard

This ability of a node to manipulate the tree formation protocol to its advantage is not unique to NICE. Rather, it is a general problem for multicast overlays that is difficult to solve without additional machinery. Here, we consider several reasonable proposals for addressing this problem and discuss the weaknesses of these solutions. For some approaches, a more in-depth discussion is warranted, but beyond the scope of this paper; here, we highlight the most salient points.

In considering the merits of each approach, we make two key assumptions. First, we assume that only a simple and lightweight solution is appropriate for this application. Clearly the problem of content dissemination could be solved with a variety of architectures.[2] Given that an end-user overlay is

[1] These cheating strategies are not designed to be optimal or the easiest, but rather simple and likely effective. In Section IV we discuss how such strategies may be stopped – but also how some cheats cannot be readily prevented.

[2] One approach is to avoid any incentive problem by using a Content Delivery Network. Here the source pays the CDN, who in turn provides and controls all the server nodes.

being used, we presume that the content and the application are such that a simpler and more light-weight approach is most appropriate. The second assumption is that Sybil attacks from a given IP address can not be detected through direct means (e.g., examining the packets) nor directly prevented (e.g., computational puzzles). In light of these two assumptions, we believe that while each of the following approaches has merit, none satisfactorily addresses this problem.

- *Identities* A direct approach to stopping Sybil attacks is through the use of identities. The identities could be tied to a form of identification difficult to forge (e.g., in-person presentation of a passport), or could simply be costly to obtain. No wide-scale system exists today, but even if such infrastructure were supported by a third-party, it would have to be integrated into the overlay which involves overhead, complexity, and implementation cost. Further, identities may be inappropriate for the application due to privacy concerns or simply the hassle relative to the value of the content.
- *Micro-Payments* The most direct approach to the incentive problem is to introduce payments into the system. Proposals for such systems include leveraging the VCG mechanism [20] or taxation schemes [10]. Any payment system would need some additional infrastructure, and would probably require an identity system as well.
- *Bilateral Monitoring and Agreements* Bilateral monitoring and rewards have been proposed in other peer-to-peer contexts such as peer-storage [6] and file-sharing networks [3]. However, multicast differs from these applications in that the flow of content is *unidirectional*– the parent never receives any content from the child. Therefore, this approach is not directly applicable. It is conceivable that bilateral exchange could be implemented by, say, using concurrent trees with the source alternating packets among the trees. However, in the absence of strong identities, it is also vulnerable to the Sybil attack.
- *Monitoring* Direct monitoring of nodes and their behaviors is an attractive, but difficult proposition. First note that in general there is no way for one node on the Internet to monitor the actions of another. However, one could imagine a system in which the source or indeed every node in the tree performed a series of tests on other nodes. For example, a monitoring node could pose as a new node to understand the tree topology, or could periodically require each node to provide information or pass a series of tests. Many of these approaches could prove hard, complex, and/or costly, and moreover they ultimately can be thwarted with Sybil attacks.
- *Reputations* Reputation schemes combine aspects of monitoring and identities. Many reputation and trust-inference systems have been proposed for peer-to-peer systems (e.g., [13]) and even multicast-overlay networks [14]. These systems are challenged in at least two ways. First, they introduce additional overhead and complexity to the system. Second, there is a significant tradeoff between efficiency and robustness when using the reputation information.
  For example, consider a large number of legitimate users in a single sub-network. The efficient solution is likely form them in a subtree that connects to the tree near the source. In this topology, they will minimally interact with the rest of the system, and thus not earn a high reputation. If they are punished, this is clearly sub-optimal. However, if they are allowed to form such a tree, then the algorithm is vulnerable to a Sybil attack.
- *Intricate Tree Structures* One way to counter the Sybil attack is to build trees that directly minimize the benefits of cheating. For example, the protocol could build trees randomly or ensure that multiple entities at a single IP address are never together in a subtree. These approaches face the same challenge as reputation schemes: such a scheme may mitigate cheating, but at a potentially heavy cost in terms of efficiency.

V. A Repeated Game Model

The negative results of the prior section are troubling for several reasons. They suggest that application overlays are not feasible in practice. Further, they do not explain the fact that, on a limited scale, such applications do exist on the Internet. A more complete model for this problem *must* explain this phenomenon of cooperation, and should ideally do so in a way that aids system designers to engender such desirable cooperation among users.

Our core observation is that *even selfish users want the system to exist in the future*. If multicast is being used, we assume it is to some degree required, or beneficial to the community using it. Reasons for this include a source trying to avoid a large bandwidth bill, or an end-user trying to avoid being shut down by an ISP or IT organization while broadcasting to a large group. In these cases, no user need care about the state of the network *per se*. However, all users want the overlay to continue to exist and thus indirectly care about the network's health. In this section, we show that this dynamic is well captured by a repeated game model.

A. The notion of repeated games

We begin with a brief overview of the concept of a repeated game. For a more thorough summary, consult [16] or [9].

A repeated game is the repeated play of a particular *stage game*. Here, we consider the Prisoners' Dilemma, a canonical example which maps nicely to our problem. Two players simultaneously choose to either cooperate (C) or defect (D), with stage-game payoffs given by Table I as ordered tuples.

TABLE I

Game Payoffs for the Prisoners' Dilemma. Player 1's moves are the rows, Player 2's moves are the columns.

	C	D
C	(1,1)	(-1,2)
D	(2,-1)	(0,0)

In the one-shot stage game, both players will play D. Regardless of what the other does, it is always in the best interest for a particular player to defect. The only Nash Equilibrium (NE) of this game is therefore (D, D). Strikingly, if the players had both played C, *both* players would have been better off.

Now consider the case where the game is played repeatedly. If the players play a fixed number of rounds (known to both players), then the unique equilibrium is still (D,D). However, when the number of rounds is infinite or unknown, other equilibrium outcomes are possible.

For example, consider the following strategy for each player:

1) Play C
2) If the other player ever plays D, then play D forever.

If the game is infinite and players do not discount future periods, then repeatedly playing (C,C) is as an equilibrium outcome. Here, the threat of punishment causes the selfish players to not defect. This class of repeated equilibria results, referred to as the Folk Theorem, can occur even when the game is finite and/or players are impatient.

