# Understanding the design tradeoffs for cooperative streaming multicast

Animesh Nandi[‡◇], Bobby Bhattacharjee[†], Peter Druschel[◇]

[◇]MPI-SWS [‡]Rice University [†]University of Maryland

Technical Report MPI-SWS-2009-002
April 2009

## ABSTRACT

Video streaming over the Internet is rapidly increasing in popularity, but the availability and quality of the content is limited by the high bandwidth cost for server-based solutions. Cooperative end-system multicast (CEM) has emerged as a promising paradigm for content distribution in the Internet, because the bandwidth overhead of disseminating content is shared among the participants of the CEM overlay network. Several CEM systems have been proposed and deployed, but the tradeoffs inherent in the different designs are not well understood.

In this work, we provide a common framework in which different CEM design choices can be empirically and systematically evaluated. Our results show that all CEM protocols are inherently limited in certain aspects of their performance. We distill our observations into a novel model that explains the inherent tradeoffs of CEM design choices and provides bounds on the practical performance limits of any future CEM protocol. In particular, the model conjectures that no CEM design can simultaneously achieve all three of low overhead, low lag, and high streaming quality.

## 1. INTRODUCTION

Video delivery over the Internet using cooperative end-system multicast (CEM) is increasingly popular, with a number of deployed services (e.g. LiveStation [23], SopCast [33], BBC iPlayer [4]). CEM systems are attractive primarily due to their low infrastructure cost, and provide the least expensive entry into the Internet video market.

The data plane for these systems can broadly be partitioned into single-tree [11, 17, 35, 2], multi-tree [9, 29], mesh-based [43, 30, 25, 31] and hybrid [3, 39, 24, 18, 37, 41, 19] approaches. Research in CEM has focused either on the design of new protocols or on comparisons of complete systems. Prior research has led to a number of partially verified "communal hypotheses", e.g. that mesh-based systems must incur high latencies and that tree-based systems are not resilient to churn. Yet, we still lacks a fundamental understanding of the CEM design space.

Gaining such an understanding is critical: the bandwidth required for streaming high quality video will remain near the limits of broadband network capabilities for the foreseeable future. From the system- and network-designer's perspective, the CEM protocol should efficiently utilize all available bandwidth. From the end-user's perspective, the protocol should have perfect continuity (i.e. streaming quality), low startup delay, and preferably low lag. Unfortunately, no single protocol meets all of these goals.

We conduct an in-depth and systematic empirical comparison of different CEM data delivery techniques, with the goal of understanding the inherent tradeoffs in CEM designs. Our approach differs from previous works that have compared CEM design choices qualitatively [1, 21] or analytically [42, 44, 7, 22], and with those that have compared specific CEM protocols empirically [26, 45].

It is not our intent to recommend any single approach or protocol. Instead, we explore the CEM design space, cleanly identify the tradeoffs that apply to these systems, tease out different components that are responsible for different aspects of observed behavior, and partition deployment scenarios into regions where different systems excel.

A systematic comparison of CEM systems is non-trivial. These systems deliver data over a diversity of data topologies (tree, multi-tree, mesh, and hybrids) which are constructed and maintained using different control and transport protocols. The overall system performance depends both on the properties of the data topology (how well it is able to use existing resources, how well it can withstand failures), and on the control protocol (how quickly the data topology is built/healed). By necessity, existing system implementations couple the data- and control-planes and often use different transport protocols.

A full exploration of the of CEM design space would involve analyzing the cartesian product of all feasible data and control planes. Such a comparison is impractical. Instead, we present our best effort at an unbiased comparison between different data topologies, by "factoring out" the effects of transport protocols and the control plane. To neutralize the effects of the control plane, we present results using two different control protocols: first, we re-implement, from scratch, representative single-tree, multi-tree, mesh-based, and hybrid protocols in their entirety using the SAAR anycast primitive [28]. The performance of our implementation is comparable to "native" implementations of each of these protocols, and provides a lower-bound on the real-world performance.

SAAR was developed exclusively as an efficient control mechanism (not in conjunction with any data plane). However, it may still introduce an unintended bias in favor of a particular data plane structure. Hence, we also experiment with an idealized control plane with perfect knowledge and a configurable response time, which allows us to control for any biases introduced by SAAR.

For each data plane we consider, we present results under diverse operating conditions, including different levels of node churn, packet loss, and stream rates. Moreover, to model the resource availability in real deployments, we rely on an empirical distribution of node forwarding bandwidths, which were obtained by measuring broadband hosts in Europe and the US [12].

A clean-room re-implementation of many different protocols is certainly labor intensive; however, we believe our approach has fundamental benefits over black-box comparisons (in which completely third-party systems are run and the results compared) previously reported in the literature [45].

Our approach enables us to compare different approaches without bias, since the control protocol used by each scheme is *exactly* the same, and all other parameters (control latency, network topology, queuing behavior, host behavior) are identical. Our experiments indicate that *all* CEM protocols are inherently limited in certain aspects of their performance, i.e., these basic limitations transcend parameter settings or control plane efficiency. We distill these observations into a novel model that, we believe, (1) explains the root causes that limit the performance of CEM protocols, and (2) provides bounds on the practical performance limits of any future CEM protocol.

The rest of the paper is organized as follows: In Section 2, we describe related work. Section 3 provides background on existing CEM designs. In Section 4, we present our methodology. Section 5 presents empirical results quantifying the tradeoffs in representative CEM design choices. In Section 6, we propose a model for understanding the inherent constraints in CEM design. Finally, we conclude in Section 7.

## 2. RELATED WORK

**Qualitative comparison** Abad et al. [1] classify thirteen different CEM protocols according to their delivery topology (mesh or tree), management protocol (distributed or centralized), scalability (large or small group sizes), etc. They further describe a number of different application scenarios including multi-player games and media distribution. They conclude that no single protocol is best for all applications. However, the paper contains no insight about why a particular protocol might be more suited to a particular application or scenario, or what advantages or disadvantages are inherent in different designs.

Liu et al. [21] review the state-of-the-art of peer-to-peer Internet video broadcast. They describe the basic taxonomy of peer-to-peer broadcast and summarize the design and deployment of these systems. Although the paper examines the tree-based and the swarming overlays and some of their hybrid variants, their goal is not to empirically compare them under different scenarios.

A number of recent mesh-based data plane protocols [43, 25, 30] have argued that tree-based schemes are not robust to scale and churn. At the same time, systems like Chunkyspread [37] and SAAR [28] have demonstrated that multi-tree data planes can both be robust and scalable. Our work explores possible causes for these discrepancies. In particular, we show that the efficiency of the underlying control plane and parameter selection greatly affects the performance observed by tree based data planes.

**Empirical comparison of specific protocols** Magharei et al. [26] compare the multi-tree and mesh-based data planes, and conclude that mesh-based systems perform better. However, their conclusions are based on an artificial scenario where a number of peers depart but the topology is not repaired. This is an extreme point in the control space (no control protocol); in contrast, we experiment with different scenarios in which the *efficiency* of the control protocol is varied — this allows us to understand how efficient the control protocols must be for acceptable performance and how much the control overhead must be at a given performance level.

We believe that the comparison in this paper is "fairer" since it neutralizes the effect of the control plane (which turns out to be a key factor in the performance of tree-based systems). We identify parameters that allow trees to outperform meshes (and vice-versa). Finally, we identify limitations with each approach that we believe are inherent, and cannot simply be overcome with better control planes or parameter tweaking.

Zhu et al. [45] compare four CEM implementations (Scattercast, Overcast, Narada, and ALMI) with respect to relative delay penalty, normalized resource usage and stress. These protocols are simulated over GT-ITM [14] topologies, after all nodes have joined and without any churn (which ends up providing an unfair advantage to tree-based protocols). In these experiments, the control protocol is different in each system, making it difficult to understand the inherent properties of the data planes.

**Analytical comparison**

A number of papers [42, 44, 7, 22] analyze the inherent properties of different overlay delivery mechanisms. For instance, Tewari et al. [34] use an analytical model to study BitTorrent-based live video streaming, and demonstrate that the swarming protocol needs a minimum number of blocks for effective utilization of peer upstream bandwidth. Bonald et al. [7] focus on the properties of different push-based data diffusion schemes. Liu et al. [22] focus on the theoretical analysis and fundamental limitations of peer-assisted live streaming using tree-based approaches.

Analytical approaches typically have to make simplifying assumptions about the protocols and workloads in order to make the analysis tractable. Our empirical study captures the complexity of real implementations and complements the insights offered by analysis.

A short, two-page extended abstract of this work has previously appeared in [27].

## 3. BACKGROUND

We present an overview of different overlay multicast data forwarding approaches. We have implemented each of these data planes using our common control plane.

**Single-tree delivery** In single-tree systems, the participating nodes form a tree, such that there is a loop-free path from the multicast source to each group member. The capacity of each tree link must be at least the streaming rate. Content is forwarded (i.e., pushed) along the established tree paths. The source periodically issues a content packet to its children in the tree. Upon receiving a new content packet, each node immediately forwards a copy to its children.

Tree data planes provide low latency, but are unable to utilize the forwarding bandwidth of all participating nodes because only interior nodes contribute. When an interior node fails, the disconnected subtree does not receive any data until the tree is repaired. As a result, trees are highly sensitive to the efficiency of the control mechanism that is used to repair the tree when a node fails. In particular, node failures or departures high up in the tree affect data delivery adversely. As a result, tree protocols are often augmented with additional, sophisticated recovery techniques. We describe these recovery techniques in Section 4.2.4.

Examples of single-tree systems include ESM [11], Overcast [17], ZIGZAG [35], and NICE [2].

**Multi-tree delivery** In multi-tree systems, each participating node joins $k$ different trees. The trees are constructed such that each member has $k$ loop-free (and optionally interior-node-disjoint) paths to the multicast source. The multicast source splits the content into $k$ "stripes". Each stripe is then disseminated in one of the trees, just as in a single-tree system.

Each member node is an interior node in some tree(s), and a leaf node in the remaining trees. Hence, as compared to a single tree, the forwarding bandwidth of each member can be utilized, and the forwarding load can be distributed more fairly among all members.

Since each stripe is on the order of $1/k$th the bandwidth of the original stream, multi-trees are able to utilize forwarding bandwidths that are a fraction of the stream rate.

The forest construction ensures that a single failure affects forwarding on only a small number (possibly one) of the $k$ stripe trees. Moreover, if the source uses redundant coding like erasure coding [6, 8] or multiple description coding (MDC) [15], then the effect of a stripe loss can be masked or limited to a reduction in streaming content quality.

Since each stripe is on the order of $1/k$th the bandwidth of the original stream, individual nodes can support more children, and the average tree depth is lower than in a single-tree system with an identical distribution of member forwarding bandwidths. Lower tree depth in turn reduces delivery delays and further increase resilience to faults.

SplitStream [9], CoopNet [29] and Chunkyspread [37] are examples of multi-tree systems.

**Mesh-based delivery** In mesh-based or swarming overlays, the group members construct a random graph. Often, a node's degree in the mesh is proportional to the node's forwarding bandwidth, with a minimum node degree (typically five [43]) sufficient to ensure the mesh remains connected in the presence of churn.

The source periodically makes a new content block available. Each node (including the source) buffers up to $b$ of the most recently published content blocks it has received. Every $r$ seconds, a node advertises to each of its mesh neighbors a bitmap indicating which of the $b$ most recently published blocks it possesses (and is willing to serve).

A missing block can be requested from any neighbor that advertises the block. Amongst the potential suppliers of the block, a node is chosen randomly. As an optimization, the random choice can be biased towards nodes with more available bandwidth. The requests for blocks are piggybacked on the bitmap advertisements to its neighbors.

Unlike in tree protocols, the randomization in the mesh data propagation ensures that blocks are disseminated throughout the mesh following random paths. Hence, mesh neighbors are likely to have received different sets of blocks at any given time, which enables them to exchange blocks. As a result, mesh-based protocols are able to utilize the forwarding bandwidth of all nodes.

The failure of a node or a network link and the efficiency of the control plane have little impact on the swarming protocol. The neighbors of a failed node or link simply fetch blocks from other mesh neighbors while they are choosing a new random overlay member as a new mesh neighbor. The delay in acquiring a new neighbor does not affect the efficiency of content dissemination (as long as it is lower than the mean node lifetime).

