

The Orchard Algorithm: P2P Multicasting without Free-riding

J.J.D. Mol, D.H.J. Epema, and H.J. Sips
Delft University of Technology
P.O. Box 5031, 2600 GA Delft, The Netherlands
j.j.d.mol,d.h.j.epema,h.j.sips @tudelft.nl

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 behaviour which is problematic for ALM: peers tend not to donate any resources (free-riding), and they arrive and depart at a high rate (churn). In this paper we propose the Orchard algorithm for ALM of video streams in P2P systems, 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. Our experiments show that Orchard is capable of providing a good quality of service to every peer, even when peers join and leave the forest at a high rate.

1. Introduction

Although the main purpose of many current peer-to-peer (P2P) networks is off-line file sharing, such networks may also be suitable environments for multicasting high-quality video streams over the Internet. However, existing multicasting solutions do not take into account the behaviour of peers as observed in existing peer-to-peer (P2P) networks. More specifically, they do not deal with *free-riding* [12] (peers unwilling to donate resources) or *churn* [12, 13] (peers arriving and departing at a high rate). In this paper, we propose the Orchard algorithm for building and maintaining multicast trees for video streams, which does deal with these two issues. 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., congestion in the Internet), it may be advantageous to split up video streams into multiple substreams and forward them independently. In this paper we assume the use of Multiple Description Coding (MDC) [8] 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.

It has been argued in the past that multicasting at the network layer is not without problems [5]. Therefore, and because we want to explore the creation and maintenance of multicast trees in a P2P-like fashion, in this paper we use Application Level Multicasting (ALM) instead. Existing P2P networks suffer from free-riding and churn, which creates problems for ALM. In conventional ALM, a single spanning tree is constructed over the peers, and the burden of forwarding the video 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, making cheating easy. If 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 recognised in BitTorrent [3], 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. This protects the system against free-riders, and, by using multiple trees, against churn.

We have performed this work in the context of the I-Share research project [2] 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 P2P-TV [11], which is a P2P network of potentially millions of nodes for sharing real-time *and*

recorded TV programs.

This paper is organised as follows. In Section 2, we further specify the problem we address. We present the Orchard algorithm in Section 3, and our experimental results in Section 4. Finally, we discuss some related work and draw conclusions in Sections 5 and 6, respectively.

2. The Problem

We will now present the problem we consider in this paper. First, we will describe the required pre-processing of video streams to split them up into substreams. Next, we will present the assumptions we make about the underlying P2P network, and finally we will state the problem we address in this paper.

2.1. 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) [8]. We will assume that these descriptions require equal bandwidths (with n descriptions, presumably a fraction of about $1/n$ 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 *colour*. Splitting the video stream adds overhead to the data stream as well as complexity to the encoder and decoder. In our experiments, the original video stream was split up into four MDC descriptions, which is a 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.

2.2. 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.

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 *neighbour set* of other interested peers. We assume an unstructured underlying P2P network: peers added to the neighbour set are considered to be selected at random. Mechanisms for discovering other interested peers are outside the scope of this paper. However, a possible way to do this is to use the mechanism presented in [7]. Then, both the peers interested in the video stream

and the peers already receiving at least one description perform limited broadcasts in order to learn about each others' existence. This allows the interested peers to create neighbour sets. In our experiments, we will use a central server to facilitate the peer discovery (see Section 4.1).

2.3. Problem Statement

Suppose we have 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 these trees. One of the main performance metrics we will use to assess the quality of our algorithm is the fraction of peers receiving all (or at least one) of the descriptions. We want our algorithm to satisfy the following conditions:

1. No peer forwards more descriptions than it receives.
2. A peer does not need to know s or any other specific peer.
3. Peer departures do not severely impact the number of descriptions received by other peers.

These requirements are necessary to ensure a fair multicast algorithm which does not depend on a central component and is resilient against peer failures.

3. The 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.

3.1. Constructing the Forest

An Orchard forest for a source s is represented by a directed graph with a set of nodes N and a set of links L . Each node $n \in N$ represents a peer. A link $e = (a, b) \in L$ represents the forwarding of a description from a to b , with $a, b \in N$. Every link in L is given the colour of the description sent over it. The subset of links of a certain colour and the nodes they connect form the multicast tree for a single description. All the trees are rooted at the same source $s \in N$. Every peer gets a colour, which is the colour 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 colour *blank*. The source has the special colour *white*.

