CONSTRUCTING RELIABLE AND EFFICIENT OVERLAYS FOR P2P LIVE MEDIA STREAMING

STEPHEN A. JARVIS, GUANG TAN, DANIEL P. SPOONER AND GRAHAM R. NUDD

Department of Computer Science, University of Warwick,
Coventry, CV4 7AL, United Kingdom
{saj,gtan,dps,grn}@dcs.warwick.ac.uk

Abstract:
For single-source, single-tree-based peer-to-peer live media streaming, it is generally believed that a short (and wide) data delivery tree provides the best comprehensive performance in terms of reliability and service delay. While a short tree directly benefits delay optimization, it is unclear whether such a structure maximizes reliability, which is sometimes more critical for a streaming Internet service. This paper compares several prevalent overlay construction algorithms in terms of (1) service reliability; (2) service delay and (3) protocol overhead. A new Heap Algorithm *is proposed to enhance reliability by leveraging the peers' time properties while maintaining a short tree, which in turn helps to reduce service delay. This new algorithm dynamically moves peers between different layers of the tree according to a simple metric called* Service Capacity Contribution *(SCC), and gradually adjusts the overlay toward a short tree with peers ordered in time. Extensive simulations show that this new algorithm achieves better comprehensive performance than existing algorithms.*

1. Introduction

There has been a good deal of research in recent years on the topic of live media streaming services over the Internet [3, 4, 12, 16]. Due to the stringent requirements on network resources and increasing market demand, the traditional client-server framework faces significant challenges. For example, during busy periods, the server's bandwidth may easily be overwhelmed by a surge in the client population. A direct solution to this problem is upgrading the server system hardware or clustering a large number of single systems into a server farm. However, this approach will only scale so far. Dedicated content distribution networks (CDN) provide an alternative solution, however its prohibitive deployment costs make it uneconomical for many small- or medium-sized sites.

Given the fact that IP multicast has not been widely deployed and this situation is unlikely to improve in the near future, the peer-to-peer (P2P) paradigm offers an attractive solution. In this architecture, the clients or peers help to relay the received content to other clients and thus form a large content distribution network. This approach is perfectly scalable in the sense that available bandwidth increases with the growth of the network. Even if the source server can support only a limited number of concurrent clients, the data can be distributed to a large network population in a self-scaling manner. In the context of live media streaming, the peer-to-peer transfer mode mostly serves to complement the client-server mode, since a number of clients will need to receive the content from the server first-hand.

An overlay of peers is often viewed as a tree rooted at the content provider. To provide satisfactory quality of service (QoS), the data delivery tree needs to address three problems: (1) to reduce the impact of peer dynamics – peers are free to join and leave at any time, and abrupt departure or failure of a node will result in service interruptions on all of its descendants in the tree. Losses due to such failures are more significant than regular packet losses in the physical network and may cause streaming breaks in the order of tens of seconds; (2) to minimize end-to-end service delay – transfers over the logical overlay generally involve longer delays than unicast in the physical network, and hence introduce a prolonged startup delay and increase network dynamics to the streams; (3) to maintain a reasonable overhead – peers may need to reconnect to other peers for the purpose of overlay adjusting, which usually requires coordination among multiple peers. In a distributed environment lacking time synchronization, such operations may require transient pauses in the streaming.

Given a set of peers with heterogenous out-degrees (limited by the actual bandwidth resource and under the condition that no network congestion occurs near the nodes), it is generally believed that a short (and wide) tree provides a good peer structure [12, 19, 16, 7] for meeting these requirements. Intuitively, the shortness helps to reduce the probability of service disruption due to the departure, failure, or congestion at an ancestor node, and hence enhances tree reliability. A short tree also means a small average hop count from the root to the peers, and this helps to minimize the average network delay if the peers are appropriately mapped to the physical network.

For the tree to be reliable, Sripanidkulchai et al. proposes another approach [16] which leverages the peer's time property: if the peers' lifetimes follow a distribution with a heavy tail, then the older peers are less likely to leave before the younger ones. This characteristic has also been observed in a number of statistical studies [20, 15].

