

The Orchard Algorithm: Building Multicast Trees for P2P Video Multicasting Without Free-Riding

Jan David Mol, Dick H. P. Epema, and Henk J. Sips

Abstract—The main purpose of many current peer-to-peer (P2P) networks is off-line file sharing. However, a potentially very promising use of such networks is to share video streams (e.g., TV programs) in real time. In order to do so, the peers in a P2P network who are interested in the same video stream may employ Application Level Multicasting (ALM). In existing P2P networks, peers may exhibit behavior which is problematic for ALM: they are not always willing to donate resources (free-riding), and they may arrive and depart at a high rate (churn). In this paper we propose the Orchard algorithm for creating and maintaining ALM trees in P2P networks, which deals with both these problems. By employing a technique called Multiple Description Coding, we split a video stream into several substreams. Orchard creates a dynamic spanning tree for each of these substreams in such a way that in the resulting forest, no peer has to forward more substreams than it receives. Based on an analysis of the expected performance of Orchard and on experiments in a real system, we find that Orchard is capable of maintaining a multicast forest, even when peers join and leave the forest at a high rate.

Index Terms—Distributed algorithms, free-riding, multicast channels, multiple description coding, peer-to-peer.

I. INTRODUCTION

ALTHOUGH the main purpose of many current peer-to-peer (P2P) networks is off-line file sharing, such networks may also provide suitable environments for multicasting high-quality video streams over the Internet. A multicast solution which aims for deployment on the Internet should take into account the behavior of peers as observed in existing P2P networks, for example, *free-riding* [1] (peers unwilling to donate resources) and *churn* [1], [2] (peers arriving and departing at a high rate). While multicasting solutions exist which address these problems [3]–[6], they either require a central authority or a distributed trust system to deal with free-riders. In this paper, we propose the Orchard algorithm for building and maintaining multicast trees for video streams in P2P networks, which deals with both free-riding and churn without the need for a central authority or a distributed trust system.

Orchard assumes a video stream to be split up into several substreams, and builds a separate spanning tree for each of these in such a way that in the resulting *forest*, the required outgoing bandwidth of every peer does not exceed the required incoming

bandwidth. In general, in an environment which is prone to errors (e.g., due to congestion in the Internet), it is advantageous to split up video streams into multiple substreams and forward these independently. In this paper we assume the use of Multiple Description Coding (MDC) [7] for this purpose. With MDC, a peer can continue viewing a video as long as it receives at least one substream, with the viewing quality increasing for every additional substream received.

For multicasting the substreams we use Application Level Multicasting (ALM), because multicasting at the network layer has been found to be problematic [8]. However, existing P2P networks suffer from free-riding and churn, which create problems for ALM. In conventional ALM, a single spanning tree is constructed over the nodes (peers) interested in a video stream, and the burden of forwarding the stream is on the interior nodes of the tree. A peer can avoid this burden and free-ride simply by refusing to forward any data. In addition, when a peer fails, the peers in the subtree below it stop receiving the video stream, making the single-tree approach vulnerable to churn. To solve the problem of free-riding, peers have to be given incentives to share their resources. This was recognized in BitTorrent [9], a download protocol in which peers exchange pieces of a file on a tit-for-tat basis. In Orchard, we combine MDC with the bartering spirit of BitTorrent. Orchard's primitives for building the spanning trees enforce that each peer has to forward a substream for every substream it wants to receive (with some minor exceptions). In the resulting forest, no peer has to forward more data than it receives, which protects the system against free-riders, and, by using multiple trees, against churn.

This paper is organized as follows. In Section II, we further specify the problem we address. We present the Orchard algorithm in Section III. In Section IV, we discuss how Orchard prevents free-riding and how it behaves under several other well-known types of attack. An analysis of the expected performance is presented in Section V. Our experimental results are presented in Section VI. Finally, we discuss some related work and draw conclusions in Sections VII and VIII, respectively.

II. THE PROBLEM

We will now specify the problem of ALM of video streams split up into multiple substreams in P2P networks. First, we will describe the required pre-processing of video streams. Next, we will present the assumptions we make about the underlying P2P network, and finally, we will give our problem statement.

A. Stream Pre-Processing

Before a video stream is distributed, it is split up into some number of substreams called *descriptions* with the technique of Multiple Description Coding (MDC) [7]. We will assume that

Manuscript received November 14, 2006; revised July 20, 2007. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Hui Zhang.

The authors are with the Department of Computer Science, Delft University of Technology, Delft, 2628 CD, The Netherlands (e-mail: j.j.d.mol@tudelft.nl).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TMM.2007.907450

these descriptions require equal bandwidths (with C descriptions, a fraction of about $1/C$ of the bandwidth of the original stream), and that they are of equal value regarding their contribution to the viewing quality. With MDC, a peer will be able to play a video as long as it receives at least any one of the descriptions. For ease of discussion, each description will be represented by a *color* (not to be confused with a color in images of the video stream). Splitting the video stream adds overhead to the data stream as well as complexity to the encoder and decoder. This overhead highly depends on the video content, the video encoding, and the number of descriptions used [10]. In our experiments, the original video stream is split up into four MDC descriptions, which is a reasonable compromise. If too few descriptions are used, the loss of one description causes a big drop in quality. If too many descriptions are used, the overhead is too large.

B. Underlying Peer-to-Peer Network

We assume the existence of a P2P network in which every peer can connect to every other peer. Every peer has sufficient incoming as well as outgoing bandwidth to receive and send all descriptions simultaneously. This implies that peers which cannot contribute as much as they consume are barred from receiving the video stream. The bandwidth bottlenecks are assumed to be at the end-user, not in the core network.

When a peer becomes interested in a video stream, it will try to contact other peers who are already receiving one or more descriptions. For this purpose, we assume that interested peers maintain a *neighbor set* of other interested peers, which are selected at random. The neighbor relation is symmetric: if a is a neighbor of b , b is also a neighbor of a . A peer is assumed to be able to keep his neighbor set populated with at least m peers. When a neighbor departs and a peer has $m - 1$ neighbors left, a new neighbor is added to bring the number of neighbors back to m . In our experiments, we use $m = 20$.

Mechanisms for discovering other interested peers are outside the scope of this paper. A possible way to do this is by using epidemic information dissemination [11], [12]. Another way is presented in [13], in which information is pushed to a random subset of peers in the underlying P2P network, and peers looking for that information query a random subset of peers. This method provides a high probability of interested peers discovering peers which receive the stream. In our experiments, for convenience, we use a central server for peer discovery (see Section VI-A).

C. Problem Statement

We suppose that there is a special peer s (the *source*) that has a video stream available for multicasting. We want to design an algorithm that creates a set of multicast trees rooted at s , one for every MDC description, with the peers interested in the video stream as the nodes in all these trees.

We focus on data dissemination and stay codec and content agnostic. What constitutes acceptable frame loss and stream latency highly depends on the codec as well as the content, so we will refrain ourselves from using them. Instead, the main performance metric we will use to assess our algorithm is the (average) number of descriptions received by the peers.

We want our algorithm to satisfy the following conditions.

- 1) *Fairness*: No peer forwards more descriptions than it receives.
- 2) *Decentralization*: A peer does not need to have s as one of its neighbors.
- 3) *Resilience*: Peer arrivals and departures do not severely impact the number of descriptions received by other peers.

III. ORCHARD ALGORITHM

In this section, we will give a detailed description of the Orchard algorithm for constructing a forest of multicast trees for distributing video streams from a single source. First, we will present the forest-building primitives of Orchard. Then we show how the forest can be repaired when peers depart. Finally, we will present some observations on the structure of the trees in Orchard.

A. Constructing the Forest

An Orchard forest for a source s is represented by a directed graph with a set of nodes (peers) P and a set of links L . Each node $a \in P$ represents a peer, and each link $e = (a, b) \in L$ represents the forwarding of a description from peer a to peer b , with $a, b \in P$. Every link and every peer is assigned a single color. Each link in L is given the color of the description sent over it. The subset of links of a certain color and the peers they connect form the multicast tree for a single description. All the trees are rooted at the source s . Each peer gets the color of the first description it receives. As long as a peer is not part of any tree, it will be considered to have the special color *blank*. The source always has the special color *white*.

When a peer p wants to join the Orchard forest, it will try to become a member of all multicast trees. In order to do so, p will query every peer q in its neighbor set to see whether q is willing to strike a *deal* with it until it has joined every tree or until its neighbor set is exhausted. The multicast forest is constructed using three types of deals.