When a peer wants to join the Orchard forest, it queries the distance of each peer in its neighbour set to the source, in

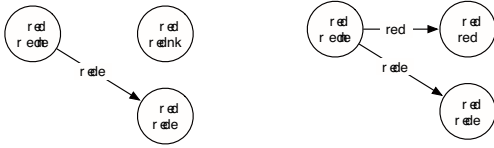


Figure 1. Joining at the source: before and after peer p joins at the source s .

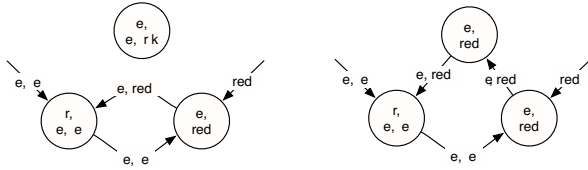


Figure 2. Redirection: before and after peer p joins the multicast tree below q .

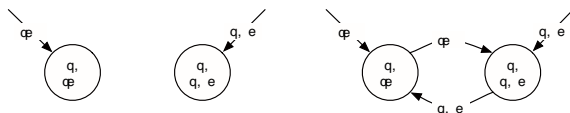


Figure 3. Exchanging descriptions: before and after two peers strike an exchange deal.

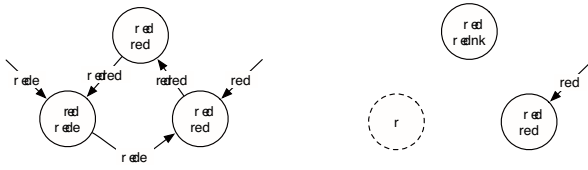


Figure 4. Peer departure: before and after peer r departs.

terms of the number of hops. It will ask each neighbour if it can join, starting with the peer closest to the source. To join a multicast tree, the peer will strike a *deal* with another peer: for each description received, one will be sent, possibly to a third peer. A deal will exist as long as both parties keep their part of the deal. Each peer keeps track of the deals it has with other peers. For each extra description a peer wants to receive, it will have to forward a description.

The mechanism of deals ensures that peers contribute as much outgoing bandwidth as they consume incoming bandwidth. The only exception to this mechanism is the source, which is willing to forward descriptions without expecting anything in return. The source makes sure it forwards every description at least once. 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 one peer redirects one of its output links through an other peer.

To strike these deals, peers exchange control information and send deal requests. Every peer exchanges updates with its neighbours about its colour and the colours it receives. If a peer receives 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 colour needs to be forwarded to whom, is sent along with the accept message.

3.2. Primitives to Build the Trees

In this section we will describe each of the three types of deals in more detail.

3.2.1. Join at the Source. If a peer p has the source in its neighbour set, and the source has spare outbound capacity, the peer can join a tree at the source. The source will then forward one of the descriptions to p . The source will favour sending a description to p that it is not yet forwarding to some other peer. Peer p will then get the colour of that description. An example of this is shown in Figure 1; here the source s decides to forward the red description to p .

3.2.2. Exchange Descriptions. To obtain another description, a peer p checks its neighbour set for a peer q , such that p does not receive q 's colour and q does not receive p 's colour. When p has found such a peer q , they strike an *exchange deal*: peer p forwards its colour to q , and vice versa, as shown in Figure 3.

3.2.3. Redirection. A peer p that has not yet joined any tree has the colour *blank*, and cannot exchange descriptions because it does not receive any. If p cannot join any tree at the source, it selects the non-blank peers in its neighbour set. It will ask each neighbour q in this set, in order of increasing hop count, whether q can accept it as a child. Just letting q forward its colour to p is not enough, as this would force q to spend more upload capacity than it uses download capacity, which is something Orchard tries to avoid. Instead, q looks at the streams it is already forwarding, and *redirects* one through p , as is shown in Figure 2. Peer q strikes a *redirection deal* with peer p , which replaces e with two links $f = (q, p)$ and $g = (p, r)$ of the same colour as e . That way q does not require additional bandwidth, and p is asked to receive the colour and also to forward it. If there is no suitable link to redirect, q will not accept p as a child.

