

Performance Analysis and Improvement of Overlay Construction for Peer-to-Peer Live Streaming

Guang Tan

Stephen A. Jarvis

Xinuo Chen

Daniel P. Spooner

Department of Computer Science

University of Warwick

Coventry, CV4 7AL, United Kingdom

gtan@dcs.warwick.ac.uk

For single-source, single-tree-based peer-to-peer live media streaming, it is generally believed that a short (and wide) tree has a good comprehensive performance in terms of reliability and service delay. While the short tree directly benefits delay optimization, it is unclear whether such a structure maximizes tree reliability, which is sometimes more critical for a streaming Internet service. This article studies several prevalent overlay construction algorithms from the aspects of (1) service reliability, (2) service delay, and (3) protocol overhead. Two types of peer layout, *bandwidth-ordered layout* and *time-ordered layout*, are identified, and their performance is evaluated. The analytical results show that, by appropriately placing peers according to their time properties, the tree achieves a much higher degree of reliability than the depth-optimized tree. This finding motivates the design of a *heap algorithm*, which aims for combining the strengths of both bandwidth ordering and time ordering. It dynamically moves peers between difference layers of the tree according to a simple metric and gradually adjusts the tree toward a layout partially ordered in time and partially ordered in bandwidth. In so doing, the tree has advantages in both service reliability and delay. Extensive simulations show that this new algorithm achieves better comprehensive performance than existing algorithms.

Keywords: Overlay construction, peer-to-peer, streaming, reliability

1. Introduction

The end-system multicast or application layer multicast (ALM) [1-3] has been shown to provide an effective architecture for large-scale Internet broadcast applications such as live media streaming [4-7]. In this architecture, the end systems 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 resource increases with the growth of the network. In practice, the “end system” that provides the multicast functionality can be a dedicated proxy-like host, a personal computer, or even a mobile device.

This article considers the single-source, single-tree-based live media streaming on such an architecture. The different types of end systems are not differentiated and therefore generically referred to as “peers.” The data delivery overlay is organized into a tree rooted at the content provider and with peers as its nodes.

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, 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 imposed on end users. Peers may need to reconnect to other peers for the purpose of overlay adjusting, which usually requires coordinations among multiple peers. In a distributed environment lacking time synchronization, such operations may require transient pauses of the streaming.

Given a set of peers with heterogeneous 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 is a good structure [6-9] 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 the 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. [7] propose another approach, which leverages the peers' time property: if the peers' lifetimes follow a distribution with a long tail, then the older peers are less likely to leave before the younger ones. This characteristic has been observed in some statistical studies [10, 11].

Hence a question arises: of these two approaches of enhancing tree reliability, which technique performs better? If the answer is the latter, then the problem becomes complex—on one hand, this approach provides good reliability, while on the other hand, it works totally blind of the peers' bandwidth properties, which determine the shape of the tree, and hence is very likely to build a tall tree, making it more difficult to maintain a small service delay.

To explore this, this article identifies two types of peer layout—namely, *bandwidth-ordered layout* and *time-ordered layout*, which represent the extreme cases for the two approaches. Stochastic models are established to analyze their performance characteristics, and the results reveal that, although resulting in a taller tree, placing peers in order of time brings much more benefit to tree reliability than placing them in order of bandwidth.

This finding motivates the design of a *heap algorithm*, which aims to combine the strengths of both bandwidth ordering and time ordering, and thus obtains performance gains in both tree reliability and service delay. To do so, the algorithm 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 that exhibits partial bandwidth order and partial time order and consequently has the advantages of high reliability and a short tree.

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

Extensive simulations have been conducted to compare the heap algorithm and various existing algorithms with respect to service reliability, service delay, and protocol overhead. The results show that the heap algorithm achieves the best comprehensive performance.

The next section introduces several existing algorithms, section 3 provides the tree model, section 4 analyzes the tree depth, section 5 discusses the tree reliability, section 6 describes the proposed algorithm, section 7 introduces the simulation methodology, section 8 presents the experimental results, and section 9 concludes the article with remarks on future work.

2. Existing Algorithms

Generally, a central part of tree management is the so-called *parent selection* strategy, which takes care of finding a parent for a newly arriving peer. This strategy is crucial to the shaping of the tree. Some commonly used methods are as follows.

The *random algorithm* [7] is the simplest method for 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.

The *high-bandwidth-first algorithm* [9] places the peers from high to low layers in a nonincreasing 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. It 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 makes all node b's children rejoin the tree. These recursive rejoins impose very high overhead on the peers and are therefore impractical for real implementations. The overhead of disconnections and reconnections for maintenance purposes is termed the *protocol overhead*, which is differentiated from the service interruption since the connection tear-downs and reestablishments can be performed in a coordinated manner and therefore avoid unexpected breaks in the streaming.