- 1) *Join at the source* is a deal a peer can strike with the source. The source forwards one description to each of the first few peers that arrive in the system.
- 2) *Exchange descriptions* is a deal in which peers exchange descriptions in a pair-wise fashion in order to obtain more descriptions.
- 3) *Redirection* is a deal in which a neighbor of a peer p redirects one of its outgoing links through p in order to have p obtain more descriptions. We will define two variations of this type of deal depending on whether p was still blank before the deal or not.

As we will see in the next section, the mechanism of deals ensures that peers contribute as much outgoing bandwidth as they consume incoming bandwidth, with the exception of the join-at-the-source deals: The source is willing to forward descriptions without expecting anything in return, and a few peers will be so lucky as to get a description for free from the source. The source makes sure it forwards every description at least once. A deal will exist as long as both parties keep their part of the deal, and each peer keeps track of the deals it has with other peers.

To strike these deals, peers exchange control information and send deal requests. Every peer exchanges updates with its neighbors about its color and the colors it receives. If a peer receives

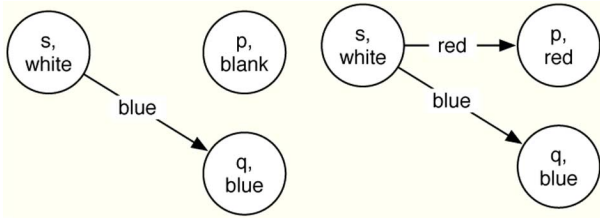


Fig. 1. Joining at the source: before and after peer p joins at the source s .

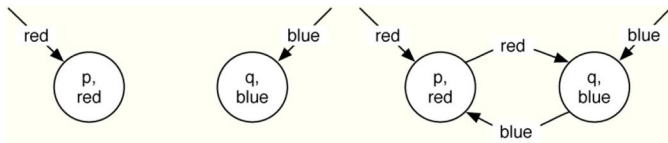


Fig. 2. Exchanging descriptions: before and after two peers strike an exchange deal.

a deal request, it will answer whether it will accept the deal. If a deal is accepted, information required by the requesting peer, such as what color needs to be forwarded to whom, is sent along with the accept message.

B. Primitives to Build the Trees

In this section we will describe each of the three types of deals in more detail. To obtain its first color, a peer has to strike either a join-at-the-source deal or a redirection deal. For every additional color, a peer has to strike an exchange deal or a redirection deal. To distinguish whether a peer strikes a redirection deal for its first or for additional colors, we will refer to the latter as *redirection through colored peers*. Every time a peer looks to strike a deal, it orders its neighbors on increasing hop-count, preferring to strike a deal with a neighbor close to the source.

1) *Join at the Source*: If a peer p has the source in its neighbor set, and the source has spare outbound capacity, the source will forward one of the descriptions to p and the peer joins the corresponding tree at the source. The source will favour sending a description to p that it is not yet forwarding to any other peer. Peer p will then get the color of that description. An example of this is shown in Fig. 1; here, the source s decides to forward the red description to p . The source will refuse to forward more than one description to the same peer.

2) *Exchange Descriptions*: To obtain a description it is not yet receiving, a peer p checks its neighbor set for a peer q such that p does not receive q 's color and q does not receive p 's color. When p has found such a peer q , they strike an *exchange deal*: peer p forwards its color to q and vice versa, as shown in Fig. 2.

3) *Redirection*: If a peer p cannot join any tree at the source and is still blank, it cannot exchange descriptions because it does not receive any. It will then ask each of the non-blank peers in its neighbor set whether it can redirect through p one of the streams it is forwarding, as is shown in Fig. 3. Here, peer q strikes a *redirection* deal with peer p , which replaces the link e with two links $f = (q, p)$ and $g = (p, r)$ of the same color as e . This way, q does not require additional bandwidth, and p is asked to receive the color and also to forward it.

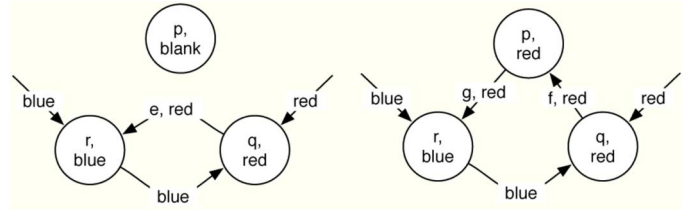


Fig. 3. Redirection: before and after peer p joins the multicast tree below q .

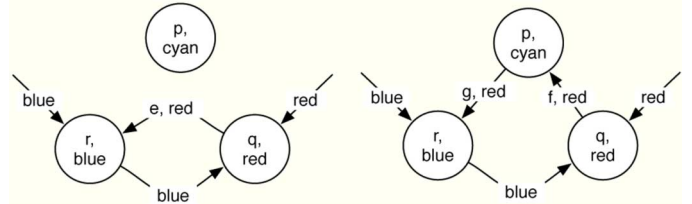


Fig. 4. Redirection through colored peers: before and after peer p joins the multicast tree below q .

Repeatedly redirecting a link $e = (q, r)$ would create a chain of peers, which is undesirable—a chain of peers between q and r would result in an increased latency as well as an increased chance of failure along the chain. To avoid this, only links which were created by an exchange deal can be redirected. So in Fig. 3, peer q cannot redirect link f , because it does not have an exchange deal with p . The same holds for p and link g .

Once a peer p has struck a redirection deal, it receives and forwards a description, and it gets the color of that description. Even though p has selected the peer closest to the source that was willing to strike a redirection deal, it is possible that p will later encounter another peer t of the same color that is even closer to the source. In that case, p switches parents by breaking the redirection deal with q and striking a redirection deal with t .

4) *Redirection Through Colored Peers*: The algorithm so far makes peers depend on exchange deals to obtain additional colors. However, not all peers will be able to obtain every additional color this way. Because a peer may not find appropriate neighbors to strike exchange deals with.

This problem could be solved by letting every peer connect to a large set (for example, 100) of neighbors, but this is undesirable from a practical point of view: if a peer has to connect to and query many neighbors in order to obtain colors, it takes longer to obtain them, especially if connecting involves DNS lookups and getting past firewalls. Also note that a large neighbor set requires more maintenance, as the chance increases that some neighbors will depart. For these reasons, Orchard has another way for peers to obtain additional colors: we also allow *redirections through colored peers*. This results in the transition shown in Fig. 4, which is similar to the one shown in Fig. 3, except that p already has a color before striking the deal with q , which is different from q 's color and which it retains after the deal.

This relaxation is provided as a backup system to allow peers to obtain all colors fast without having to query many neighbors, but makes the colored peers compete with the blank peers for redirection deals. To prevent this, peers will break a redirection deal through a colored peer if it can be replaced with a redirection through a blank peer. Because colored peers know this can happen, they will give priority to striking exchange deals.

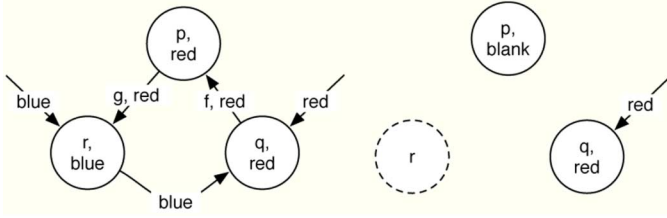


Fig. 5. Peer departure: before and after peer r departs.

C. Resulting Trees

We will now argue that in each tree, a peer forwards its own color to at most $C - 1$ other peers, where C is the number of colors in the system. By construction of Orchard, the out-degree of every peer is at most C . There are two ways in which a peer p can obtain its first color. Either p joins at the source, or it strikes a redirection deal for its first color. In the first case, p receives its color for free. In the second case, p joins through a redirection deal, and as part of this deal has to forward its own color to a peer of another color (see Fig. 3). In both cases, p receives at least one color without forwarding anything to a peer of the same color in return, which proves our assertion. Because a peer has at most $C - 1$ children of the same color, the system needs to contain at least three colors, for otherwise all trees are linear chains.

Let us take the color red as an example to further analyze the structure of the trees in Orchard. Every red peer gets its own color either from the source (see Fig. 1), or from another red peer (see Fig. 3). The non-red peers that receive the color red are either leaves (see Fig. 2), or peers that forward red to precisely one leaf (see Fig. 4). In other words, the ‘core’ of the red tree consists of red peers and the source, while the leaves, and in some cases peers one level above the leaves, are peers of other colors. As a consequence, if a single red peer p fails, most peers in the subtrees below p stop receiving at most one color (red) until the forest is repaired.