This paper presents a new overlay construction algorithm, namely the *heap algorithm*, which leverages peers' properties in both bandwidth and time. It moves high-bandwidth and long-lived peers upward in the tree according to a metric called *service capability contribution* (SCC), which is defined as the product of a peer's outbound bandwidth (or simply called bandwidth) and its age in the overlay. This way the tree is gradually adjusted toward a layout which exhibits partial time order and partial bandwidth order, and consequently has the advantages of high reliability and a short tree. Besides the overlay-level operations, the heap algorithm uses a simple parent switching technique to re-map the parents to children so that the actual network delay can be minimized.

When designing the algorithm, the overlay adjusting cost (called the protocol overhead in this paper) is also an important consideration, since it has an immediate effect on the overlay optimization quality, and also reflects the overhead imposed on the end-users and is thus directly related to the QoS.

Simulations have been conducted to compare the performance of different algorithms. The results show the advantages of the heap algorithm over existing schemes in a variety of performance respects.

The rest of the paper is structured as follows. Section 2 introduces several existing methods; Section 3 gives a detailed description of the algorithm; Section 4 introduces the simulation methodology; Section 5 presents the experimental results and Section 6 concludes the paper.

2. Existing algorithms

A peer tree has its root at the content provider, and organizes all M peers in layers $L_0, L_1, \cdots, L_N$, with L_0 consisting of the root, L_1 consisting of all peers directly connected with L_0, and so on. Generally, $L_i (i >= 1)$ receives data from L_{i-1} and forwards it to L_{i+1}. Each peer has an *out-degree* $d \geq 0$, which is defined as the number of children it can serve simultaneously. A peer in the tree is also called a *node*.

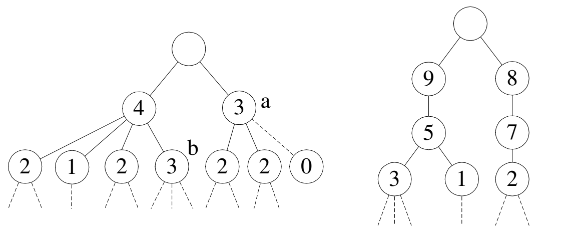
A central part of the tree management is the so-called *parent selection* strategy, which identifies a parent for a newly arriving peer. This strategy is crucial to shaping the tree. A selection of the more significant existing algorithms include the:

- **Random algorithm** that provides the the simplest approach [16] to parent selection. It randomly chooses a node with spare bandwidth capacity as the parent for a new peer. Clearly this algorithm is efficient and requires no global topological knowledge, but it results in a large tree depth and thus performs badly in almost all other performance respects.

- **High-bandwidth-first algorithm** [7] that places the peers from high to low layers in a non-increasing order of outbound bandwidths, that is, peers do not have more bandwidth capacity than any peer higher up in the tree. See Figure 1 (a) for an example. This algorithm allows later arriving peers to preempt the positions of existing peers with smaller bandwidths. This approach can achieve a minimum tree depth, but needs frequent disconnections and reconnections between peers to maintain such a globally ordered layout. For example, if node *a* in Figure 1 (a) leaves, then node *b* should be moved to node *a*'s position, which further forces all of node *b*'s children rejoin the tree. This recursive rejoin imposes very high overheads on the peers and is therefore impractical for real implementations. The overhead of disconnections and reconnections for maintenance purposes is termed the *protocol* *overhead*, which should be differentiated from service interruption since the connection teardowns and re-establishments can be performed in a coordinated manner and therefore avoid unexpected breaks in the streaming.

- **Minimum depth algorithm** obtains a tradeoff between simplicity and high overheads [7, 12, 16]. It searches from the tree root downward to the leaf layer to identify a parent with spare bandwidth capacity for a new peer. A variant of this approach is also proposed [11] so as to reduce the reliance on an understanding of the global overlay topology; this algorithm combines the heuristics of the minimum depth algorithm with some randomness, i.e., it first selects a number of peers randomly from the overlay and then performs the minimum depth algorithm.

- **Longest-first algorithm** [16] is intended to minimize service interruptions incurred by the departure of peers. It selects the longest-lived peer as the new peer's parent; the intuition behind this is that when the peers' lifetime follows a heavy-tailed distribution, the older peers generally remain longer than younger peers. This approach has been verified by the experimentation found in [16], the algorithm does not however guarantee that an older peer can always be identified.



