mTreebone: A Hybrid Tree/Mesh Overlay for Application-Layer Live Video Multicast

Feng Wang[1]*, Yongqiang Xiong[2], Jiangchuan Liu[1]†

[1]School of Computing Science
Simon Fraser University
Burnaby, B.C. Canada
{fwa1, jcliu}@cs.sfu.ca

[2]Wireless and Networking Group
Microsoft Research Asia
Beijing, China
yqx@microsoft.com

Abstract

Application-layer overlay networks have recently emerged as a promising solution for live media multicast on the Internet. A tree is probably the most natural structure for a multicast overlay, but is vulnerable in the presence of dynamic end-hosts. Data-driven approaches form a mesh out of overlay nodes to exchange data, which greatly enhances the resilience. It however suffers from an efficiency-latency tradeoff, given that the data have to be pulled from mesh neighbors with periodical notifications.

In this paper, we suggest a novel hybrid tree/mesh design that leverages both overlays. The key idea is to identify a set of stable nodes to construct a tree-based backbone, called treebone, with most of the data being pushed over this backbone. These stable nodes, together with others, are further organized through an auxiliary mesh overlay, which facilitates the treebone to accommodate node dynamics and fully exploit the available bandwidth between overlay nodes.

This hybrid design, referred to as mTreebone, is braced by our real trace studies, which show strong evidence that the performance of an overlay closely depends on a small set of backbone nodes. It however poses a series of unique and critical design challenges, in particular, the identification of stable nodes and seamless data delivery using both push and pull methods. In this paper, we present optimized solutions to these problems, which reconcile the two overlays under a coherent framework with controlled overhead. We evaluate mTreebone through both simulations and PlanetLab experiments. The results demonstrate the superior efficiency and robustness of this hybrid solution.

1. Introduction

Large-scale live video multicast is one of the most attractive network applications, which unfortunately has yet to be fulfilled over the Internet. Earlier proposals attempted to use IP multicast as the underlying vehicle. While IP multicast is efficient, its deployment remains limited in reach and scope due to many practical and political issues. Recently, focus has been shifted to the use of application-layer overlays for multicast data delivery, where end-hosts act as not only receivers but also relays to forward the received data to others [6]. This peer-to-peer solution enables quick and easy deployment of multicast applications, for there is no involvement of routers. Nevertheless, a new set of challenges have to be addressed in this communication paradigm, in particular, the dynamics of the autonomous end-hosts. The high data volume and stringent timing requirements of video applications further complicate the overlay design.

A large body of application-layer multicast proposals suggest the end-hosts be organized into a tree for data delivery [3][6][13]. This is probably the most natural and efficient structure in terms of bandwidth and delay optimization. The tree structure, however, is vulnerable to dynamics, and the leave or failure of a node, particularly one close to the source, may cause data outage in all its descendants. The tree thus has to repair frequently, which brings extra costs and renders the structure to be sub-optimal.

A possible enhancement to the single tree structure is to employ multiple disjoint trees [5][15][18], each of which delivering a sub-stream of the video. While the multi-tree can remarkably improve resilience, it is more complex to be constructed and maintained. Optimizing the multiple trees

[*] This work was mainly performed when Feng Wang was a visiting student at Microsoft Research Asia.

[†] J. Liu's work was supported by a Canadian NSERC Discovery Grant, an NSERC Research Tools and Instruments Grant, a Canada Foundation for Innovation (CFI) New Opportunities Grant, a BCKDF Matching Grant, and an SFU President's Research Grant.

as a whole without violating the disjoint property can be quite difficult in the presence of dynamic nodes [4][14].

Recently, data-driven randomized overlays have been proposed as an alternative solution with better resilience [11][10][16][23]. Such systems essentially construct a mesh out of overlay nodes, where each node independently selects some other nodes as neighbors and exchanges data with them. A drastic difference between the tree and the mesh overlays lies in their data delivery strategies. In a tree or multi-tree, a video stream is basically *pushed* along well-defined routes, i.e., from parents to their children. In a mesh, given that multiple and dynamic neighbors may have video data available to send, a node has to *pull* data to avoid significant redundancies. A mesh-based system is therefore more robust, but experiences longer delays and higher control overhead. More explicitly, there is an efficiency-latency tradeoff [19][22]: if the mesh nodes choose to send notifications for every data block arrival, the overhead will be excessive; Periodical notifications containing buffer maps within a sliding window, as suggested in [16][23], reduce the overhead, but increase the latencies.

