High-Bandwidth Mesh-based Overlay Multicast in Heterogeneous Environments

Bartosz Biskupski, Raymond Cunningham, Jim Dowling, and René Meier
Distributed Systems Group
Trinity College Dublin, Ireland
{biskupski,racunnin,jpdowlin,rmeier}@cs.tcd.ie

ABSTRACT

In this paper we present MeshCast, a peer-to-peer (p2p) multicast protocol for applications requiring high bandwidth (such as live video streaming) from a server to a large number of receivers. Traditional tree-based approaches to overlay multicast inefficiently utilise the outgoing bandwidth of participating nodes and poorly adapt to node membership churn. In contrast, MeshCast is based on Chainsaw mesh-based approach to data delivery that better utilises bandwidth and provides excellent adaptation properties. In this paper we identify properties that enable mesh-based overlay multicast protocols to better utilise the available bandwidth and consequently support higher data stream rates in heterogeneous environments. MeshCast uses a gossip-based algorithm to adapt the overlay to peer heterogeneity, while still preserving the advantages of a mesh-based overlay. Our experiments show that MeshCast can support 68% higher stream rates and provides a 22% improvement in buffering delay over the recently proposed Chainsaw protocol for a heterogeneous node bandwidth distribution.[1]

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms

Algorithms, Design, Performance

Keywords

Heterogeneity, multicast, multimedia streaming, peer-to-peer systems

[1]The work described in this paper was partly supported by the "Information Society Technology" Programme of the Commission of the European Union under research contract IST-507953 (DBE)

http://doi.acm.org/10.1145/1190183.1190187
AAA-Idea '06, October 10, 2006, Pisa, Italy

1. INTRODUCTION

There has been a growing interest in peer-to-peer multicast for large, both in scale and volume, content delivery due mainly to the lack of widespread support for IP multicast and the high cost of infrastructure-based approaches using dedicated/replicated servers. Peer-to-peer multicast is attractive in this setting because the bandwidth available to serve content scales with demand (i.e., the self-scaling property). Therefore, a grand challenge for p2p multicast protocols is to efficiently utilise the available bandwidth of participating nodes.

A common approach to p2p multicast is to organise nodes into a tree-structured overlay with a root at the source node [4]. The tree structure defines the routing decisions – a node receives data from its parent and forwards it to all its children. This approach, however, has a number of problems. Firstly, it does not utilise the outgoing bandwidth of a large fraction of nodes that are leaves in the tree. Secondly, the received bandwidth is limited by the minimum bandwidth on the path to the source and any loss in the upper level of the tree reduces bandwidth available to nodes lower in the tree. Finally, it has poor resilience to churn (a node departure results in the data stream being lost at all its descendants until the tree is fixed). Multiple tree approaches [3] aim at solving the first two problems, but do not solve the third problem as they require more tree structures to be maintained.

Recently many researchers have switched focus to build multicast protocols using overlay meshes to overcome the aforementioned problems (e.g., [5, 7, 10] outlined in Section 5). The approach of these systems is that the nodes organise into an overlay mesh by selecting multiple nodes as neighbours to exchange data with. In order to enable data exchange between neighbours, the disseminated data is split into smaller data blocks or chunks. Each time a node receives a data block, it informs its neighbours and they can request this missing block. This improves resiliency to churn and enables the bandwidth of all nodes to be utilised. With a sufficient number of neighbours, the failure of a node goes unnoticed in the system because neighbouring nodes can choose to download data from other neighbours. The contributions of this paper are:

- we identify properties that enable mesh-based overlay multicast protocols to better utilise the available bandwidth of nodes in a heterogeneous environment
- we propose to apply a simple topology adaption algorithm to mesh-based multicast streaming to enable

the efficient utilisation of the available bandwidth and evaluate this resulting protocol

In Section 2, we introduce the general problem of mesh-based overlay multicast on the example of the Chainsaw protocol [10] and highlight issues that arise in heterogeneous environments. Section 3 describes our protocol that addresses these issues. In Section 4, we present experimental results. Section 5 outlines recent related work. In Section 6, we outline some future work and conclude the paper.