(a) Bandwidth-ordered tree (b) Time-ordered tree

Figure 1. Examples of the bandwidth-ordered and time-ordered trees. The numbers in (a) and (b) represent the peers' outbound bandwidths and ages, respectively.

Of these algorithms, the high-bandwidth-first algorithm and the random algorithm achieve optimal tree depth and protocol overhead, respectively. The longest-first algorithm can be easily extended to generate a more reliable tree by placing the peers in order of arrival time (or ages) order, just as in the bandwidth ordering performed by the high-bandwidth-first algorithm. Figure 1 (b) gives an example of this type of tree. Clearly, the time ordering may result in a tall tree because it arranges the peers regardless of the peers' bandwidth properties, which themselves determine the tree shape. Moreover, this approach requires position adjusting when peers rejoin the tree after failures occur, and thus incurs higher protocol overheads. Hereinafter, the extended longest-first algorithm is termed the *time-ordered algorithm*, and a tree constructed by such an algorithm is termed a *time-ordered tree*. Likewise, the high-bandwidth-first algorithm is termed the *bandwidth-ordered algorithm* which builds a *bandwidth-ordered tree*.

While the bandwidth-ordered tree achieves a short tree which helps minimize service delay, it is unclear how tree reliability can be maximized: on the one hand, the short tree reduces the average number of peers affected by a failed node, while on the other hand, the time-ordered tree, at the expense of a large depth, enhances the reliability of an arbitrary top-down tree path. The main driver of this research is the following: Is it possible to construct a peer tree that achieves time ordering to some degree, while attaining the characteristics of a short tree; that is, can reliability and service delay can be improved at the same time?

3. The Heap Algorithm

This section describes the proposed heap-based approach. Its performance implications are also discussed qualitatively.

3.1. Fundamental approach

The heap algorithm uses the same strategies for peer joining and leaving as the minimum-depth algorithm. The only difference lies in the sift-up procedure during the normal streaming process. The criteria guiding the sift-up procedure is a metric $SCC = B \times T$, where B is the outbound bandwidth of a peer and T is its age. As such, SCC can be alternatively interpreted as the volume of media data one peer has helped to (or can) forward, and thus can be regarded as its "service capacity contribution" to the peer community. The basis of the algorithm is to move peers with large SCC's higher in the tree so that better service quality (less service interruptions and possibly smaller service delay) can be offered to these peers. This has an interesting result: since either a large bandwidth or a long service time helps to increase SCC, a peer can be encouraged to contribute more bandwidth resource or longer service time as a trade for service quality. From the user perspective, this forms an incentive mechanism that encourages cooperation among peers and helps increase overall system resources. Note that the use of a dynamic metric combining both bandwidth and time properties differentiates this mechanism from other incentive schemes [4, 10], which themselves usually consider only a static metric such as bandwidth.

3.2. The sift-up operation

The root is pre-assigned an infinite SCC, and always remains at the top of the tree. When a peer enters the network, its SCC is 0, and it will be placed using the same join operation as in the minimum-depth algorithm. In most cases, the high layers are occupied and the new peer becomes a leaf node. As time continues, the SCC increases at a rate proportional to its bandwidth. If its bandwidth is larger than its parent, then there must exist some time in the future when its SCC exceeds its parent. At that time the algorithm will exchange the roles of these two nodes. Figure 2 gives an example of this operation.

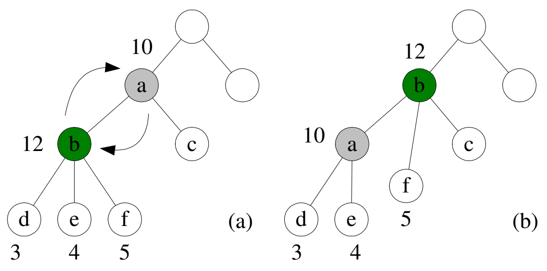


Figure 2. Illustration of the sift-up operation. (a) Before sift-up; (b) After sift-up. The numbers beside the nodes represent the SCCs.

In Figure 2 (a), node a's SCC is 10 and has an out-degree of 2; node b has an SCC of 12 and an out-degree of 3. Node b is therefore moved up to become the parent and node a is moved down to become the child. Now that node a can support only two of the three nodes d, e, f, one child must be assigned a new parent. The algorithm chooses f, the node with the largest SCC and reconnects to node b, which now has a spare out-degree. Node f is promoted because it has contributed the largest "service capacity" among all its siblings.