We can formally analyze this class of equilibria by defining the user's decision making process. The standard approach in repeated game theory is to model the user as attempting to maximize the discounted sum of all rounds' payoffs. Thus, if $p_i^{(k)}$ represents player i's payoff in the kth round, player i's total payoff P_i is given by

$$P_i = \sum_{k=1}^{\infty} \delta^{k-1} p_i^{(k)}$$

Here, the parameter δ satisfies $0 \leq \delta \leq 1$; values of δ strictly less than 1 can be used to capture the probability of the game ending after each round, a real or psychological discounting that player i applies to future payoffs, or some combination of these effects. In this case, the players will cooperate only if δ is sufficiently large relative to the parameters of the game and the particular strategy being played. This model explains the emergence of cooperation among the players in this game and permits analysis of the relationship between the game, strategies, and outcomes.

B. Repeated-game modeling of overlays

We now turn to repeated-game modeling of the multicast overlay systems. Note that a user's options of following the protocol or cheating map to the "cooperate" and "defect" strategies of the prisoner's dilemma, although there are multiple possible cheating strategies. A user's payoff at each step is determined by her position in the multicast tree. The payoffs are represented by her utility function. As discussed, it is natural to assume that a user's utility decreases with increasing depth or increasing number of children to be served.

In our model, we assume that the original need for the multicast network, and thus the indirect motivation for cooperation, is network load. We define network load as the total number of unique packet-hops to deliver all the traffic, which is analogous to average stress. If the load is too high, or close enough to the unicast load, the future existence of the network is in jeopardy. There are several potential reasons for this phenomenon. An ISP or IT department might shut the overlay down if there is too much traffic; the source might stop using an inefficient network; or, the community might abandon the system for a competing system.[3]

The second key aspect of the system is that the performance of the entire overlay network affects this likelihood of it ceasing to exist. We model this by postulating a (*e.g.*, ISP) *Response Function*. This function determines, for a given value of the network load, the probability that the system will be shut down after the current round.

For this feedback to work, the nodes require some signal of the load, and also some knowledge of the response function. In practice, users can obtain a reasonable indication of the network's state from the source, or from traversing the network itself. Similarly, the (non-strategic) source could provide information on the response function. This could be a data-feed or a more crude method, such an email to users regarding the pressure the source is getting from its ISP. Finally, we note that in practice, the users do not even need a noisy signal of the entire response function, but only the local derivative. For simplicity, in our model, we assume that the users have perfect information as to the state of the network and the direct impact of their actions. (We will relax these assumptions in Section VIII.) In our model, users do not consider the impact of their actions on other users' willingness to cheat. (We discuss this assumption more formally in Section VI-I.)

Combining these key components, our model can explain cooperation in overlay networks without relaxing the standard model of an individual as a rational (selfish) utility-maximizer. This idea is deeply rooted in the game theory literature. However it contrasts starkly with the Computer Science literature on this class of problems that uses *exogenous types*. For example, Mathy *et al*[17] models cheating in multicast overlay trees by randomly assigning a given percentage of users to be cheaters. These cheating nodes always cheat and the other nodes are not selfish and always faithful. Similarly, Feldman *et al* look at the related problem of cooperation in the context of peer-to-peer file sharing. They develop threshold [8] and randomized (mixed) [7] strategies which produce a cooperative equilibrium. While the notion of repetition is implicit in their models, the relevant parameters (the threshold or mixing proportion) is again a type exogenous to the model. By contrast, *in our model, the motivation for cooperation is endogenous*. All users are selfish, yet their desire to cheat is tempered by their concern for the network and informed by their current and potential future payoffs. This in turn is a function of the overlay network topology, facilitating analysis of protocol decisions.

[3]In addition to network load, alternative motivations could and likely do exist. These could include concerns about source load or about degradation of quality in the face of fixed capacity and other resources. We believe that these models can be shown to be functionally equivalent. However, we omit such a discussion due to the space constraints, and for clarity.

We believe that a complete model of user behavior likely includes such generosity (captured in altruism types) and non-zero action costs (captured in epsilon-equilibria [4]). Here, we focus on the repeated game, which we feel is a simpler explanation and a more useful tool. The repeated game cleanly explains why these networks work well for few users but do not necessarily scale without additional mechanisms. Further, by making the incentive to cooperate endogenous, it facilitates system analysis and architecture comparison.

C. Formalizing the Repeated Model

Based on these assumptions, we can now model the entire game. We group the definitions and discussions based on what they impact the most. We then present the equilibrium equations.

1) Instance Definition: An instance of the game is defined as follows:

- A network, $G = (V, E)$, V and E are finite.
- A set of nodes N to be served, $N \subset V$
- A single source, $s \notin N$, $s \in V$ which sends a stream of data to all nodes directly connected to it. The source has infinite capacity and constant utility.
- A single, atomic, piece of content to be sent from the source to the end-users.

2) Tree Related Definitions: Given a problem, we have an algorithm, A, which constructs an overlay topology, T.

Definition 1: The load of a given link, ℓ for a topology T and network graph G is denoted as $L(\ell, T, G)$. The load of an overlay topology, denoted $L(T, G)$ or $L(T)$, is defined as the sum of the edge links:

$$L(T, G) = \sum_{\ell \in E} L(\ell, T, G)$$

Definition 2: A *valid* tree is one that satisfies the following properties:

1) All N nodes are connected by $T = A(G)$; and
2) All nodes satisfy the individual rationality constraint, that is, $u_i(d_i, c_i) \geq 0 \ \forall i$.

We now define the ISP response function, which represents the probability that the game will end in a given period. In our analysis we examine particular response functions, but in general, we could use any response function R that satisfies certain properties. We formalize this by defining T_{max} to be the unicast tree. That is, $T_{max} = \{(s, i) \forall i \in N\}$. T_F is defined to be the faithfully constructed multicast overlay tree, adjusted by connecting nodes who are further from their parent than the root to the root. Let $L_F = L(T_F)$ and $L_{max} = L(T_{max})$.

1) $R(L_F) = 1.0$
2) $R(L_{max}) < 1.0$ and $R(L_{max}) \geq 0$
3) $\frac{dR(L)}{dL} \geq 0$ for $L_F \leq L \leq L_{max}$

The first requirement says that the network will exist if users are faithful. [4] The second says the unicast network will continue to exist with some probability strictly less than 1.0. And the third adds that for valid L, $R(L)$ increases with L.

[4]We could allow for $R(L_F) < 1.0$, but choose not to for clarity.

3) User Model: Given an overlay tree, T, we have:

- $d_i(T)$ denotes the depth of node i in tree T, and $c_i(T)$ denotes the number of children that i supports in T.
- $u_i(T) = u_i(d_i(T), c_i(T))$. $u(.)$ is strictly decreasing in both d_i and c_i.

As discussed in Section V, users have knowledge of $L(T)$ and $R(.)$, but not T nor the history of plays that produced T.