Both the delivery delay and join delay in swarming protocols are proportional to the size of the content buffer $b$. The delivery delay in meshes is larger than in tree-based systems, because blocks are not immediately forwarded.

Examples of mesh-based systems are CoolStreaming [43], Chainsaw [30], PRIME [25], and PULSE [31].

**Hybrid tree-mesh delivery** Hybrid data planes attempt to combine the advantages of tree- and mesh-based systems by employing a tree backbone and an auxiliary mesh structure. Typically, blocks are "pushed" along the tree edges (as in a regular tree protocol) and missing blocks are "pulled" from mesh neighbors (as in a regular mesh protocol). The tree overlay could be either a single-tree or a multi-tree, resulting in a single-tree-mesh or a multi-tree-mesh hybrid.

Normally, blocks are delivered along the tree edges, yielding low delay. Blocks that do not arrive via the tree due to failures are recovered via the mesh, thereby increasing robustness. Moreover, the forwarding bandwidth not used for transmitting packets along tree edges can be utilized by the auxiliary mesh structure to provide missing blocks requested by mesh neighbors, thereby increasing the bandwidth utilization.

Examples of single-tree-mesh systems are mTreeBone [39] and Pulsar [24]. Bullet [18] is also a single-tree mesh but instead of relying on the primary tree backbone to deliver the majority of blocks, random subsets of blocks are pushed along a given tree edge and nodes recover the missing blocks via swarming. PRM [3] is a probabilistic single-tree mesh system. Along with tree delivery, each node pushes data blocks to mesh neighbors with a configurable probability.

Chunkyspread [37], GridMedia [41] and Coolstreaming+ [20, 19] are multi-tree-mesh systems. In these systems, the multi-tree structure is embedded in a random mesh; the stripe trees are not interior-node-disjoint.

# 4. METHODOLOGY

We describe our experimental methodology including our implementation of a common control plane and the various data planes. It was not clear to us, a priori, which specific data planes ought to be implemented to provide a representative sampling of the many overlay protocols that have been proposed. Instead of implementing each different protocol, we have meticulously implemented three basic data planes: single-tree, multi-tree and mesh-based delivery. One (and sometimes a hybrid) of these three paradigms form the basis for every protocol in the literature. The protocols differ in their control (how the delivery structure is formed and maintained) and in their recovery mechanisms (how missing data handled).

Along with the base protocols, we have implemented a range of recovery strategies like ephemeral forwarding [3, 37], randomized forwarding [3], and mesh recovery [39]. We experiment with tree-based systems augmented with these recovery techniques (mesh-based forwarding natively incorporates "recovery"). We study hybrid protocols that augment mesh-based systems with tree backbones to lower latency. And, we investigate protocols that combine multiple recovery strategies, for instance, PRM [3] uses ephemeral forwarding, randomized forwarding, and mesh recovery.

By combining these base protocols and recovery techniques, we cover the major CEM protocols and approaches that have been published. Table 1 shows the range of protocols our implementations cover. We believe our results are representative of the state-of-the-art in CEM protocol design.

By design, our data planes use a common control plane, because the goal is to understand the inherent performance characteristics of the data planes. Published performance results from the prior, native implementations indicate that our common control plane is comparable or better than the native implementations.

Our implementations can be executed on Planetlab [32], Emulab [40], ModelNet [36], or deployed on the general Internet. The implementations can also be run on top of a network emulator, which executes the actual protocol code atop an emulated network with a given distribution of link delays and bandwidths.

We next describe our control plane (Section 4.1) and data plane (section 4.2) implementations, followed by a description of primitives (such as heartbeats) that are common to all protocols.

## 4.1 Control planes

Virtually every CEM protocol deployed or described in the literature has its own control plane, making it difficult to isolate the

| Protocol | Base Data Plane | Recovery Strategy: Ephemeral Forwarding | Recovery Strategy: Randomized Forwarding | Recovery Strategy: Mesh Recovery |
|---|---|---|---|---|
| ESM [11], Overcast [17], ZIGZAG [35], NICE [2], FatNemo [5] | Single-tree | | | |
| Bullet [18], mTreeBone [39] | Single-tree | | | ✓ |
| Pulsar [24] | Single-tree | | ✓ | ✓ |
| PRM [3] | Single-tree | ✓ | ✓ | ✓ |
| SplitStream [9], CoopNet [29] | Multi-tree | | | |
| Chunkyspread [37] | Multi-tree | ✓ | | |
| GridMedia [41], Coolstreaming+ [20, 19] | Multi-tree | ✓ | | ✓ |
| Coolstreaming [43], Chainsaw [30], PRIME [25], PULSE [31] | Mesh | | | |

**Table 1: CEM Protocols and Recovery Mechanisms**

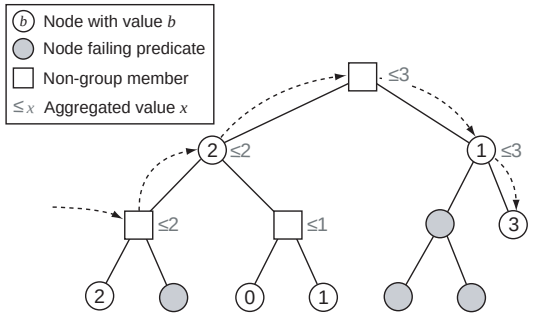



**Figure 1: Anycast traversal example: Given an anycast request issued at the leftmost interior node in the group spanning tree, the anycast traverses the tree in a depth-first search. The search only visits subtrees with members that satisfy the predicate and whose value exceeds that of the current best known member.**

performance of its data plane. Our goal is to present a realistic evaluation that neutralizes the effect of the control plane without affecting data plane performance. Towards this end, we use SAAR [28] as a common control plane that allows us to isolate the data plane performance. SAAR implements a decentralized anycast primitive for overlay neighbor acquisition, and can efficiently support multicast overlays with diverse structures [28].

Our implementations are comparable (and often better) than the native implementations of each data plane. Conservatively, SAAR provides an upper bound on the achievable control overhead. (A control plane implementation may exist that is more efficient or produces better overlay neighbors). The resulting data plane performance achieved by our implementations represent lower bounds.

We have also implemented a centralized control plane (for network emulations only), which can respond to anycast requests with a configurable anycast response time. Varying the anycast response time enables us to infer the extent to which CEMs depend upon the efficiency of the underlying control mechanisms.

### *4.1.1 SAAR*

SAAR is a decentralized control plane based on a structured overlay network. All nodes participate in the SAAR control overlay, regardless of which content they are currently receiving. This overlay implements an anycast primitive, which in turn supports efficient and flexible selection of data dissemination peers. The SAAR overlay also performs efficient, proactive state dissemination and aggregation. This aggregate state is used to increase the efficiency of the anycast primitive.

**Group abstraction:** The key abstraction provided by SAAR is a *group*. A group represents a set of nodes that are members of one data dissemination overlay, i.e., that subscribe to a particular data channel. The group's control state is managed via a spanning tree that is embedded in the control overlay and rooted at a random member of the control overlay.

A set of *state variables* is associated with a group. Each group member holds an instance of each state variable. Typical examples of state variables are a node's forwarding capacity, current load, streaming loss rate, tree-depth in a single-tree data plane, etc.

SAAR can aggregate state variables in the spanning tree. Each state variable $g$ is associated with an *update propagation frequency* $f_{up}$, a *downward propagation frequency* $f_{down}$ and an *aggregation operator* $A$. The values of a state variable are periodically propagated upwards towards the root of the group spanning tree, with frequency at most $f_{up}$. (The propagation is suppressed if the value of a variable has not changed). At each interior node, the values received from each child are aggregated using the operator $A$. The aggregated value at the root of the spanning tree is propagated down the tree with frequency at most $f_{down}$. State variables for which no aggregation operator is defined are propagated only one level up from the leaf nodes.

**Anycast:** SAAR's anycast operation takes as arguments a *group identifier* $G$, a *constraint* $p$, an *objective function* $m$ and a *traversal threshold* $t$. The primitive "inspects" group members whose state variables satisfy $p$ and returns the member whose state maximizes the objective function $m$ among the considered members. To bound the anycast overhead, at most $t$ nodes are visited during a traversal of a group control tree that SAAR maintains for each group. If $t = \perp$, the first considered node that satisfies the predicate is selected. By default, the anycast primitive inspects candidate nodes in order of increasing delay from the invoking node. Figure 1 illustrates an example anycast traversal.

State aggregation allows SAAR to optimize its anycast. For instance, when the aggregated state indicates that no members within a certain subtree satisfy the constraint, then the entire subtree can be pruned from an anycast search. As a result, a search typically considers many more nodes than it visits.

**Example anycast traversal** Figure 1 shows an example group spanning tree. A new node wants to join the data overlay and seeks a parent that maximizes the value of an integer state variable among the nodes that satisfy a given constraint. There are six members that satisfy the constraint. Given an anycast request issued at the leftmost interior node in the spanning tree, the anycast traverses the tree in a DFS, pruning subtrees that contain no eligible members with a value of the variable that exceeds that of the current best known member. In the example shown, the anycast stops after visiting five nodes, and yields the rightmost leaf node with the value

3. Had the anycast been invoked with a value of $t < 5$, then the anycast would have stopped after visiting $t$ nodes, and yielded the leftmost leaf node with value 2.

## 4.2 Data plane implementations

We next describe our data plane implementations.

### 4.2.1 Single-tree data plane (sT)

Nodes use a SAAR anycast to select a parent when initially joining the overlay, when replacing a failed or departed parent, and when choosing a new parent to improve performance. The anycast constraint ensures that the parent has spare forwarding capacity and does not result in a loop. The objective function is chosen to preferentially select parents that have low depth in the tree. When a node joins or recovers from a disconnect, it uses a traversal threshold $t = \perp$ to find an eligible parent as quickly as possible. A node with forwarding bandwidth $B$ and the streaming rate is $S$ takes on at most $\lfloor B/S \rfloor$ children.

In addition to the anycasts for tree repair, the nodes also issue anycasts for preemption and periodic data plane optimizations to improve tree quality (tree depth, biasing high bandwidth nodes towards the top of the tree).

### 4.2.2 Multi-tree data plane (mT)

The multi-tree data plane maintains $k$ separate trees, each forwarding a different *stripe* of $1/k$th of the stream rate. The constraint and objective function is the same as in the single-tree data plane. A node joins all $k$ trees but forwards data (i.e., accepts children) only in its *primary* stripe. This construction ensures interior-node-disjoint stripe trees: a node is an interior node in at most one stripe tree and a leaf in all other stripe trees. If a node has forwarding bandwidth $B$ and the streaming rate is $S$, then the node takes on at most $\lfloor B/(S/k) \rfloor$ children.

When a node joins, it biases its choice of a primary stripe towards stripes with relatively low total forwarding capacity, in order to balance the available forwarding capacity amongst the stripes. In particular, amongst the stripes whose resources $R_i$ (i.e. total forwarding capacity) are lower than the average stripe resources $R_{avg}$, a stripe $i$ is chosen as primary with a probability proportional to $(R_{avg} - R_i)$.

Even with this flexible choice of a primary stripe, it is still possible that the departure of a node causes a stripe to be momentarily left with no forwarding capacity until another node joins. We implement the tree transformations described in SplitStream [9] to address these cases. As a last resort, a child relaxes the predicate to select a parent with forwarding capacity in a different stripe, at the expense of interior-node-disjointedness. Our data plane behaves like SplitStream in this respect, except that the adaptive primary stripe selection significantly reduces the likelihood of stripe resource exhaustion.

### 4.2.3 Mesh-based data plane (pM)

In our mesh implementation, a node maintains neighbors proportional to its forwarding bandwidth, or a minimum $k$ (unless otherwise stated, $k = 5$). Nodes use a SAAR anycast to maintain at least $l$ neighbors of good quality and accept up to $u$ neighbors. A node with forwarding bandwidth $B$ accepts at most $u = \lfloor k * B/S \rfloor$ neighbors. However, in order to ensure that requests for neighbors can be satisfied, the number of neighbors proactively established is slightly lower than the maximum number of neighbors that can be supported. Therefore, nodes proactively establish $l = Max(k, u-2)$ neighbors.