The sift-up is performed periodically over all peers. At set time intervals, the algorithm scans from the leaf layer to the first layer and updates the SCCs of all peers. At the same time, it checks if a node has a smaller SCC than one of its children. If so, it picks the child with the largest SCC and compares its own bandwidth with that child. If the child's bandwidth is larger, the sift-up will be performed between these two nodes. The bandwidth comparing avoids unnecessary sift-up since if the child has a smaller bandwidth, the SCC will eventually be exceeded by the parent, and it will ultimately be placed below the parent. The sift-up operation is illustrated in Algorithm 1.

Algorithm 1 Sift-up

```
1:  for $i = N$ to $i = 1$ do
2:    for all $P(j)$ in $L(i)$ do
3:      $Scc(j) \leftarrow B(i) \times A(i)$ {update SCC}
4:      $c \leftarrow$ the first child of $P(j)$
        {find the child with maximum SCC}
5:      for all $P(k)$ that is $P(j)$'s child do
6:        if $Scc(k) > Scc(c)$ then
7:          $c \leftarrow k$
8:        end if
9:      end for
        {sift-up the child peer $c$}
10:     if $Scc(c) > Scc(j)$ and $B(c) > B(j)$ then
11:       for $r = 1$ to $r = D(j)$ do
12:         $s \leftarrow$ child of $P(c)$ with minimum $SCC$
13:         $P(s).parent \leftarrow P(j)$
14:         remove $P(s)$ from $P(c)$'s children list
15:       end for
16:       $grandp \leftarrow P(j).parent$
17:       $P(grandp).child \leftarrow P(c)$
18:       $P(c).parent \leftarrow P(grandp)$
19:       $P(c).child \leftarrow P(j)$
20:       $P(j).parent \leftarrow P(c)$
21:     end if
22:   end for
23: end for
```

With the sift-up operations, the tree nodes will be placed in the tree from the high to low layers in decreasing order of SCCs. This ordering process is analogous to the sift-up operation in the conventional Heap Sort algorithm, and thus we name this new approach the "heap algorithm".

The algorithm moves peers up the tree in a gradual manner. This accounts for the fact that many peers may leave within a short time after their arrival [20, 17], resulting in a large number of service interruptions if they are placed high in the tree upon arrival. In contrast, placing a new peer at the leaf layer first and then adjusting its position according to its behavior can reduce this risk. The longer a peer stays in the network, the safer it is to be moved up the tree.

The sift-up procedure requires a per-node operation which involves an updating of the SCC, and potentially a peer exchange. The peer exchange requires $\overline{d}$ time, where $\overline{d}$ is the average out-degree of the peers. So each pass of the sift-up operation requires $O(M)$ time.

3.3. Topology-aware delay optimization

A short overlay path does not necessarily mean a short network delay due to the mismatching between the logical overlay network and the physical underlying network [9]. The heap algorithm addresses this problem in two ways. First, when a peer initially joins the network, the algorithm provides a number of candidate parents, of which the nearest will be chosen. Second, for peers that are already in the network, the algorithm uses a parent switching technique to dynamically re-connect the peers so that the average network delay between the peers remains optimal.

Parent switching is performed between any two consecutive layers, (e.g., L_k and L_{k+1}). An example is given in Figure 3: due to changes in the underlying network, the optimized mapping between the three pairs of peers in (a) is transformed to the mapping in (b) to minimize the average delay. It can be seen that this combinatorial optimization problem is NP-hard and when the number of peers in each layer is large (e.g. 2000), it can only be resolved using some approximate algorithms.

The heap algorithm assigns to each peer upon arrival an *optimization agent*, which is selected from the peers that have existed for a relatively long time and are thought to be stable. During the streaming service, peers exchange neighbors information between each other and periodically measure the network distances (in terms of delay) between themselves and other peers, including their immediate neighbors, and the neighbors' neighbors, and so on. These data are reported to their optimization agent, which periodically computes for a good solution for the peer mapping problem using a genetic algorithm. The agent does not necessarily compute for all its associated peers, instead it randomly selects a subset of peers, which form an *optimization group*. A limited population size in the genetic algorithm allows a good solution to be obtained quickly, and thus is more suitable for a dynamic environment. When a solution is obtained, the agent coordinates the peers to adjust their connections. If an agent leaves, its associated peers simply request from the server for a new agent. Figure 3 (c) illustrates the optimization agent and grouping.