The *minimum depth algorithm* obtains a trade-off between simplicity and high overhead [6, 7, 9]. It searches from the tree's root downward to the leaf layer to look for a parent with spare bandwidth capacity for a new peer to join. If there are multiple choices, the new peer chooses the nearest parent in the underlying network (in terms of network delay). A variant of this algorithm [12] first selects a number of peers randomly from the overlay and then performs the minimum depth algorithm.

The *longest-first algorithm* [7] is intended to minimize the number of service interruptions incurred by the departures 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 tend to stay longer than younger peers. The

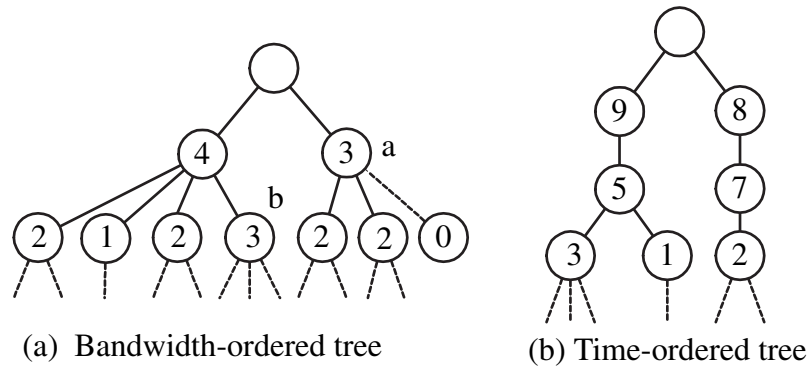


Figure 1. Examples of the bandwidth-ordered and time-ordered trees. The numbers in (a) and (b) represent the peers' outbound bandwidths and ages, respectively.

algorithm does not, however, guarantee that a peer can always find an older peer.

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 build a more reliable tree by placing the peers in a strict order of peers' arrival times (or ages), just as the bandwidth ordering performed by the high-bandwidth-first algorithm. Figure 1(b) gives an example for this type of tree. Clearly, the time ordering requires position adjusting when peers rejoin the tree after failures occur and thus incurs higher protocol overhead. Hereinafter, the extended longest-first algorithm is called the *time-ordered algorithm*, and a tree constructed by such an algorithm is called a *time-ordered tree*. Likewise, the high-bandwidth-first algorithm is called the *bandwidth-ordered algorithm*, which builds a *bandwidth-ordered tree*.

Thus, the bandwidth-ordered algorithm, the time-ordered algorithm, and the random algorithm form three baselines for streaming performance, from which many variants can be developed. This article focuses on the first two algorithms, with the emphasis on service reliability and service delay. The properties of the random algorithm can be inferred from the results obtained.

3. The Tree Model

Consider a peer tree organizing its M peers, or nodes, into N layers L_0, L_1, \dots, L_N , with L_0 consisting of only the root node, L_1 consisting of all peers directly connected with L_0 , and so on. Peers in layer i ($i \geq 1$) receive data from peers in L_{i-1} and forward them to L_{i+1} . Each peer's bandwidth is equal to the number of children it can serve simultaneously under a unit media streaming rate. Peers' bandwidths are assumed to be an independent identically distributed (i.i.d.) random variable (r.v.) $\xi_1, \xi_2, \dots, \xi_M$, with

common mean μ_ξ and variance σ_ξ . Let r.v.s X_k, B_k , and A_k denote the number of peers, the aggregate bandwidth of peers, and the average out-degree of peers in L_k , respectively. Peers enter the tree as a Poisson process of rate λ . Table 1 summarizes the notations used in the discussion.

4. Tree Depth Analysis

This section analyzes the average depths of the time-ordered and bandwidth-ordered trees. The tree depth only depends on the layout of peers' bandwidths. Since the time-ordered algorithm only considers peers' time property, which is independent of their bandwidths [13], it can be assumed to have a random bandwidth layout.