We use a SAAR anycast to choose random mesh neighbors rather than the commonly used random walk [38] or gossiping [13] techniques, in order to ensure identical conditions for all data planes. The anycast is deliberately not biased towards nearby nodes, to provide high path diversity and to form a more robust mesh. In addition, each node periodically refreshes its neighbor list, even if it has $l$ or more neighbors of good quality. Without this periodic update (and especially with low churn), nodes that joined early lack links to nodes that joined late, resulting in low path diversity and high mesh diameter.

The swarming algorithm operates as follows. The source publishes a new content block every $p$ seconds (typically, $p = 1$). Every swarming interval $r$, mesh neighbors exchange their list of available blocks (using a bitmap) within a sliding window of blocks covering $b$ seconds (typically, $b = 60$). The leading edge of the window at a given node is defined by the most recent block available at the node's mesh neighbors.

After exchanging the lists, each node chooses one random block from the intersection of the set of blocks it is missing and the set of nodes that are advertised by some neighbor. The choice of a specific neighbor from which to request the block is also random, but biased towards the neighbors with the lowest bandwidth utilization. The block requests to a neighbor are piggybacked on the periodic bitmap advertisements.

### 4.2.4 Recovery Mechanisms

Tree-based protocols often use sophisticated recovery mechanisms (Table 1) to mask delivery problems on the data path. We next describe our implementation of these mechanisms. In each case, we classify the recovery strategy as "reactive" (recovery starts after a failure is detected) or "proactive" (recovery is part of base forwarding).

**Ephemeral forwarding (EF) [Reactive]** EF was introduced in [3], and attempts to provide an uninterrupted data stream while the data plane is being repaired (after a node departs). In EF, when a node $n$ does not receive data from its parent, it tries to locate an *ephemeral parent* that can provide the data while the overlay reconstruction protocol "fixes" the data plane. Obviously, EF is effective only if it allows $n$ to find a suitable ephemeral parent (one with sufficient forwarding bandwidth) quicker than the standard overlay recovery protocol can find a new parent.

In order to quickly locate an ephemeral parent, nodes maintain a set of mesh neighbors. Upon detecting a disconnection (lack of a data packet from the tree parent), the root of the disconnected subtree ($n$) immediately tries to obtain the stream from its mesh neighbors. EF (ephemeral forwarding) is successful as long as any one of the mesh neighbors can temporarily supply the stream. Node $n$ chooses only one ephemeral parent if more than one mesh neighbor is capable. (Node $n$ continues to send heartbeats down its subtree while locating an ephemeral parent to preclude nodes in the affected subtree from trying to institute their own recovery).

If none of node $n$'s mesh neighbors are able to provide a stream, then node $n$ sends a "delegate" message to its tree children, who then try to find an ephemeral parent using their mesh neighbors. Upon success, the child sends the ephemeral stream up to $n$ (and down into its own subtree). In this manner, one successful recovery is sufficient for "patching" the entire subtree.

EF provides a quick fix for maintaining the tree while the data plane recovers and finds the most suitable parent (one that might optimize criteria such as latency and underlying bandwidth usage). EF also enables the control protocol to use larger timeouts and reduces the latency demands on tree recovery.

**Randomized forwarding (RF) [Proactive]** Like EF, Randomized forwarding (RF) was introduced in PRM [3], and uses the auxiliary mesh structure. In RF, each overlay node, with a small probability (usually 1-3%), proactively forwards the data packets received on the tree to mesh neighbors. The intuition (proven in PRM) is that if a large subtree is affected due to a node failure, then, with high probability, a proactive recovery packet will be incident upon at least one node in the subtree. When a node receives a recovery packet (i.e. a packet not from its tree parent), it forwards it down its subtree, and also up to its tree parent. This process recurses and a single RF recovery packet is sufficient for recovering the entire affected subtree. RF recovery packets also serve as a trigger for starting EF recovery. In our implementation, RF packets are sent only if a node has sufficient spare bandwidth left after it forwards packets on the primary data path.

**Mesh recovery (MR) [Reactive]** Mesh recovery systems augment the primary tree backbone with a mesh. Blocks are "pushed" down the tree links (as in a regular tree protocol), and missing blocks are "pulled" from mesh neighbors (as in a regular mesh protocol). In each case, the tree and mesh components are maintained and used as described in the base mechanism descriptions, except for the following differences.

In normal operation, no blocks are advertised and no blocks are exchanged among mesh neighbors. MR piggybacks the buffer advertisement message on the heartbeat sent every $h$ (typically $h = 1$) secs to keep the mesh neighbor connections alive. When a node realizes that a block from (one of) its parent(s) is overdue, it requests the block from a mesh neighbor that has the block. If it has no recent buffer advertisements from the mesh neighbors, then it sends a block advertisement to all of its mesh neighbors, who respond with their own block advertisements.

As long as all blocks are delivered in a timely fashion within the tree, MR systems behave like a pure tree-based plane, except that a small overhead is being incurred for maintaining mesh neighbors. When a node misses a block in the tree, then after at most one round-trip time, the node and its neighbors have the same information as they would in a pure mesh-based data plane. As a result, the behavior of MR approaches that of a pure tree-based data plane under low loss or churn, and approaches that of a pure mesh-based data plane under high loss or churn.

**Source Coding** A range of different coding schemes such as Reed Solomon Codes [6], Digital Fountain codes [8], multi-descriptive codes [15] have been used by different systems [29, 9]. Each coding scheme has a pre-defined overhead that inflates the stream bandwidth. This overhead is proportional to the erasure tolerance of the scheme (how many packets can be lost before there is loss in signal), processing overhead (order of the computation at the source and especially receivers) and decoding latency (how many packets must be received before decoding can commence). Unfortunately, the wide variety of codes in terms of overhead, decoding latency, and processing requirements (and for some codes, their availability) rendered it infeasible for us to experiment with specific implementations. Instead, in our experiments, we simply report the number/fraction of bits received at each node without considering the ultimate decodability of the video stream. This means that systems that rely on source coding are not penalized (in terms of coding overhead) in our evaluation; we assume that the decoding algorithm is able to extract useful information from every received bit.

#### *4.2.5 Combining recovery strategies*

Many systems, for instance PRM [3] combine multiple of these recovery strategies, and must choose how spare bandwidth is allocated to these different recovery schemes.

When multiple recovery schemes are used, we prioritize them as follows: ephemeral forwarding (EF) has the highest priority followed by mesh recovery (MR) and finally randomized forwarding (RF). EF has the highest priority because the data items recovered via EF will be pushed down the primary data delivery path and can potentially assist multiple nodes (the entire disconnected subtree). MR is guaranteed to assist at least one node (whereas RF is not), and hence MR has higher priority than RF.

### 4.3 Common Primitives

We conclude with a description of functions that are shared among all data plane implementations.

**Heartbeats** In each data plane, overlay neighbors exchange heartbeat messages every $h$ seconds (typically $h = 1$). However, if another control message or data message has been sent to a neighbor during the last $h$ seconds, then the heartbeat message is suppressed, since the message counts as an implicit heartbeat. If a node has not heard from its neighbor for $t * h$ seconds (typically, $t = 4$), it presumes the neighbor has failed and it initiates an anycast operation to locate a new neighbor.

**Dynamic estimation of available forwarding bandwidth** As described in the previous sections, the number of tree- or mesh neighbors a node accepts is based on the forwarding bandwidth available for data traffic (i.e. $B$). Therefore, nodes have to account for their control traffic in order to compute the available bandwidth for data.

Each node's control bandwidth usage is measured periodically (typically every 5 sec) and rounded to an integral multiple of 4 kbps (i.e. 1% of a 400 kbps stream data rate). A node periodically adjusts its estimate of $B$ based on the measured control traffic.

**Uplink bandwidth sharing** Our network emulator multiplexes transmissions from multiple flows over the same network link in a round-robin fashion at a granularity of 1500 bytes (i.e typical IP MTU). Since the emulator does not model TCP/UDP level packet dynamics, the round robin scheduler is intended to approximate the behavior of multiple TCP connections sharing the node's uplink.

## 5. EXPERIMENTAL EVALUATION

In this section, we present the results of a systematic empirical evaluation of different data plane designs for cooperative streaming multicast.

### 5.1 Experimental setup

We performed experiments using the FreePastry network emulator and on PlanetLab. Unless otherwise noted, the results reported here are from the emulator since it allowed us to explore many configurations and parameters, including system sizes up to 10,000 nodes.

**Emulated network** In our emulations, we assume that the network core is well provisioned, that bottlenecks occur only on the access links at the edge of the network, and that the upstream link (rather than the downstream link) at the edge of the network limits the available bandwidth. A backbone network connects 500 stub nodes with unlimited bandwidth and with pair-wise delays drawn from the King [16] data set of measured Internet delay data. That delay set has a mean one-way delay of 79 ms. Each end node is

| Stream rate (kbps) | 437 | 400 | 350 | 300 | 262 |
|---|---|---|---|---|---|
| Resulting RI | 1.2 | 1.31 | 1.5 | 1.75 | 2.0 |

**Table 2: Stream rates and resulting RIs (Monarch distribution)**

connected to a randomly selected stub using a dedicated access link with 1 ms delay, infinite downstream bandwidth and an upstream bandwidth assigned according to an empirical distribution described below. The network emulator models unreliable packet delivery via IP, but does not model TCP/UDP transport level effects.

**Upstream bandwidth distributions** We assigned upstream bandwidths to the access links using an empirical distribution, measured by the Monarch [12] project. The distribution is based on measurements of 1894 residential broadband hosts in Europe and North America; the average (median) upstream bandwidth is 525 (381) kbps, respectively. We use different stream rates (see Table 2) to achieve a given resource index RI (i.e., the ratio of total supply of upstream bandwidth to the total demand for bandwidth). In all experiments, the multicast source has an upstream bandwidth that is five times the stream rate.

Figure 2 shows the Monarch upstream bandwidth distribution and the resulting node RI distribution when using a stream rate of 400 kbps (node RI is a node's upstream bandwidth normalized to the stream rate).

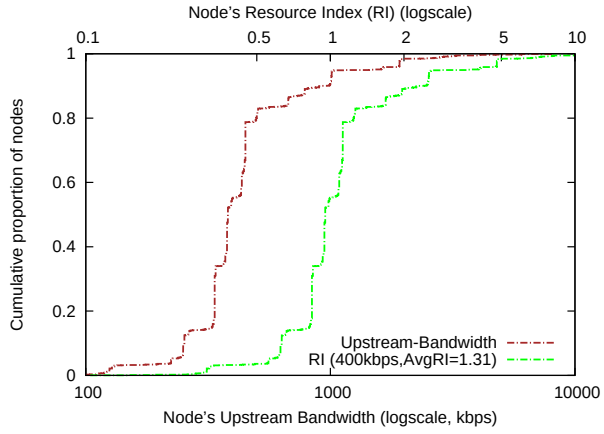



**Figure 2: Upstream Bandwidths and RI assignment**

**Session time** We model different rates of churn in the group membership by varying the session time, i.e., the average time for which a node remains subscribed to a group. Session times are chosen from an exponential distribution with a mean of $S$ seconds and a minimum of 1 second. We present results with $S = 120$ seconds and $S = 300$ seconds. To maintain a large instantaneous group size, nodes re-join the same channel 15 sec after leaving. The chosen session time distribution is consistent with findings from a recent analysis of an IPTV system [10], which shows that most sessions are short due to channel surfing, and a small proportion of sessions last tens of minutes or more.

**Packet loss rate** We model packet loss using an exponential distribution with a mean loss rate of $l$. We experiment with $l = 0$, 1, 3, and 5%.

**Control plane** All data planes use the SAAR control plane to acquire overlay neighbors. At the start of the experiment, a node joins the shared control plane, and then joins data channels for a time determined by the session time distribution. We pessimistically assume that nodes leave a data channel abruptly (i.e., without notifying their overlay neighbors). Nodes use a timeout of four seconds to declare an unresponsive data overlay neighbor as dead and then use a SAAR anycast to replace the dead neighbor.