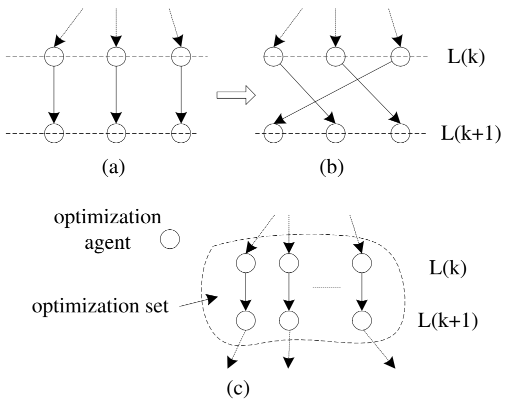


Figure 3. Topology-aware delay optimization.

The approach of switching trees has also been studied in [8, 2, 9]. While these studies make use of fully distributed optimization schemes, the heap algorithm uses a hybrid approach: a subset of peers are optimized by a single agent, and there are multiple such agents in the network. This mechanism has the advantage of being simple to implement. For example, Banerjee et al. define five deterministic local transformations and one probabilistic transformation [2], which make the protocol much more complicated. The switch-tree in [8] also defines multiple transformations. These distributed algorithms generally have a slower convergence speed than a centralized scheme, which means they may require more reconnections between peers than the proposed hybrid approach.

3.4. Discussion

Accurate bandwidth estimates are critical for the heap algorithm and they should not entirely rely on the users' settings. In the heap algorithm, the user-advertised bandwidth is only taken as an upper bound; actual bandwidth estimates are derived from the active end-to-end measurements as described in [5]. When a peer joins, its outbound bandwidth is set to zero, and an active measurement is launched between itself and another peer in the leaf layer. The measured bandwidth is then used in the calculation of SCC. To keep the estimate up to date, the bandwidth is measured periodically for peers whose bandwidths have not been fully utilized. The techniques of choosing the end host to transfer testing data, smoothing estimation and estimate discretization all follow the methods introduced in [5].

A peer tree resulting from the sift-up operation integrates the characteristics of both the bandwidth-ordered and time-ordered tree, since the peers are adjusted with a metric that mixes bandwidth and time properties, and those peers at the higher layers either possess high-bandwidths or long lifetimes, or both. As a hybrid of the two types of baseline tree, the new tree is expected to inherit their merits in both tree depth and tree reliability.

4. Simulation methodology

An event-driven simulator has been developed to study the performance of the different algorithms. The following five algorithms are implemented:

- Minimum-depth algorithm: This algorithm follows that in [16, 12, 7], but with a minor modification – when a layer that can support a new peer is found, the new peer chooses the nearest peer in terms of network delay (from up to 200 peers) in that layer as its parent. Since in practise a tree hierarchy may have thousands of peers in a layer, imposing a limit to the number of candidates would be more practical for implementation;

- Longest-first algorithm: This follows the scheme presented in [16]. When a new peer chooses its parent from the highest possible layer, it always chooses the oldest peer (from up to 200 peers) in that layer.

- *Relaxed bandwidth-ordered algorithm* and *Relaxed time-ordered algorithm*: These are two variants of the bandwidth-ordered and time-ordered algorithms as introduced in Section 3. The (strict) bandwidth-ordered and time-ordered trees are found to have a extremely high protocol cost, which makes them unacceptable in practise and only of theoretical value. Therefore, a modification is made to make the compared scenarios more realistic – when a peer joins/rejoins the tree, it always searches from the high to low layers to see if there is a smaller-bandwidth or younger peer, and if so, the identified peer is replaced with the new one. The evicted peer, and possibly together with some of its children in the case of time ordering, are forced to rejoin the tree. This results in bandwidth/time ordering locally within each layer and among parents and children, but not in a strict hierarchical structure; that is, a peer may have a smaller bandwidth/age than another non-child peer in the next layer. Since they still follow the basic ideas of bandwidth/time ordering, they are used for performance comparisons.