Repeatedly redirecting a link $e = (q, r)$ would create a chain of peers, which is undesirable. To avoid this, only

links which were created by an exchange deal are redirected. Once such a link is redirected, it cannot be redirected again. So in Figure 2, 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. Peer p will get the colour 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 colour that is even closer to the source. In that case, p can switch parents by breaking the redirection deal with q , and striking a redirection deal with t .

3.2.4. Redirection Deals through Coloured Peers. The algorithm so far makes peers depend on exchange deals to obtain additional colours. For example, a blue peer p has to find a red neighbour which does not have an exchange deal for blue. If many red peers have such a deal, p has to contact many red peers to find one which does not have such a deal. If there are fewer red peers than blue peers, some blue peers will not be able to strike an exchange deal for red.

This problem could be solved by letting every peer connect to a large set (for example, 100) of neighbours, but this is undesirable from a practical point of view: if a peer has to connect to and query many neighbours in order to obtain colours, it takes longer to obtain them, especially if connecting involves DNS lookups and getting past firewalls. Also note that a large neighbour set requires more maintenance, as the chance increases that some neighbours will depart. For these reasons, Orchard has another way for peers to obtain additional colours: we also allow redirections through a non-blank peer p . This results in a transition similar to the one shown in Figure 2, except that p is (for instance) yellow on both the left- and the right-hand side.

This relaxation is provided as a backup system to allow peers to obtain all colours fast without having to query many neighbours, but makes the coloured peers compete with the blank peers for redirection deals. To prevent this, peers will break a redirection deal through a coloured peer if it can be replaced with a redirection through a blank peer. Because a coloured peer knows this can happen, it will give priority to obtaining the red colour through an exchange deal instead.

3.3. The Resulting Trees

We will now argue that in each tree, a peer forwards its colour to at most $d - 1$ other peers, where d is the number of colours in the system. By construction of Orchard, the out-degree of every peer is at most d . Two cases can be distinguished. Either a peer p joined at the source, or it joined further down the tree. In the first case, p received its own colour for free. In the second case, p joined through a

redirection deal, and as part of this deal has to forward its own colour to a peer of another colour (see Figure 2). In both cases, p receives at least one colour without forwarding anything to a peer of the same colour in return. Because a peer has at most $d - 1$ children of the same colour, the system needs to contain at least three colours, for otherwise all trees are linear chains.

Not considering the redirection deals through coloured peers, the interior nodes of the tree of colour c are all peers of colour c . This causes a single peer failure to result in the loss of at most one colour for other peers. Also, it results in shallow trees. This 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 m -ary tree of n nodes. The height of the tree can be expressed as $h(n) = \log_m(1 + (m - 1)n)$, and every node at distance d from the root is the child of d nodes. Let $c(n)$ be the average number of children of all nodes. Then, $c(n) = (1/n) \sum_{d=1}^{h(n)-1} dm^d$, which can be shown to be equal to $c(n) = h(n) + (h(n)/n - m)/(m - 1)$, which is of order $\log_m n$. So, $c(n)$ decreases when m increases.

Due to the construction, the Orchard algorithm thus meets the first two conditions stated in Section 2.3. The shallow trees, along with the tree repair mechanisms and the experiments that will follow, make a case for Orchard meeting the third condition.

3.4. Repairing the 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 Figure 2 on the right, if peer p fails, the redirection deal with q is cancelled. This causes q to restore the exchange deal with r as it was before it got redirected through p .

If the deal to be cancelled is an exchange deal, any redirection deal dependent on it is also cancelled. An example of this is shown in Figure 4. If peer r departs, q will break its exchange deal with r , which forces q to also break its redirection deal with p . Peer q was the parent of p in the tree of p 's colour. This causes p to stop receiving its colour, and makes p unable to maintain any exchange deals with other peers. Peer p can still maintain the redirection deals through coloured peers (p does not need its own colour for that). For simplicity, we force p to break all its deals and become blank.

These breaks propagate down the subtrees below each peer that stops receiving its own colour. In order to counter this massive failure, each peer keeps track of a set of *backup parents*: other peers in the tree of its own colour.