Nodes remain in the shared SAAR control plane throughout our experiments. This is the intended usage for SAAR. When users switch channels, there is no need to leave the control plane. Even when a user stops watching channels, there is no need to leave the control plane, because membership in the control plane has very little overhead. In a real deployment, nodes sometimes leave the control plane involuntarily due to node or network failure. To account for this and other factors that could impair the performance of the control plane, we perform additional experiments with an artificially inflated anycast response time.

**Other parameters** A single source node publishes the content at intervals of $p$ *sec*. Unless otherwise stated, the number of multi-tree stripes (5), minimal number of mesh neighbors $k$ (5), swarming interval $r$ (1 sec), swarming buffer size $b$ (45 blocks) and various timeouts were set to reasonable values under the given conditions. All network emulation experiments were repeated 3 times with different random number generator seeds. Each reported datapoint is the mean of the measured values; we computed the 95% confidence intervals and they were extremely tight (within 1% of the mean in all cases). Therefore, they are not shown in the plots.

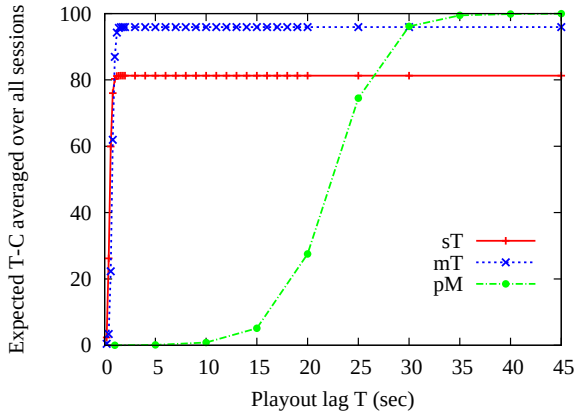



**Figure 3: T-continuity as a function of playout lag [$N = 1000, RI = 1.5, S = 300, l = 0, p = 40$ $msec(sT, mT), p = 1$ $sec(pM), r = 1$ $sec, b = 45$ $blocks$]**

## 5.2 Streaming quality

We first consider what proportion of the streamed content different data planes are able to deliver within a given time period. This proportion has a direct influence on the quality of the displayed video, because it determines how much of the streamed information is available to the player by the playout deadline. We use the following metric:

**T-continuity (T-C)** For each session, T-C is the proportion of the streamed bits that have arrived at the receiving node within $T$ seconds of their first transmission by the source. Assume the source generates independently decodable blocks once every $w$ seconds. The instantaneous T-C measures what fraction of a block is available to a receiver $T$ secs after the source finished transmitting the

| | Graceful departures | | | | | Abrupt departures | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | SAAR | C: 0.25 s | C: 1 s | C: 2 s | C: 4 s | SAAR | C: 0.25 s | C: 1 s | C:2 s | C:4 s |
| sT | 81.8 | 81.7 | 79.8 | 77.9 | 75.6 | 79.8 | 79.2 | 77.1 | 75.2 | 71.0 |
| mT | 99.1 | 98.9 | 98.1 | 97.0 | 94.6 | 95.4 | 95.6 | 95.3 | 94.1 | 91.9 |
| pM | 99.9 | 99.9 | 99.9 | 99.9 | 99.9 | 99.9 | 99.9 | 99.9 | 99.9 | 99.9 |

**Table 3: Average T-continuity ($T = 45sec$) with different control planes (SAAR or centralized C: d with 'd' sec of anycast response time) [$N = 1000, S = 300, l = 0, p = 40$ $msec(sT, mT), p = 1$ $sec(pM), r = 1$ $sec, b = 45$ $blocks$]**

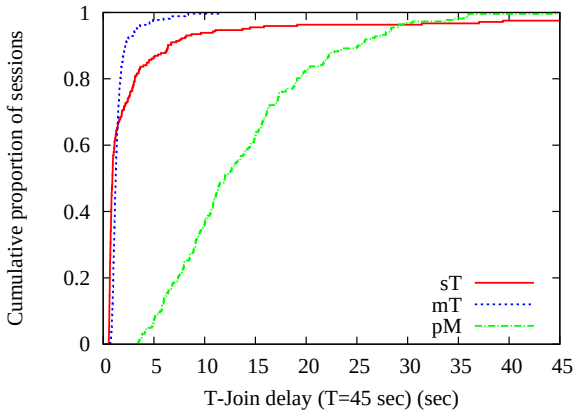



**Figure 4: T-join delay [$N = 1000, RI = 1.5, S = 300, l = 0, p =$ 40 $msec(sT, mT), p = 1$ $sec(pM), r = 1$ $sec, b = 45$ $blocks$]**

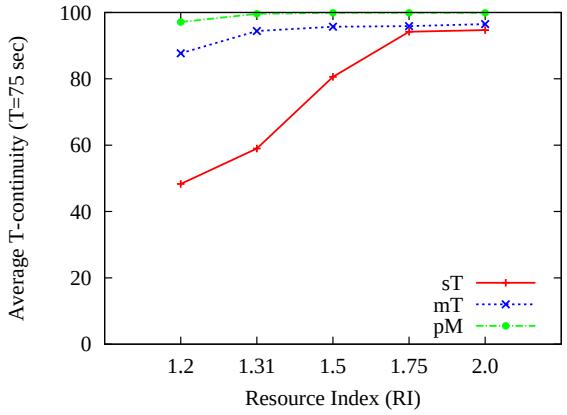



**Figure 5: Average T-continuity for $T = 75$ $secs$ as a function of the RI [$N = 1000, S = 300, l = 0, p = 40$ $msec(sT, mT), p =$ 1 $sec(pM), r = 1$ $sec, b = 45$ $blocks$]**

block. The parameter $T$ can be interpreted as the lag with which a receiver plays out the stream. T-C specifies how much of the stream is available to the player for a given playout lag $T$. The *expected T-continuity* for a session is the mean of the instantaneous T-continuity values over the course of the session.

In our first set of experiments, we use a group size of $N = 1000$, a stream rate of 350 $kbps$ (which yields an $RI = 1.5$), a mean session time $S = 300$ $sec$ and no packet loss. We use small data blocks ($p = 40$ $ms$) in tree-based systems (sT and mT) and large data blocks ($p = 1$ $sec$) in the mesh (i.e. pM). In the mesh, the swarming interval $r = 1$ $sec$ and the swarming buffer size $b = 45$ $blocks$.

Figure 3 shows the T-continuity as a function of the playout lag $T$. The mesh (pM) achieves almost perfect continuity at a lag of 45 seconds. The tree-based data planes (sT, mT), on the other hand, cannot achieve a perfect T-C for any playout lag. sT maxes out at 80% T-continuity, mT at 95%. The primary reason is that churn affects pure tree-based systems. In an otherwise identical experiment without churn, mT achieves 99.9% continuity; sT achieves only 85.6%, but the reason is that it is resource-bound at $RI = 1.5$. At an $RI = 1.75$ and no churn, sT also achieves an almost perfect 99.8%.

The mesh requires a playout lag that is almost an order of magnitude higher than the tree-based data planes. This result shows a fundamental tradeoff between pure tree-based systems and pure swarming meshes: the former achieve low delay by pushing data along optimized distribution paths, but suffer when these paths are disrupted by churn. The latter route packets dynamically and opportunistically, which makes them less vulnerable to churn but incurs higher delays.

Among the tree-based systems, the multi-tree achieves a higher T-continuity than the single-tree at a marginally higher lag. Two factors contribute to this result: (i) most node failures affects only one stripe and (ii) a failure tends to affect fewer nodes. Because most mT nodes contribute their entire forwarding bandwidth to a single stripe, their degree exceeds that of the same node in sT by a factor $s$, the number of stripes. Thus, the stripe trees are shallower and correspondingly more *robust* than the corresponding single tree. This is because the expected number of nodes affected by the failure of a random node decreases as the average interior node degree in a tree increases. (Intuitively, the higher the average interior node degree, the shallower a tree which implies that a larger proportion of nodes are leaves or have few children.)

The mT's greater robustness to churn follows from the (mostly) interior-node-disjointedness of our multi-tree data plane. A multi-tree data plane that does not maintain this property (e.g. Chunkyspread [37]) is not necessarily more robust to churn than a single tree (though it still achieves much better resource utilization than a single tree). To confirm this, we performed an experiment with a multi-tree system that does not attempt to build interior-node-disjoint stripe trees (mT-nind). In the same scenario as used in Figure 3, mT-nind maxes out at a T-continuity of only 86.9%, as compared to 95% with the interior-node-disjoint mT. As explained above, the deeper stripe trees in mT-nind (with an average stripe tree depth of 8.5 in mT-nind, as compared to only 3.8 in mT) are more vulnerable to churn.

## 5.3 Join delay

The delay required to join a given content channel is another important aspect of CEM performance. In IPTV, for instance, users expect to be able to switch between content channels rapidly. Thus, a CEM system suitable for IPTV must be able to join and start displaying the content of a channel quickly. We use the following metric to evaluate join delay:

**T-Join delay** Elapsed time between the instant when a node initiates the process of joining a channel to the instant when the node's instantaneous T-continuity first reaches a value within one standard

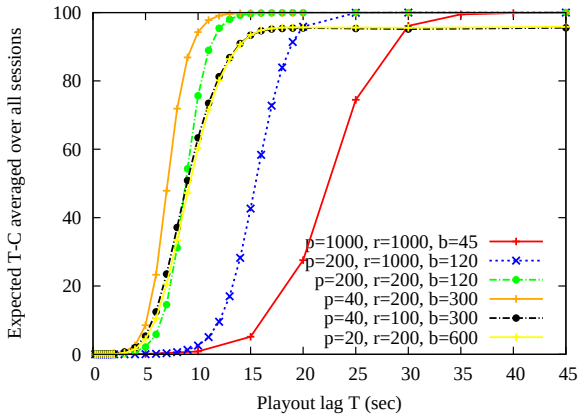



**Figure 8: Mesh T-continuity: some examples of blocksize ($p$ msec), swarming interval ($r$ msec) and swarming buffer size ($b$ blocks) [$N = 1000, RI = 1.5, S = 300, l = 0$]**

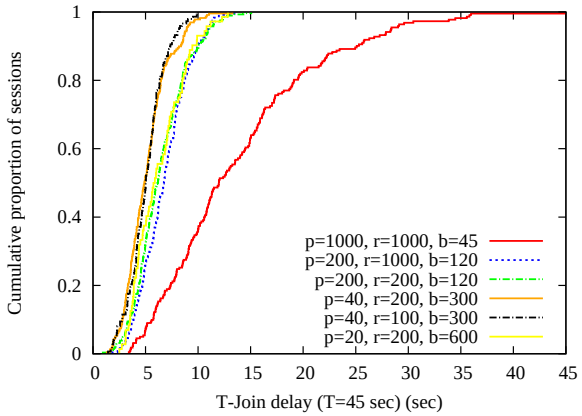



**Figure 9: Mesh T-join delay: some examples of blocksize ($p$ msec), swarming interval ($r$ msec) and swarming buffer size ($b$ blocks) [$N = 1000, RI = 1.5, S = 300, l = 0$]**

deviation of the expected T-continuity over the course of a session. Given a desired playout lag of $T$ seconds and the resulting expected T-continuity in steady state, this metric shows how long it takes a node to approach the steady state when joining a channel.

Figure 4 shows the cumulative distribution of T-Join delays over all sessions. The results show that the join delays for the tree-based systems are an order of magnitude lower than those of the mesh. About 65% of the sT session have a join delay that is slightly less than that of mT. The reason is that unlike an sT node, which needs to find one parent, an mT node needs to join several stripe trees in parallel, and the slowest of the join events contributes to the join delay. However, the remaining 35% of sT sessions had considerably longer join delay than mT. The reason is that sT is resource-bound at the relatively low RI=1.5 in this experiment, and these sT nodes have difficulty finding a parent with sufficient forwarding capacity.