Table 1. Notation and variable definitions

Notation	Definition
r.v.	Random variable
p.d.f.	Probability density function
c.d.f.	Cumulative distribution function
i.i.d.	Independent, identically distributed
$\text{LN}(\mu, \sigma)$	Lognormal distribution with parameters μ and σ
$\text{BP}(\alpha, u, v)$	Bounded Pareto distribution with shape α , scale u , and upper bound v
M	Number of peers in the tree
L_k	The k th layer of the tree
N	r.v., number of layers in the tree
X_k	r.v., number of peers in L_k
B_k	r.v., aggregate bandwidth of all peers in L_k
A_k	r.v., average bandwidth of peers in L_k
C	Constant, bandwidth of the root node
ξ_i	r.v., bandwidth of peer i
λ	Arrival rate of peers

4.1 The Time-Ordered Tree

Under the assumptions of the tree model, the number of peers at layer $i + 1$ is determined by the aggregate bandwidth provided by layer i , which in turn is equal to the sum of bandwidths of all peers in layer i . Mathematically, $X_{k+1} = B_k = \sum_{j=1}^{X_k} \xi_j$. It is natural to describe the process $\{X_k, k = 1, 2, \dots\}$ as a *branching process* [14], which is a special type of Markov chain. According to the properties of the branching process, the mean and variance of X_k can be determined recursively as $E[X_{k+1}] = \mu_\xi E[X_k]$, and $Var[X_{k+1}] = E[X_k]\sigma_\xi^2 + \mu_\xi^2 Var[X_k]$. With the initial conditions that $E[X_1] = C$ and $Var[X_1] = 0$, the recurrence can be resolved as

$$E[X_k] = C\mu_\xi^{k-1}, \tag{1}$$

and

$$Var[X_k] = \begin{cases} kC\sigma_\xi^2, & \text{if } \mu_\xi = 1; \\ \frac{(\mu_\xi^{k-1} - 1)\mu_\xi^{k-2}C}{\mu_\xi - 1}\sigma_\xi^2 & \text{if } \mu_\xi \neq 1. \end{cases} \tag{2}$$

By equation (1), the total number of peers in a tree with N layers has a mean of $(\mu_\xi^N - 1)C/(\mu_\xi - 1)$. Now define \bar{N} as the minimum integer that satisfies

$$\frac{\mu_\xi^{\bar{N}} - 1}{\mu_\xi - 1}C \geq M,$$

which gives

$$\bar{N} = \left\lceil \frac{\ln[M(\mu_\xi - 1)/C + 1]}{\ln \mu_\xi} \right\rceil, \tag{3}$$

then \bar{N} is the minimum depth with which a tree has a mean number of nodes no less than M . In other words, a depth of \bar{N} is the lowest requirement for a tree to accommodate M peer nodes on average (over multiple tree instances). This definition of “tree depth” will be used in the rest of the article.

4.2 The Bandwidth-Ordered Tree

By the definition of the bandwidth-ordered tree, the bandwidth values of the peers from the high to low layers in nonincreasing order are $b_1 \geq b_2 \geq \dots \geq b_M$. It is easy to prove:

THEOREM 1. A bandwidth-ordered tree has a minimum depth.

To compute the depth of a bandwidth-ordered tree, B_k again needs to be determined. In essence, this is a problem

of *order statistics* [15] and can be treated using related theories. However, when M is very large (generally of tens of thousands), an exact numerical solution for such a problem becomes infeasible. Hence, in the following, an alternative approach is used to approximate the average value of B_k and, furthermore, the tree depth, while the detailed distribution of B_k is ignored.

Suppose $\xi_1, \xi_2, \dots, \xi_M$ have a common continuous cumulative distribution function (c.d.f.) $F_\xi(y)$ on (u, v) ; then the approximation is conducted as follows. First consider B_1 , the sum of b_1, b_2, \dots, b_{X_1} . Since $F_\xi(u) = 0$ and $F_\xi(v) = 1$, there must exist some $z_1, u < z_1 < v$ such that $F_\xi(v) - F_\xi(z_1) = X_1/M$. In other words, there exists a range (z_1, v) , in which ξ_i ($1 \leq i \leq M$) falls with probability $p_1 = X_1/M$. If the number of ξ_i values that lie between (z_1, v) is denoted by Z_1 , then Z_1 follows a M -step Bernoulli distribution (M, p_1) . By the DeMoivre-Laplace theorem, when M is large, Z_1 approximately follows a normal distribution $N(Mp_1, Mp_1(1 - p_1))$. In view of the mean value of $Mp_1 = X_1$, one can approximate the range that covers b_1, b_2, \dots, b_{X_1} using (z_1, v) .¹ Given this range, theorem 2 offers an approximation of the average of b_1, b_2, \dots, b_{X_1} .

THEOREM 2. Let X_1, X_2, \dots, X_n be i.i.d. r.v.s with probability density function (p.d.f.) $f_X(x)$ and c.d.f. $F_X(x)$ is defined on (u, v) , which has finite mean and variance. Let $S = \{X_i \mid a < X_i < b\}$, where $u \leq a < b \leq v$, and $\bar{S} = \frac{1}{|S|} \sum_{X_i \in S} X_i$, and then as $n \rightarrow \infty$, with probability 1,

$$\bar{S} \rightarrow \frac{1}{F_X(b) - F_X(a)} \int_a^b f_X(x)x \, dx.$$

Proof. Refer to Appendix A.