4) User Decision: At any point in time, a user may accept the faithful position in the tree (d, c) with associated load L, or cheat and obtain some alternative (d', c') with load L'. By construction we have $d' < d$ and/or $c' < c$. We have $u(d', c') \geq u(d, c)$ but such a deviation will produce a new load $L' > L$. As discussed above, while we later relax these assumptions in simulation, we assume that all users have perfect knowledge of $R(.)$, L, and $L' - L$. We also assume that in practice they can observe δ and c.

If a user accepts a (d, c) (other than $(1, 0)$), decreasing the chances of the network's future existence must outweigh the benefits of changing the position in the tree. We model this formally below. In doing so, we assume that users have a limited understanding of the impact their actions on other users. In particular, each user moves as if her move will be the last. In practice, when one user cheats, this may precipitate more cheating by others. Reasoning about this process is however not simple for a user, especially given the subtleties of protocols such as NICE. Therefore, our model assumes limited rationality – users have no ability to reason about this impact and do not consider it. The effect of this is that users are more likely to cheat – making our results of cooperation conservative. We call this assumption the Restructuring Assumption and formally state it in Section VI-I.

Taking these factors, we have that for a user i not to cheat, the following must hold:

$$\sum_{t=0}^{\infty} \delta^t [R(L)]^t u_i(d, c) \geq u_i(d', c') + \sum_{t=1}^{\infty} \delta^t [R(L')]^t u_i(d', c') \quad (1)$$

The left-hand term represents the discounted payoffs to cooperating. The first right-hand term represents the one-stage payoff to cheating and the last term represents the continuation payoff from that cheat.

We simplify:

$$\frac{u_i(d, c)}{1 - \delta R(L)} \geq u_i(d', c') + \delta R(L') u_i(d', c') * \frac{1}{1 - \delta R(L')} \quad (2)$$

And now put the equation in standard repeated game form:

$$u_i(d, c) \geq (1 - \delta R(L)) u_i(d', c') + \delta R(L') u_i(d', c') * \frac{1 - \delta R(L)}{1 - \delta R(L')} \quad (3)$$

Quick informal analysis of Eqn(3) provides intuition on the impact of the various parameters have on a user's desire to cheat. Informally:

- *Patience of Users* As $\delta \to 0$, the right hand side goes to $u_i(d', c')$. Since $u_i(d', c') > u_i(d, c)$, we see that users become more likely to cheat.

- *Benefit to Cheating* The benefit to cheating is $(u(d', c') - u(d, c))$. As this increases, the right-hand side becomes large relative to the left, meaning that users are more likely to cheat.
- *Cost to Cheating* By contrast, as the cost of cheating $R(L') - R(L)$ becomes large, the final term becomes small, making users less likely to cheat.

VI. The Simulator Methodology

This problem of multicast overlays is complex with many variable attributes including the number of players, utility functions, overlay tree topology, and underlying network topology. As a result of this complexity, it is not possible to theoretically analyze the behavior of the entire system. Therefore, we use simulation to better understand the system dynamics and to gain intuition for design.

In this section, we describe a simulator designed to capture and vary these parameters. First, we overview the simulator. We then discuss several key assumptions implicit and explicit in the simulator design.

A. Overview

The goal of the simulator is to model the decisions, actions, and interactions of self-interested user-nodes who are part of a single overlay topology.

1) The simulator takes a set of inputs:
   - A synthetic network topology.
   - A user utility function $(u(d, c))$
   - A number of nodes (N)
   - A discount factor (δ)
   - A tree formation algorithm
2) The simulator randomly selects a source and N end-user nodes in the topology, and constructs the overlay tree using the specified algorithm.
3) Each node learns its depth and number of children and receives a signal of the efficiency of the overlay system.
4) The simulator considers each node in the topology and allows it to take action. The permissible actions are: 1) connect to root and 2) drop child.
5) Step 4 is repeated until no node wishes to act.
6) Statistics are collected and reported.

B. Implementation of NICE

When the tree formation algorithm used is NICE, the simulator uses a custom-built implementation of the NICE protocol. The NICE modules takes as input a network topology, a set of N nodes, a source, and a cluster size parameter (k). The simulator then builds the NICE tree, bottom up, using a distance metric of hop count, which corresponds to our definition of load (as discussed in Section V-B). As nodes enter and leave the tree, the NICE module performs the appropriate cluster joins and splits as specified by the protocol. Furthermore, to simulate the limit of the constant refinement process, upon any change to the tree, the tree is rebuilt from the bottom (layer 0) of each rooted subtree.

One necessary modification of the NICE protocol is that we allow for multiple top-layer nodes in steady-state. This is done for multiple reasons. In order that no node has negative utility, the depth of the tree and the number of children of any node must be bounded. Yet, the tree protocol must always allow for additional children of the source as the number of nodes increases. Furthermore, if the NICE protocol did not allow for this, a pathological behavior could result where nodes kept increasing their claimed layer (though fabricated nodes) to claim leadership of the top level, thereby creating an instability in the tree. Finally, note that multiple source children *are* allowed in the standard NICE protocol in cases such as partitioned networks. Therefore, it is possible to for cheating nodes to produce such a topology if they need to.

C. Implementation of Cheating

The simulator allows for two potential cheating actions: connecting to the source and dropping a child. As discussed, there are a variety of ways to cheat and a variety of partial defenses that could be used. Our simulator's implementation of the cheats is designed to be simple but representative.

Nodes are free to connect directly to the source whenever they want. When a nodes does so, it maintains the structure beneath itself. As such, this essentially creates another NICE tree rooted at the same source. This tree continues to follow the NICE protocol.

Dropping a child is a slightly more subtle operation due to the complexity in reconfiguring a tree. While there are multiple subtle ways to drop a child, the simplest is to simply stop sending the child any data. In NICE, the orphaned node would attempt to perform a new leader election within its cluster and connect to the higher layer. However, this higher layer could also thwart the orphan. While it is conceivable that we could detect when the leader at the higher layer would also reject it, this procedure of determining the exact reconnection point can be quite complex. Therefore, for simplicity, when a child is orphaned, the simulator reconnects it directly to the source. Further, after dropping a child, the parent creates an artificial child to avoid receiving a new child. Given this behavior, a parent when considering whether or not to drop a child, first considers the child who would generate the least amount of additional load at the source. This maximizes the expected system lifetime, and hence, the future payoff of the parent.