Because peers prefer to join close to the source, the trees created are shallow, which is desirable because of reduced latency and packet loss, but also because on average, nodes in a shallower tree have smaller subtrees below them. To see this, take a full w -ary tree of n nodes. The height of the tree can be expressed as $h(n) = \log_w(1 + (w - 1)n)$, and every node at distance d from the root is a descendent of d nodes. Let $c(n)$ be the average number of children of all nodes. Then, $c(n) = (1/n) \sum_{d=1}^{h(n)-1} dw^d$, which can be shown to be equal to $c(n) = h(n) + (h(n)/n - w)/(w - 1)$, which is of order $\log_w n$. So, $c(n)$ decreases when w increases.

D. Repairing Trees

When a peer p departs or fails, other peers will stop receiving the descriptions they get from p . Any peer that has a deal with p will cancel that deal. For example, in Fig. 3 on the right, if peer p fails, the redirection deal with q is canceled. This causes q to restore the exchange deal with r as it was before it got redirected through p .

If the deal to be canceled is an exchange deal, the redirection deals dependent on it (there can be at most two) have to be canceled. An example of this is shown in Fig. 5. Here, if peer r departs, q will break its exchange deal with r , which forces q to

also break its redirection deal with p . This causes p to stop receiving its color, and makes p unable to maintain any exchange deals with other peers. Even if p is still receiving other colors, we then force p to break all its deals and become blank.

These breaks of deals propagate down the subtrees below each peer that stops receiving its own color. In order to counter this, we will now discuss three methods of maintaining the trees.

1) *Backup Parents*: In order for a peer to be able to quickly repair the loss of its parent of the same color, each peer keeps track of the backup parents in its neighbor set. A peer q is a *backup parent* of p , if it is of the same color and the path from the source to q in that color does not contain p 's parent.

When the parent of p fails, p asks each of its backup parents, in order of increasing hop count from the source, whether it can strike a redirection deal. If p happens to have the source as a backup parent, it will first try to join at the source. Peer p will join the tree at the first backup parent that allows it. If no backup parent allows p to join, p will break all its deals, become blank, and try to rejoin the system.

Because no descendant of the departed parent is allowed to be a backup parent, no cycles are created in the multicast tree. To satisfy this condition, every peer keeps its neighbors informed about its paths to the source.

2) *Peers Changing Their Color*: Unfortunately, the backup parent strategy is not enough. If a peer of color c near the root of a tree fails and many peers below it cannot find a backup parent, those peers are all forced to rejoin the system and obtain a different color. The color c then becomes more rare, and it is possible that at some point not all peers will be able to receive it. In the worst case, the description could entirely disappear from the system.

To counter this, every peer keeps track of the set of all the colors its neighbors are receiving. Once this set stabilizes, a peer checks whether switching to a rarer color is beneficial. This is the case if it will be able to receive the rare color, and will be able to strike more exchange deals than it could before switching. Once a peer switches colors, it breaks the deals with those peers that are not willing to accept the color change, and strikes new deals with peers that are willing to. In our experiments, the neighbor set is considered stable if the colors they receive does not change for 3 s.

3) *Neighbor Set Maintenance*: When too many peers become blank, there may be peers which have too many blank neighbors to obtain all colors. To counter this system degeneration, a peer removes a neighbor from its neighbor set if that neighbor stays blank for a certain length of time, which, in our experiments, is 3 s. Note that since we assume that each peer can keep its neighbor set populated, the neighbor set will not become depleted if peers are removed.

IV. ATTACKING ORCHARD

In this section, we state explicitly in what way Orchard prevents free-riding. In addition, we discuss to what extent Orchard is resilient to several other types of attack.

A. Free-Riding

In this paper, we consider a peer to be a free-rider if it provides fewer resources for forwarding the video stream than it consumes. Because the source forwards data for free, there must

be peers which receive data for free, and these peers can thus be said to free-ride to some degree. In Orchard, the peers which have struck a join-at-the-source deal are those which receive data for free. However, the effect of this is limited as the source forwards every color to a limited set of peers, and forwards at most one color to any peer.

To obtain a color from a peer other than the source, a peer always has to forward a color in return, either to the same peer (exchange deal) or to a different peer (redirection deal). This forces peers to contribute if they want to obtain any color from another peer, making it impossible for them to free-ride.

B. Other Types of Attacks

A well-known method of attacking P2P networks is by taking control of multiple identities, either by a single peer emulating them (Sybil attack [14]) or by cooperation amongst peers (collusion attack). In Orchard, the exchange and redirection deals function regardless of whether the peers cooperate. The upload-download balance is always maintained for any peer and thus for any subset of peers as well. Both types of attack thus have no impact on either type of deal. The join-at-the-source deal is different, because the source grants only one such deal to any peer. A peer which can fake multiple identities can thus fool the source and potentially obtain multiple colors for free. In systems where this is a problem, the source can require the peers with which it strikes a join-at-the-source deal to send a stream with random data of the same size back to it. Although this wastes resources, it removes any bandwidth advantage to join at the source.

The source is, by definition, a single point of failure in any multicast algorithm, but fortunately, in Orchard peers do not need to know the source to obtain the video stream. The probability that a new peer meets the source thus decreases when the number of receivers increases.

Many more types of attacks exist in P2P networks. Peers could enter the system purely for malicious purposes, with no interest in the video stream itself. A malicious peer could for example offer fake deals or forward fake data. For an attack to be most effective, its effects have to propagate down the trees below its victim. To do this, it has to convince a victim peer to switch parents. However, when a victim peer detects it is receiving no data or fake data, it will disconnect from the malicious peer. If the victim peer cannot repair in time, its children will also detect the absence of data, and will look for a different parent themselves. This will cause an attack to be quarantined quickly. Additionally, peers could employ a policy of making the swap of parents definitive only when verified data starts flowing.

Of course, a sufficiently powerful malicious peer can attack enough peers in the system as to render it unusable. In such cases, the help of the underlying P2P network is needed to be able to shut out such peers.

V. EXPECTED PERFORMANCE

In this section, we will analyze the expected performance of the Orchard algorithm. As a performance metric, we will use the average number of colors received by a peer. First, we will describe the parameters used and make some general observations. Then, we will analyze the expected number of exchange

deals in the system as peers arrive and depart, as well as the expected number of redirection deals, for both variants.

A. Parameters of the Model

We define two types of events: peer arrivals and peer departures. Furthermore, we will assume an event is completely processed before the next event occurs. We will use the following parameters in our model:

- N is the number of peers in the system;
- C is the number of descriptions (colors);
- m is the size of the neighbor set a peer obtains upon arrival;
- n_i is the fraction of the peers that are of color i ;
- e_{ij} is the fraction of the peers that are of color j and have an exchange deal for color i ($e_{ii} = 0$ and $e_{ij} = e_{ji}$);
- $\alpha_i = \sum_{j=1}^C e_{ij}$ is the fraction of the peers that have an exchange deal for color i ;
- r_i is the fraction of the peers that are not of color i but have color i redirected through them;
- $\rho_i = n_i + \alpha_i + r_i$ is the fraction of the peers that receive color i .

These parameters represent the situation before an event occurs, and primed versions of the parameters (n'_i , etc.) will describe the situation in the system after an event has been processed. For instance, when a peer arrives, $N' = N + 1$, and when a peer leaves, $N' = N - 1$.

As a metric, we will use $R = \sum_{i=1}^C \rho_i$, which is the number of colors the peers receive on average. We will assume that the system is *stable*. We define a system to be stable when (for large N) the expected values of the parameters n_i , e_{ij} , α_i and r_i (and so, also ρ_i and R) do not change under peer arrivals and departures. System stability with respect to the fractions of peers of the same color ($E[n'_i] = n_i$) is found to occur in our experiments, which even seem to indicate that the distribution of colors over the peers is almost uniform (see Fig. 10). For settings in which the distribution of colors nevertheless becomes nonuniform, peers could keep each other informed about the color distribution using gossip-based protocols [15]. An arriving peer can then decide its color based on more than its local observations, rebalancing the system.

The performance in a stable system depends on the arrival and departure patterns of the peers. We shall analyze two extremes: a system in which peers only arrive, and a system in which peers only depart. For both scenarios, we will analyse for which values of e_{ij} and r_i the system is stable.