Since \bar{S} converges to a certain value π as $n \rightarrow \infty$, $\bar{S}|S|$, the sum of all conditional samples, has a relative difference from $\pi|S|$ converging to 0 as $n \rightarrow \infty$.

Theorem 2 gives a reasonable approximation for the average of b_1, b_2, \dots, b_{X_1} when M is large. Let this average be A_1 ; B_1 is then approximately given by A_1X_1 . With $X_2 = B_1$, it is possible to compute A_2 as follows. Now choose range (z_2, z_1) such that $F_\xi(z_1) - F_\xi(z_2) = X_2/M$, and then this range is again used to approximate the interval that covers $b_{X_1+1}, b_{X_1+2}, \dots, b_{X_1+X_2}$. Using theorem 2, A_2 is obtained, leading to $B_2 \approx A_2X_2$. Continuing in this way, all X_i can be computed.

1. The larger the X_1 , the more accurate the approximation. Recall that $\sigma = \sqrt{Mp_1(1 - p_1)} \approx \sqrt{X_1}$, and Z_1 has a (minimum) 95% confidence interval of $[X_1 - 2\sqrt{X_1}, X_1 + 2\sqrt{X_1}]$. Normalized over μ , the interval length is $\frac{4}{\sqrt{X_1}}$, which is nontrivial for a small X_1 . Nevertheless, this approximation is used here as it gives an average-case profile of the problem with a much more concise form than the strict (and sometimes intractable) probabilistic solution.

Especially, if $F_\xi(y)$ is a bounded Pareto distribution on (u, v) , the following corollary holds.

COROLLARY 1. Let X_1, X_2, \dots, X_n be i.i.d. r.v.s with a bounded Pareto distribution whose p.d.f. is $f_X(x) = \frac{\alpha u^\alpha x^{-\alpha-1}}{1 - (u/v)^\alpha}$, $u \leq x \leq v$. Let $S = \{X_i \mid a < X_i < b\}$, where $u \leq a < b \leq v$, and $\bar{S} = \frac{1}{|S|} \sum_{X_i \in S} X_i$; then, as $n \rightarrow \infty$, with probability 1,

$$\bar{S} \rightarrow \frac{\alpha}{\alpha - 1} \frac{ab^\alpha - a^\alpha b}{b^\alpha - a^\alpha}.$$

Proof. Refer to Appendix B.

If ξ_i follows a bounded Pareto Distribution,² X_k ($k \geq 1$) can be computed as follows. Since $X_{k+1} = B_k$, which in turn can be determined from z_{k+1} and z_k by corollary 1, z_1, z_2, \dots need to be computed first. From the analysis above, $F_\xi(z_{k-1}) - F_\xi(z_k) = X_k/M$; that is,

$$\frac{(u/z_k)^\alpha - (u/z_{k-1})^\alpha}{1 - (u/v)^\alpha} = \frac{X_k}{M}.$$

Solving for z_k gives

$$z_k = \frac{u}{\sqrt[\alpha]{\frac{X_k}{M} \left[1 - \left(\frac{u}{v}\right)^\alpha \right] + \left(\frac{u}{z_{k-1}}\right)^\alpha}}. \quad (4)$$

By corollary 1,

$$A_k = \frac{\alpha}{\alpha - 1} \frac{z_{k-1}^{\alpha-1} - z_k^{\alpha-1}}{z_{k-1}^\alpha - z_k^\alpha} z_k z_{k-1}. \quad (5)$$

Furthermore,

$$X_{k+1} = B_k = A_k X_k. \quad (6)$$

Under the initial conditions $z_0 = v$ and $X_1 = C$, Equations (4) through (6) determine all X_k for $k \geq 1$.

With all X_k in hand, the depth of the tree is readily given by counting the minimum number of X_i whose sum is no less than M .

4.3 Tree Depth Comparison

Figure 2 gives the comparison of tree depths for the bandwidth-ordered and time-ordered trees. Peers' bandwidth follows a BP(1.2, 0.4, 100) or BP(1.2, 0.7, 100); $C = 400$. The generated bandwidth skewness among the peers is comparable to the findings from Sripanidkulchai,

2. Previous studies [7, 11, 16] have shown that the bandwidths of peers exhibit characteristics of heavy-tailed distributions, a typical example of which is Pareto distribution. Considering the practical limit of possible bandwidth values, a bounded Pareto distribution is used to model the peers' bandwidths.