2. P2P MULTICAST IN HETEROGENEOUS ENVIRONMENTS

In this paper we consider a cooperative mesh-based overlay multicast in which a single data stream is generated "live" at a source node and is disseminated to all nodes. The bandwidth distribution of nodes is considered to be heterogeneous with nodes having the incoming bandwidth larger, equal or lower than their outgoing bandwidth. Nodes contribute their outgoing bandwidth irrespective of how much bandwidth they receive, however, support for fairness over multiple multicast sessions can be added through a credit-based reputation system (e.g., [9]). We are interested in designing a protocol that maximises the stream rate that all nodes receive. In order to do this, the system needs to satisfy two conditions – maximise the node outgoing bandwidth utilisation and deliver the same data rate to all nodes. The theoretical upper bound on this data stream rate is given by the formula $\frac{\sum_{i=1}^{N} outbw_i}{N-1}$, where $outbw_i$ is the node i outgoing bandwidth and N is the total number of nodes in the system (the source node contributes its outgoing bandwidth, but does not consume any). This is under an assumption that all participating nodes incoming bandwidth is not lower than this stream rate (which is usually accomplished due to the popularity of asymmetric Internet connections with high download capacity). Our goal is to approach this theoretical limit.

In contrast to mesh-based file-sharing systems such as BitTorrent [5], the entire file is not available to live streaming protocols and thus it cannot be split into blocks for distribution throughout the network. In order to leverage mesh-based delivery, streaming protocols require that there is a delay between the stream creation time at the source node and the receiver playback time. The data stream produced within this delay is split into small blocks and distributed throughout the network similarly to how entire file blocks are distributed in mesh-based file-sharing protocols. In the Chainsaw protocol nodes maintain *sliding buffers* (called windows of interest) that reflect this delay and define which blocks have been already received and which are still missing. The buffers move forward with the speed of the original video rate that is globally known. The beginning of the buffer points at the block currently being played at the node and the end of the buffer is the currently generated block at the source node. Blocks that do not arrive in time (are outside the sliding buffer) are lost and result in degraded video quality.

The mesh overlay is created in Chainsaw in a random fashion by joining nodes connecting with randomly selected nodes. Neighbouring nodes maintain local knowledge about blocks they possess by informing each other whenever they receive a new block. Nodes request missing blocks from neighbours applying some *block selection strategy*. In Chainsaw, nodes pick blocks at random in order to increase the disjointness of blocks possessed by neighbouring nodes. The ability to concurrently upload/download data blocks from many neighbours (called also swarming) is one of the advantages of mesh-based systems. It enables node incoming bandwidth to be maximised (i.e., using the outgoing bandwidth of many neighbours) and improves resilience to congestion (i.e., if one uploader becomes congested, some traffic can be easily shifted to another uploader).

The above general scheme for mesh-based multicast, which is employed by Chainsaw, achieves excellent bandwidth utilisation (and consequently supports high data stream rates) in homogeneous environments where all nodes have the same upload and download capabilities [10]. However, this is not a realistic scenario as today's Internet consists of heterogeneous hosts having asymmetric upload and download capacities [12]. The following properties should be addressed in heterogeneous environments to improve bandwidth utilisation:

- the sender and the receiver of each data transfer should be bandwidth-matched. The incoming bandwidth allocated by the receiving node to enable a data transfer should not be lower than the outgoing bandwidth allocated by the sending node. Otherwise, a receiver with insufficient allocated incoming bandwidth would constrain the sender's outgoing bandwidth utilisation

- a topology has to guarantee that all nodes are able to receive data at a desired rate. A randomly formed topology may result in some nodes having a majority of neighbours with low upload bandwidth and are thus unable to receive the stream at the desired rate. Moreover, much of their outgoing bandwidth is unused since many blocks are not needed by their neighbours by the time these nodes manage to download them