## 5.4 Varying conditions

**Stream rates** We repeated the previous experiments at different stream rates (and consequently different amounts of available resources). Figure 5 shows the average T-Continuity for $T = 75$ *sec* as a function of the stream rate. We use $T = 75$ *sec* in contrast to the $T = 45$*sec* used earlier, because at the low RI=1.2, pM does not reach its maximum continuity at a lag of T=45 sec, but almost (i.e. within 0.5% from the maximum) does so at T=75 sec. The results show that the pure mesh is virtually unaffected by the stream rate down to an RI of 1.2. The mT declines somewhat below an RI of 1.5, while sT deteriorates rapidly below an RI of 1.75. The reason is that the data planes differ in their ability to utilize the available forwarding bandwidth, as explained in Section 3. Note that the trees cannot achieve perfect continuity even at an RI of 2.0—the reason is that pure trees (without any recovery) cannot achieve perfect continuity under churn.

**Interior-node-disjointedness** As in Section 5.2, we performed experiments with mT-nind, the multi-tree system that does not attempt to build interior-node-disjoint stripe trees. In contrast to mT, mT-nind's T-continuity benefits much more from an increased RI, since the average fanouts of the stripe trees improve, resulting in shallower stripe trees, which are more robust to churn. The average T-continuity of mT-nind at $RI = 1.2$, 1.31, 1.5, 1.75 and 2.0 were 60.4, 75.1, 86.9, 89.6 and 90.1, respectively. The corresponding average stripe tree depths were 11.6, 11.1, 8.5. 8.1 and 6.7, respectively. In contrast, mT even at RI=1.31, had an average T-continuity of 94.5 and a corresponding average stripe tree depth of only 4.03. These results demonstrate the benefit of the interior-node-disjoint property in building shallower stripe trees that are more robust to churn.

**Churn and packet loss** We also experimented with higher churn ($S = 120$ *sec*) and packet loss ($l = 3\%$). The results confirm the tradeoffs between continuity, lag and join delay for tree-based versus mesh data planes that we had identified in the previous experiments. Higher churn, packet loss or fewer resources reduce the continuity and slightly increase the lag and join delay of the tree-based systems, while the mesh is not significantly affected. More resources benefit particularly the single-tree, because its ability to exploit available forwarding bandwidth is limited.

**Anycast response time** Finally, we varied the anycast response time of the control plane. We experiment with a response time $d = 0.25, 1, 2, 4$ *secs*. We also experiment with graceful departures where nodes send explicit departure notifications to its overlay neighbors when leaving a group, rather then relying on the neighbor detection timeout of 4 secs. Table 3 shows the results under conditions identical to those used in Section 5.2, except for differences in the control plane. We see that the pM is not affected by deteriorating control efficiency or abrupt departures, whereas the trees are affected by both.

Our results raise the question as to whether the observed performance trends of mesh and tree-based systems are fundamental, or if they can be overcome with appropriate protocol design. More specifically, we ask the following questions. Can the lag and join delay in mesh-based systems be reduced to the level of tree-based systems? Can the tree-based systems match the near-perfect continuity of a mesh by incorporating recovery strategies? We consider these questions next.

## 5.5 Reducing mesh lag

We first consider swarming mesh systems. There are three ways by which one could try to reduce the lag in the swarming mesh: reduce the block size $p$, reduce the swarming interval $r$ or reduce the size of the swarming buffer $b$.

The results in Fig 6 show the impact of the block size ($p$) and swarming buffer size ($b$) on the average T-continuity (for $T = 45$*sec*) and the average delivery delay of blocks, respectively. We see that reducing the size of the swarming buffer reduces the delivery delay

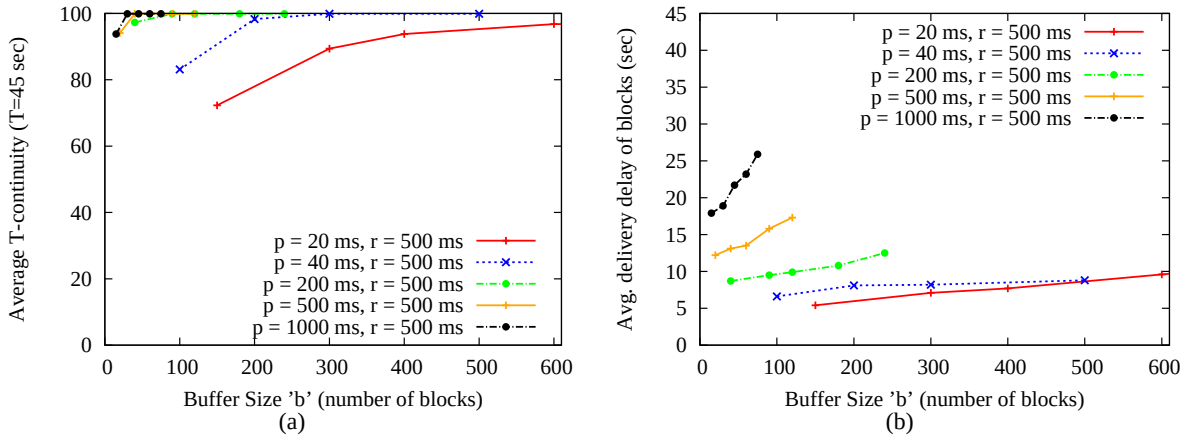



**Figure 6: Effect of varying the swarming buffer size in mesh-based systems [$N = 1000, RI = 1.5, S = 300, l = 0$]**

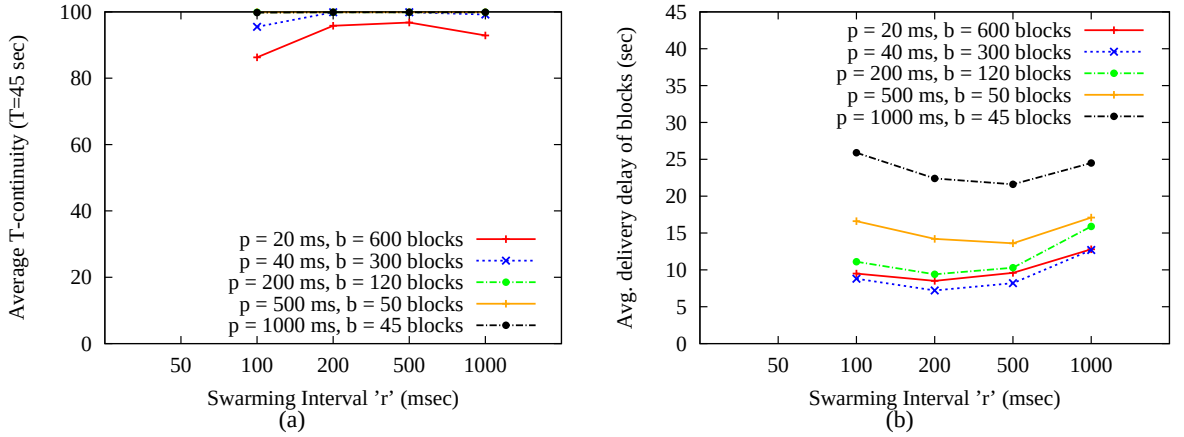



**Figure 7: Effect of varying block size and swarming interval in mesh-based systems [$N = 1000, RI = 1.5, S = 300, l = 0$]**

to a point, beyond which the continuity decreases. The reason is as follows. A smaller buffer reduces the expected time until a block is picked by a neighbor who needs it, thereby reducing per-hop delays. However, the swarming buffer must have a certain minimal size to ensure that the forwarding paths of different data blocks are sufficiently randomized. Random forwarding paths in turn ensure that mesh neighbors tend to have disjoints sets of blocks available, which allows them to utilize their forwarding bandwidth to exchange blocks. Therefore, the continuity diminishes when the buffer becomes too small.

Figures 7 shows the impact of changing the block size ($p$) and the swarming interval ($r$) on the average T-continuity (for $T = 45sec$) and the average delivery delay of blocks, respectively. For each data point, we set the size of the swarming buffer ($b$) so as to achieve the minimal delay while not sacrificing continuity, as per the results in Figure 6.

At a high level, the results show that the average delivery delay (which affects lag) can be reduced by reducing the swarming interval or employing smaller blocks down to a point, beyond which the continuity decreases and delivery delay increases again. The reason in each case is that smaller blocks or more frequent swarming increase overhead due to headers and control messages. When these overheads become too large, they reduce the bandwidth available for data transmissions.

Figure 8 shows the extent to which the performance of mesh-based systems can be improved by an optimal choice of block size, swarming interval and swarming buffer size. The figure shows the T-continuity as a function of the playout lag in pM, for several choices of block size, swarming interval and swarming buffer size. The line 'p=1000, r=1000, b=45' corresponds to a typical configuration used in deployed mesh-based system like CoolStreaming [43]. As the results show, it is possible to reduce the lag by a significant amount with an optimized configuration (see 'p=40, r=200, b=300').

In summary, an optimal configuration of the swarming mesh yields a significant reduction in the delivery delay and lag. However, the delays remain significantly higher than those achieved by tree-based systems.

## 5.6 Reducing join delays

Figure 9 shows that the T-Join delays can also be reduced substantially with the same configuration that minimizes the delivery delay. Again, however, the delays remain significantly higher than those achieved by tree-based systems.

We investigated if the join delays in mesh protocols can be further reduced by incorporating ideas from tree-based protocols targeted at reducing the join delay. Specifically, we implemented an optimization that allows a joining node to use ephemeral tree par-

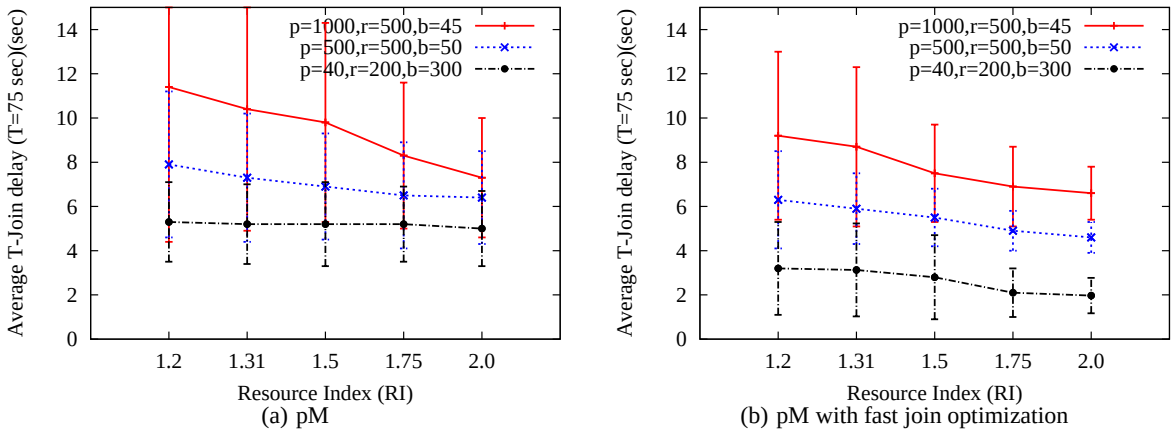



**Figure 10: Effect of fast mesh startup optimization [$N = 1000, S = 300, l = 0$]**

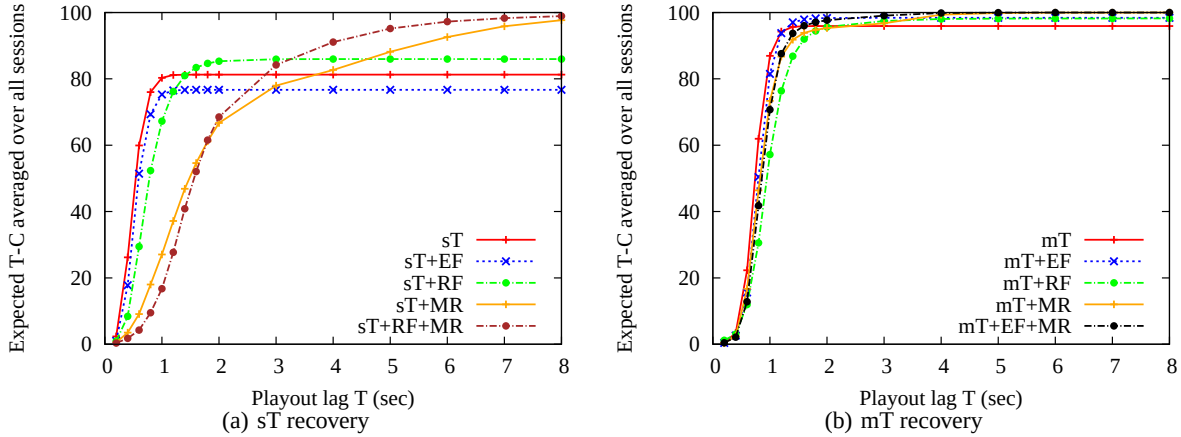



**Figure 11: sT and mT recovery strategies [$N = 1000, RI = 1.5, S = 300, l = 0, p = 40$ *msec*$, r = 500$ *msec*$, b = 300$ *blocks*]**

ents to quickly fill a prefix of its buffer, in order to start replay as quickly as possible.