D. Randomization

For a given experiment (which defines N, δ, and $u(.)$), each run is defined by a network topology, a source node, and a set of user nodes. The data presented varies the inputs in two dimensions:

- Network Topology: As discussed in Section VI-G, we use three different network topologies. Each is randomly generated by BRITE, with the same parameters.
- Random Seed: Each run is defined by a random seed. The seed determines the source and the users from the network topology. It also defines the order via which the simulator will iterate through the users.

Unless otherwise noted, each experiment uses 30 random seeds and 3 topologies for 90 runs per data point. To facilitate comparison, we use the same topologies and seeds across all of our experiments.

Most graphs present only the mean value over the trials. However, the individual trials of all graphs are qualitatively similar. Therefore, we feel that the mean is a sufficient metric, unless otherwise noted.

E. Utility Function

The assumptions about QoS and the incentive for cheating is embodied in the utility function. We use utility functions of the following form:

$$u(d, c) = \gamma - \lambda\sqrt{d} - \beta c$$

where γ, λ, and β are all parameters configurable in the simulator. The selection of the square root function for the depth is designed to capture the decreasing marginal disutility of increasing depth. The linear function over the number of children is based on the fact that some of the costs of supporting children (e.g., CPU, bandwidth) scale linearly with load. Finally, we have the constant γ term to capture the fact that while depth and children do matter, another important component of the utility function is simply obtaining the content. Unless otherwise noted, the simulations in this paper use $\gamma = 10$, $\lambda = 1.0$, and $\beta = 0.25$; in Section VIII-A, we discuss a range of alternate utility functions.

F. Response Function

The response function of the ISP plays an important role in the behavior of the system. The simulator does not model the actions of the ISP per se, but rather uses the response function when evaluating the utility of different actions. Unless otherwise noted, the simulator uses the following linear response function:

$$R(T) = \frac{L(UNI) - L(T)}{L(UNI) - L(F)}$$

where $L(.)$ is the load operator, UNI is the unicast tree, and F is the tree produced by faithful users.

Section VIII-B presents an analysis of a wider space of response functions, demonstrating the relative insensitivity of the conclusions to the choice of functions.

G. Network Topologies

To generate the underlying network topologies, the simulator uses the Boston university Representative Internet Topology gEnerator (BRITE) [18] . BRITE uses heavy-tailed models to produce inter-AS and intra-AS models. In our simulations we use the BRITE Barabasi model, a preferential attachment model, for the inter-AS connectivity. For intra-AS (router) connectivity, we use the BRITE Waxman model. All models have 3000 nodes.

H. Alternative Design Decisions

The simulator, as presented, embodies a few key design decisions and assumptions regarding the approach to finding trees and user behavior within the trees. However, we feel that the conclusions drawn are qualitatively similar to those which would have been obtained from alternative sound assumptions.

One implicit decision of the simulator is to construct trees instead of searching the space of potential trees. For many formation protocols, various faithful trees could be constructed depending on ordering. Once built, the ordering of the cheating could also cause different stable topologies result. Therefore, an alternative approach is to construct all possible topologies and evaluate which are stable. This has two downsides in that it is computationally difficult and also provides little justification for why one topology may arise versus another.

Another key decision is whether or not actions, once taken, are permanent. In practice, when a node cheats, for example, inventing children to move up the tree, this cheat need not be permanent. After some period of time, based on the actions other nodes, the node could choose to become faithful. However, such a strategy would likely be quite complicated for a node to implement effectively. Instead, our model takes the approach that cheating is a permanent decision, though (as described above) after cheating the user may still choose to faithfully implement NICE in her new subtree.

I. Proof of Equilibrium Results

For simplicity, the strategies used by the agents here do not anticipate the reactions of other players. Therefore, there may be additional, more efficient, equilibria which are possible. In our analysis, we examine properties of the stable points of the simulator – when no user wants to cheat. Under reasonable assumptions, these stable points are equilibrium outcomes even if players had a perfect ability to understand future reactions.

While a formal proof follows, the intuition is relatively simple. Consider an overlay topology T^* that is stable in the simulator. That is, no player wishes to either root itself or drop a child considering only that action in isolation. If T^* were not an equilibrium outcome (given perfect foresight), then a player must be able to make a beneficial move, considering the reactions of the other players. Since the only difference between these two cases is the other players' reactions, this requires a player i to be able to move in a way that precipitates a move by another player, j, which is beneficial to i. We can show that under reasonable assumptions, this is impossible.

In this paper we consider two tree formation protocols, NICE, which has been already presented, and Naive Min Cost, which we present in Section VII-B.1. One salient feature of NICE is that while most changes to not affect the tree, some changes, over time can produce intricate reorganizations. Thus, in rare cases, it may be possible for a node to move which causes the protocol to further rearrange the tree in a way that benefits the node. We call such events *beneficial induced restructuring events*. In practice, not only are such events relatively rare (since it involves at least two cheating moves

leading to a positive outcome), it would also be very difficult for a node to properly foresee such a move. Further, the move will likely need to involve nodes not nearby in the tree as simple moves (like disconnecting from the parent or having a child leave) can be accomplished directly by the node. We therefore feel the following assumption is very reasonable:

Restructuring Assumption: We assume that no node can cause a beneficial induced restructuring event. That is, there is no action for any player i that initially reduces her total payoff, but induces the tree-formation protocol to restructure the tree, which in turn induces some node j to cheat, and finally results in a topology where the discounted payoff to i is greater than in the original topology.

Again, this assumption is only needed for protocols like NICE which attempt to re-balance the tree. For protocols which produce a static tree or for architectures in which there is no tree formation protocol, this assumption is not required. We can now use the restructuring assumption to formalize the preceding argument into the following theorem.

Definition 3: Let α be the strategy represented by the logic of the simulator. That is, α prescribes that i moves from (d, c) to (d', c') iff:

$$u_i(d,c) \geq (1-\delta R(L))u_i(d',c')+\delta R(L')u_i(d',c')*\frac{1-\delta R(L)}{1-\delta R(L')}$$

Further, per the implementation of the simulator, α first considers rooting itself and then considers child drops, in order from the least rooting cost to the most.

Theorem 1: Let T^* represent a stable point for players in the simulator (playing α). Assume that the restructuring assumption holds. Then, starting from T^*, always playing "Stay" is a Subgame Perfect Equilibrium.