Maggs, and Zhang [13] and Sen and Wang [11]. It can be seen that the bandwidth-ordered tree is 50% to 66% shorter than the time-ordered tree.

5. Service Reliability Analysis

Given the fact that the time-ordered tree is more than two times as high as the bandwidth-ordered tree, one question arises: will the time-ordering process overcome this disadvantage and ultimately result in a more reliable tree than the bandwidth-ordered tree? This section seeks to answer this question.

5.1 Reliability of Individual Peers

The lifetime of a peer in the overlay is a r.v. usually modeled with a lognormal distribution $LN(\mu, \sigma)$ [10, 13], where σ is the shape parameter and μ is the location parameter. Under this assumption, the lifetime p.d.f. is

$$g(t) = \frac{1}{\sqrt{2\pi} \sigma t} \exp\left[-\frac{(\ln t - \mu)^2}{2\sigma^2}\right], \quad t > 0, \quad (7)$$

and the c.d.f. is

$$G(t) = \Phi\left(\frac{\ln t - \mu}{\sigma}\right), \quad t > 0, \quad (8)$$

where Φ is the standard normal distribution function. Suppose that at some time point, peer i has stayed in the system for a period of t_i (having an age of t_i), then the *hazard rate* (or failure rate) [14] at t_i is

$$p_{t_i} = \frac{g(t_i)}{1 - G(t_i)}. \quad (9)$$

By the property of the hazard function of the lognormal distribution, p_{t_i} is increasing for a small value x and then decreasing. Therefore, for peers whose ages are larger than x , the older ones have smaller (instantaneous) probabilities to leave than the younger ones. When σ chooses a realistic value (e.g., 1.28 or 2.03 as in [10]), x becomes so small that a simple rule can be made to guide practical system design: the larger the t_i , the smaller the p_i . This rule has been used in the longest-first algorithm and will also be assumed in the following discussion.

5.2 Reliability of Peers in a Path

To show how path length affects the reliability of a peer tree, consider a root-to-leaf path consisting of m nodes—namely, $1, 2, \dots, m$. For any node i in the path, its departure will bring a service interrupt to all its descendants in the tree (not only on the path). Let $D(i)$ denote the number of all descendants of node i and R the number of descendants that encounter service interruptions introduced by all possible failures on the path, and then the *reliability index*

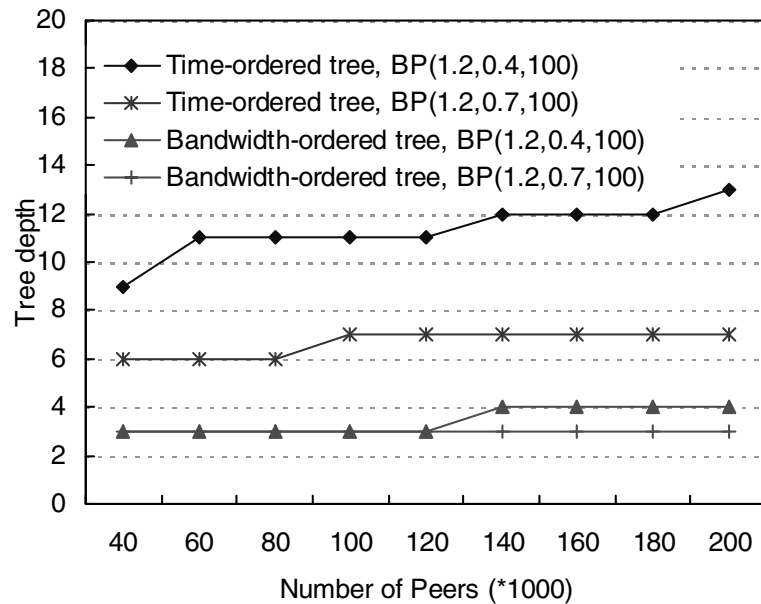


Figure 2. Analytical comparison of tree depths

of the root-to-leaf path, defined as the mean of R , is a function of $D(i)$ and the failure rates of all peers on the path, p_1, p_2, \dots, p_m . Formally,

$$\begin{aligned}
 & E[R(D(\cdot), p_1, p_2, \dots, p_m)] \\
 &= \sum_{k=1}^m \Pr\{\text{node } k \text{ departs while all node } l (l < k) \\
 &\quad \text{remain}\} \times D(k) \\
 &= \sum_{k=1}^m p_k D(k) \prod_{j=1}^{k-1} (1 - p_j) \tag{10}
 \end{aligned}$$

For brevity, $E[R(D(\cdot), p_1, p_2, \dots, p_m)]$ is also denoted by $E[R]$. Clearly, the smaller the $E[R]$, the more stable a tree. For a bandwidth-ordered tree, $D(i)$ can be computed from the average node bandwidth (out-degree) of layer L_i, L_{i+1}, \dots, L_N , which are given by equation (5), whereas for a time-ordered tree, $D(i)$ can be approximated using the total number of nonroot nodes in the k -ary subtree rooted at node i , where k is the mean value of the peers' bandwidth distribution. It is easy to see that $D(i)$ monotonically decreases over i , and it can be proved that

THEOREM 3. $E[R]$ is minimum when $p_1 \leq p_2 \leq \dots \leq p_m$.