A joining node seeks to find $k = 5$ ephemeral parents, in addition to its mesh neighbors. As in a multi-tree protocol, the node requests a different subset of the most recent blocks from each of its ephemeral parents, while starting the swarming. To ensure that the optimization does not interfere with the base swarming protocol, ephemeral parents prioritize mesh request over transmissions to ephemeral children. The joining node discontinues the use of ephemeral parents as soon as it reaches steady state or the swarming window has advanced by one full buffer size $b$. Also, a node accepts ephemeral children only once it is in steady state.

Figure 10 shows the average T-Join delay ($T = 75sec$) for the mesh, with and without the optimization, for different configurations and a range of RIs. First, we see that regardless of the optimization, the average join delay can be significantly improved with an appropriate configuration of block size, swarming interval and buffer size. Moreover, the same configuration that yields the best continuity (Figure 8) also yields the best join delays.

Second, the fast join optimization yields a noticeable reduction in average join delay across the board, between 11% and 63%. However, at low RI, the join delays have high variance, because only a fraction of the nodes are able to successfully acquire ephemeral parents.

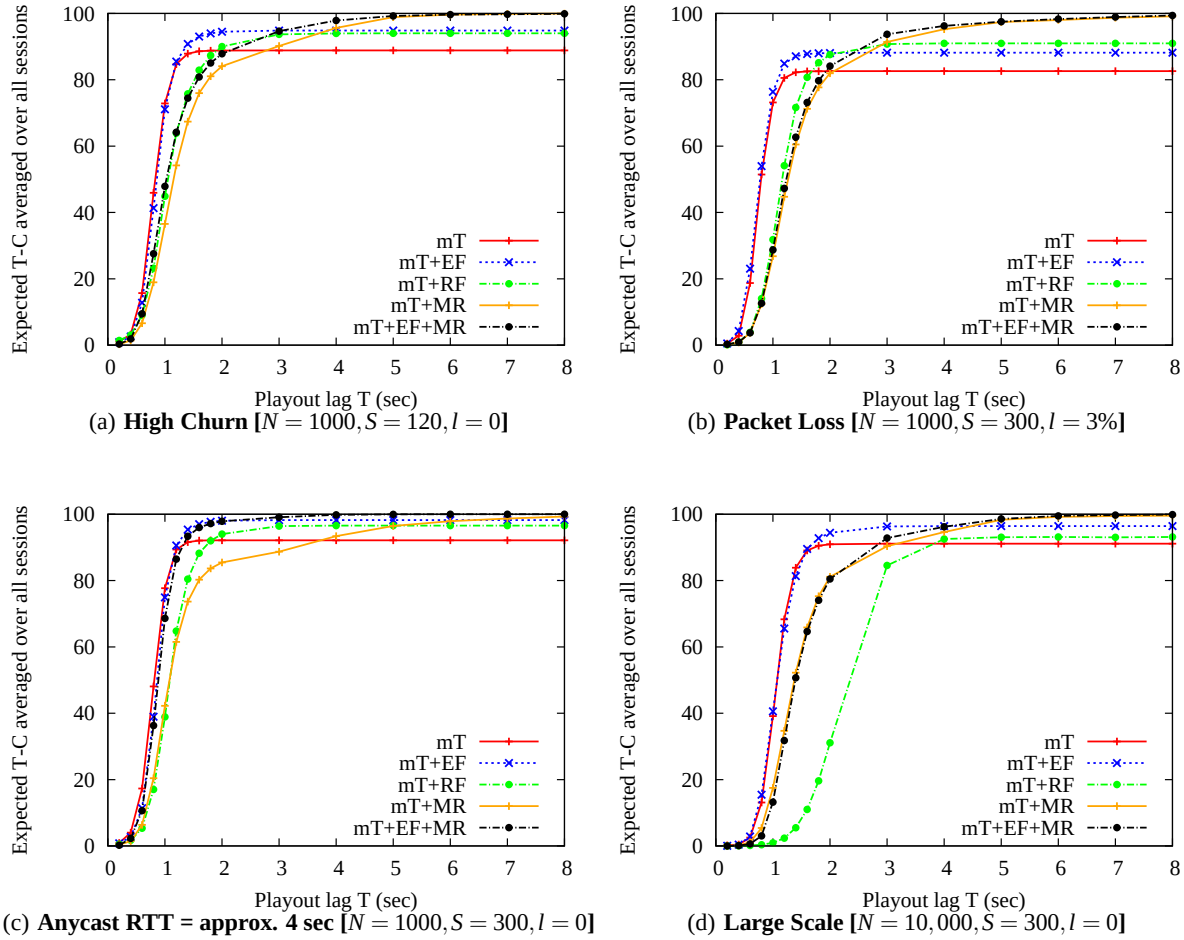
Third, additional resources (higher RI) tend to reduce the average join delays. With the optimization, additional resources reduce the join delay for all configurations, while they only benefit the poor configurations without the optimization. With the best configuration, the fast join optimization and an RI of 1.75 or more, the join delay approaches that of tree-based systems.

We conclude that the fast join optimization allows meshes to achieve low join delays that approach those of tree-based systems, but these results can only be achieved when resources are abundant.

## 5.7 Improving tree continuity

Next, we investigate to what extent the continuity of tree-based systems can be improved using different recovery strategies. Specifically, we use the recovery techniques for tree-based systems described in Section 4. Figure 11 shows the T-Continuity as a function of lag for sT and mT with different recovery techniques at $RI = 1.5$.

All recovery strategies yield some increase in continuity, with the exception of EF applied to the single-tree. The reason is that ephemeral parents in a single tree must have sufficient available bandwidth to support the full stream rate. Due to the large number of nodes with forwarding bandwidth less than the stream rate, there is a shortage of eligible parents. Worse, "ephemeral children" can occupy resources that could be used for permanent parents, which




(a) **High Churn [**$N = 1000, S = 120, l = 0$**]** (b) **Packet Loss [**$N = 1000, S = 300, l = 3\%$**]**

(c) **Anycast RTT = approx. 4 sec [**$N = 1000, S = 300, l = 0$**]** (d) **Large Scale [**$N = 10,000, S = 300, l = 0$**]**

**Figure 12: mT recovery under more severe conditions [**$RI = 1.5, p = 40$ *msec*$, r = 500$ *msec*$, b = 300$ *blocks*$]$

explains the loss of performance with EF. In the multi tree system, however, the bandwidth requirement for an ephemeral parent is only the stream rate divided by the number of stripes, yielding many more eligible parents.

Reactive mesh recovery (MR) achieves perfect continuity, but only at a substantial lag. This is not surprising, because a mesh is used to recover blocks that do not arrive via the tree. Random forwarding (RF) is very effective with sT. The reason is that the many leaf nodes in the single tree with available bandwidth below the stream rate can contribute to the system via random forwarding. However, the additional stream data received via RF comes with additional lag.

Focusing on the upper left part of the plots (which shows what fraction of the stream is delivered with low lag), we see that ephemeral forwarding (EF) is the only recovery technique that increases the proportion of the stream data delivered with low lag, when applied to mT.

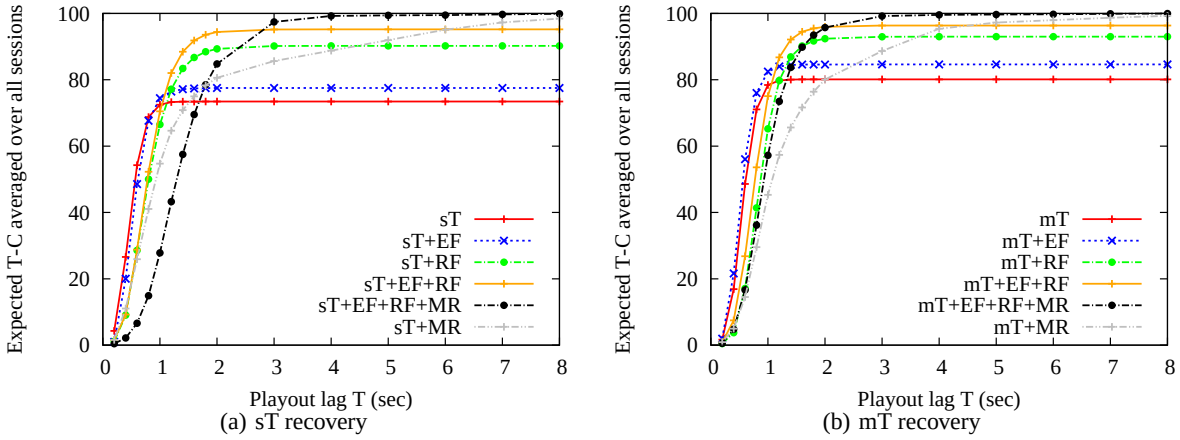
When optimizing for lag, EF is the best technique for mT and RF is best for sT, under the given conditions. When optimizing for continuity, MR is best. The combinations of EF+MR for mT and RF+MR for sT constitute a compromise that achieves perfect continuity, albeit at a larger lag.

## 5.8 Recovery under severe conditions

Figure 12 shows the results for mT recovery techniques under more severe conditions, namely high churn, packet loss, large group size and inflated anycast response times. In general, the same trends hold: EF remains the most effective technique when lag matters, while MR remains most effective for continuity.

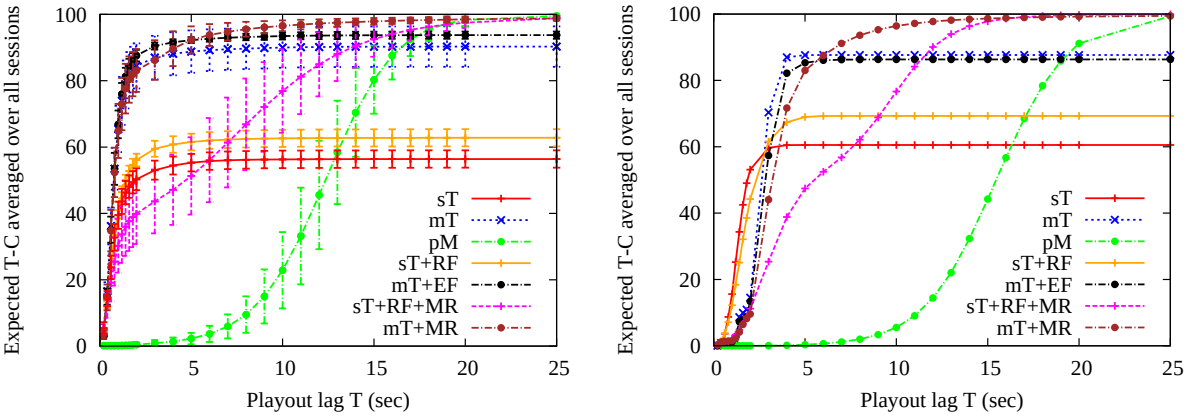
Random forwarding (RF) adds significant lag at large scale, because it increases the lengths of the forwarding paths. A slow control plane (4 seconds anycast response time) affects the pure tree-based data planes, because it increases tree repair time. EF, however, does not depend on the control plane and masks the effects of a slow control plane almost completely. Large scale has the strongest effect on the lag, while packet loss has the strongest effect on continuity. However, recovery remains effective in all cases.