- Heap algorithm: This is implemented as introduced in Section 3, but with the parent switching disabled. It should be noted that such a technique can be applied to any of the other algorithms previously documented. Disabling this component helps to reveal the performance of the proposed algorithm in its "naive" form, and also avoids any bias in comparison with other algorithms.

The GT-ITM transit-stub model [21] is used to generate an underlying network topology consisting of 15600 nodes. Link delays between two transit nodes, transit nodes and stub nodes, and two stub nodes are chosen uniformly between $[15, 25]$ ms, $[5, 9]$ ms and $[2, 4]$ ms, respectively. Of all the 15360 stub nodes, a fraction of them are randomly selected to participate in the peer tree. The server's location is fixed at a randomly chosen stub node.

In all simulations, the root node's bandwidth is set to 100; the peers' outbound bandwidths follow a Bounded Pareto distribution[1] with shape, lower bound and upper bound parameters set to 1.2, 0.5 and 100 respectively (denoted by BP(1.2, 0.5, 100)), with which 55.5% of the peers have out-degrees less than 1 and are therefore termed "free-riders"; the peers' lifetimes follow a lognormal distribution with the μ (location parameter) and σ (shape parameter) set to 6.0 and 2.0 respectively (denoted by LN(6.0, 2.0)), which are chosen according to the findings in [20].

The simulation considers different network scales in terms of the average number of peers M in a steady state. According to *Little's Law*, the peer arrival rate λ is determined from M divided by the mean value of LN(6.0, 2.0). For the heap algorithm, the sift-up operation is performed every time 200 new peers join by default. Other selections of parameters are also tested and the results are found to be consistent.

5. Performance evaluation

This section presents and discusses the performance results with respect to service reliability, service delay, protocol overhead and the impact of sift-up frequency on tree depths.

[1]Previous studies [15, 16, 14] have shown that the bandwidths of peers exhibit characteristics of heavy-tailed distributions, a typical example of which is the Pareto distribution. Considering the practical limits of possible bandwidth values, a bounded Pareto distribution is used to model the peers' bandwidths.

5.1. Service reliability

Service reliability is measured by the average number of service interruptions experienced by a single peer during its lifetime in the steady state of a tree. The experiments consider the extreme case in which every peer departs abruptly without notification to others, and hence results in a service interruption on each of its descendants. This metric reflects the stability of a tree in the most uncooperative and dynamic environment.

Figure 4 compares the performance of the five algorithms under different network sizes.

As expected, the minimum-depth algorithm performs the worst in most cases, because it is designed completely blind of reliability. The longest-first algorithm has very limited improvement over the minimum-depth algorithm, as it operates in a very conservative way when ordering the peers' times.

It is surprising to see that the relaxed time-ordered algorithm performs worse than the heap algorithm. There are two reasons for this: first, the heap algorithm also achieves a partial time order in the process of sift-ups (the older peers are gradually moved upward in the tree), and consequently benefits from this in terms of reliability; the second reason is that the heap algorithm builds a much shorter tree than the relaxed time-ordered algorithm, which means that a failed node generally introduces fewer service interruptions to its descendants.

Therefore, with the advantages of both bandwidth-ordering and time-ordering, the heap algorithm appears to be a scheme that produces the most reliable tree among all the algorithms examined.

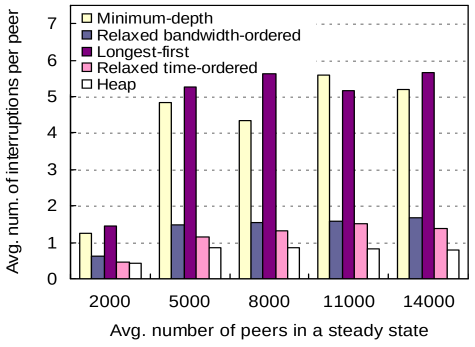


Figure 4. Comparison of reliability.