In this paper, we present a novel approach toward a hybrid overlay design. The key idea is to identify a set of stable nodes to construct a tree-based backbone, called *treebone*, with most of the data being pushed over this backbone. These stable nodes, together with others, are further organized through an auxiliary mesh overlay, which facilitates the treebone to accommodate dynamic nodes and also to fully explore the available bandwidth between node pairs.

Our design, referred to as *mTreebone*, is largely motivated by a series of trace studies of a practical overlay streaming system[20]. Our results provide strong evidence that most of the data blocks delivered through a mesh overlay essentially follow a specific tree structure or a small set of trees. The similarity of the trees, defined as the fraction of the common links, can be as high as 70%. The overlay performance thus closely depends on the set of common internal nodes and their organization. If such a set consists mainly of the stable nodes, with others being organized through a mesh, we can expect high efficiency with low overhead and delay simultaneously.

In this hybrid tree/mesh design, however, a series of unique and important issues need to be addressed. First, we have to identify the stable nodes in an overlay, and gradually build up the treebone; Second, we need to reconcile the push and pull data delivery in the two overlays, respectively. They should work complementarily to maximize the joint efficiency in the presence of autonomous nodes. In this paper, we derive an optimal age threshold for identifying stable nodes, which maximizes their expected service times in the treebone. We then propose a set of overlay construction and evolution algorithms to enable seamless treebone/mesh collaboration with minimized control overhead and transmission delay. Finally, we present a buffer partitioning and scheduling algorithm, which coordinates push/pull operations and avoids data redundancy.

We extensively evaluate the performance of mTreebone and compare it with existing mesh and tree based solutions. The simulation results demonstrate the superior efficiency and robustness of this hybrid solution. Such results are reaffirmed by our experimental results of an mTreebone prototype over the PlanetLab network [1].

The remainder of this paper is organized as follows: In Section 2, we introduce the background and related work. Section 3 summarizes our trace studies and overviews the framework of mTreebone. Details about treebone evolution and its interactions with the mesh are discussed in Sections 4 and 5, respectively. We evaluate the performance of mTreebone in Section 6. Finally, Section 7 concludes the paper and offers potential future research directions.

2. Background and Related Work

Many application-layer multicast protocols have been proposed for media streaming, which can be broadly classified into two categories according to their overlay structures [13][14], namely, *tree-based* and *mesh-based*. The former, like IP Multicast, uses a tree rooted at the source as the data delivering structure; typical examples include ESM [6] and NICE [3]. However, unlike IP multicast with dedicated routers, the nodes in an application-layer overlay are autonomous end-hosts, which join or leave at will or crash without notification. Later studies, e.g., SplitStream [5], CoopNet [15] and THAG [18], rely on multiple disjoint trees to mitigate the impact of such overlay churns. Recently, an robust yet simple alternative, data-driven design, is proposed. It essentially constructs a mesh out of the overlay nodes, with each node having a small set of neighbors to exchange data. Examples include Bullet [11], Bullet' [10], Chainsaw [16], CoolStreaming [23] and AnySee [12].

Both tree/multi-tree and mesh solutions have shown their success in practical deployment[13], and yet neither completely overcomes the challenges from the dynamic peer-to-peer environment. The selling point for the data-driven mesh overlays is their robustness, but the lack of a well-ordered parent/children relation implies that data have to be pulled from neighbors, which suffers the efficiency-latency tradeoff as discussed before. The push delivery in a tree is efficient, but has to face data outage in descendants when an internal node fails. The pre-defined flow direction also prevents the overlay from fully utilizing the bandwidth between node pairs, e.g., that between two leaf nodes.

Given the pros and cons of the two approaches, a natural question is whether we can combine them to realize a hybrid overlay that is both efficient and resilient. An early attempt toward such combination is HON [24], which fo-

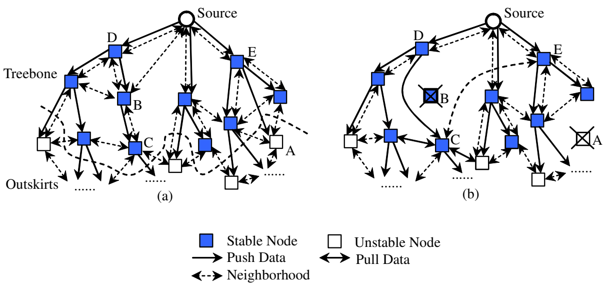

Figure 1. mTreebone framework. (a) A hybrid overlay; (b) Handling node dynamics.

cuses on on-demand streaming. For live streaming, a recent work is ChunkySpread [19]. Our work differs in that we explicitly classify stable and transient nodes, and focus on the effective use of stable nodes. We demonstrate that a treebone consisting purely of stable nodes remarkably boosts the overlay performance.