Proof. Refer to Appendix C.

With the relationship between p_i and t_i , Theorem 3 can be equivalently stated as follows: when $t_1 \geq t_2 \geq \dots \geq t_m$, $E[R]$ is minimized. This justifies the intuition behind the longest-first algorithm. In the construction of a peer tree, placing the oldest peers on the highest layers of the tree maximizes the stability of a tree in terms of reliability index.

Substituting p_i with equation (9), we can rewrite $E[R]$ as

$$E[R] = \sum_{k=1}^m \frac{g(t_i)}{1 - G(t_i)} D(k) \prod_{j=1}^{k-1} \left[1 - \frac{g(t_i)}{1 - G(t_i)} \right]. \tag{11}$$

5.3 Reliability Comparison

Since the reliability index reflects the impact of a root-to-leaf path on the whole tree's reliability, it is used as a metric to evaluate the reliability of a tree. Consider again a root-to-leaf path in a tree of depth m . $E[R]$ is actually an r.v. governed by the joint distribution of i.i.d. r.v.s T_1, T_2, \dots, T_m , which represent the ages of the m peers. Thus, to evaluate $E[R]$, the distribution of T_i for $1 \leq i \leq m$ needs to be determined first. As assumed before, the peers enter the network as a Poisson process of rate λ . For a specific path, however, the initial parent assignments, departures, and rejoins of peers all introduce random effects to the path

structure; hence, for simplicity, it is assumed that the m peers arrive at the path also as a Poisson process, but with a rate τ . At the same time, peers leave the path according to a lifetime distribution.

Let $N(t)$ denote the point process formed by all peers on a specific path under the conditions that $t < T_{now}$ and $N(T_{now}) = m$, where T_{now} is an observation time point in a steady state. It can be shown that $N(t)$ is a nonhomogeneous Poisson process with rate function [17]

$$\lambda(t) = \tau \{1 - G(T_{now} - t)\}, 0 < t < T_{now}. \quad (12)$$

When $T_{now} \rightarrow \infty$, the mean rate $\int_0^{T_{now}} \tau \lambda(t) dt = \int_0^{\infty} \tau \lambda(t) dt = \tau E[T_i] = \tau e^{\mu + \sigma^2/2}$.

Knowing the characteristic of this process, we can readily obtain the distribution of peers entering times W_i , and hence T_i for $1 \leq i \leq m$. Based on this, the mean of reliability index $E[E[R]]$ can be computed. On $(0, \infty)$, a sequence of W_1, W_2, \dots, W_m is generated [14], and T_1, T_2, \dots, T_m are computed. The sequence of T_i is in random order for a bandwidth-ordered tree and in increasing order for a time-ordered tree. With this sequence, $E[R]$ can be obtained. An average over a large number of runs of such computation is expected to approach the actual value of $E[E[R]]$.

Figure 3 compares the reliability indices of the two kinds of tree in a steady state. The lifetime distribution parameters are chosen close to those reported in Veloso et al. [10], and the bandwidth distribution is the same as in section 4.3. All results of $E[E[R]]$ are averages over 100,000 instances of $E[R]$, after which the values keep quite stable.

It can be seen that a large difference exists between the reliability indices of the two kinds of tree: the bandwidth-ordered tree has a reliability index two to four times as high as that of the time-ordered tree. Even though the latter has a tree 57% to 73% deeper than the former, the effect of time ordering in the time-ordered tree quickly offsets the disadvantages of the large tree depth. The reason can be further attributed to the characteristics of the lognormal lifetime distribution. With a long distribution tail, the peers' ages exhibit great variability, and there are always a number of peers with very long lifetimes serving as stable parent nodes in the high layers of the tree. Since all short-lived and unstable peers are moved to the lower layers, their failures only affect few downstream peers, and thus the average service interruption rate is reduced.

Hence, the conclusions are as follows: in terms of reliability, the time-ordering process of the time-ordered algorithm brings benefits that far outweigh the loss brought by its large tree depth in comparison with the bandwidth-ordered tree.