B. Peer Arrivals and Exchange Deals

A peer p that arrives and becomes of color i (by joining at the source or by a redirection deal). will try to obtain additional colors by striking exchange deals with its neighbors. Let us consider color $j \neq i$. If p has a neighbor which is both of color j and does not have an exchange deal for color i yet, an exchange deal will be struck. Since a random neighbor has probability $n_j - e_{ij}$ of falling into that category, p will have a probability of $1 - (1 - n_j + e_{ij})^m$ of striking an exchange deal for color j . Note that this probability depends only on n_j and e_{ij} , not on the distributions of the other colors and exchange deals.

Now, consider e'_{ij} . The number of exchange deals between colors i and j can increase if a peer arrives and becomes of

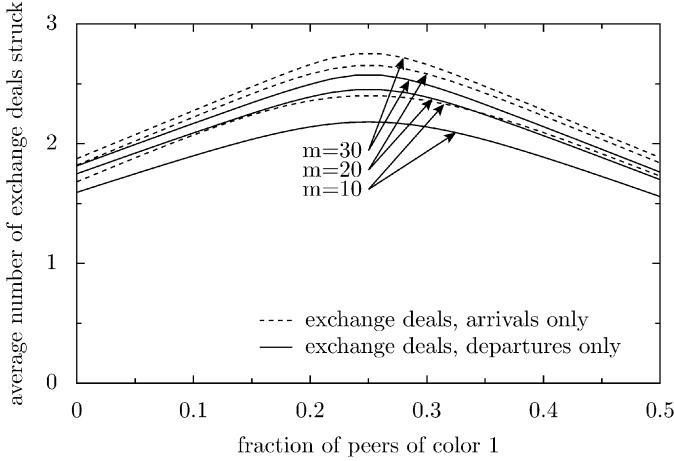


Fig. 6. Expected number of exchange deals struck in a system with 4 colors for various numbers of neighbors m ($n_2 = n_3 = n_4$).

color i or j . Let $f_{ij}(e_{ij})$ be the expected increase if there are $e_{ij}N$ exchange deals between peers of colors i and j already. Then, the following holds:

$$f_{ij}(e_{ij}) = n_i(1 - (1 - n_j + e_{ij})^m) + n_j(1 - (1 - n_i + e_{ij})^m).$$

Because $E[e'_{ij}N'] = e_{ij}N + f_{ij}(e_{ij})$ with $N' = N + 1$, we have

$$E[e'_{ij}] = e_{ij} + \frac{f_{ij}(e_{ij}) - e_{ij}}{N + 1}.$$

Note that e_{ij} is bounded by $0 \leq e_{ij} \leq \min\{n_i, n_j\}$, because there cannot be more exchange deals than peers of color i or j . Let $\bar{e}_{ij} = \min\{n_i, n_j\}$.

In a stable system, $E[e'_{ij}] = e_{ij}$, which holds if $f_{ij}(e_{ij}) = e_{ij}$. The value of e_{ij} for which this equation holds is unique, because f_{ij} decreases from $f_{ij}(0) \geq 0$ to $f_{ij}(\bar{e}_{ij}) \leq \bar{e}_{ij}$.

This allows us to calculate $\alpha_i = \sum_{j=1}^C e_{ij}$, which is the fraction of all peers obtaining color i through an exchange deal. Furthermore, we can calculate $\sum_{i=1}^C \alpha_i$, which is the average number of exchange deals struck by a peer. In Fig. 6, we plot this average using dashed lines, for various values of n_1 and m . In all cases, $C = 4$ colors are assumed, with $n_2 = n_3 = n_4$ and $\sum_{i=1}^C n_i = 1$ (that is, there are no blank peers). Note that a peer can strike at most $C - 1$ exchange deals, as the first color is obtained by either a redirection or a join-at-the-source deal.

We draw three conclusions from Fig. 6. First, the optimum is clearly attained when all colors are uniformly distributed ($n_1 = 0.25$). Second, having more neighbors per peer indeed increases performance, but with diminishing returns. Third, the exchange deals enable peers to receive a rather high number of colors, but there is still some room for improvement, as we will see in Section V-E.

C. Peer Departures and Exchange Deals

In this section we consider the effect of peer departures on the fractions of peers with exchange deals in the system. In this section, we will assume peers are only departing, not arriving.

Every peer p maintains a neighbor set and makes sure it contains at least m peers (see Section II-B). If a neighbor departs

and p has only $m - 1$ neighbors left, it is replaced by a new one selected at random from the peers still in the system. We will assume that eventually, with only peer departures and no arrivals, every peer has exactly m neighbors.

When a peer p departs, its children of the same color will try to repair the tree by contacting backup parents. Those who do not succeed in obtaining a new parent this way will drop their color and rejoin the system; this can be considered (for analysis purposes) as a peer departure followed by a peer arrival.

We will again consider the fraction e_{ij} when a random peer p departs. There are two possibilities.

- 1) p is of color j and has an exchange deal for color i , or vice-versa. This is the case with probability $e_{ij} + e_{ji} = 2e_{ij}$.
- 2) Otherwise.

In case 1, which has probability $2e_{ij}$, an exchange deal between colors i and j is lost. Let q be the neighbor with which p had this deal. Then, q will obtain a new neighbor to replace p , and will attempt to strike an exchange deal with any of its neighbors, which succeeds with probability $f_{ij}(e_{ij})$.

In addition to q , there are $m - 1$ other peers which had p as a neighbor, and which will replace p with a new neighbor. If such a peer is of color i or j and does not have an exchange deal yet for the other color, it will try to strike one with its newly acquired neighbor. For each of the $m - 1$ peers, this happens with a probability of $v_{ij} = (n_i - e_{ji})(n_j - e_{ij}) + (n_j - e_{ij})(n_i - e_{ji}) = 2(n_i - e_{ij})(n_j - e_{ij})$. This is an approximation, because the neighbor sets may intersect, but for large N , the error will be small.

In case 2, which has probability $1 - 2e_{ij}$, there is no exchange deal between colors i and j lost when p departs, and there are m peers which had p as a neighbor. For each of these peers, there is again a probability of v_{ij} that they have no exchange deal between color i and j but will strike one with their new neighbor. To sum up, the expectation of the new number of exchange deals is approximately

$$\begin{aligned} E[e'_{ij}N'] &\approx e_{ij}N - 2e_{ij}(1 - f_{ij}(e_{ij})) + 2e_{ij}(m - 1)v_{ij} \\ &\quad + (1 - 2e_{ij})mv_{ij} \\ &= e_{ij}N - 2e_{ij}(1 - f_{ij}(e_{ij})) + (m - 2e_{ij})v_{ij} \end{aligned}$$

with $N' = N - 1$.

The system is stable if $E[e'_{ij}] = e_{ij}$, and it can again be shown that e_{ij} is uniquely determined. This implies that $\sum_{i=1}^C \alpha_i$, which is the average number of exchange deals struck by a peer, is also uniquely determined. In Fig. 6, we plot this average using solid lines for various values of n_1 and m . Again, $C = 4$ colors are assumed, with $n_2 = n_3 = n_4$ and $\sum_{i=1}^C n_i = 1$. The performance is slightly lower than in the previous section, where peers were only arriving.

D. Redirection

We will now analyze the probability that an arriving peer will obtain its first color. The probability of a peer joining at the source is small if N is large, so we will focus on a peer joining the forest by striking a redirection deal.

Redirection deals take precedence over redirection deals through colored peers, so we can safely ignore the latter in this

section. For an arriving peer p to strike a redirection deal, it needs a neighbor which has an exchange deal which is not yet redirected (*free*). There are $(1/2) \sum_{i=1}^C \alpha_i N$ exchange deals, which provide $\sum_{i=1}^C \alpha_i N$ potential redirections. Of these, $\sum_{i=1}^C n_i N$ are redirected to give peers their first color. This leaves $\sum_{i=1}^C (\alpha_i - n_i) N$ potential redirections available when p arrives. The distribution of these is skewed—peers prefer to join a tree as close to the source as possible, so peers close to the source are less likely to have any free exchange deals left. Since a peer can have at most $C - 1$ exchange deals, the Pigeonhole Principle ensures us that at least $\sum_{i=1}^C (\alpha_i - n_i) N / (C - 1)$ peers have at least one free exchange deal regardless of the distribution of these deals over the peers. The probability of p having a neighbor which has a free exchange deal is at least $1 - (1 - \sum_{i=1}^C (\alpha_i - n_i) / (C - 1))^m$. As an example, when using $m = 20$ and the same conditions as before ($C = 4, n_2 = n_3 = n_4$, and $\sum_{i=1}^C n_i = 1$), the probability of p striking a redirection deal upon arrival is higher than 0.99 for $0 \leq n_1 \leq 0.55$.