3. Overview of mTreebone

We consider a live video streaming system using application-layer multicast. There is a single dedicated source node, which is persistent during the streaming session. The video stream, originated from the source, is divided into equal-length blocks. The clients that are interested in the video form an overlay network rooted at the source, and each overlay node, except for the source, acts both as a receiver and, if needed, an application-layer relay that forwards data blocks. In addition, these nodes can join and leave the overlay at will, or crash without notification.

The primary issue for building such a system lies in the construction of a resilient overlay structure with low overhead and short delay. In our proposed *mTreebone*, we advocate a hybrid tree/mesh overlay design that combines the best features of both approaches to meet such demands.

3.1. Treebone: A Stable Backbone Overlay

The core of our design is a tree-based backbone, referred to as *treebone*. Unlike existing tree-based approaches, this backbone consists of only a subset of the nodes, in particular, the stable nodes. Other non-stable nodes are attached to the backbone as outskirts. Most of the streaming data are pushed through the treebone and eventually reach the outskirts, as shown in Fig. 1.

It is worth noting that even a small set of stable nodes is sufficient to support the entire overlay. As an illustration, consider a simple K-ary tree of height H. The fraction of its internal nodes, i.e., those belong to the backbone, is no more than $1/K$ if the tree is complete and balanced ($\sum_{i=0}^{H-2} K^i / \sum_{i=0}^{H-1} K^i < \frac{1}{K}$). As such, the construction and maintenance overheads for the treebone are relatively low, particularly considering its nodes are stable, while the data delivery is efficient.

The critical question here is thus how to identify stable nodes. Recent studies have found that, in overlay multicast systems, nodes with a higher age tend to stay longer [4]. It offers a hint to identify stable nodes by periodically examining their ages. With this hint, we devise an optimal threshold-based method, which is discussed in Section 4.1.

3.2. Mesh: An Adaptive Auxiliary Overlay

The treebone, however, cannot completely eliminate repairing operations because the nodes are not absolutely persistent. In addition, the potential bandwidth between the unstable nodes are largely ignored by the treebone.

To improve the resilience and efficiency of the treebone, we further organize all the nodes into a mesh overlay. Similar to CoolStreaming [23], in this auxiliary mesh overlay, each node keeps a partial list of the active overlay nodes and their status. This local list facilitates the node to locate a set of mesh neighbors as well as its dedicated treebone parent. To keep the list updated, a light-weighted, scalable, random gossip algorithm, SCAMP [7], is used for the nodes to periodically exchange their status information. The mesh neighbors also periodically exchange their buffer maps. Unlike existing data-driven systems, a node will not actively schedule to fetch data blocks from neighbors using such data availability information. Such a fetch is invoked only if data outage occurs in the treebone.

Fig. 1(a) illustrates this hybrid mTreebone design. When an unstable node, such as node A fails or leaves, it will not affect the data pushed along the treebone. On the other hand, the treebone nodes are stable and seldom leave; even a leave happens, the impact can be remarkably mitigated with the help from the mesh overlay. For example, consider the leave of node B, shown in Fig 1(b). While node C is affected, it can easily pull the missing data from its mesh neighbors before it re-attaches to the treebone.

4. Treebone Construction and Optimization

To realize such a hybrid overlay for live streaming, a series of unique and important issues have to be addressed. First, we have to identify the stable nodes in the overlay; Second, we have to position the stable nodes to form the treebone, which should also evolve to optimize its data delivery; Third, we have to reconcile the treebone and the mesh overlays, so as to fully explore their potentials. In this section, we present our solutions for the construction and optimization of the treebone. Its interactions with the mesh will be detailed in the next section.

4.1. Optimal Stable Node Identification

Intuitively, the stability of a node is proportional to its duration in the overlay, which unfortunately cannot be known before the node actually leaves. We thus resort to a practical prediction using a node's age in the session, i.e., the time elapsed since its arrival. As mentioned earlier, existing studies have shown that the nodes already with higher ages tend to stay longer [4]; hence, a node's age partially reflects its stability, and if its age is above a certain threshold, a node can be considered stable and moved into the treebone. Once a stable node is in the treebone, it remains there until it leaves or the session ends.