Proof: The action space available to each node is {Drop Child j, Root Self, Stay}. Assume that the strategy of all players always playing "Stay" is not a subgame-perfect equilibrium. Then, there must be player i who can improve her payoff by deviating from this strategy at some point in time. Consider the first player i who can profit from a deviation; it follows that i can profit from the deviation in the first round after reaching topology T^*, because "Stay" plays do not alter the state of the game. Let us call the deviating action – which must be "Root Self" or "Drop Child" – D

Let x be the total payoff to node i from playing "Stay", and let x' be the total payoff to node i from playing D, *assuming that all other nodes continue to play "Stay"*. But this is exactly what the simulator examines and by definition it already considered and rejected D. Thus, if T^* is stable in the simulator, it must be the case that $x \geq x'$.

Now let us consider the impact of playing D on other nodes. Let x'' be the payoff to i, taking into account the impact of playing D on other nodes. Using the restructuring assumption, this payoff cannot be greater than x– there are no indirect benefits to i. Thus, we must have $x \geq x''$ – contradicting the assumption that D was a profitable deviation. ■

VII. Core Results

In this section, we present the three core results from the simulator and discuss their significance for system design. Our three core results are:

1) System efficiency decreases with δ.
2) System efficiency decreases with N.
3) For the NICE protocol, under reasonable assumptions system efficiency for sufficiently impatient users increases with k, the cluster size.

The first two results are fundamental to the thesis of this paper and validate our basic assumptions and model – that concerns about future existence can mitigate cheating in the absence of external enforcement mechanisms. This is in line with standard repeated game results. The third result is novel insight gained specifically from this problem. In short, we find that the NICE protocol naturally exploits the tradeoff between depth and number of children by assigning nodes at higher levels more children. This means that with even mild amounts of selfishness, NICE trees can outperform a centralized min-cost spanning tree protocol. Further, since this tradeoff is already an explicit parameter in the protocol (cluster size), it can be adjusted in various ways.

To provide insight and intuition, the results presented in this section consider a simple model. In particular, we consider the basic utility function (from Section VI-E), no noise, and linear reaction curve. In Section VIII, we relax these assumptions to show that the main conclusions are robust to a wide range of operating environments consistent with our base assumptions.

Many of the graphs in this section plot the impact of a particular parameter on system efficiency. Efficiency in this context is defined to be the fraction of the improvement provided by the overlay that is realized in the face of selfish users. Formally, we define it as:

Definition 4: The efficiency of a particular tree, T, is defined as

$$\frac{L(U)-L(T)}{L(U)-L(F)}$$

where $L(.)$ calculates the network load of a tree, U is the unicast tree, and F is the overlay tree for the same network and users, with all users being faithful.

A. Cooperation in Practice

As expected, system efficiency decreases with δ. This can be seen simply in Figure 2, which examines a topology of 50 nodes.[5] The steep slope as $\delta \rightarrow 1.0$ results from the $\frac{1}{\delta}$ terms in Equation (3). It is also useful to note that $\delta = 0$ corresponds to the negative result from the one-shot game presented in Section IV.

Our second core result is that efficiency decreases with the number of nodes. This is important to understand as it is an important dimension to consider for scalability concerns. Figure 3 builds on Figure 2 by adding $N = 10$ and $N = 100$. Increasing N has two impacts on efficiency – it increases the

[5]As discussed previously, each data point represents the average of 90 separate randomly generated trials.

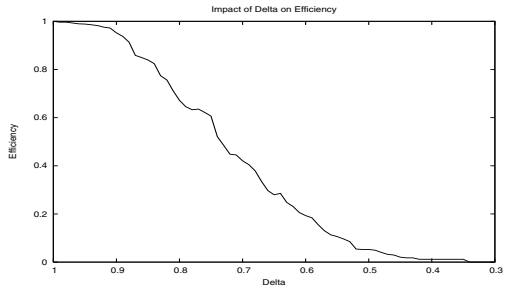


Fig. 2. Efficiency versus Delta. Note that $\delta = 0$ corresponds to the one-shot game.

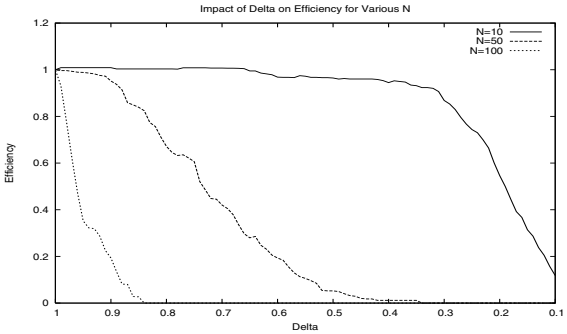


Fig. 3. Efficiency versus Delta for N $\in \{10, 50, 100\}$.

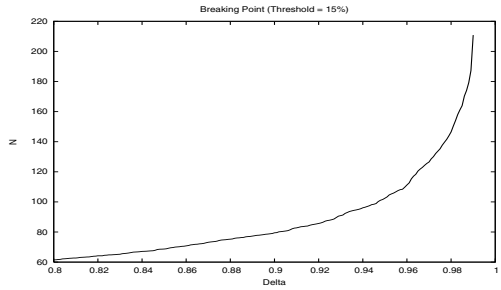


Fig. 4. Breaking Point as a Function of δ. The breaking point is the minimum value of N such that efficiency falls below 15%

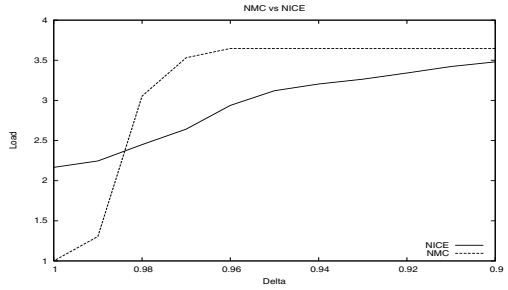


Fig. 5. Performance of Naive Min-Cost vs NICE. While the NMC is superior for faithful users, with even modest discounting NICE performs better.

minimum δ at which cheating occurs and increases the rate at which efficiency approaches zero.

An alternative way to view the impact of the number of users is to examine the maximum value of delta such that efficiency falls below a given threshold. We define this critical point to be the "breaking point." Figure 4 plots the breaking point as a function of delta with a threshold of 15%. For example the graph contains the point (0.94, 100). This means that with $\delta = 0.94$ the maximum number of nodes in a network such that the efficiency is at least 15% is 100. As would be expected, the curve is very steep as $\delta \to 1.0$.

B. Robustness of Tree Formation Protocols

The first two core results beg the question *"How does the shape of tree impact its robustness given impatient users?"* Of course different tree formation algorithms and protocols can form trees with vastly different structure. Thus, instead of heavyweight mechanisms (e.g., payment schemes or monitoring) an alternative approach to building more robust trees is to alter the protocol or algorithm appropriately.