E. Redirection Through Colored Peers

If a peer p cannot obtain a certain color by striking an exchange deal, it will try to strike a redirection deal for that color instead. However, unlike exchange deals, such redirection deals can be broken in the future even if peers do not depart. For instance, p drops an exchange deal in favour of an exchange deal it can strike when it is contacted by arriving peers. For this reason, we will look at the system at a single point in time and analyze the expected number of redirection deals for a certain color. The following analysis will provide a lower bound on this expectation.

Let us consider color i . Of the N peers in the system, $n_i N$ are of color i , and $\alpha_i N$ receive color i through an exchange deal. Let D_i be the rest of the peers, with $|D_i| = (1 - n_i - \alpha_i) N$. These peers try to strike a redirection deal with their neighbors. We will approximate the fraction of peers in D_i that is expected to succeed. To do this, we start with a system without redirection deals through colored peers for color i . Then, we consider the peers in D_i in a random order and let them try to strike a redirection deal with their neighbors.

For a peer in D_i to strike a redirection deal for color i , it needs a neighbor of color i with a free exchange deal. By using the same argument as in the previous section, but focusing on color i , it can be shown that there are at least $(\alpha_i - n_i) N / (C - 1)$ such peers if there would be no redirection deals through colored peers for color i .

Now we consider each peer in D_i in turn, in a random order, and analyze the probability it will strike a redirection deal. Let y_k be the fraction of the k peers that have struck a redirection deal for color i when k peers have been considered. Then, at least $F_i(y_k) = ((\alpha_i - n_i) N - y_k k) / (C - 1)$ peers have at least one free exchange deal. We will focus on this worst case and use the results as a lower bound on the expected performance in general. We have

$$E[y_{k+1}(k+1)] = y_k k + 1 - \left(1 - \frac{F_i(y_k)}{N}\right)^m$$

and by defining $G_i(y_k) = 1 - (1 - (F_i(y_k)/N))^m$, we have

$$E[y_{k+1}] = y_k + \frac{G_i(y_k) - y_k}{k+1}.$$

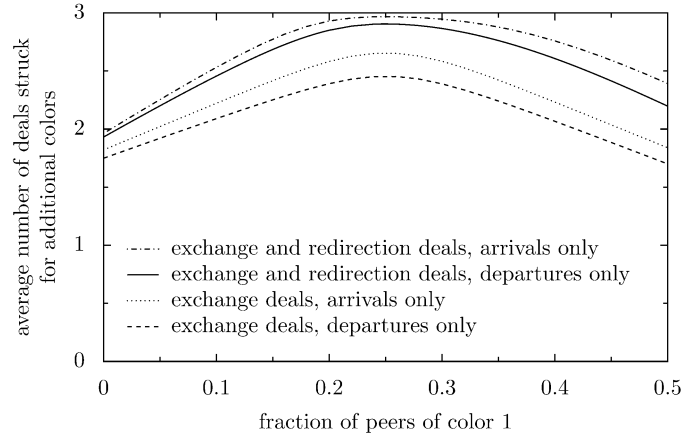


Fig. 7. Expected number of deals struck to obtain additional colors in a system with only peer arrivals or only peer departures. Every peer maintains 20 neighbors ($C = 4, n_2 = n_3 = n_4$).

Now, let γ_i be the solution to $G_i(x) = x$ with $0 \leq \gamma_i \leq 1$. Because $G_i(x)$ decreases over that interval, γ_i is uniquely defined.

We can now bound $E[y_{|D_i|}]$ as follows. Every time a redirection deal is struck, the probability for subsequent peers to strike one decreases. This leads to $E[y_{k+1}] \leq E[y_k]$. In order for that to hold, $G_i(y_k) \leq y_k$ and so $y_k \geq \gamma_i$. The same holds for $y_{|D_i|}$, so we expect to have $y_{|D_i|} \cdot |D_i| \geq \gamma_i \cdot |D_i|$ redirection deals through colored peers for color i . This translates to $r_i = y_{|D_i|} \cdot |D_i| / N \geq \gamma_i \cdot |D_i| / N = (1 - \alpha_i - n_i) \gamma_i$ as a lower bound on r_i .

In Fig. 7 we show the performance of a system with and without redirection deals through colored peers, in a stable system with only peer arrivals or only peer departures. As in the previous figure, $C = 4$ colors are used, $n_2 = n_3 = n_4$, and there are no blank peers, so $\sum_{i=1}^C n_i = 1$. Every peer thus receives at least one color by definition. Fig. 7 only plots the number of additional colors obtained.

The upper two lines indicate the total expected number of deals struck by a peer to obtain additional colors (equal to $\sum_{i=1}^C (\alpha_i + r_i) = R - 1$), when peers are only arriving, and when peers are only departing. The lower two lines indicate the performance when redirection deals are not considered. In both cases, the advantage over having only exchange deals is clear. In the following section, we will compare these bounds to measured results.

VI. EXPERIMENTS

In this section we present our experimental setup and the results of five experiments for assessing the multicast forest constructed by Orchard. We conclude this section with a discussion about possible issues when the systems contains a larger number of peers than we employed in our experiments.

A. Experimental Setup

We have tested the Orchard algorithm in two emulation environments. The first environment is the DAS2 cluster computer [16], using 20 processing nodes. The second environment is a wide-area network which we refer to as Delft-37, which consists of hosting accounts at various ISPs around the globe. These accounts range from web accounts to virtual private servers. All

TABLE I
NUMBER OF PEERS EMULATED IN DELFT-37

Delft-37 node	# of peers	Delft-37 node	# of peers
AZ: USA (Arizona)	9	UK	8
NY: USA (New York)	14	DE: Germany	15
WA: USA (Washington)	11	SG: Singapore	4
GE: USA (Georgia)	5		

Delft-37 accounts have limited resources as they have to share the same machines with other accounts. The reason for creating Delft-37 rather than using PlanetLab [17] is a closer resemblance to real peers on the Internet. The nodes in PlanetLab generally have better hardware and Internet connections than the average end system on the Internet. In our experiments, we use 7 nodes of Delft-37 located in the USA, the UK, Germany and Singapore (see Table I). The bandwidth between these nodes was 20–600 kbit/s, and the latency between these nodes varied between 30–400 ms.

We have implemented the Orchard algorithm in Python. The emulator is started on every host computer, and emulates multiple peers on every one of them. Every peer sets up TCP connections with each neighbor to exchange control information. We used this emulator to do five experiments, which evaluate the number of descriptions received under varying conditions. In the first experiment, peers only arrive and do not depart. The second and third experiment assess the performance of Orchard under flash crowds and churn. The fourth experiment measures the performance when multicasting a 4 Mbit/s data stream using UDP. The last experiment tests the performance of Orchard on Delft-37, a wide-area network which we will describe in the following section. All experiments except the fourth will only check the membership of peers of the multicast trees by having the emulator send one packet per second for each description instead of using a real data stream.

In our experiments, we assume four MDC descriptions. The source forwards each description at most three times. We have a central rendezvous server which keeps track of all peers interested in the video and which sends the sets of random, uniformly selected peers upon request. By keeping in touch with the rendezvous server, a peer can replace departed neighbors to make sure it always maintains a neighbor set of at least 20 peers ($m = 20$).

Peer arrivals are modeled as a Poisson process with an average of two per second. In a recent measurement of a large-scale Video-on-Demand system [18], it was shown that about half the peers disconnect within 10 min. Our measurements take a more pessimistic scenario into account, in order to be able to cope with zapping behavior which we expect to rise if P2P video multicasting becomes more popular. In those tests where peers depart, we let peers stay in the system for 120 s on average, using an exponential distribution.