The effectiveness of the treebone clearly depends on the age threshold. If the threshold is too low, many unstable nodes would be included in the treebone; on the other hand, given a high threshold, few nodes could be considered stable. Our objective is thus to optimize the Expected Service Time (EST) of a treebone node by selecting an appropriate age threshold.

Let $f(x)$ be the probability distribution function (PDF) of node duration, and L be the length of the session. Since a node starts serving in the treebone when its age exceeds the corresponding threshold, for a treebone node arriving at time t, its expected service time $EST(t)$ can be calculated as the expected duration minus the corresponding age threshold, $T(t)$, i.e.,

$$EST(t) = \frac{\int_{T(t)}^{L-t} x f(x) \mathrm{d}x + \int_{L-t}^{\infty} (L-t) f(x) \mathrm{d}x}{\int_{T(t)}^{\infty} f(x) \mathrm{d}x} - T(t)$$

Previous studies on video client behavior have suggested that node durations generally follow a heavy-tailed distribution [2][17][20], in particular, the Pareto distribution with parameters k and x_m (k is a shape parameter that determines how skew the distribution is, and x_m is a location parameter that determines where the distribution starts). Given this model, we have the following expression,

$$EST(t) = \frac{T(t)}{k-1}[1 - (\frac{T(t)}{L-t})^{k-1}]$$

To maximize $EST(t)$ with respect to $T(t)$, we have

$$EST(t)'_{T(t)} = \frac{1}{k-1} - \frac{k}{k-1}(\frac{T(t)}{L-t})^{k-1} = 0,$$

which follows that $T^{*}(t) = (L-t)(\frac{1}{k})^{\frac{1}{k-1}}$.

For the typical k value close to 1, $EST(t)$ is maximized when $T(t)$ is roughly about $0.3(L-t)$. In other words, the age threshold for a node arriving at time t is 30% of the residual session length. This is the default setting used in our experiments. In practice, we can also online estimate k and adjust the threshold accordingly.

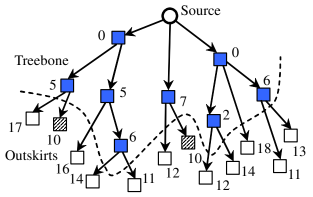

Figure 2. An example of treebone evolution. Numeric labels indicate node arrival times.

4.2. Treebone Bootstrapping and Evolution

Given the optimal age threshold, we now discuss how the nodes evolve into a stable treebone. We assume that initially only the source node is in the treebone. Each newly joined node obtains L and t from the source, as well as a partial list of existing overlay nodes, at least one of which is in the treebone. The new node then attaches itself to one of the treebone nodes and locates mesh neighbors using the list.

If a node is not in the treebone, it will periodically check its own age in the overlay. Once its age exceeds the threshold $T(t)$, it will promote itself as a treebone node. Fig. 2 shows an example, where the numeric label of each node is its arrival time.

In this basic promotion method, before time $T(0)$, no node but the source is included in the treebone, which reduces the efficiency of data delivery in this period. To alleviate this problem, we introduce a randomized promotion for the initial period of the session. For a node arriving at time t, the algorithm strikes to achieve a probability $s/T(t)$ for the node to be in the treebone when its age is s. Specifically, each non-treebone node independently checks its status per unit time; for the s-th check (i.e., at time $t+s$), it will be promoted to the treebone with probability $1/(T(t) - s + 1)$ (and 0 for $s = 0$). Such an early promotion will speed up the establishment of the treebone. And it is fully distributed with no extra message exchange among the overlay nodes. In addition, as suggested by observations from [8], the built-in randomness of the promotion will also reduce the churn of the treebone.

4.3. Treebone Optimization

The treebone constructed by the above basic algorithm does not necessarily minimize the latency for data delivery. In particular, two non-optimal substructures could exist, as shown in Fig. 3 and 4. In the first case, a node has more children than its parent, and a swap of them can reduce the average depth of the treebone nodes. In the second case, a treebone node closer to the source may still be able to accept new children; a node can use this chance to reduce its depth in the treebone. We now introduce two localized algorithms that implement such optimizations.

High-Degree-Preemption. Each treebone node x periodically checks whether it has more children than a node that is closer to the source in the treebone. Such a node, referred to as y, could either be the parent of x in the treebone, or a node known from x's local node list. If so, node x will then preempt y's position in the treebone, and y will re-join the treebone. In practice, y can simply attach itself to x, as illustrated in Fig. 3.

Low-Delay-Jump. Each treebone node x periodically checks whether there are nodes closer to the source than its parent. If so and one such node, say y, has enough bandwidth to support a new child, node x will leave its original parent and attach itself to y as a child, as illustrated in Fig. 4.