3. MESHCAST

In this section we present the MeshCast protocol that exhibits the properties covered in the previous section. We assume that each node estimates its maximum outgoing (*outbw*) and incoming (*inbw*) bandwidth (e.g., using bandwidth estimation tools such as [13]). Each node in MeshCast maintains two sets of neighbours – *receivers*, which are the neighbours that it can potentially upload data to and *senders*, which are the neighbours that it can potentially download data from. Naturally, one node has another node in its receivers set precisely when the latter has the former in its senders set. Whenever a node receives a new data block, it informs all its receivers. Nodes request missing blocks in a random order from their senders (i.e., use the random block selection strategy).

MeshCast adaptability to bandwidth heterogeneity is based on an observation that all nodes can maximise their outgoing bandwidth utilisation and receive the same aggregated data rate if the following conditions are satisfied:

1. all nodes have the number of receivers proportional to their outgoing bandwidth, i.e.,

$$\forall_{i,j} \frac{outbw_i}{|receivers_i|} = \frac{outbw_j}{|receivers_j|}$$

```
loop
  wait( δ time units )
  p ← random peer from view
  send view ∪ {(myAddress, 0, |senders|)}  to p
  receive view_p from p
  view ← view ∪ view_p
  view ← view / {(myAddress, *, *)}
  view ← select s freshest unique descr. from view
  view ← increase each descriptor's age in view
  balanceSenders()
end loop
```

Algorithm 1: sending thread

and they share the outgoing bandwidth equally among the receivers

2. all nodes have an equal number of senders, i.e.,

$$\forall_{i,j}|senders_i| = |senders_j|$$

Therefore, nodes in MeshCast maintain a fixed number of receivers equal to $\frac{outbw}{\alpha}$ and upload data to each receiver with the speed of α (where the value of α expresses the trade-off between what should be the lowest bandwidth that enables a node to contribute versus the maximum number of concurrent connections that a high-bandwidth node needs to support). In order to continuously balance (equalise) the number of senders at nodes, MeshCast employs an underlying gossip-based topology adaptation algorithm. As a side effect, the gossip protocol in each gossip round provides nodes with a fresh random sample of nodes. These are used for building random sets of receivers and replacing receivers that leave the overlay (stop requesting blocks) with new ones in order to keep a fixed number of them.

The algorithm we use for gossiping is based on Newscast [6]. Nodes maintain *partial views* that are fixed-size sets of node *descriptors* (s is the size of the views). Each descriptor contains the address of a node, age of the descriptor (increased with each gossip) and a number of senders that the node had when the descriptor was created. Periodically (every δ time units), each node generates a fresh descriptor (with the age set to zero and its updated number of senders), adds it to its view and exchanges the view with a randomly selected neighbour (see Algorithms 1 and 2 for the sending and receiving threads). After the view is exchanged, a gossiping node selects s freshest unique descriptors (with minimal age values) that do not represent itself for the new view, increases each descriptor's age and initiates the senders balancing algorithm. The presented gossiping algorithm has a number of nice properties such as self-healing (descriptors of nodes that leave the system are eventually removed from partial views), resilience even to catastrophic failures (where a high proportion of nodes fail at the same time) and low communication overhead [6].