Figure 13 shows how increasing the RI to 2.0 (i.e., lowering the stream rate) affects the performance of the various recovery techniques with the single-tree and multi-tree data plane. We use more severe conditions of both high churn and packet loss in this experiment, to see whether additional resources allow the recovery techniques to mask these. The results show that all recovery techniques are effective, but RF in particular is able to take advantage of the additional resources. Moreover, the combination of the recovery techniques works very well: sT+EF+RF+MR and mT+EF+RF+MR approach almost perfect continuity at a lag of only 4 respectively 3 seconds under these harsh conditions! Also note that under an RI=2.0, the performance of sT is not very different from that of mT, whereas at RI=1.5 the sT is resource bound because of its in-



(a) sT recovery (b) mT recovery

**Figure 13: sT and mT recovery strategies under high RI, high churn and packet loss [$N = 1000, RI = 2.0, S = 120, l = 3\%, p = 40$ *msec*$, r = 500$ *msec*$, b = 300$ *blocks*]



(a) Planetlab results (b) Corresponding emulation results ($l = 2\%$)

**Figure 14: Planetlab vs emulation results: T-Continuity as a function of playout lag T[$N = 350, RI = 1.5, S = 300, Monarch, p = 200$ *msec*$, r = 1$ *sec*$, b = 120$ *blocks*]**

ability to utilize resources effectively.

The overall conclusion we can draw is that although effective recovery techniques exist that can increase the continuity of the tree-based systems, no combination of recovery techniques can simultaneously match, when resources are constrained, the near-perfect continuity of a swarming mesh and the low delivery delay, lag and join delay of a tree-based system. However, when resources are abundant (e.g. $RI = 2.0$), then tree-based systems with recovery can achieve low lag, join delay and high continuity even under adverse conditions. Mesh-bases systems, on the other hand, achieve high continuity under high churn, packet loss and constrained resources, but at the expense of higher lag and join delay.

When lag and join delay are not an issue in a given application, then pure mesh-based systems are superior to trees, because they deliver almost perfect continuity under a wide range of conditions. Tree-based systems are interesting when lag and join delay are important. Moreover, when resources are abundant, the combination of tree-based techniques and recovery techniques can achieve low lag, low join delay and high continuity.

## 5.9 Planetlab experiments

To validate our network emulation results, we also performed experiments with a deployment on 325 nodes in the Planetlab testbed. Planetlab is a live testbed with concurrent experiments that compete for CPU and network bandwidth. Therefore, experiments are subject to some degree of packet loss. As a result, we compared the Planetlab results with results of our network emulation at a packet loss rate of $l = 2\%$. At this loss rate, the results of the two experiments matched very well.

Among the set of Planetlab nodes (across all continents) with reasonable load averages, we randomly chose 325 nodes. We limit, using a token bucket, the upstream bandwidth of each node as per the Monarch distribution. In addition, we had to cap bandwidths in the Monarch distribution greater than 1 Mbps, because Planetlab limits the per-node bandwidth available to an experiment. As a result, we had to use a lower streaming rate of 300 kbps (instead of the default of 350 kbps used in earlier experiments) to achieve an $RI = 1.5$. We used the same bandwidth caps and streaming rates in the corresponding emulation experiments. In the experiments, we use a block size $p = 200$ *msec* and a swarming interval of $r = 1$ *sec*.

Fig 14 shows T-continuity as a function of the playout lag, comparing the result obtained in Planetlab with the network emulation results. In the Planetlab plots, each data point is the mean of five runs, with the 95% confidence intervals shown in the error bars.

At a high level, the Planetlab results show the same trends as the network emulation. However, there is one noteworthy difference across all data planes - the lag in the Planetlab experiments is lower than in our network emulations. This is because, most Planetlab nodes have very high forwarding bandwidth, which results in much lower transmission delays than in the network emulation.

We also observe that the continuity achieved by single-tree based systems (i.e. sT and sT+RF) in Planetlab is lower than that achieved in the network emulations. The reason for this was the higher anycast response time in the Planetlab environment (e.g. for sT, the anycast response time was 2.54 sec in Planetlab versus 757 msec in the emulations) due to overloaded Planetlab nodes. To confirm this hypothesis, we performed an additional emulation experiment in thes same configuration, but using the centralized control plane with configurable response time. The results showed that the continuity of single-tree based systems reduce by approximately 5% when the anycast response time increases from 750 msec to 2.5 sec. A similar trend can be observed in Table 3.

Additionally, we investigated why the continuity of single-tree based systems (both for Planetlab as well as emulations) was significantly inferior than what we had observed in earlier experiments (e.g. Section 5.2) that used the un-capped bandwidth distribution of Figure 2. Our hypothesis was that, as compared to the un-capped bandwidth distribution, the capped-bandwidth distribution used here results in deeper trees, which are more vulnerable to churn and packet loss. To verify this hypothesis, we looked at the distribution of tree depths of nodes in the capped and un-capped bandwidth distribution respectively. We observed that only 38% of nodes were within tree depth of 4 in the capped distribution, as compared to 62% of nodes within a tree depth of 4 in the un-capped distribution.

In summary, accounting for unavoidable differences between the PlanetLab testbed and our emulation environment, the PlanetLab results confirm the trends we had observed in the emulation results.

# 6. DESIGN CONSTRAINTS IN CEMS

In Section 5, we have presented results of our experiments with CEM systems. In this section, we distill our observations and reasoning into a simple model that identifies design constraints and fundamental tradeoffs for CEM systems.

For instance, our results (regardless of parameter or protocol variation) consistently indicate that in a resource constrained system, tree-based data planes are not able to provide high continuity, and mesh-based systems are not able to provide low lag. According to our model, these limitations are inherent and are a by-product of a set of underlying constraints that we describe next.

## 6.1 Model

Our model is based on a set of constraints that we assert no CEM design can violate. The constraints are depicted in Figure 15 as a pair of inter-related triangles. We begin with a description of the vertices:

- **High continuity**: Continuity is a measure of the fraction of playable bits received by a node. Higher continuity is preferable.
- **High resource utilization**: Resource utilization specifies how well *global* resources (forwarding bandwidth of all nodes in the system) are utilized. High resource utilization is preferable.
- **Low lag**: Lag is the delay from the instant when a data item was first transmitted at the source to the instant when it is played out by the media player at a given node. Lower lag is preferable.
- **Low overhead**: Overhead measures the number of extra bits transferred in the system, not counting the original data. Overhead includes control messages, coding for data recovery, and duplicate data packets. Lower overhead enables more of the available bandwidth to be used for media delivery.

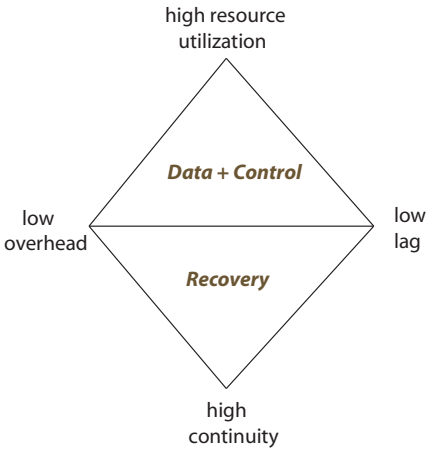



**Figure 15: Constraint Model**

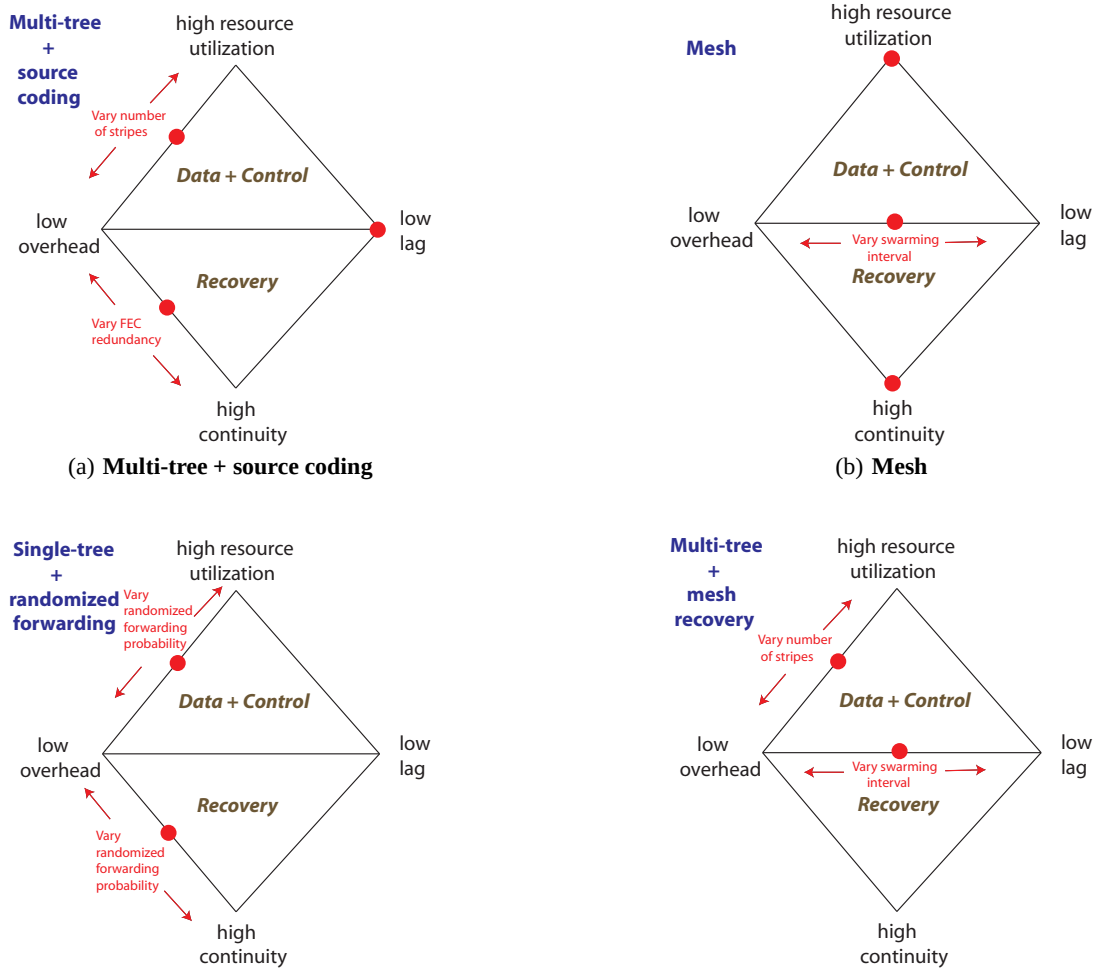
We conjecture that the triangles are, in fact, *impossibility* triangles, in that CEM systems (and indeed any streaming system) can choose to optimize at most two properties from each triangle, but *never* all three. A protocol may, however, trade off two (or more) of the properties in either triangle.

The constraints are perhaps individually obvious; presented together, they provide a clear basis for putting our results in context. Moreover, they assert that no amount of parameter tweaking or protocol engineering will be sufficient to change some of the trends we have observed. In the rest of this section, we discuss the constraints imposed by each triangle, our explanation of why these constraints arise, and how these constraints apply to the systems we have studied.

## 6.2 The constraint triangles

**The *Data + Control* triangle states that no data plane design can simultaneously achieve all three of low lag, high global resource utilization, and low overhead.** For example, a single tree minimizes lag but cannot provide high utilization. As multiple trees are introduced, resource utilization increases but so does overhead. Meshes provide essentially perfect utilization, but must incur either high overhead (due to frequent swarming exchanges) or high lag. The underlying reason for this triangle is as follows: to achieve high resource utilization, a data plane must be *dynamic*, i.e., be able to use upload bandwidth of all nodes even during periods of high churn. Such a data plane cannot maintain statically computed paths; the price for this flexibility must be paid in terms of coordination overhead on the data path. This overhead can be amortized but doing so necessarily increases lag.