1) NICE vs Naive Minimum Cost Spanning Tree: To begin our analysis, we compare NICE to a centralized algorithm which creates the minimum cost spanning tree. As in [19], this centralized algorithm may be run at the source and with faithful nodes, will minimize the cost metric, network load. Since selfish agents can perturb the tree, we instead call this algorithm Naive Min-Cost Spanning Tree (NMC).

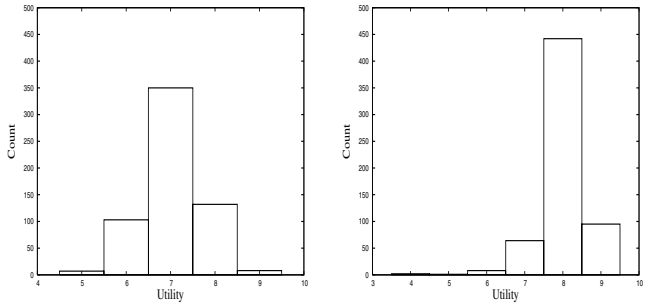
There are however several ways in which a node can cheat this centralized algorithm, depending on the particulars of the algorithm. If the nodes simply report their distances, as in [19], then a cheating node can deflate its distance to the root and inflate its distances to its neighbors [17]. Alternatively, if the algorithm limits the number of children assigned to a node, the basic NAT-based Sybil attack will enable the node to have no children. Further, if the algorithm attempts to minimize or otherwise bound the depth of the tree (to reduce stretch or to preserve quality), the NAT attack will enable a node to connect directly to the root. For the purposes of this analysis, we assume that one or more of these techniques will permit the action space that we simulate for NICE – a node can drop any children and/or connect to the root if desired.

Not surprisingly, for $\delta = 1.0$, NMC outperforms NICE, as can be seen in Figure 5. The fact that NICE performs worse is based on the fact that NICE makes myopic decisions based on only local information. However, note that as in [2], NICE still performs relatively well.

However, we see that NICE is far more robust, and thus far more efficient, when faced with selfish users. Figure 5 plots the load on the network for both NMC and NICE relative to the load of the minimum cost tree, varying delta.[6] While the NMC algorithm outperforms NICE for $\delta = 1.0$, its efficiency rapidly decreases. NICE is therefore able to perform better for a wide range of δ.

Careful inspection of the tree structure lends insight into why NICE is robust but NMC is fragile to selfishly minded users. With the NICE trees, a cluster leader at depth k is by

[6]Because we are comparing two different topologies, we plot load on the y-axis.



(a) Distribution of Utility for the Naive Min-Cost Algorithm (μ = 7.46, σ = 0.61)

(b) Distribution of Utility for the NICE Algorithm (μ = 8.37, σ = 0.54)

Fig. 6. Utility Distribution for NMC and NICE at $\delta = 1.0$. NICE induces homogeneity across positions in the tree.

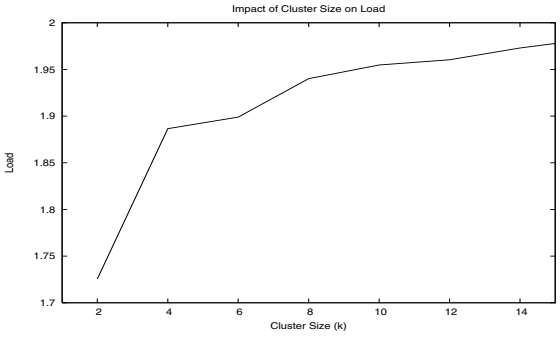


Fig. 7. Load of NICE versus Cluster Size (k) at $\delta = 1.0$ (normalized to the load of NMC at $\delta = 1.0$.)

definition a cluster leader at all depths $k' > k$. As such, the benefits to being higher in the tree are mitigated by supporting more children. In contrast, the minimum cost tree has no such guaranteed structure. While of course it is possible by sheer luck to have a robust topology, this is unlikely. We can further appreciate this relationship by looking at the distribution of utilities in the faithful trees. Figure 6 the per node utilities (bucketed to the integer value) of the node at tree-creation time. Here we see that the disparity in user utilities is greater for the NICE algorithm (μ = 8.37, σ = 0.54) than for the Naive Min-Cost Algorithm (μ = 7.46, σ = 0.61).

2) Applying the Lessons to NICE: This intuition can be used to increase the robustness of NICE itself. A structured tree where nodes near the top are asked to bear an appropriate fraction of the burden is more robust against user incentives. This suggests that just as we gained robustness moving from NMC to NICE, we can, in some cases, also gain by increasing the cluster size.

Recall from Section II that the cluster size of the NICE algorithm defines the tree structure and the efficiency of the tree when users are faithful. Figure 7 depicts this relationship. In general, we find that low k, 2 or 3, minimizes load. (While [2] and [17] do not present an analysis of load, they do use

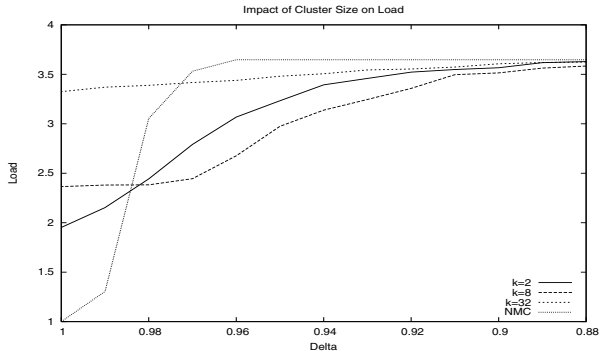


Fig. 8. Load of NICE $k \in \{2, 8, 32\}$ and NMC versus δ (normalized to the load of NMC at $\delta = 1.0$.)

$k = 3$.) Obviously for $k > N$, we essentially have a unicast tree with the source sending the stream to one node who then sends it to everyone else. For $N > k > 3$ efficiency degrades.

However, Figure 8 shows how load scales as we decrease delta for various k and the NMC algorithm. A load of 1.0 represents the load of NMC at $\delta = 1.0$. The topologies with larger k degrade less perform worse for faithful users but degrade significantly less as $\delta \to 0$. This suggests that over a range of δ values, while increasing k may decrease efficiency for faithful users, it can be used to increase the robustness – and thus the realized efficiency – of the system.