When too many peers become blank, there may be peers which have too many blank neighbours to obtain all colours. To counter this system degeneration, a peer removes a

neighbour from its neighbour set if that neighbour stays blank for a certain length of time, which, in our experiments, is three seconds. Note that since we assume each peer can keep its neighbour set populated, the neighbour set will not become depleted if peers are removed.

3.4.1. Backup Parents. In order for a peer to be able to quickly repair the loss of its parent of the same colour, each peer keeps track of which peers in its neighbour set are backup parents. To be a *backup parent* of p , a peer q has to be of the same colour, and the path from q to the source in that colour cannot contain p 's parent. The latter condition avoids the creation of cycles, and ensures that if p 's parent fails, each backup parent will still receive the stream. To facilitate this, every peer keeps its neighbours informed about its paths to the source.

When the parent of p fails, p takes its set of backup parents and asks each one, in order of increasing hop count from the source, whether p can join the tree below it. Peer p will join the tree at the first potential parent that allows it. If no potential parent allows p to join, p will break all its deals and tries to rejoin the system by becoming another colour.

3.4.2. Peers Changing their Colour. Unfortunately, the backup parent strategy is not enough. If a peer of colour c near the root of a tree fails, and the peers below it cannot connect to an independent subtree, those peers are all forced to rejoin the system and obtain a different colour. The colour 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 colours its neighbours are receiving. Once this set stabilises, a peer checks whether switching to a rarer colour is beneficial. A peer benefits from switching if it will be able to receive the rare colour, and will be able to strike more exchange deals than it could before switching. Once a peer switches colours, it breaks the deals with those peers that are not willing to accept the colour change, and strikes new deals with peers that are willing to.

4. Experiments

In this section we present our experimental setup and the results of four experiments with Orchard. The first experiment shows the increase in performance by allowing redirection deals through coloured peers, as mentioned in Section 3.2.4. Experiments two and three assess the performance of Orchard under flash crowds and churn, respectively. The fourth experiment tests the performance of Orchard on Delft-37.

Delft-37 node	peers	Delft-37	peers
USA (Arizona)	9	UK	8
USA (New York)	14	Germany	15
USA (Washington)	11	Singapore	4
USA (Georgia)	5		

Table 1. The workload of Delft-37.

4.1. Experimental Setup

The Orchard algorithm has been tested in two emulation environments. The first environment is the DAS2 distributed computer [1], using 20 processing nodes. The second environment in which we have tested the Orchard algorithm is a wide-area network which we refer to as Delft-37, which consists of hosting accounts we use at various ISPs around the globe. These accounts range from web accounts to virtual private servers. All 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 [6] 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 1). The bandwidth between these nodes was 20–600 kbit/s, and the latency between these nodes varied between 30–400 ms.

We implemented the Orchard algorithm in Python. When it is started, it emulates a single peer which sets up TCP connections with each neighbour to exchange control information. We used this emulator to do four experiments, which evaluate the number of descriptions received under varying conditions. In these experiments we 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 video stream.

In our experiments, we assume four MDC descriptions. Every peer maintains a neighbour set of at least 15 peers. We have a central rendezvous server which keeps track of all peers interested in the video and which sends sets of random, uniformly selected peers upon request. Peers keep their neighbour sets sufficiently populated (by contacting the server) when links to peers in the set are lost. If a peer does not know 15 neighbours, it obtains a set of at most 20 extra neighbours from the central server.

Peers arrival is modelled as a Poisson process with an average of two per second. In a recent measurement of a large-scale Video-on-Demand system [15], it was shown that about half the peers disconnect within 10 minutes. Our measurements take a more pessimistic scenario into account, in order to be able to cope with zapping behaviour which we expect to rise if P2P video multicasting becomes more mainstream. In those tests were peers depart, we let peers stay in the system for 120 seconds on average, using

number of descriptions	fraction of time		
	(Figure 5(b))	(Figure 6)	(Figure 8)
at least 1	0.995	0.989	0.987
at least 2	0.994	0.987	0.982
at least 3	0.993	0.978	0.968
4	0.984	0.874	0.846

Table 2. The cumulative distribution of the average number of received descriptions.

an exponential distribution.