B. Arrivals Only

In the first test we let 500 peers arrive but not depart. The performance results are shown in Table II, which contains the fractions of peers which receive at least a certain number of descriptions. More than 97% of the peers was able to obtain all

TABLE II
CUMULATIVE DISTRIBUTION OF THE AVERAGE NUMBER OF DESCRIPTIONS RECEIVED

number of descriptions	fraction of time		
	Arrivals only	Flash crowd (Fig. 8)	Churn (Fig. 11)
at least 1	0.987	0.989	0.986
at least 2	0.987	0.985	0.983
at least 3	0.986	0.975	0.971
4	0.974	0.874	0.853

TABLE III
AVERAGE NUMBER OF DESCRIPTIONS RECEIVED AND THE COLOR DISTRIBUTION FOR THE COLORED PEERS. THE COLOR DISTRIBUTION IS FIXED FOR THE ANALYSES AND MEASURED FOR THE EXPERIMENTS

Type	Scenario	Color distribution $\{n_1, n_2, n_3, n_4\}$	Avr. nr. of desc.
Measured Analysis	Arrivals only	$\{0.251, 0.251, 0.249, 0.248\}$	3.984
	Arrivals only	$n_1 = 0.25, n_2 = n_3 = n_4$	3.968
Measured	Flash crowd (Fig. 8)	$\{0.254, 0.244, 0.252, 0.249\}$	3.867
Measured Analysis	Churn (Fig. 11)	$\{0.252, 0.252, 0.249, 0.248\}$	3.846
	Departures only	$n_1 = 0.20, n_2 = n_3 = n_4$	3.851
	Departures only	$n_1 = 0.25, n_2 = n_3 = n_4$	3.905
	Departures only	$n_1 = 0.30, n_2 = n_3 = n_4$	3.866

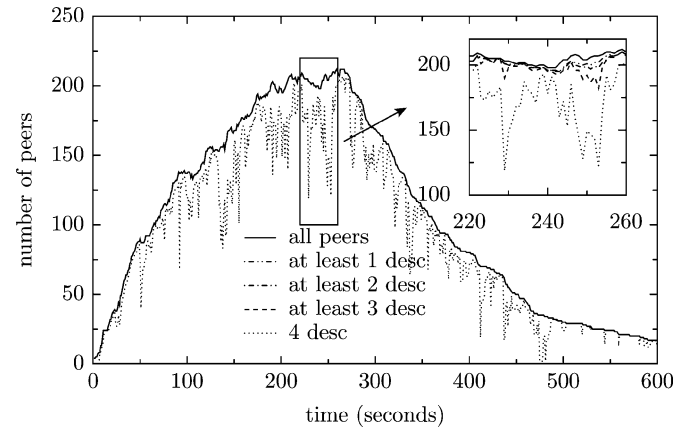


Fig. 8. Number of descriptions received under flash crowds.

four descriptions. In Table III, we compare the measured result (first row) against the expected performance as derived in Section V (second row). The table shows the average number of descriptions received by a peer at any moment in time, as well as the average color distribution. On the second row, the table shows the lower bound on the expected performance as derived in Section V-E. Both rows are restricted to those peers that receive at least one description to make the numbers comparable to Fig. 7. The measured results are consistent with the derived lower bound.

C. Flash Crowds

In the second test, we let 500 peers arrive and depart. As the mean inter-arrival time is very short, initially there is indeed a fast build up of the number of peers. In Fig. 8, we show the number of peers which receive a certain number of descriptions over time for a typical run. To keep the main graph readable,

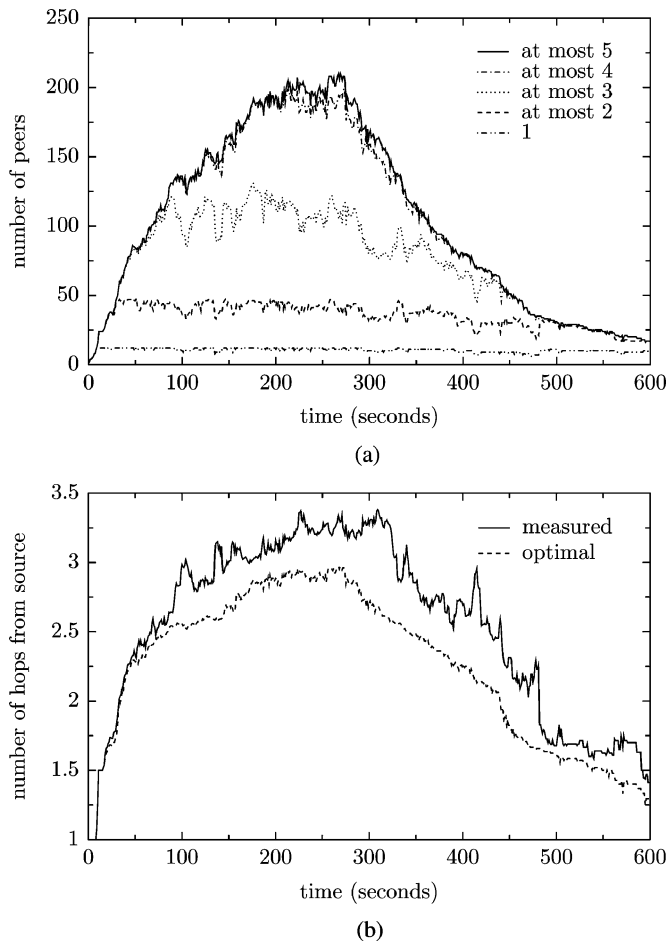


Fig. 9. Distance from the source to the peers in the trees of their own colors. (a) Distribution of the path lengths. (b) Measured and the optimal average path lengths.

only the number of peers receiving all four descriptions and the total number of peers are represented. The inset shows a detailed sample which includes all curves.

All peers receive at least three out of four descriptions most of the time, but spikes can be observed in the number of peers which receive all descriptions. Peer departures, especially high up in trees, can cause many other peers to temporarily stop receiving one of their descriptions. However, the system recovers fast. In Table II, we show the percentage of time a peer receives a certain number of descriptions, averaged from the moment the peer joins until it departs.

The number of hops from the source to every peer indicates the shallowness of the trees. The source forwards each color to at most three peers, and every peer forwards its own color to at most three peers as well (since there are four colors). The tree of every color is thus ternary, and the minimal average distance from the source to each peer can be calculated accordingly (see Section III-C).

The height of the trees over time in the same experiment is shown in Fig. 9(a). Each line shows the number of peers at most a certain number of hops from the source, in the tree of their own color. Even though peers are constantly arriving to and departing from the system, the trees do not degenerate. This is due

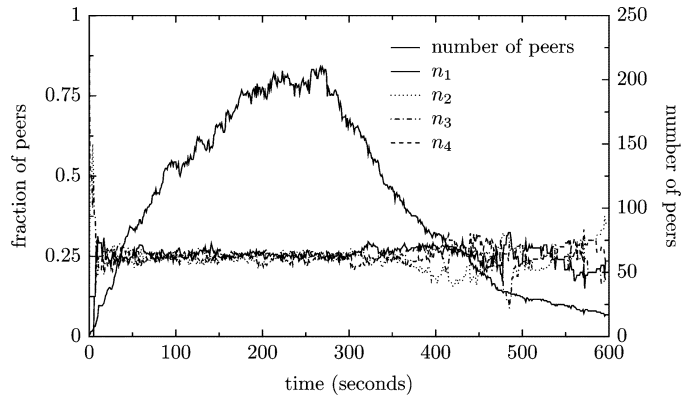


Fig. 10. Distribution of the colors of the peers. The number of peers is plotted as well (scale on the right-hand side of the figure).

to the fact that peers switch to parents of the same color but closer to the source, whenever possible. We compare the average measured distance from a peer to the source against the optimal value in Fig. 9(b). As can be seen in the figure, every peer is on average at most one hop further away from the source than in the optimal case.

In Fig. 10, the distribution of the colors of the peers is shown. The number of peers in the system is plotted as well. The colors stay approximately evenly distributed. Of course, fluctuations are larger than when the number of peers is low.

In Table III, we compare the measured results (third row) against the lower bounds on the expected performance as derived in Section V-E (rows 5–7). The table shows the average number of descriptions received by a peer at any moment in time, again restricted to those peers that receive at least one description. The color distribution in this experiment fluctuates slightly around a uniform distribution. Because of this, we compare the measurements to Fig. 7 using slight fluctuations as well, while $n_1 = 0.20, 0.25, 0.30, n_2 = n_3 = n_4$. The measured results are close to the derived lower bound on the expected performance in a stable situation. Also, it must be noted that in the analysis, every event is dealt with instantaneously, while in practice, events take time to process. This can cause peers to be in the process of repairing the tree for a certain amount of time. This performance loss is not taken into account in the analysis.

D. Churn

In the third experiment, we let 1200 peers arrive and depart. Initially, the departure rate is low, causing a flash crowd. Once the departure rate is (about) equal to the arrival rate of $2/s$, there is churn, which is the rapid arrival and departure of many peers in the network. The result of this experiment is shown in Fig. 11, in which we only consider the data after 200 s, when the number of peers is (reasonably) stabilized. On average, the peers are still able to receive at least one description 98.6% of the time (see also Table II). The continuous peer departures cause frequent big drops in the number of peers which are receiving all four colors, but the system also recovers quickly. This shows that the Orchard algorithm can handle sustained peer arrivals and departures and still deliver part of the video stream to most of the users, most of the time.