The above two algorithms will be executed by the overlay nodes iteratively until no node can further locate candidates for swapping. The average depth of the treebone nodes is monotonically decreasing in the iterations, leading to a minimal average depth. More explicitly, we have the following theorem to show the minimal average depth is actually minimum:

Theorem 4.1 *The average depth of the treebone is minimized when high-degree-preemption and low-delay-jump terminate at all treebone nodes.*

Due to the concern of space, we omit the proof here. A full version of the proof can be found in [21].

5. Collaborative Push/Pull Data Delivery

We next discuss the collaboration between the treebone and the mesh within the mTreebone framework. Such collaboration reflects in two aspects, namely, delivering video data and handling node dynamics.

5.1. Seamless Push/Pull Switching

In mTreebone, the data blocks are delivered by two means. In general, they are pushed over the treebone. And if a gap appears in the stream received by a node, due to either temporal capacity fluctuation in the treebone or node dynamics as discussed later, the node may pull the missed blocks through the mesh overlay. We introduce a seamless push/pull buffer that coordinates the treebone and the mesh to make data delivery efficient yet resilient against failure.

Fig. 5 illustrates the push/pull switching, where a *tree-push* pointer is used to indicate the latest data block delivered by the push method, and a *mesh-pull* window facilitates the pull delivery. When a node is temporarily disconnected from the treebone, its tree-push pointer will be disabled and only the mesh-pull window works to fetch data from its mesh neighbors. When it connects to the treebone again, the tree-push pointer will be re-activated. The mesh-pull window is always kept behind the tree-push pointer so as not to request data currently being delivered by the treebone. Therefore, no duplicated data blocks are received from both treebone and mesh.

5.2. Handling Node Dynamics

A node may gracefully leave the overlay, or abruptly fail without any notification. In the former, the node will proactively inform its mesh neighbors and its treebone children if it resides in the treebone. In the latter, the abrupt leave can be detected by the mesh neighbors after a silent period with no control message exchange, or by the children in the treebone after observing persistent losses. In either case, its mesh neighbors need to re-establish neighborships with other known nodes in their local node lists, and, if the node is in the treebone, its children have to relocate parents.

If the affected child is an unstable node in the outskirts of the treebone, it will check its local node list and directly attach to one node that is nearest to the source with enough available bandwidth. On the other hand, if it is a stable node, it has to re-join the treebone. To this end, the node will first locate a treebone node with enough available bandwidth and then attach itself. If no such treebone node is known, the node needs to preempt the position of an unstable node that is currently a child of a treebone node.

In the above process, mesh overlay will temporarily take over data delivery before the treebone is repaired, as discussed previously. Our simulation and experimental results demonstrate that this 2-tier overlay effectively reduce data losses in the tree repairing process. Meanwhile, its control overhead is kept at a reasonable level, which is lower than relying on a mesh only.

6. Performance Evaluation

To evaluate mTreebone, we have conducted extensive simulations and PlanetLab-based experiments. We have also implemented two state-of-the-art application layer multicast systems for comparison, namely, CoolStreaming [23] and ChunkySpread [19]. The former is a typical data-driven mesh system, and the latter is a multi-tree system, which also adopts an auxiliary neighboring graph to facilitate tree construction.

In our evaluation and comparison, we use the following three typical metrics, which combined reflect the quality of service experienced by overlay nodes.

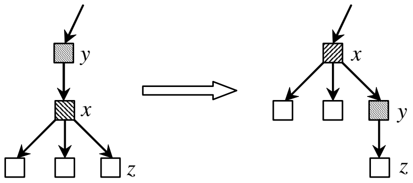

Figure 3. An illustration of *high-degree-preemption***.**

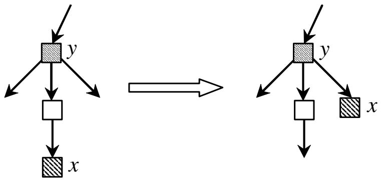

Figure 4. An illustration of *low-delay-jump***.**

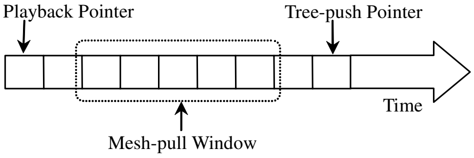

Figure 5. Design of the push/pull switch buffer.