The random samples of nodes produced by gossiping are used for balancing the number of senders among all nodes (see Algorithm 3). A node running the balancing algorithm compares its own number of senders to the number of senders stored in each of the descriptors to find a neighbour with maximally different number of senders. It requests the selected neighbour to transfer its updated number of senders (as it might have changed since the time the descriptor was produced) and equalises the number of senders for both

```
loop
  receive view_p from p
  send view ∪ {(myAddress, 0, |senders|)}  to p
  view ← view ∪ view_p
  view ← view / {(myAddress, *, *)}
  view ← select s freshest unique descr. from view
  view ← increase each descriptor's age in view
  balanceSenders()
end loop
```

Algorithm 2: receiving thread

```
p ← peer with most different number of senders
//p is not a source node
k ← request the p's number of senders
toTransfer ← ⌞ | k - |senders| | / 2⌟
if (k > |senders|) then
  request toTransfer senders from p
  //also request each of these senders
  //to replace receiver p with this node
else
  transfer toTransfer senders to p
  //also request each of these senders
  //to replace this receiver with node p
end if
```

Algorithm 3: balanceSenders()

nodes by transferring some senders to or from the neighbour. The transferred sender nodes are informed that their sets of receivers have changed.

4. EVALUATION

4.1 Simulation Details

We developed a discrete-time simulator for both MeshCast and Chainsaw [10] evaluations. Our simulator is able to model heterogeneous networks with different node downlink and uplink bandwidths. The simulator uses the bandwidth settings to appropriately calculate the amount of data that can be transfered between nodes in each simulation step. This requires that for each data transfer the simulator takes into account the number of flows concurrently sharing the uplink and downlink at either end. Since providing a realistic model of the network can be prohibitively expensive in terms of simulation, we decided to introduce some simplifications. Similarly to [1] we assume that one uplink or downlink is shared by all flows equally and bottlenecks appear either at the uplink of the sending node or downlink of the receiving node (i.e., the Internet core has a sufficient bandwidth capacity). These simplifications improve the scalability of our simulator and enable us to model mesh-based multicast streaming for 10000 nodes.

In our evaluation we use a node bandwidth distribution derived from the Gnutella p2p system measurements [12] (see Table 1). Similarly to [1], we discretise the CDFs presented in [12], excluding the tail of the distribution. Most of the excluded nodes have download bandwidth not sufficient for receiving the data stream. The source node has been set to belong to the first category with 5 Mbps uplink. The protocol parameters used in the experiments (unless specified otherwise) are presented in Table 2. In the experiments we ignore node membership churn and assume that all nodes

Downlink	Uplink	Ratio
10 Mbps	5 Mbps	20%
3 Mbps	1 Mbps	30%
1.5 Mbps	384 Kbps	50%

Table 1: Node bandwidth distribution

Parameter	Value
Number of nodes	10000
Streaming duration	1000 sec
Block size	16 KB
α	128 Kbps
Gossip view size (s)	30
Gossip interval (δ)	2 sec
Avg. neighbourhood size in Chainsaw	20

Table 2: Protocol parameters

join at the beginning of the experiments and stay until the end (with the exception of the experiments in Section 4.4). In experiments with Chainsaw we assume that the initial topology is random with some average node neighbourhood size.