VIII. Insensitivity of Results

While the previous section examines a single model, the core results and intuition presented apply to a much larger space of assumptions. In particular, for simulation, we had to assume specific forms for the utility function and the response function. In this section, we investigate to what extent our results are artifices of our specific assumptions, and to what extent they capture general characteristics of the system itself. To do this, we study the impact of changing the parameters of the utility function and the shape of the response function, and also of adding noise to the response that users receive about the performance of the system. For each change, we find that the key qualitative results are unchanged, although the quantitative results do change.

A. Changing the Utility Functions

Recall that the utility function used in the paper thus far is:

$$u(d, c) = \gamma - \lambda\sqrt{d} - \beta c$$

with $(\gamma, \lambda, \beta) = (10, 1.0, 0.25)$. This function captures our assumptions regarding the importance and direction of depth and number of children on the user's experience. However, the intuition and conclusions presented extend to a large class of utility functions that correspond to our assumptions.

Figure 9 plots efficiency versus delta for a variety of parameter settings for the utility function. (Each utility function is represented in the legend by a (γ λ β) tuple.) As the parameter values increase, the rate at which efficiency diminishes also

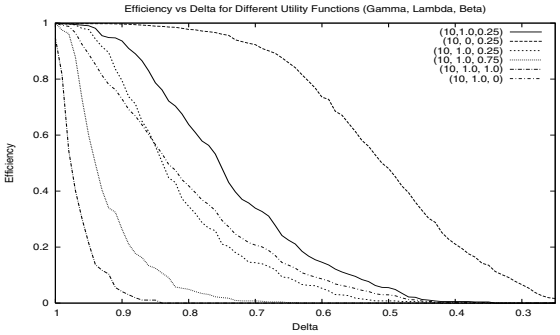


Fig. 9. The relationship between efficiency and delta for N=100. Each line is represented by a (γ λ β) tuple.

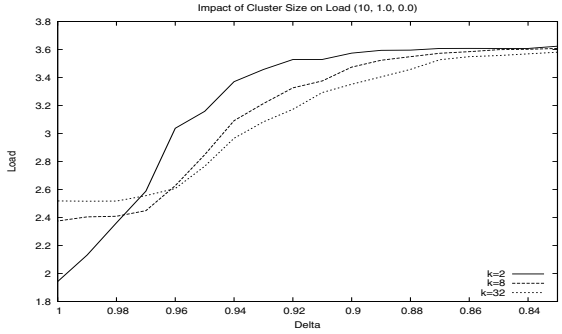


Fig. 10. Sensitivity of k with Alternative Utility Function

increases. This is as expected as higher parameters correspond to a greater incentive to cheat, and vice versa.

We also examined the impact of the utility function on the relationship between k and efficiency, over the same range of utility functions. In all cases, we found that at $\delta = 1.0$ the smaller values of k perform better. However, as δ increases, this performance difference can significantly diminish, and in some cases become inverted. Due to space constraints, we only include the graph for the parameter values $(10, 1, 0)$ in Figure 10. (Note that the scales have been adjusted as appropriate to provide a clearer view of the data.) In this case, as with our original utility function, $k = 8$ is the most efficient value over a large range of δ.

We also consider the extreme end of the parameter spectrum with $\beta = 0.5$ and $\beta = 0.75$. For perspective, moving from a depth of $d = 2$ to $d = 1$ increases utility by $\sqrt{2} - \sqrt{1} \approx 0.41$ and from $d = 3$ to $d = 2$ is $\sqrt{3} - \sqrt{2} \approx 0.32$. Therefore, $\beta = 0.5$ implies that dropping a child is always more important than moving up the tree and that for $\beta = 0.75$ it's more than twice as important to drop a child than to move from depth 3 to depth 2. Since it hard to support children in this space, it is not surprising that $k = 8$ and $k = 32$ perform relatively poorly. However, even here, we found that the gap between $k = 2$ and the other settings diminishes quickly and that for a range of δ, $k = 32$ outperforms $k = 2$.

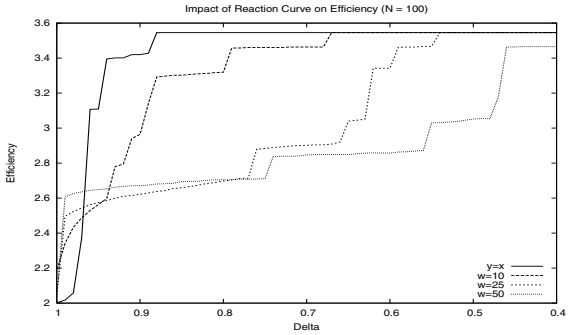


Fig. 11. The Impact of Different Response Functions on Efficiency and Robustness

B. Alternative Response Functions

The analysis above uses a linear reaction curve. This embodies the assumption that all load is equally harmful and noticed. In some circumstances, a threshold-like behavior might be more appropriate: As the load increases, there is initially no effect on the expected lifetime of the system, but if the load crosses a critical value, the expected lifetime drops sharply. In this section, we use a parametrized sigmoid function to explore the range between linear and threshold functions:

$$r(x) = 1 - \frac{1}{1 + exp(\omega(x - .5))}$$

The ω parameter serves to modulate the slope of the function; for higher ω, the response is more non-linear and threshold-like.

Figure 11 plots efficiency versus delta. Here we see two effects of the sigmoid. The flat, initial, region of the response function causes an increased amount of initial cheating. However, the sharp drop in the probability helps to induce cooperation and faithfulness. As ω increases, both of these factors become more prominent.

C. Information and Noise

The simulator embodies two key assumptions about information. First, we assume the nodes have perfect information about the current level of efficiency in the network. Second, we assume that the nodes perfectly understand the reaction curve. It is more reasonable to assume that in practice, the nodes will have a noisy signal of both.

To implement such noise in the simulator, we introduce a noise term, $\epsilon \in [0, 1]$. Instead of assuming that the load signal, ℓ is correct, the agent treats the signal as noisy and instead assumes that the true load, $\hat{l} \in U(\ell - \epsilon\Delta, \ell + \epsilon\Delta)$ where $U(.)$ is the uniform distribution and $\Delta = L(U) - L(F)$, the improvement over unicast provided by faithful users. Therefore, when evaluating the current and/or potential scenarios, the agent will take the expected value over this range.[7] While other noise

[7]There are two important implementation notes. First, the expected value is calculating by evaluating the given function at a discrete number of points in the range and averaging. Second, only points in the range [L(Faithful), L(UNI)] are considered. For points outside that range, the nearest end-point of the range is used.