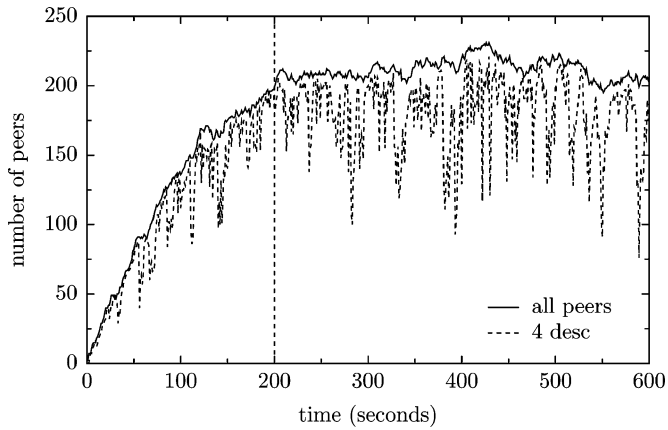


Fig. 11. Number of descriptions received under churn. The initial 200 s are disregarded.

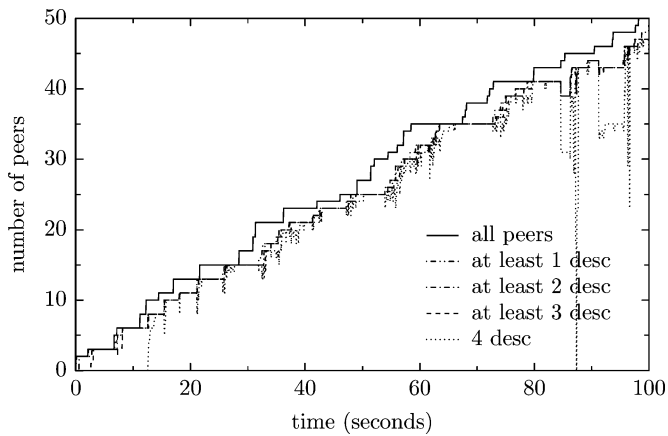


Fig. 12. Number of descriptions received when streaming 4 Mbit/s video.

The measured results are again compared to the lower bound on the expected performance in Table III on row 4 and rows 5–7, respectively. Although the performance for this experiment is a bit less than in a flash crowd, it still comes close to the values derived in Section V-E.

E. Real Streaming

In this experiment, we let 50 peers arrive. A 4 Mbit/s data stream (1 Mbit/s per description) was distributed using UDP. The result of this experiment is shown in Fig. 12. The spikes are due to packet loss. When streaming these amounts of data, the peers require a bit more time to obtain their first description. The huge amount of UDP traffic at each peer slows down the ability of the emulator to initiate TCP connections. Thus, it takes a peer somewhat longer to establish connections with its neighbors, which causes additional delay between arriving in the system and obtaining the first description.

We do not do any retransmits of dropped UDP packets, nor do we define how much packet loss a peer tolerates from its parent. These parameters depend on the network (such as router configuration, network congestion, and the latencies between the peers) and local parameters (such as the codec used and the buffer size). Considering these parameters is outside the scope of this paper.

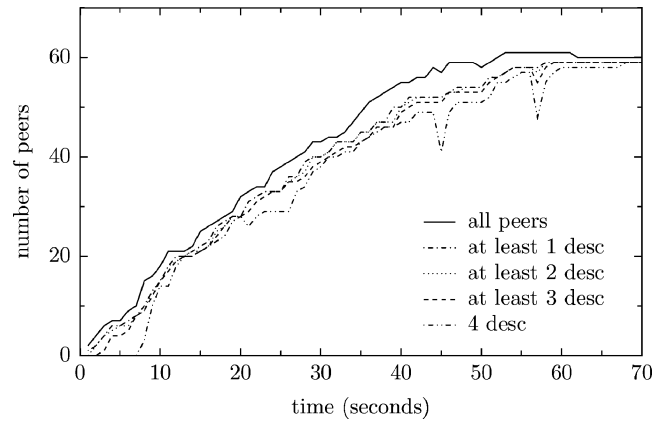


Fig. 13. Number of descriptions received in Delft-37.

F. Delft-37

In the last experiment, we distribute the peers over the Delft-37 network. The Delft-37 nodes we used cannot all host the same number of peers due to different amounts of resources being available on the nodes. In this test, we started 65 peers, with a distribution as shown in Table I. The results of this experiment are shown in Fig. 13.

Every Delft-37 node forms the end point of a large number of TCP connections, since every peer maintains a TCP connection to all of its neighbors. Five of the peers (two in Washington, one in Georgia, two in Germany) were able to start, but were not able to set up any TCP connections. This caused four of these peers to depart, explaining the three small dips in the ‘all peers’ curve. We suppose this happened due to the high number of TCP connections already started on these nodes. The 60 peers that did not depart were all able to obtain all colors.

G. Scalability of Orchard

In the experiments we have presented above, there are at most a few hundred peers in the system at a single point in time. If the number of peers is significantly larger, the behavior of Orchard could change. The analysis in Section V is independent on the number of peers and thus scales, but it does not cover all aspects of Orchard. More specifically, the trees created by Orchard become deeper if the number of peers in the system increases, and so the average distance from a peer to the source increases. As a result, the differences in latencies with which a peer receives the colors may increase. The amount of buffering done at a peer has to be sufficient to cope with these differences. Also, the probability that a peer departs along the path from a peer to the source increases, which is likely to result in more packet loss, or even the complete loss of a color. However, these problems are not specific to Orchard, but they will occur in any ALM streaming algorithm that employs trees.

VII. RELATED WORK

The idea of using MDC [7] in ALM has been proposed before [19]–[21]. Pouwelse *et al.* [20] propose the use of a bartering mechanism, but do not provide an algorithm. SplitStream [19] uses MDC to create a forest interior-node-disjoint trees and can be modified in such a way that every node forwards as much as

it receives, if such a forest can be constructed. However, Split-Stream was designed for cooperative environments, and lacks the mechanisms to enforce a fair bandwidth contribution. Peers respect the forwarding capacity claimed by their neighbors, and a peer can thus easily free-ride by lying about its forwarding capacity. In contrast, Orchard forces the peers to contribute, and avoids relying on the peers' own claims.

It is not necessary to use MDC to create resilience against peer failure. In Chainsaw [22], the video stream is cut into pieces, with peers forwarding to each other the most recent pieces as long as bandwidth allows. Each peer requests the pieces it is still missing from its neighbors. This makes it very resilient against peer failure, because a failing neighbor causes a peer to (re-)request the pieces from someone else.

The traditional single tree approach is the basis of many algorithms. In NICE [23] and ZigZag [24], a hierarchy of groups ensures members of a group can replace each other in case of peer failures.

None of the algorithms mentioned above provides a mechanism to enforce a fair resource contribution. All of the algorithms allow a peer to define its own maximum out-degree, and count on peers to be honest about their latency to others. This makes cheating easy [25]. Peers can simply pretend to be unable to forward data, and will still be served. Unlike those algorithms, Orchard enforces fair resource contribution and is still resilient against high-rate peer arrivals and departures.

BAR gossip [26] is a multicast solution based on an epidemic approach, in which fairness is encouraged by letting peers exchange pieces of the video stream and providing countermeasures against malicious peers. However, this algorithm relies on a central, trusted auditor to evict malicious peers from the system.

Several algorithms have been proposed to combat free-riding in multicast systems. Some [3], [6] assign the role of central authority to the source, letting it manage the tree or enforce other rules on the other peers. However, this puts an additional load on the source, which it may not be able to handle.

Habib *et al.* [4] let peers keep each other informed about the behavior of others, and punish free-riders. This creates an incentive for peers to contribute resources, but requires a central authority or the deployment of a distributed trust system to avoid peers spreading false information about others. A distributed trust system adds complexity, it needs to exist beforehand, and it needs to cover all the receiving peers. This is not always feasible or practical.

Ngan *et al.* [5] propose a system in which the multicast tree is periodically reshuffled. If a free-rider refuses to forward to another peer, that other peer could become the free-rider's parent after a reshuffle, and subsequently deny serving the free-rider. Such a system is very vulnerable to Sybil attacks, in which a free-rider assumes a new identity. To combat this, the authors propose the use of certified nodeIds. However, this requires a central authority or a distributed trust mechanism as well.

VIII. CONCLUSIONS AND FUTURE WORK

When multicasting video streams using Application Level Multicasting, each byte received by a peer has to be uploaded

by another peer. Through the Orchard algorithm, we have shown that it is possible to construct an ALM system in P2P networks which is *fair, decentralized* and *resilient*.