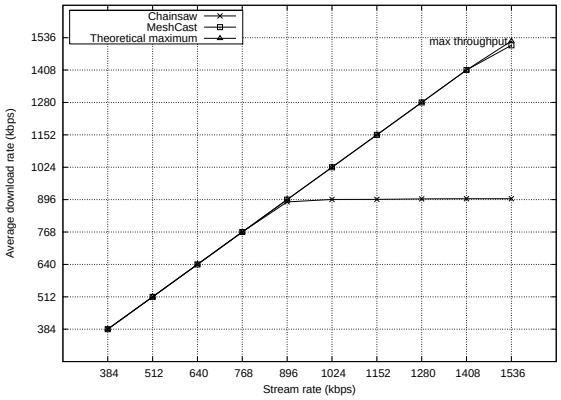
4.2 Maximum Stream Rate

In Section 2 we showed a formula for the theoretical upper bound on the stream rate to each node in any multicast protocol. For our considered node bandwidth distribution this limit reaches 1523 Kbps. Notice, that the incoming bandwidth of all nodes is large enough to support this rate. In Figure 1(a) we compare the average download bandwidth achieved by nodes in Chainsaw and MeshCast protocols as a function of the stream rate. Clearly, the node download rate is limited by the stream rate as it is not possible to receive data faster than it is produced. In the next subsection we show that all nodes in MeshCast receive data at the same rate. Figure 1(b) compares the outgoing bandwidth utilisation that is defined as the ratio of the total amount of data uploaded to the total available upload in the system. The results show that MeshCast can support 1509 Kbps stream rates, which is 99% of the theoretical maximum, outperforming Chainsaw that supports data streams at the maximum 899 Kbps rate in the same settings. This is a 68% improvement in the system throughput after the topology adaptations performed by MeshCast.

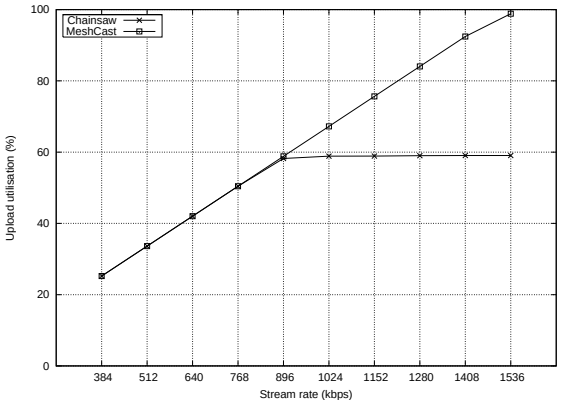
4.3 Minimum Buffering Delay

Mesh-based streaming protocols require a buffering delay since nodes download data blocks in non-sequential order, while the video playback requires sequential data ordering. We define a *minimum buffering delay* as the minimum length of the buffer in seconds required to avoid any data loss. A receiving node's *progress* is defined as the source creation time of the latest block of contiguous data downloaded (or produced in the case of the source node) by this node (see Figure 2). Thus, the maximum difference between the node's progress and the source's progress represents the minimum buffering delay.

In Figure 3 we show the average progress of nodes for different streaming rates compared to the source node progress. It can be seen that the minimum tolerable buffering delay of an average node for the highest considered stream rate



(a) Average download rates



(b) Outgoing bandwidth utilisation

Figure 1: Maximum stream rates for Chainsaw and MeshCast protocols

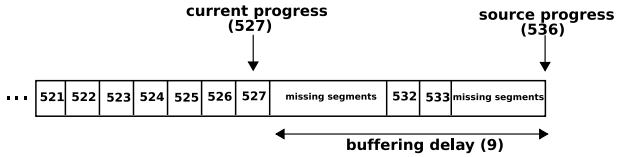


Figure 2: Buffer and progress of a node

(1408 Kbps) is 10 seconds (i.e., the difference between the source's progress and the earliest possible playback time). In order to show that the variance of the progress between nodes is small and that all nodes receive data at the desired rate, we show the earliest playback time for the worst node in the system (it is only 2 seconds behind the average node for the 1408 Kbps stream rate).

Figure 4 compares the average and the worst node's buffering delay in MeshCast and Chainsaw protocols for the 768 Kbps stream rate, which is a rate still supported by Chainsaw. The results show that MeshCast exhibits 22% shorter average buffering delay than Chainsaw and additionally the variance between nodes is smaller (only 1.3 seconds difference between the average and the worst node).

4.4 Topology Adaptation

Figure 5 investigates the performance of the senders balancing algorithm for an initial uniformly random topology.