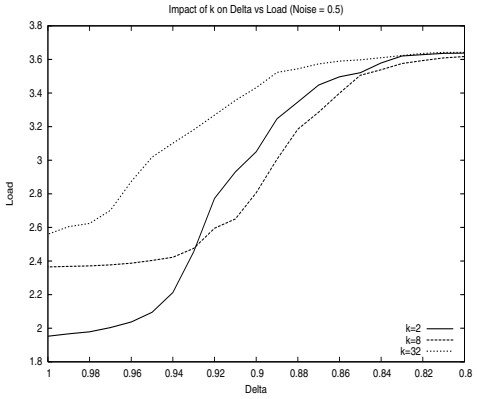


Fig. 12. Impact of k with Noise

models could be used, this model captures uncertainty both in the current position and in the exact nature of the reaction curve with a single parameter. We consider the impact of k as a function of noise and see that the fundamental relationships still hold; we depict the results for $\epsilon = 0.5$ in Figure 12.

In this model, the impact of noise varies on the reaction function, but does not qualitatively change the qualitative results of the model. We saw that with the linear response function, increased noise causes users to be more cautious – and thus more faithful. By contrast, the increased noise diminishes the impact of the sigmoid. This is expected, as the addition of noise has the effect of smoothing out the sharp drop in the sigmoid.

IX. Conclusions

Application overlay multicast has the potential to be a lightweight alternative to IP multicast. In order to design robust overlays we need to take into account the users' incentives. In this paper, we have described an approach to analyzing user incentives using repeated games. The critical feature of our model is that it captures the endogenous, system-induced, motivation for users to cooperate. This allowed us to identify system features and protocol parameters that can increase the robustness to selfish users. We analyzed the NICE protocol, and found that its layered branching structure helps to make the protocol more robust to selfish users; further, under reasonable assumptions, the cluster-size parameter can be used to make the protocol even more robust.

One important direction for future work is to conduct further empirical validation and calibration of the predictions made by this model. This would be very useful to measure to what extent real user cooperation is system-dependent, and then build a composite model that includes both altruism and repeated-game effects.

Acknowledgments

We would like to thank Robert Beverly, Steve Bauer, Dave Clark, Dina Katabi, Drew Fudenberg, and John Wroclawski for their valuable comments and suggestions.

References

[1] Afergan, M. *Applying the Repeated Game Framework to Multiparty Networked Applications*. PhD thesis, MIT, August 2005.

[2] Banerjee, S., Bhattacharjee, B., and Kommareddy, C. Scalable Application Layer Multicast. In *ACM SIGCOMM 2002*.

[3] Beverly, R. Reorganization in Network Regions for Optimality and Fairness. Master's thesis, MIT, August 2004.

[4] Christin, N., Grossklags, J., and Chuang, J. Near Rationality and Competitive Equilibria in Networked Systems. In *PINS '04: Proceedings of the ACM SIGCOMM Workshop on Practice and Theory of Incentives in Networked Systems* (New York, NY, USA, 2004), ACM Press, pp. 213–219.

[5] Chu, Y.-H., Ganjam, A., Ng, T. S. E., Rao, S. G., Sripanidkulchai, K., Zhan, J., and Zhang, H. Early Experience with an Internet Broadcast System Based on Overlay Multicast. In *USENIX Annual Technical Conference, General Track* (2004), pp. 155–170.

[6] Cox, L. P., and Noble, B. D. Samsara: Honor Among Thieves in Peer-to-Peer Storage. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (Bolton Landing, NY, USA, October 2003), pp. 120–132.

[7] Feldman, M., Lai, K., Stoica, I., and Chuang, J. Robust Incentive Techniques for Peer-to-Peer Networks. In *EC '04: Proceedings of the 5th ACM Conference on Electronic Commerce* (2004), ACM Press, pp. 102–111.

[8] Feldman, M., Papadimitriou, C., Chuang, J., and Stoica, I. Free-riding and Whitewashing in Peer-to-Peer Systems. In *PINS '04: Proceedings of the ACM SIGCOMM Workshop on Practice and Theory of Incentives in Networked Systems* (2004), ACM Press, pp. 228–236.

[9] Fudenberg, D., and Tirole, J. *Game Theory*. MIT Press, 1991.

[10] hua Chu, Y., Chuang, J., and Zhang, H. A Case for Taxation in Peer-to-Peer Streaming Broadcast. In *PINS '04: Proceedings of the ACM SIGCOMM Workshop on Practice and Theory of Incentives in Networked Systems* (New York, NY, USA, 2004), ACM Press, pp. 205–212.

[11] hua Chu, Y., and Zhang, H. Considering Altruism in Peer-to-Peer Internet Streaming Broadcast. In *NOSSDAV '04: Proceedings of the 14th International Workshop on Network and Operating Systems Support for Digital Audio and Video* (New York, NY, USA, 2004), ACM Press, pp. 10–15.

[12] Huang, E., Crowcroft, J., and Wassell, I. Rethinking Incentives for Mobile Ad Hoc Networks. In *PINS '04: Proceedings of the ACM SIGCOMM Workshop on Practice and Theory of Incentives in Networked Systems* (New York, NY, USA, 2004), ACM Press, pp. 191–196.

[13] Kamvar, S. D., Schlosser, M. T., and Garcia-Molina, H. The Eigentrust Algorithm for Reputation Management in P2P Networks. In *WWW '03: Proceedings of the 12th International Conference on World Wide Web* (New York, NY, USA, 2003), ACM Press, pp. 640–651.

[14] Lee, S., Sherwood, R., and Bhattacharjee, S. Cooperative Peer Groups in NICE. In *Proceedings of INFOCOM 2003* (2003).

[15] LimeWire User Guide. http://www.limewire.com/english/content/ug_options.shtml.

[16] Mas-Colell, A., Whinston, M. D., and Green, J. R. *Microeconomic Theory*. Oxford University Press, New York, 1995.

[17] Mathy, L., Blundell, N., Roca, V., and El-Sayed, A. Impact of Simple Cheating in Application-Level Multicast. In *Proceedings of IEEE Infocom* (2004).

[18] Medina, A., Lakhina, A., Matta, I., and Byers, J. BRITE: Universal Topology Generation from a User's Perspective. Tech. Rep. 2001-003, 1 2001.

[19] Roca, V., and El-Sayed, A. A Host-Based Multicast (HBM) Solution for Group Communications. In *ICN '01: Proceedings of the First International Conference on Networking-Part 1* (London, UK, 2001), Springer-Verlag, pp. 610–619.

[20] Selwyn Yuen, B. L. Market-driven Bandwidth Allocation in Selfish Overlay Networks. In *Proceedings of INFOCOM 2005* (2005).