The Orchard algorithm is fair, because peers are forced to share the burden of uploading equally: no peer has to upload more data than it downloads. We achieved this by using Multiple Description Coding, and by letting the peers strike deals to exchange and redirect descriptions. Every deal ensures that a peer will not have to forward more than it receives. The Orchard algorithm is decentralized as it does not need any central component. Although the source is a special component, no peer is required to know it. The Orchard algorithm is resilient against peer arrivals and departures due to the use of multiple descriptions. This was confirmed by analyzing the expected performance of Orchard as well as through emulation.

We have performed this work in the context of the I-Share research project [27] on sharing technologies in virtual communities (e.g., the sets of users interested in the same content). As a research vehicle for I-Share, we are designing and implementing Tribler [28], [29], which is a P2P network of potentially millions of nodes for sharing live *and* recorded TV programs. We plan to incorporate Orchard into Tribler.

REFERENCES

- [1] S. Saroiu, P. Gummadi, and S. Gribble, "Measurement study of P2P file sharing systems," *Proc. SPIE*, vol. 4673, pp. 156–170, 2002.
- [2] D. Stutzbach and R. Rejaie, Characterizing churn in peer-to-peer networks Univ. Oregon, Eugene, Tech. Rep. CIS-TR-2005-03, 2005.
- [3] Y. Chu, J. Chuang, and H. Zhang, "A case for taxation in peer-to-peer streaming broadcast," in *Proc. ACM SIGCOMM Workshop Practice Theor. Incentives Networked Syst.*, 2004, pp. 205–212.
- [4] A. Habib and J. Chuang, "Incentive mechanism for peer-to-peer media streaming," in *Proc. 12th Int. Workshop Quality Service (IWQOS)*, 2004, pp. 171–180.
- [5] T.-W. J. Ngan, D. S. Wallach, and P. Druschel, "Incentives-compatible peer-to-peer multicast," in *Proc. 2nd Workshop Econ. Peer-To-Peer Syst. (P2PEcon)*, 2004.
- [6] V. Padmanabhan, H. Wang, and P. Chou, "Resilient peer-to-peer streaming," in *Proc. 11th IEEE Int. Conf. Network Protocols (ICNP)*, 2003, pp. 16–27.
- [7] V. Goyal, "Multiple description coding: Compression meets the network," *IEEE Signal. Process. Mag.*, vol. 18, no. 5, pp. 74–93, 2001.
- [8] Y. Chu, S. Rao, and H. Zhang, "A case for end system multicast," in *Proc. ACM SIGMETRICS*, 2000, pp. 1–12.
- [9] N. Andrade, M. Mowbray, A. Lima, G. Wagner, and M. Ripeanu, "Influences on cooperation in bittorrent communities," in *Proc. ACM SIGCOMM*, 2005, pp. 111–115.
- [10] F. Fitzek, B. Can, R. Prasad, and M. Katz, "Overhead and quality measurements for multiple description coding for video services," in *Wireless Personal Multimedia Commun.*, 2004, pp. 524–528.
- [11] A. Alagoz, O. Ozkasap, and M. Caglar, "SeCond: A system for epidemic peer-to-peer content distribution," in *Proc. ISCN'06 7th Int. Symp. Comput. Networks*, Istanbul, Turkey, 2006, pp. 248–253.
- [12] P. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié, "Epidemic information dissemination in distributed systems," *Computer*, vol. 37, no. 5, pp. 60–67, May 2004.
- [13] R. Ferreira, M. Ramanathan, A. Awan, A. Grama, and S. Jagannathan, "Search with probabilistic guarantees in unstructured peer-to-peer networks," in *Proc. 5th Int. Conf. Peer-To-Peer Comput.*, 2005, pp. 165–172.
- [14] J. Douceur, "The sybil attack," in *Proc. 1st Int. Workshop Peer-To-Peer Syst. (IPTPS)*, 2002.
- [15] M. Jelasity and A.-M. Kermarrec, "Ordered slicing of very large-scale overlay networks," in *Proc. 6th IEEE Int. Conf. Peer-To-Peer Comput. (P2P'06)*, Cambridge, U.K., 2006, pp. 117–124.

- [16] *DAS: Distributed ASCII Supercomput.*, [Online]. Available: <http://www.cs.vu.nl/das2>
- [17] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: An overlay testbed for broad-coverage services," *ACM Comput. Commun. Rev.*, vol. 33, no. 3, pp. 3–12, 2003.
- [18] H. Yu, D. Zhang, B. Zhao, and W. Zheng, "Understanding user behavior in large scale video-on-demand systems," in *Proc. 1st ACM EuroSys*, 2006, pp. 333–344.
- [19] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-Bandwidth multicast in a cooperative environment," in *Proc. 19th ACM Symp. Oper. Syst. Principles (SOSP)*, 2003, pp. 298–313.
- [20] J. Pouwelse, J. Taal, R. Lagendijk, D. Epema, and H. Sips, "Real-time video delivery using peer-to-peer bartering networks and multiple description coding," in *Proc. IEEE Conf. Systems, Man, Cybern.*, 2004, pp. 4599–4605.
- [21] J. Taal and R. Lagendijk, "Fair rate allocation of scalable multiple description video for many clients," in *Proc. Visual Commun. Image Process.*, 2005, pp. 2172–2183.
- [22] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr, "Chainsaw: Eliminating trees from overlay multicast," in *Proc. 4th Int. Workshop Peer-To-Peer Syst. (IPTPS)*, 2005, pp. 127–140.
- [23] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan, "Resilient multicast using overlays," in *Proc. ACM SIGMETRICS*, 2003, pp. 102–113.
- [24] D. Tran, K. Hua, and T. Do, "ZIGZAG: An efficient peer-to-peer scheme for media streaming," in *Proc. 22nd IEEE INFOCOM*, 2003, pp. 1283–1292.
- [25] L. Mathy, N. Blundell, V. Roca, and A. El-Sayed, "Impact of simple cheating in application-level multicast," in *Proc. IEEE INFOCOM*, 2004, vol. 2, pp. 1318–1328.
- [26] H. Li, A. Clement, E. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin, "BAR gossip," in *Proc. 7th USENIX Oper. Syst. Design Implementation (OSDI)*, 2006, pp. 191–206.
- [27] *I-Share*, [Online]. Available: <http://ishare.ewi.tudelft.nl>
- [28] P. Garbacki, A. Iosup, D. Epema, and M. van Steen, "2Fast : Collaborative downloads in P2P networks," in *Proc. 6th IEEE Int. Conf. Peer-To-Peer Comput. (P2P'06)*, Cambridge, U.K., 2006, pp. 23–30.
- [29] J. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. Epema, M. Reinders, M. van Steen, and H. Sips, "Tribler: A social-based peer-to-peer system," *Concurrency and Computation: Practice and Experience* to appear.



Jan David Mol completed his master's project at Philips Research and received the M.Sc. degree in computer science from Delft University of Technology, Delft, The Netherlands, in 2004. He is currently pursuing the Ph.D. degree in the Department of Computer Science, Delft University of Technology.

His research focuses on video distribution in P2P systems.



Dick H. P. Epema received the M.Sc. and Ph.D. degrees in mathematics from Leiden University, Leiden, the Netherlands, in 1979 and 1983, respectively.

From 1983 to 1984, he was with the Computer Science Department, Leiden University. Since 1984, he has been with the Department of Computer Science, Delft University of Technology, Delft, The Netherlands, where he is currently an Associate Professor in the Parallel and Distributed Systems Group. During the academic year 1987–1988, the fall of 1991, and the summer of 1998, he was also a Visiting Scientist at the IBM T. J. Watson Research Center, Yorktown Heights, NY. In the fall of 1992, he was a visiting professor at the Catholic University of Leuven, Leuven, Belgium. His research interests are in the areas of performance analysis, distributed systems, peer-to-peer systems, and grids.



Henk J. Sips is full Professor of computer science and Head of the Parallel and Distributed Systems Group at Delft University of Technology, Delft, The Netherlands. His research interests include parallel computer architectures, parallel programming, parallel algorithms, mobile computing, and distributed systems. Research projects he is involved in cover subjects such as language design and compiler technology for parallel systems, low power computing, mobile computing, and P2P systems. He has authored and co-authored over 140 scientific publications. He is a member of the editorial board of the journals *Scientific Programming* and *Concurrency & Computation*.

Professor Sips has served on numerous program committees of scientific conferences.