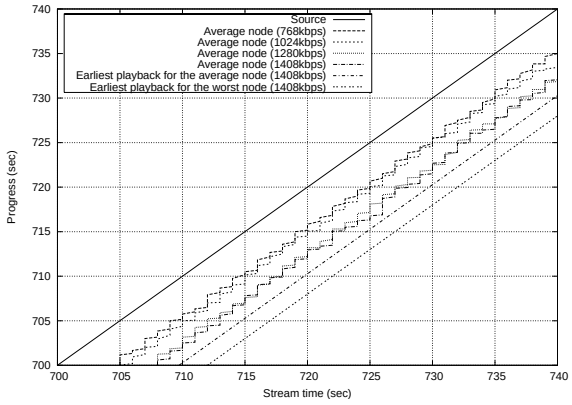


Figure 3: Average node progress as a function of time in MeshCast

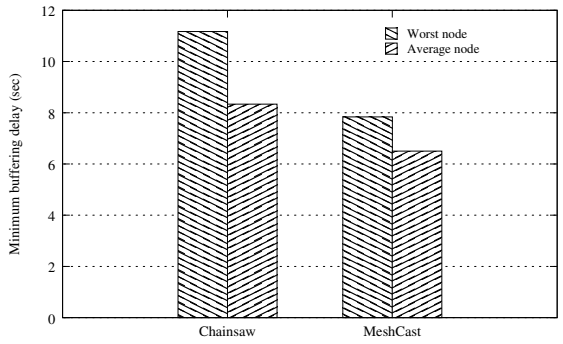


Figure 4: Comparison of the minimum buffering delay for the 768 Kbps data rate (rate still supported by Chainsaw)

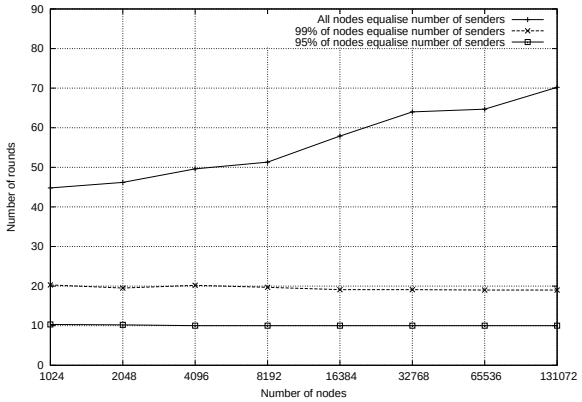


Figure 5: Number of gossip rounds required to reach the optimal number of senders by nodes as a function of the number of nodes.

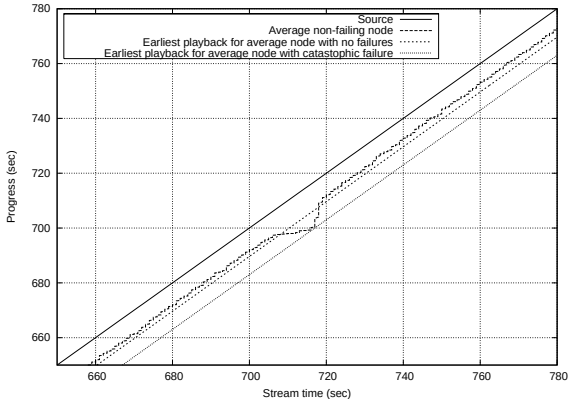


Figure 6: Impact of a catastrophic failure (50% nodes fail in the 700th second) on the average node progress (for the 1408 Kbps stream rate)

It shows the number of gossip rounds required to reach the balanced (equal) number of senders by all nodes, 99% of nodes and 95% of nodes as a function of the total number of nodes in the system (increasing exponentially). It can be seen that while the optimal adaptation is a logarithmic function of the total number of nodes, the almost optimal adaptation is constant and is reached in 19 gossip rounds by 99% of all nodes.

We investigated the impact of a catastrophic failure, where suddenly 50% of nodes fail in the 700th second, on the average node progress for the 1408 Kbps stream rate. It can be seen from Figure 6 that the topology managed to reconstruct itself and nodes came back to the normal progress within 12 seconds. The buffering delay required to absorb the catastrophic failure, such that playback is not disturbed, should be 7 seconds longer.