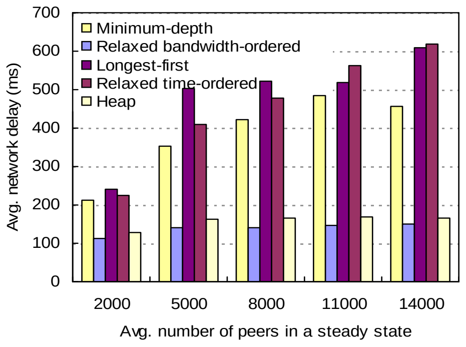


Figure 5. Comparison of service delays.

5.2. Service delay

The metric *average service delay* measures the actual physical network delay from the root to the peers. Figure 5 plots the results obtained under different network sizes. All the values are averages over a certain number of samples in a steady network state. It can be seen that the heap algorithm significantly outperforms all other algorithms with the exception of the relaxed bandwidth-ordered algorithm. This shows how bandwidth-ordering benefits the tree depth, even though it is only implicitly and partially realized.

Compared with the relaxed bandwidth-ordered tree, the heap algorithm has a small increase of 10-15%. This is because the heap algorithm optimizes the layout in a more confined space (only along the child-parent paths regardless of the bandwidth order between siblings), and hence yields a more sub-optimal bandwidth layout.

5.3. Protocol overhead

Both bandwidth ordering and time ordering require re-establishment of connections between certain peers to optimize the layout of the tree, thus introducing a protocol overhead. This overhead is measured in the average number of re-connections imposed on a single peer during its lifetime. Figure 6 compares the protocol overhead of the five algorithms. Note that the minimum-depth algorithm and the longest-first algorithm do not impose any protocol overheads at all; they are plotted in the figure with small values for convenience of observation.

The results show that the relaxed time-ordered algorithm yields the highest overhead, and the heap algorithm is better the relaxed bandwidth-ordered algorithm. Besides which, the relaxed bandwidth-ordered algorithm and the heap algorithm both require less

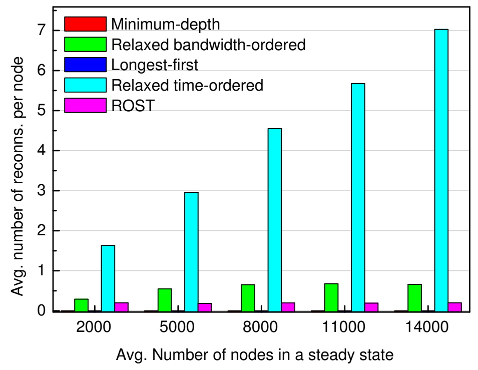


Figure 6. Comparison of protocol overheads.

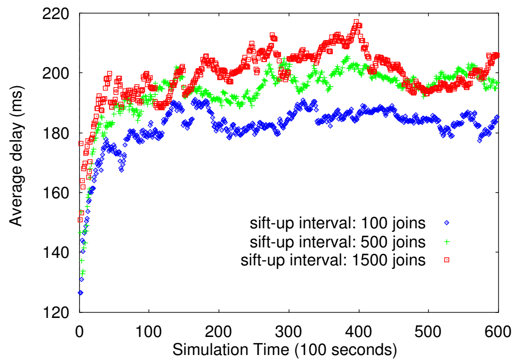


Figure 7. Average service delays changing over time under different sift-up frequencies.

than one reconnection for a single peer during its lifetime. This should be at an acceptable level for a practical system.

5.4. Impact of sift-up frequency

Intuitively, the more frequently the sift-up operations are performed, the more chance there is for the tree to be optimized, and consequently the higher overhead on the tree manager. To show how the sift-up frequency impacts on the service delay, Figure 7 plots the average network delays changing over a time interval of 16.7 hours with different sift-up frequencies. The network is fixed at 10,000 peers. The sift-up interval is measured by the number of newly joining peers. It can be seen that a higher sift-up frequency achieves smaller average delays. The protocol overhead corresponding to three intervals, 100 joins, 500 joins and 1500 joins, are 0.069, 0.17 and 1.338, respectively. This reveals the trade-off between the overlay performance and the required protocol overhead.

6. Conclusions

This paper presents an analysis for the performance of overlays constructed by several algorithms for P2P live media streaming. Three performance criteria are considered: (1) service delay; (2) service reliability and (3) protocol overhead. Two principles of peer placement, bandwidth ordering and time ordering, are discussed and their impact on different aspects of the overlay's performance are analyzed.