4.2. Redirection Deals through Coloured Peers

In the first test, we measure the decrease in performance if redirection deals through coloured peers are disallowed (Limited Orchard). This is compared to the Orchard algorithm, where peers can also gain additional colours by striking redirection deals.

We let 500 peers arrive in the system and not depart. We measured the number of peers receiving a certain number of descriptions over time, using the two algorithms. The result of this is shown cumulatively in Figure 5(a) and 5(b).

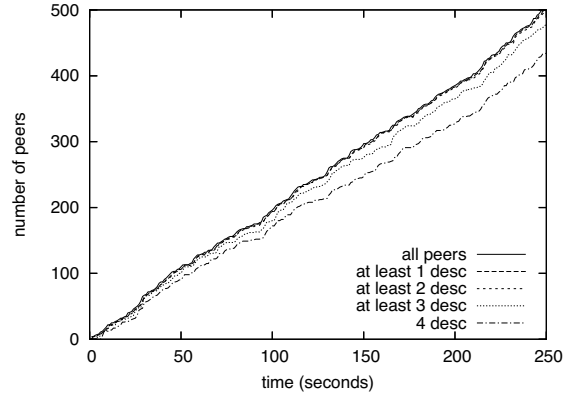
The contrast between these two graphs is clear: with the limited algorithm, not all peers can obtain all descriptions. Once 500 peers have arrived, only 84% of them receive all four descriptions. When using the full algorithm the situation improves considerably: 98.4% of the peers is able to receive all four descriptions once all 500 peers have arrived.

4.3. Flash Crowds

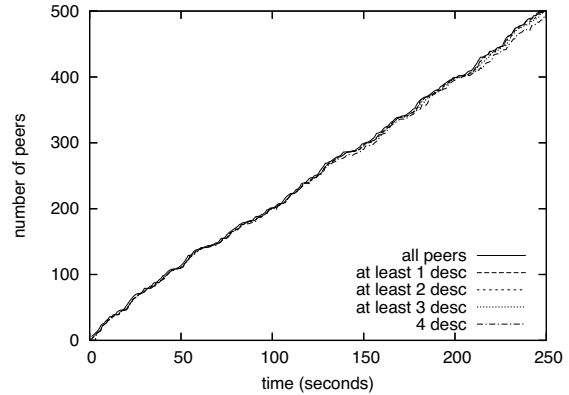
In the second test we let 500 peers arrive and depart. This creates a more realistic situation than in the previous experiment. In Figure 6, we show the number of peers which receive a certain number of descriptions over time for a typical run. To keep the main graph readable, only the number of peers receiving all four descriptions and the total number of peers are represented. The sub-graph 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 failures, 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 2, 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. It is desirable to have shallow trees, as argued in Section 3.1. The optimal value for the average number of hops from the source to a peer can be calculated as follows. In Section 3.3, we argued that a peer forwards its colour to at most $d - 1$ other peers of the



(a) Limited Orchard.



(b) Orchard.

Figure 5. The number of received descriptions.

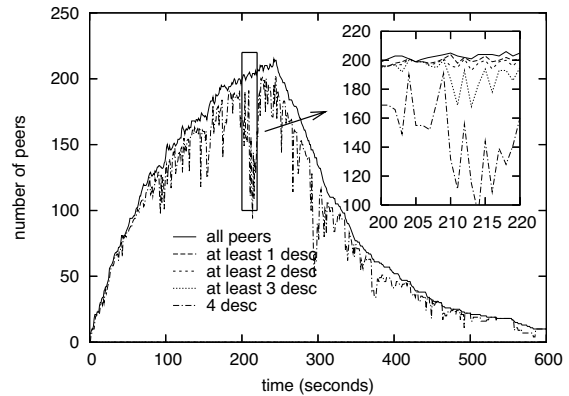
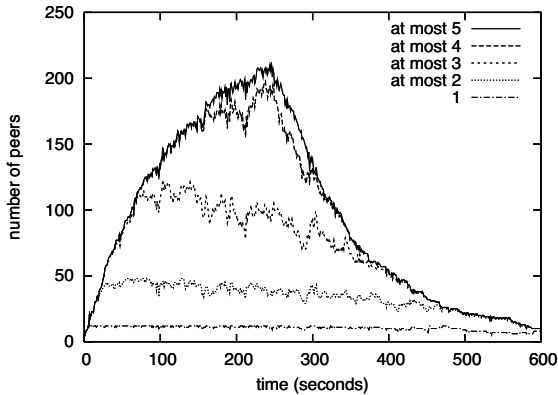
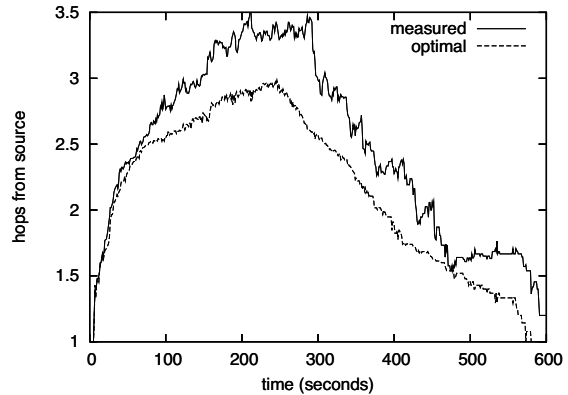