Startup latency, which is the time taken by a node between its requesting to join the session and receiving enough data blocks to start playback;

Transmission delay, which is the time to deliver a data block from the source to the node. Due to the buffer-and-relay nature in overlay networks, the end-to-end transmission delay is orders of magnitude higher than that in IP multicast, thus we use *second* as the unit, as for startup latency;

Data loss rate, which is defined as the fraction of the data blocks missing their playback deadlines, i.e., either lost during transmission or experienced excessive delays.

We adopt a dynamic scenario for the evaluation, where the overlay nodes arrive at different times and may also leave or fail before the session ends. As discussed in Section 4.1, we use the Pareto distribution to model the durations of the nodes in a session, which is also suggested by many empirical trace studies [2][17]. The default parameters of the distribution are adopted from the recently modeling work for peer-to-peer streaming in [4] as well as trace studies of PPLive, a popular commercial peer-to-peer streaming system [9][20]. We also investigate the impact of other parameter settings, in particular, that for the skew factor k, which reflects the churn rate of the clients.

6.1. Simulation Results

Unless otherwise specified, the following default parameters are used in our simulation, most of which follow the typical values reported in [2][9][20]. The session length L is set to 6000 seconds and each data block is of 1-second video; there are 5000 overlay nodes; the maximum end-to-end delay is 1000 ms between two overlay nodes, and the maximum upload bandwidth is uniformly distributed from 4 to 12 times of the bandwidth required for a full streaming. For comparison, the number of partners in CoolStreaming is set to 5 and the number of substreams in ChunkySpread is set to 16. The details about these two parameters and their setting guidelines can be found in [23][19], respectively.

We set the default age threshold $T(t)$ to 30% of the residual session length, which corresponds to the optimal setting (see Section 4.1) when k is close to 1. We will also examine other settings of $T(t)$ as well as the impact of k.

Fig. 6 shows the CDF of the startup latency. We can see that mTreebone has the lowest latency, followed by CoolStreaming and then by ChunkySpread. This is because, when two events (join and leave) occur closely, the tree repair process in ChunkySpread may delay the newly joined node from receiving data. In addition, the new node needs to receives sub-streams across multiple trees to assemble the original stream, which further increases the delay. A CoolStreaming node, however, has to locate multiple mesh neighbors and establish relations, and the data pull operation also slows down the process. On the other hand, a new node in mTreebone can receive data in the tree structure even before establishing the mesh neighborship, and the *high-degree-preemption* and *low-delay-jump* minimize the delay of the treebone. Similar reasons also explain the results of transmission delay in Fig. 7. The data loss rate for the three systems are given by Fig. 8. mTreebone also outperforms CoolStreaming and ChunkySpread, validating the advantage of the hybrid design.

Fig. 9 shows the data loss rate in mTreebone with different age threshold. Although the data loss rates are generally below 1% for a wide range of thresholds, the minimal appears at 30% of the residual session length, which corresponds to the optimal threshold setting. Also, we have conducted simulations with different k values. In general, a larger k means the overlay is of a higher churn rate, i.e., more dynamic. We show the data loss rate of mTreebone as a function of k in Fig. 10. When k is no greater than 1.5, our mTreebone is fairly stable, and the default value of 1 is thus representative for performance evaluation. It is worth noting that, although the data loss rate noticeably increases for k over 1.5, it remains less than 2%, implying that our hybrid design resists to node dynamics well.

6.2. PlanetLab-based Experimental Results

To further investigate the performance of mTreebone, we have implemented a prototype and conducted experiments on the PlanetLab. Besides the three typical metrics for overlay nodes, we also examine the cost of the whole system in such a real network, i.e., the message overhead to construct, maintain, and repair the overlay. When a mesh is applied, the overhead also includes the messages to exchange data availability and to request blocks from neighbors. Such

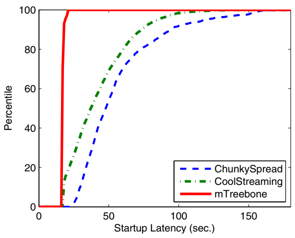

Figure 6. CDF of startup latency (simulation).

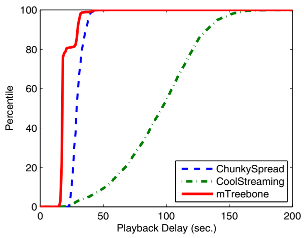

Figure 7. CDF of transmission delay (simulation).

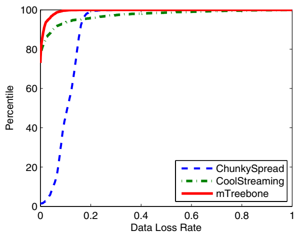

Figure 8. CDF of data loss (simulation).