Based on this, a new algorithm called the *heap algorithm* is devised with the objective of taking advantage of both bandwidth ordering and time ordering. It adjusts peers in the tree during the normal streaming process according to a metric that combines both bandwidth and time properties of a peer, and employs a technique to optimize the mapping of overlay connections to physical network connections so as to minimize the actual network delay. Simulations show that the heap algorithm achieves superior comprehensive performance in comparison with existing algorithms.

REFERENCES

[1] S. Banerjee, B. Bhattacharjee, C. Kommareddy. Scalable Application Layer Multicast. *Proc. of ACM SIGCOMM 2002*, August 2002

[2] S. Banerjee, C. Kommareddy, K. Kar, S. Bhattacharjee, and S. Khuller. Construction of an efficient overlay multicast infrastructure for real-time applications. *Proc. of IEEE INFOCOM, 2003*.

[3] Y. Chawathe. Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service. *Ph.D. Thesis, University of California, Berkeley*, Dec. 2000.

[4] 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. *Proc. of USENIX 2004 Annual Technical Conference*.

[5] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the Internet using an overlay multicast architecture. *Proc. of ACM SIGCOMM 2001*.

[6] Y. Chu, S. Rao, and H. Zhang. A Case for End System Multicast. *Proc. of ACM SIGMETRICS*, June 2000.

[7] M. Guo, M. Ammar. Scalable live video streaming to cooperative clients using time shifting and video patching. *Proc. of IEEE INFOCOM 2004*.

[8] D. Helder and S. Jamin. End-host Multicast Communication Using Switch-tree Protocols. In *Proc. of Internation Conference on Global and Peer-to-Peer Computing on Large Scale Distributed Systems, 2002*.

[9] Y. Liu, Z. Zhuang, Li Xiao. A Distributed Approach to Solving Overlay Mismatching Problem. In *Proc. of 24th International Conference on Distributed Computing Systems (ICDCS'04)*

[10] Wei Tsang Ooi. Dagster: contributor-aware end-host multicast for media streaming in heterogeneous environment. *Proc. of Multimedia Computing and Networking (MMCN)*, 2005.

[11] Venkata N. Padmanabhan, Helen J. Wang, Philip A. Chou. Resilient Peer-to-Peer Streaming. *11th IEEE International Conference on Network Protocols (ICNP)*, 2003.

[12] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai. Distributing Streaming Media Content Using Cooperative Networking. *ACM NOSSDAV*, May 2002.

[13] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An Application Level Multicast Infrastructure. In Proc. of *3rd Usenix Symposium on Internet Technologies and Systems (USITS)*, March 2001.

[14] S. Saroiu, P. Gummadi and S. Gribble A Measurement Study of Peer-to-Peer File Sharing Systems. *Proc. of Multimedia Computing and Networking (MMCN)*, 2002.

[15] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. *IEEE/ACM Trans. on Networking.* Vol. 12, No. 2, April 2004.

[16] K. Sripanidkulchai, A. Ganjam, B. Maggs and H. Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. *Proc. of ACM SIGCOMM*, 2004, Portland, Oregon, USA.

[17] K. Sripanidkulchai, B. Maggs and H. Zhang An analysis of live streaming workloads on the Internet. *Proc. of the 4th ACM SIGCOMM IMC*, Oct., 2004. Italy.

[18] G. Tan, S. A. Jarvis, D. P. Spooner and G. R. Nudd. On Efficient and Robust Overlay Construction for Large-scale P2P Live Media Streaming. *TR-2005-02*, Dept. of Computer Science, University of Warwick, UK.

[19] D. A. Tran, K. A. Hua, and T. T. Do. A peer-to-peer architecture for media streaming. *IEEE Journal on Selected Areas in Communications (JSAC), Special Issue on Recent Advances in Service Overlay Networks*. 22, Jan. 2004.

[20] E. Veloso, V. Almeida, W. Meira, A. Bestavros, and S. Jin. A Hierarchical Characterization of A Live Streaming Media Workload. *IEEE/ACM Trans. on Networking*, 12(5), 2004.

[21] E. W. Zegura, K. Calvert and S. Bhattacharjee. How to Model an Internetwork. *Proc. of IEEE INFOCOM '96*, San Francisco, CA.