4.5 Protocol Overhead

In order to calculate the protocol communication overhead, notice that for each received block a node sends $\frac{outbw}{\alpha}$ control packets to inform all its receivers. An additional one control packet per non-source node is required to request each block from a sender. Thus, in a network with N nodes, each block generates $(\frac{\sum_{i=1}^{N} outbw_i}{\alpha}) + N - 1$ control packets, which for the particular node bandwidth distribution and protocol parameters from Table 2 is approximately equal to $13 * N$. In contrast, the Chainsaw overhead depends on its neighbourhood size (20 used in our experiments) and approximately equals to $21 * N$. A nice property of MeshCast is that the communication overhead incurred on a node is proportional to its bandwidth capacity. We believe that this communication overhead is acceptable compared to the total data being transmitted, considering that the application domain is high-bandwidth streaming.

MeshCast incurs also communication overhead caused by the gossip-based senders balancing algorithm. The gossip algorithm requires on average two packets sent for exchanging the view per one gossip interval [6] (one packet sent by the sending thread and one by the receiving thread). Figure 7 presents the communication overhead of the senders balancing algorithm as a function of the number of nodes (exponentially increasing). It shows the average number of packets sent in total by each node in order to equalise the number of senders at all nodes, 99% of nodes and 95% of nodes. The presented overhead includes packets used to request the current number of senders from neighbours as well as packets used to transfer senders and notify these senders about the change in their receivers sets. The overhead of both the gossip and the balancing algorithm is small, especially compared to the high data stream rates.

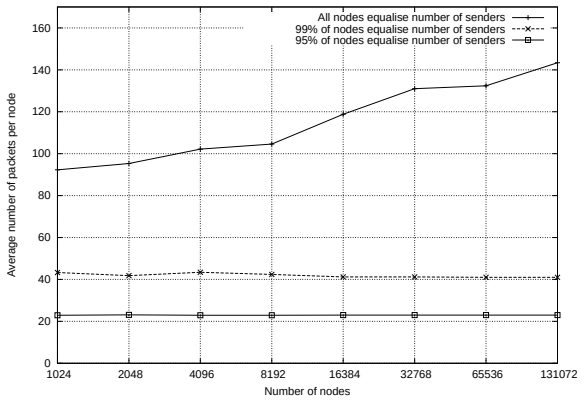


Figure 7: Average number of packets sent in total by a node in the balancing algorithm

5. RELATED WORK

SplitStream [3] is a multiple tree based multicast protocol based on top of the Scribe [4] single tree-based system, which is in turn built on top of a Distributed Hash Table (DHT). In SplitStream, the original data stream is divided into several substreams, each having an individual distribution tree. In an effort to utilise outgoing bandwidth of all nodes, trees are built such that each node is an interior node in exactly one tree. However, SplitStream has poor adaptation to node failures compared to mesh-based protocols. Moreover, it has been shown that multicast protocols built on DHTs suffer from a mismatch between the DHT identifier space and the node outgoing bandwidth, resulting in limited system throughput [2].

Bullet [8] splits the stream into blocks and uses a single tree on top of a mesh. Nodes receive a subset of blocks from their parents in the tree, while the remaining blocks are recovered from nodes in the mesh overlay. This has been shown to improve bandwidth utilisation compared to single tree approaches. Authors also noticed the need for adaptation to node bandwidth. Thus, nodes in the tree compute the best rate to send data to each of their children, while nodes in the mesh overlay connect to the most useful neighbours. However, Bullet wastes bandwidth on receiving duplicate packets and its reliability to node failures is lower than in pure mesh-based protocols because of the requirement of maintaining the tree structure.