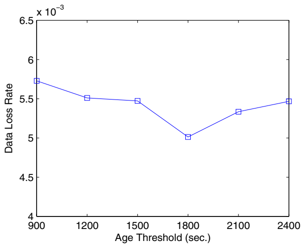

Figure 9. Avg. data loss w/ different $T(t)$ (simulation).

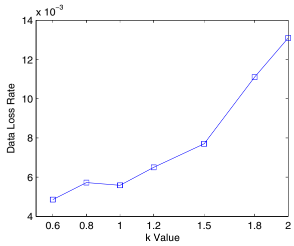

Figure 10. Avg. data loss w/ different k (simulation).

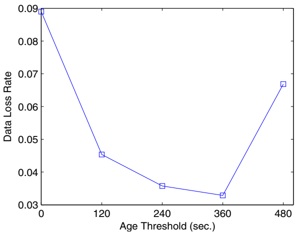

Figure 11. Avg. data loss w/ different $T(t)$ (PlanetLab).

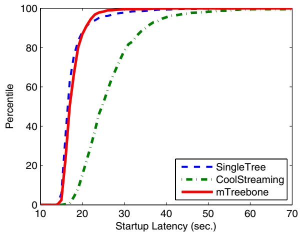

Figure 12. CDF of startup latency (PlanetLab).

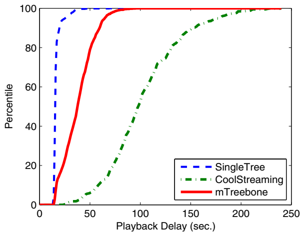

Figure 13. CDF of transmission delay (PlanetLab).

messages are generally of small sizes, but their excessive total number could still impose heavy load to routers. Hence, in our evaluation, we use the total number of the control messages as the metric to measure their impact.

In each experiment, we let 200 PlanetLab nodes stay in the session following the Pareto distribution. To accumulate enough join and leave events for evaluation, we also allow a node to re-join the overlay after it leaves the overlay for a while. We first vary the age threshold to see its impact on data loss rate. The result is shown in Fig. 11, where the threshold at 30% again leads to the minimum loss rate. Hence, we still use it as the default setting in mTreebone.

Figs. 12–14 give the results of startup latency, transmission delay, and data loss rate of mTreebone. For comparison, we also implement CoolStreaming and a single tree based system, and present their respective results in the figures. From these results, we can see that the startup latency of mTreebone is very close to that of the tree-based approach, and is much lower than that of CoolStreaming. Similar observation applies to transmission delay. On the other hand, mTreebone and CoolStreaming both have much lower data loss rates, as compared to the single tree. Comparing with the simulation results, the data loss rates on the PlanetLab are generally higher. This is due to the influences of background traffic.

Fig. 15 compares the control overhead of the three systems. It is not surprising that mTreebone has a higher overhead than the single tree; but, even though mTreebone has to maintain two overlays, its overhead is still lower than CoolStreaming. This result suggests that the stable treebone is the major delivery path in mTreebone. Also the total traffic volume of the (small) control messages are indeed quite low, which generally less than 1% of the total data traffic of mTreebone in our experiments. We thus believe that the control overhead of mTreebone is acceptable.

To further demonstrate the resilience of mTreebone against node dynamics, Fig. 16 shows the data loss rates for different values of k. The results reaffirm that mTreebone is quite stable for a wide range of churn levels.

7. CONCLUSION AND FUTURE WORK

In this paper, we explored the opportunity to leverage both tree and mesh approaches within a hybrid framework, *mTreebone*. We presented effective and coherent solutions to reconcile the two overlays in the mTreebone framework. Specifically, we derived an optimal age threshold to identify stable nodes, which maximizes their expected service time in the treebone. We designed a set of overlay construction and evolution algorithms, which minimize the startup and transmission delays. Finally, we gave a buffer partitioning and scheduling algorithm, which enables seamless treebone/mesh collaboration in data delivery.

We extensively evaluated the performance of mTreebone and compared it with existing mesh and tree based solu-

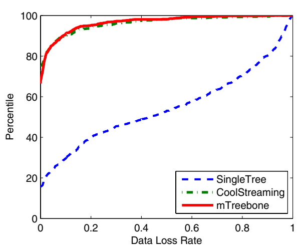

Figure 14. CDF of data loss (PlanetLab).