Up to now, the bandwidth-ordered and time-ordered algorithms separately show their own strengths. A further question, then, is the following: is it possible to combine these two aspects in a new algorithm so that a tree can have both advantages of high reliability and small depth? This will be explored in the following sections.

6. The Heap Algorithm

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

6.1 Fundamental Idea

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 criterion 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 basic idea of the algorithm is to move peers with large SCCs higher in the tree so that better service quality (fewer 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 staying time helps to increase SCC, a peer can be stimulated to contribute more bandwidth resource or longer serving 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 [5, 18], which usually consider only a static metric such as bandwidth.

6.2 The Sift-Up Operation

The root is preassigned 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 goes by, the SCC increases in a rate proportional to its bandwidth. If its bandwidth is larger than its parent, then there must be a time point in the future when its SCC exceeds its parent. At that time, the algorithm will exchange the roles of these two nodes. Figure 4 gives an example of this operation.

In Figure 4(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. So node b is moved up to become the parent, and node a is moved down to become the child (Figure 4(b)). 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 one with the largest SCC to reconnect to node b , which now has a spare out-degree. Node f is promoted because it has contributed the most "service capacity" among all its siblings.

The sift-up is performed periodically over all peers. For every interval of a certain time, the algorithm scans from

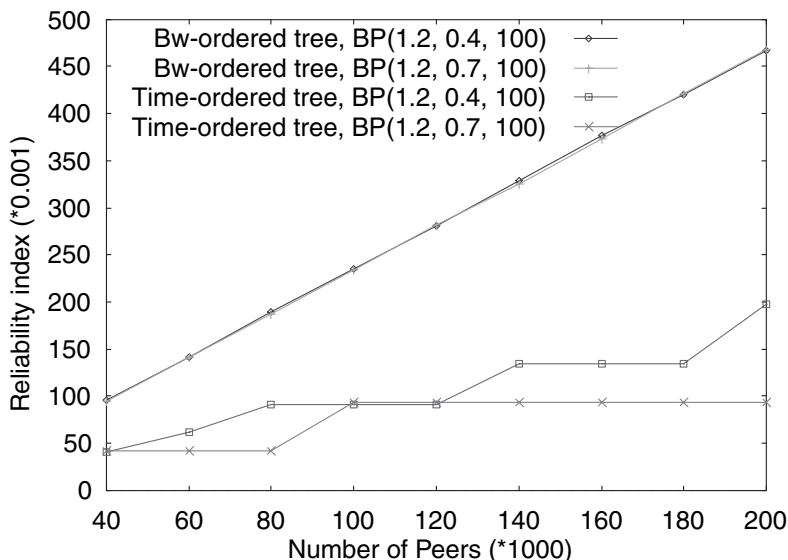


Figure 3. Analytical comparison of reliability indices. Peers' lifetime \sim LN(5.0, 2.0).

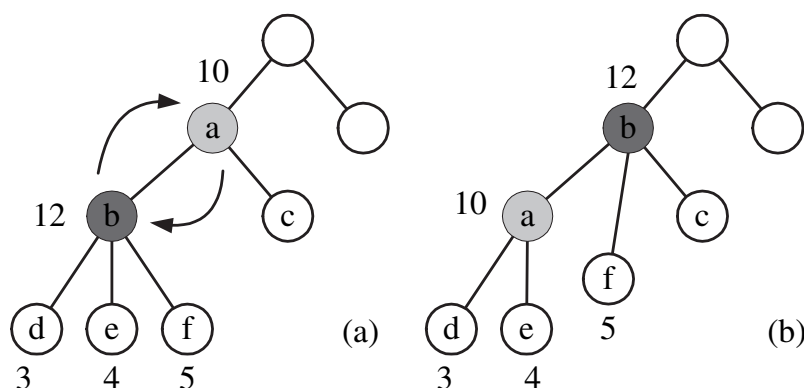


Figure 4. Illustration of the sift-up operation. (a) Before sift-up; (b) after sift-up. The numbers beside the nodes represent service capability contributions (SCCs).

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 no less than the parent's bandwidth, 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 in the future, and it will ultimately be placed below the parent.

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 has the name "heap algorithm."

The algorithm moves peers up the tree in a gradual manner. This considers the situation that many peers may leave within a short time after their arrival [10, 13], 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 that involves an updating of SCC and potentially a peer exchange. The peer exchange requires μ_{ξ} time, where μ_{ξ} is the average out-degree of the peers. So each pass of the sift-up operation requires $O(M)$ time.