BitTorrent [5] is a file-sharing protocol. It uses an optimistic unchoking mechanism that explores potential neighbours in order to match nodes with similar bandwidth and disjoint data and consequently better utilise their bandwidth. The optimistic unchoking exploration, however, requires uploading to suboptimal nodes and wastes outgoing bandwidth. Bharambe et al. [1] suggested matching nodes based on their bandwidth when they join the overlay. This, however, results in a set of completely disjoint bandwidth matched clusters of nodes forming. Thus, only a fraction of a node's neighbourhood can be matched and the rest have to keep the network connected. In contrast, MeshCast avoids network clustering by remaining a random topology influenced by only the node fan-out. Finally, optimistic unchoking works relatively well in file-sharing protocols, but is not suitable for live multicast streaming protocols. This is because streaming requires short buffers that results in two nodes having a few unique blocks at a time and consequently frequent switching between neighbours.

MeshCast is similar in its approach to Bullet' [7] in that it splits neighbours into senders and receivers as well as adapts the number of concurrent uploads and downloads to the available bandwidth. However, Bullet' is a file-sharing protocol rather than a multicast streaming protocol and, similarly to BitTorrent, benefits from long lasting connections between nodes. Thus, nodes explore potential senders and receivers to select the best performing ones.

Finally, in Sec. 2 we presented the related Chainsaw protocol [10] that MeshCast extends.

6. CONCLUSIONS AND FUTURE WORK

In this paper we identified properties that improve bandwidth utilisation in heterogeneous environments and consequently enable support for higher data stream rates. We presented MeshCast that uses a gossip-based overlay adaptation to take advantage of node heterogeneity. We evaluated the protocol using a custom built simulator in an example environment derived from a particular node bandwidth distribution. The simulation results show that our protocol supports stream rates 68% higher than Chainsaw and within 99% of the theoretical upper bound, provides a 22% improvement in buffering time over Chainsaw and self-adapts the topology even in the case of catastrophic failures.

In the future, we are planning to implement MeshCast in the Mace/MACEDON framework [11] and to deploy it in a wide-area network emulator such as ModelNet [14] to better model network dynamics and to compare it to other existing protocols (e.g., SplitStream and Bullet). We are also investigating methods to further reduce the required buffering delay through the topology adaptation and improved block selection strategies.

7. REFERENCES

[1] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and improving a BitTorrent network's performance mechanisms. In *Proceedings of INFOCOM'06*, 2006.

[2] A. R. Bharambe, S. G. Rao, V. N. Padmanabhan, S. Seshan, and H. Zhang. The impact of heterogeneous bandwidth constraints on DHT-based multicast protocols. In *IPTPS*, pages 115–126, 2005.

[3] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: high-bandwidth multicast in cooperative environments. In *SOSP'03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 298–313, New York, NY, USA, 2003.

[4] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized publish-subscribe infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC)*, 20(8):1489–1499, October 2002.

[5] B. Cohen. Incentives build robustness in BitTorrent. In *the 1st Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, June 2003.

[6] M. Jelasity, W. Kowalczyk, and M. van Steen. Newscast computing. Technical Report IR-CS-006, Dept. of Computer Science, Vrije Universiteit Amsterdam, The Netherlands, 2003.

[7] D. Kostic, R. Braud, C. Killian, E. Vandekieft, and J. W. Anderson. Maintaining high bandwidth under dynamic network conditions. In *Proceedings of 2005 USENIX Annual Technical Conference*, 2005.

[8] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 282–297, 2003.

[9] A. Nandi, T.-W. Ngan, A. Singh, P. Druschel, and D. S. Wallach. Scrivener: Providing incentives in cooperative content distribution systems. In *Middleware*, pages 270–291, 2005.

[10] V. S. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. Chainsaw: Eliminating trees from overlay multicast. In *IPTPS*, pages 127–140, 2005.

[11] A. Rodriguez, C. Killian, S. Bhat, D. Kostic, and A. Vahdat. Macedon: Methodology for automatically creating, evaluating, and designing overlay networks. In *Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation (NSDI '04)*, San Francisco, CA, USA, March 2004.

[12] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking*, 2002.

[13] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *IMC'03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 39–44, New York, NY, USA, 2003.

[14] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. *SIGOPS Oper. Syst. Rev.*, 36(SI):271–284, 2002.