Figure 6. The number of received descriptions under flash crowds.

same colour. In our tests, we have $d = 4$ colours, and the source forwards each colour to at most 3 peers. The nodes of a single colour (plus the source) thus form a ternary tree. The minimal average distance from the source to each peer



(a) The distribution of the path lengths.



(b) The optimal and measured average path lengths.

Figure 7. The distance from the source to the peers, in the trees of their own colours.

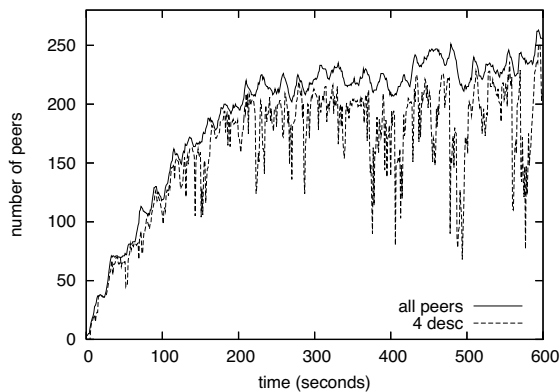


Figure 8. The number of received descriptions under churn.

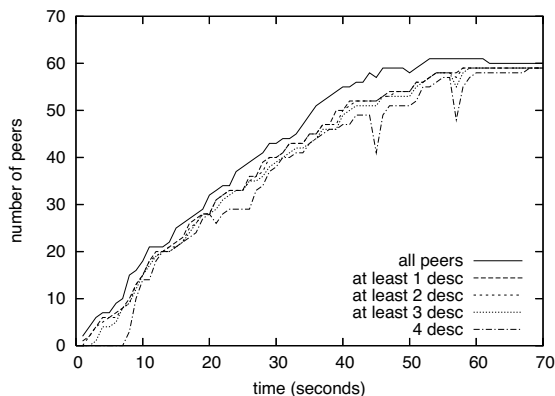


Figure 9. The number of received descriptions when using Orchard in Delft-37.

can be calculated accordingly.

Measurements in the same experiment of the height of

the trees over time is shown in Figure 7(a). Each line shows the number of peers at most a certain number of hops from the source, in the tree of their own colour. Even though peers are constantly arriving to and departing from the system, the trees do not degenerate. This is due to the fact that peers switch to parents of the same colour but closer to the source, whenever possible.

We compare the average measured distance from a peer to the source against the optimal value in Figure 7(b). As can be seen in the figure, every peer is at most one hop further away from the source than in the optimal case.

4.4. Churn

In the third test, we let 1200 peers arrive and depart to analyse the resilience of Orchard under churn, which is the rapid arrival and departure of many peers in the network.

The result of this test is shown in Figure 8. On average, the peers are still able to receive at least one description 99% of the time (see also Table 2). The continuous peer departure causes big drops in the number of peers which are receiving all four colours, 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.

4.5. Delft-37

In the last experiment, we distributed the peers over the Delft-37 network. The Delft-37 nodes we used cannot all host the same number of peers. In this test, we started 65 peers, with a distribution as shown in Table 1. The results of this experiment are shown in Figure 9.