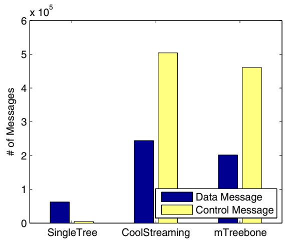

Figure 15. Message overhead (PlanetLab).

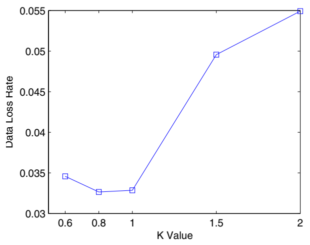

Figure 16. Avg. data loss w/ different k (PlanetLab).

tions. The simulation results demonstrated the superior efficiency and robustness of this hybrid solution, which were further validated by our experiments of an mTreebone prototype over the PlanetLab network.

It is worth emphasizing that diverse optimization techniques could be carried over the treebone, for its relatively smaller scale and inherent stability. In our future work, we plan to examine advanced node organization/re-organization methods to further improve its efficiency and its interactions with the mesh. We will also explore the use of multi-tree-based backbone, which may lead to more balanced load and finer-grained bandwidth control. Finally, we are interested in experiments of larger scales with our prototype as well as real deployment over the global Internet.

References

[1] *PlanetLab Website: http://www.planet-lab.org/*.

[2] K. C. Almeroth and M. H. Ammar. Collecting and modeling the join/leave behavior of multicast group members in the mbone. In *IEEE International Symposium on High Performance Distributed Computing (HPDC)*, 1996.

[3] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *ACM SIGCOMM*, 2002.

[4] M. Bishop, S. Rao, and K. Sripanidkulchai. Considering priority in overlay multicast protocols under heterogeneous environments. In *IEEE INFOCOM*, 2006.

[5] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *ACM SOSP*, 2003.

[6] Y. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *ACM SIGMETRICS*, 2000.

[7] A. J. Ganesh, A. M. Kermarrec, and L. Massoulie. Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, (2):139–149, February 2003.

[8] P. B. Godfrey, S. Shenker, and I. Stoica. Minimizing churn in distributed systems. In *ACM SIGCOMM*, 2006.

[9] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross. Insights into pplive: A measurement study of a large-scale p2p iptv system. In *Workshop on Internet Protocol TV (IPTV) services over World Wide Web*, 2006.

[10] D. Kostic, R. Braud, C. Killian, E. Vandekieft, J. W. Anderson, A. C. Snoeren, and A. Vahdat. Maintaining high bandwidth under dynamic network conditions. In *USENIX Anual Technical Conference*, 2005.

[11] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *ACM SOSP*, 2003.

[12] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng. Anysee: Peer-to-peer live streaming. In *IEEE INFOCOM*, 2006.

[13] J. Liu, S. G. Rao, B. Li, and H. Zhang. Opportunities and challenges of peer-to-peer internet video broadcast. In *Proceedings of the IEEE*, 2007.

[14] N. Magharei, R. Rejaie, and Y. Guo. Mesh or multiple-tree: A comparative study of p2p live streaming services. In *IEEE INFOCOM*, 2007.

[15] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanid-kulchai. Distributed streaming media content using cooperative networking. In *ACM NOSSDAV*, 2002.

[16] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. Chainsaw: Eliminating trees from overlay multicast. In *IPTPS*, 2005.

[17] K. Sripanidkulchai, B. Maggs, and H. Zhang. An analysis of live streaming workloads on the internet. In *Internet Measurement Conference*, 2004.

[18] R. Tian, Q. Zhang, Z. Xiang, Y. Xiong, X. Li, and W. Zhu. Robust and efficient path diversity in application-layer multicast for video streaming. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(8):961–972, August 2005.

[19] V. Venkataraman, P. Francis, and J. Calandrino. Chunkyspread: Multi-tree unstructured peer-to-peer multicast. In *IPTPS*, 2006.

[20] F. Wang and J. Liu. A trace-based analysis of packet flows in data-driven overlay networks. Technical report, 2006.

[21] F. Wang, Y. Xiong, and J. Liu. mTreebone: A hybrid P2P multicast framework for live streaming. Technical report, 2006.

[22] 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 *ACM Multimedia*, 2005.

[23] X. Zhang, J. Liu, B. Li, and T. P. Yum. Coolstreaming/donet: A data-driven overlay network for efficient live media streaming. In *IEEE INFOCOM*, 2005.

[24] M. Zhou and J. Liu. A hybrid overlay network for video-on-demand. In *IEEE ICC*, 2005.