**The *Recovery* triangle states that it is impossible to simultaneously achieve low overhead recovery, low lag, and high continuity.** Reactive recovery strategies either incur high lag (since the receiver must detect a missing packet or heartbeat) or high overhead (lag can be reduced by increasing heartbeat frequency). Proactive recovery strategies have relatively low lag but must perform "blind"




(a) **Multi-tree + source coding**

(b) **Mesh**

(c) **Single-tree + randomized forwarding**

(d) **Multi-tree + mesh recovery**

**Figure 16: Constraint triangles for CEM protocols. A red dot on a vertex means that the protocol optimizes the associated metric. A red dot on an edge connecting two vertices means that the protocol can trade off between the two metrics, by varying the indicated protocol parameter.**

repairs (without a-priori knowledge of what data was lost). Proactive repair strategies that provide high continuity (without increasing lag) necessarily incur high overhead.

**Performance bounds for existing CEM protocols** The constraint triangles allow us to reason about the inherent performance limitations of all existing CEM protocols. These protocols (whether by design or otherwise) choose specific "vertices" on the triangles that largely determine their relative performance. We will demonstrate this using some example CEM protocols depicted in Figure 16.

Multi-tree systems, as shown in Figure 16(a), utilize resources better than single-tree systems. However, this comes at the cost of increased overhead for stripe tree maintenance. The multi-tree system can trade off overhead and resource utilization in the data triangle, by varying the number of stripes. As per the recovery triangle, since the multi-tree is already optimized for low lag, in order to get high continuity the multi-tree must incur high overhead. The multi-tree can, however, trade off continuity and overhead by varying the amount of source coding (e.g. FEC) overhead.

Pure mesh systems, as shown in Figure 16(b), achieve high resource utilization (data triangle), but this means they must either have high lag or incur high overhead on the data path. The mesh can, however, trade off lag and overhead by varying the swarming interval. Keeping continuity constant, the overhead in a mesh can be decreased at the cost of lag (by increasing the swarming interval). The recovery triangle and the data triangle both demonstrate this tradeoff.

Single tree-based systems with randomized forwarding, as shown in Figure 16(c), cannot match the low lag of the best tree-based systems, but are able to trade off resource utilization, overhead and continuity depending on how many packets are being proactively forwarded.

Multi tree-based systems combined with mesh recovery, as shown in Figure 16(d), exhibit a similar lag versus overhead tradeoff as a pure mesh. They can reduce the lag of the packets recovered via the auxiliary mesh by operating the mesh swarming protocol at a higher swarming rate. This is not surprising, because the packets recovered via the mesh are expected to show the same tradeoffs of a pure mesh system. The multi-tree system combined with mesh recovery also demonstrates a tradeoff similar to that of a pure multi-tree system, wherein it can trade off the resources utilized to push

tree packets and the overhead of tree maintenance, by varying the number of stripe trees.

### 6.3 Implications for future protocols

The constraint triangles imply that existing and future hybrid systems that combine trees and meshes cannot *fundamentally* improve performance since each component of the hybrid is subject to the constraint triangles. For instance, packets in a single-tree with mesh hybrid system follow the tree triangles for the packets that go along the tree and the mesh triangles for packets that are recovered using the mesh.

We note that the triangles do not preclude the design of *adaptive* protocols that change the data topology from a tree to a mesh depending on system conditions. Such a protocol can optimize for current system conditions (e.g. provide low lag using a tree when churn is low and provide high resilience using a mesh when churn is high), but will again be unable to simultaneously provide all of low lag, high continuity, and low overhead.

The constraint triangles model we have presented was originally inspired by observations based on our experiments. We used feedback from the model development to direct our experiments, and the results from our experiments to refine the model. We believe the triangles, as presented, reflect an accurate synopsis of our results and intuitively present the reasons behind inherent limitations of CEM data planes.

## 7. CONCLUSION

We have performed a systematic empirical study of CEM data delivery techniques. By factoring out the control plane, we were able to isolate the inherent performance characteristic of competing data plane designs and recovery techniques. We evaluate the design choices under a range of conditions that are likely to arise in a practical deployment.

Our study covers all basic dataplane designs, hybrid designs, and all major recovery techniques. Moreover, we study new combinations of recovery techniques and contribute a novel optimization to reduce the join delay of mesh dataplanes. Our empirical results demonstrate the inherent tradeoffs of CEM design choices. Although some of these tradeoffs were expected, this is the first work that systematically explores the design space to demonstrate that these tradeoffs are inherent.

Finally, we condense our findings into a simple model that identifies what we conjecture to be fundamental constraints that no CEM design can violate. In particular, the model asserts that no CEM design can simultaneously achieve all three of low overhead, low lag, and high continuity.

## 8. REFERENCES


[1] C. Abad, W. Yurcik, and R. Campbell. A survey and comparison of end-system overlay multicast solutions suitable for network centric warfare. *Proceedings of SPIE'04*, pages 215–226, 2004.

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable Application Layer Multicast. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'02)*, August 2002.

[3] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan. Resilient multicast using overlays. In *Proceedings of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'03)*, June 2003.

[4] BBC iPlayer. `http://www.bbc.co.uk/iplayer/`.

[5] S. Birrer, D. Lu, F.E. Bustamante, Y. Qiao, and P. Dinda. Fatnemo: Building a resilient multi-source multicast fat-tree. In *Proceedings of 9th International Workshop on Web Content Caching and Distribution*, 2004.

[6] R.E. Blahut. *Theory and Practice of Error Control Codes*. Addison Wesley, MA, 1994.

[7] T. Bonald, L. Massoulie, F. Mathieu, D. Perino, and A. Twigg. Epidemic live streaming: Optimal performance trade-offs. In *Proceedings of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'08)*, June 2008.

[8] J. Byers, M. Luby, and M. Mitzenmacher. A digital fountain approach to asynchronous reliable multicast. *Proceedings of IEEE Journal on Selected Areas in Communication (JSAC'02)*, 20(8), October 2002.

[9] M. Castro, P. Druschel, A.M Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in a cooperative environment. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, October 2003.

[10] M. Cha, P. Rodriguez, J. Crowcroft, S. Moon, and X. Amatriain. Watching television over an IP network. In *Proceedings of the ACM/USENIX Internet Measurement Conference (IMC'08)*, October 2008.

[11] Y. Chu, A. Ganjam, T.S.E. Ng, S.G. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang. Early Experience with an Internet Broadcast System Based on Overlay Multicast. In *Proceedings of USENIX Annual Technical Conference (USENIX'04)*, June 2004.

[12] M. Dischinger, A. Haeberlen, K.P. Gummadi, and S. Saroiu. Characterizing residential broadband networks. In *Proceedings of the ACM/USENIX Internet Measurement Conference (IMC'07)*, October 2007.

[13] A.J. Ganesh, A.M. Kermarrec, and L. Massoulie. Scamp: Peer-to-peer lighweight membership service for large-scale group communication. In *Proceedings of the 3rd International Workshop on Networked Group Communications (NGC'01)*, London, UK, November 2001.

[14] Geogia Tech Internet topology model. `http://www.cc.gatech/fac/Ellen.Zegura/graphs.html/`.

[15] V.K. Goyal. Multiple description coding: Compression meets the network. *Proceedings of IEEE Signal Processing Magazine*, 18(5):74–93, September 2001.

[16] K.P. Gummadi, S.Saroiu, and S.D. Gribble. King: Estimating latency between arbitrary Internet end hosts. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop (IMW'02)*, November 2002.

[17] J. Jannotti, D. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole Jr. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of the 4th USENIX Symposium on Operating Systems Design and Implementation (OSDI'00)*, October 2000.

[18] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, 2003.

[19] B. Li, S. Xie, Y. Qu, G.Y. Keung, C. Lin, J. Liu, and X. Zhang. Inside the new Coolstreaming: Principles, measurements and performance implications. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'08)*, April 2008.

[20] B. Li, K. Yik, S. Xie, J. Liu, I. Stoica, H. Zhang, and

X. Zhang. Emperical study of the Coolstreaming system. In *Proceedings of IEEE Journal on Selected Areas in Communication (JSAC'07), Special Issues on Advance in Peer-to-Peer Streaming Systems*, 2007.

[21] J. Liu, S.G Rao, B. Li, and H. Zhang. Opportunities and challenges of peer-to-peer internet video broadcast. In *Proceedings of IEEE, Special Issue on Recent Advances in Distributed Multimedia Communications*, 2007.

[22] S. Liu, R.Z. Shen, W. Jiang, J. Rexford, and M. Chiang. Performance bounds for peer-assisted live streaming. In *Proceedings of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'08)*, June 2008.

[23] Livestation: Be there now. `http://www.livestation.com`.

[24] T. Locher, R. Meier, S. Schmid, and R. Wattenhofer. Push-to-pull peer-to-peer live streaming. In *Proceedings of International Symposium of Distributed Computing*, September 2007.

[25] N. Magharei and R. Rejaie. PRIME: Peer-to-peer Receiver-drIven MEsh-based Streaming. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'07)*, May 2007.

[26] N. Magharei, R. Rejaie, and Y. Guo. Mesh or multiple-tree: A comparative study of live p2p streaming approaches. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'07)*, May 2007.

[27] A. Nandi, B. Bhattacharjee, and P. Druschel. What a mesh: Understanding the design tradeoffs for streaming multicast. In *Proceedings of ACM SIGMETRICS Performance Evaluation Review, special issue on the SIGMETRICS 2009 poster session*, Seattle, WA, USA, June 2009.

[28] A. Nandi, A. Ganjam, P. Druschel, T.S.E. Ng, I. Stoica, H. Zhang, and B. Bhattacharjee. SAAR: A shared control plane for overlay multicast. In *Proceedings of the 4th Symposium on Networked Systems Design and Implementation (NSDI'07)*, April 2007.

[29] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *Proceedings of 12th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'02)*, Miami Beach, FL, USA, May 2002.

[30] V.S. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A.E. Mohr. Chainsaw: Eliminating trees from overlay multicast. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems (IPTPS '05)*, Ithaca, NY. USA, February 2005.

[31] F. Painese, D. Perino, J. Keller, and E. Biersack. PULSE: An adaptive, incentive-based, unstructured p2p live streaming system. In *Proceedings of IEEE Transactions on Multimedia, Special Issue on Content Storage and Delivery in Peer-to-Peer Networks, Volume 9*, November 2007.

[32] Planetlab. http://www.planet-lab.org/.

[33] Sopcast. `http://www.sopcast.com`.

[34] S. Tewari and L. Kleinrock. Analytical model for bittorrent-based live video streaming. In *Proceedings of IEEE NIME 2007 Workshop*, January 2007.

[35] D. Tran, K. Hua, and T. Do. ZIGZAG: An efficient peer-to-peer scheme for media streaming. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'03)*, 2003.

[36] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI'02)*, December 2002.

[37] V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast. In *Proceedings of the 14th IEEE International Conference on Network Protocols (ICNP '06)*, November 2006.

[38] V. Vishnumurthy and P. Francis. On heterogeneous overlay construction and random node selection in unstructured p2p networks. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'06)*, Barcelona, Spain, April 2006.

[39] F. Wang, Y. Xiong, and J. Liu. mTreebone: A hybrid tree/mesh overlay for application-layer live video multicast. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS'07)*, June 2007.

[40] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI'02)*, Boston, MA, December 2002.

[41] M. Zhang, J.G. Luo, L. Zhao, and S.Q. Yang. A peer-to-peer network for live media streaming - using a push-pull approach. In *Proceedings of ACM Multimedia*, 2005.

[42] M. Zhang, Q. Zhang, L. Sun, and S. Yang. Understanding the power of pull-based streaming protocol: Can we do better ? *Proceedings of IEEE Journal on Selected Areas in Communication (JSAC'07), Special Issue on Advances in Peer-to-Peer Streaming Systems*, 2007.

[43] X. Zhang, J. Liu, B. Li, and T.S.P. Yum. Coolstreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'05)*, Miami, FL, USA, March 2005.

[44] Y. Zhou, D. Chiu, and J. Lui. A simple model for analysis and design of p2p streaming protocols. In *Proceedings of IEEE International Conference on Network Protocols (ICNP'07)*, October 2007.

[45] Yan Zhu, Min-You Wu, and Wei Shu. Comparison study and evaluation of overlay multicast networks. In *ICME '03: Proceedings of the 2003 International Conference on Multimedia and Expo - Volume 3 (ICME'03)*, pages 493–496. IEEE Computer Society, 2003.