Every Delft-37 node forms the end point of a large number of TCP connections, since every peer maintains a TCP connection with all of its neighbours. 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 presume this happened due to the high number of TCP connections already started on that node. The 60 peers that did not depart all obtained all colours.

5. Related Work

The idea of using MDC [8] in ALM has been proposed before [4, 11, 14]. Pouwelse et al. [11] propose the use of a bartering mechanism, but do not provide an algorithm. SplitStream [4] uses MDC to create interior-node-disjoint trees, but does not provide mechanisms to enforce a fair resource contribution.

It is not necessary to use MDC to create resilience against peer failure. In Chainsaw [10], 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 neighbours. This makes it extremely resilient against peer failure, because a failing neighbour causes a peer to (re-)request the pieces from someone else.

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 [9]. Unlike those algorithms, Orchard enforces fair resource contribution and is still resilient against high-pace peer arrivals and departures.

6. 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 which the burden of uploading is shared 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. Also, the use of multiple descriptions makes the forest robust against peer failure.

Through emulation, we have shown that the Orchard algorithm can operate under flash crowds and churn. The trees constructed by Orchard do not degenerate when there are many peer arrivals and departures. By testing our implementation on our Delft-37 network, we have shown that Orchard can operate under realistic conditions.

In the future, we will extend this research with a theoretical analysis of Orchard's performance. Although Orchard in

itself is not complicated, such an analysis is non-trivial due to the highly dynamic nature of a P2P environment. Next to this, we will implement a complete application for P2P real-time video distribution, using Orchard.

References

- [1] DAS: Distributed ASCII Supercomputer. <http://www.cs.vu.nl/das2/>.
- [2] I-Share. <http://ishare.ewi.tudelft.nl>.
- [3] N. Andrade, M. Mowbray, A. Lima, G. Wagner, and M. Rippeanu. Influences on Cooperation in BitTorrent Communities. In *Proc. of ACM SIGCOMM*, pages 111–115, 2005.
- [4] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth multicast in a cooperative environment. In *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, pages 298–313, 2003.
- [5] Y. Chu, S. Rao, and H. Zhang. A Case for End System Multicast. In *Proc. of ACM SIGMETRICS*, pages 1–12, 2000.
- [6] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM Computer Communication Review*, 33(3):3–12, 2003.
- [7] R. Ferreira, M. Ramanathan, A. Awan, A. Grama, and S. Jagannathan. Search with Probabilistic Guarantees in Unstructured Peer-to-Peer Networks. In *Proc. of the 5th Int. Conf. on Peer-to-Peer Computing*, pages 165–172, 2005.
- [8] V. Goyal. Multiple Description Coding: Compression Meets the Network. *IEEE Signal Processing Magazine*, 18(5):74–93, 2001.
- [9] L. Mathy, N. Blundell, V. Roca, and A. El-Sayed. Impact of Simple Cheating in Application-Level Multicast. In *Proc. of IEEE INFOCOM*, volume 2, pages 1318–1328, 2004.
- [10] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr. Chainsaw: Eliminating Trees from Overlay Multicast. In *Proc. of the 4th Int. Workshop on Peer-To-Peer Systems (IPTPS)*, pages 127–140, 2005.
- [11] 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 *IEEE Conference on Systems, Man and Cybernetics*, pages 4599–4605, 2004.
- [12] S. Saroiu, P. Gummadi, and S. Gribble. Measurement Study of P2P File Sharing Systems. In *Proc. of SPIE, Multimedia Computing and Networking Conference (MMCN)*, volume 4673, pages 156–170, 2002.
- [13] D. Stutzbach and R. Rejaie. Characterizing Churn in Peer-to-Peer Networks. Technical Report CIS-TR-2005-03, University of Oregon, 2005.
- [14] J. Taal and R. Lagendijk. Fair Rate Allocation of Scalable Multiple Description Video for Many Clients. In *Proc. of Visual Communications and Image Processing*, pages 2172–2183, 2005.
- [15] H. Yu, D. Zhang, B. Zhao, and W. Zheng. Understanding User Behavior in Large Scale Video-on-Demand Systems. In *Proc. of ACM EuroSys (to appear)*, 2006.