6.3 Discussion

Accurate bandwidth estimates are critical for the heap algorithm. They should not entirely rely on the users' settings. In the heap algorithm, the user-advertised bandwidth is only taken as an upper bound; the actual bandwidth estimate is accomplished by the active end-to-end measurements, as described in Chu et al. [19]. 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 used up. The techniques of choosing the other end host to transfer the testing data, smoothing estimate, and estimate discretization all follow the methods introduced in Chu et al. [19].

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 possess high bandwidths, 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.

7. Simulation Methodology

This section presents the simulation methodology, including the various algorithms to be compared, the topology generation, and parameter selection.

7.1 Schemes Compared

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

- **Minimum-depth algorithm:** The basic idea follows that in other works [6, 7, 9], but with a minor modification: when a layer that can support a new peer is found, the new peer chooses the nearest one in terms of network delay from up to 200 peers in that layer as its parent. Since in practice, 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 Sripanidkulchai et al. [7]. When a new peer chooses its parent from the highest possible

layer, it always chooses the oldest one 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 2. The (strict) bandwidth-ordered and time-ordered trees are found to have protocol costs as high as $O(M)$ in terms of average number of reconnections for a single peer during its lifetime, which makes them unacceptable in practice and only of theoretical value. 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 found peer is replaced with the new one. The evicted peer, and possibly together with some of its children in the case of time ordering, is 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 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 6.

7.2 Topology Generation and Parameters Selection

The GT-ITM transit-stub model [20] is used to generate an underlying network topology consisting of 15,600 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 15,360 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; peers' (outbound) bandwidths follow BP(1.2, 0.5, 100), with which 55.5% of the peers have outbound bandwidths less than 1 and therefore cannot forward data to other peers (these are so-called free-riders); peers' lifetimes follow LN(6.0, 2.0). The simulation considers different network scales in terms of the average number of peers M in a steady state. According to *Little's law*, 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.

8. Performance Evaluation

This section presents and discusses the performance results with respect to service reliability, service delay,

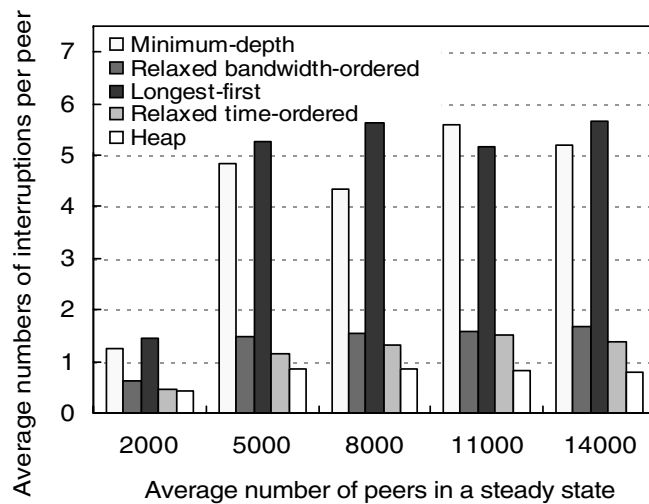


Figure 5. Comparison of reliability

protocol overhead, and the impact of sift-up frequency on tree depths.

8.1 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 5 compares the performance of the five algorithms under different network scales.

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.

The relaxed time-ordered algorithm has a better result than the relaxed bandwidth-ordered algorithm, but the improvement is not as significant as expected when compared with the reliability index results in Figure 3. To understand this, samples of peers' entering times along an arbitrary root-to-leaf path have been collected during the simulations. It is found that, while on some paths the sample values are distributed widely and unevenly as expected, on a large number of paths, they are clustered in a narrower range near the current time and clearly do not conform to a nonhomogeneous Poisson process. Since a low reliability index depends on the existence of a few long-lived peers on the higher layers, the resulting paths have much higher reliability than that of a strict time-ordered tree, where the

upper-layer peers always have smaller arrival times than lower-layer ones, and the time distribution along a root-to-leaf path is more likely to be widely distributed over $(0, \infty)$. The distortion of the time distribution is due to the relaxed rule of time ordering in the implementation. How to optimize this is left as a further research topic.

Even though the relaxed time-ordered algorithm performs far from optimally, it is surprising to see that it is still 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 other 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 turns out to be a scheme producing the most reliable tree among all the algorithms examined.

8.2 Service Delay

Three metrics are used to evaluate the effectiveness of various algorithms with respect to service delay. The *average overlay path length* is defined as the average number of hops from the root to all peers in the tree. Although it does not immediately translate to the underlying network delay, it reflects the shape of the overlay constructed and suggests some relationship between the overlay structure and the average network performance.

The *average service delay* measures the actual physical network delay from the root to the peers. The *average stretch* is the average of all peers' stretches, which is

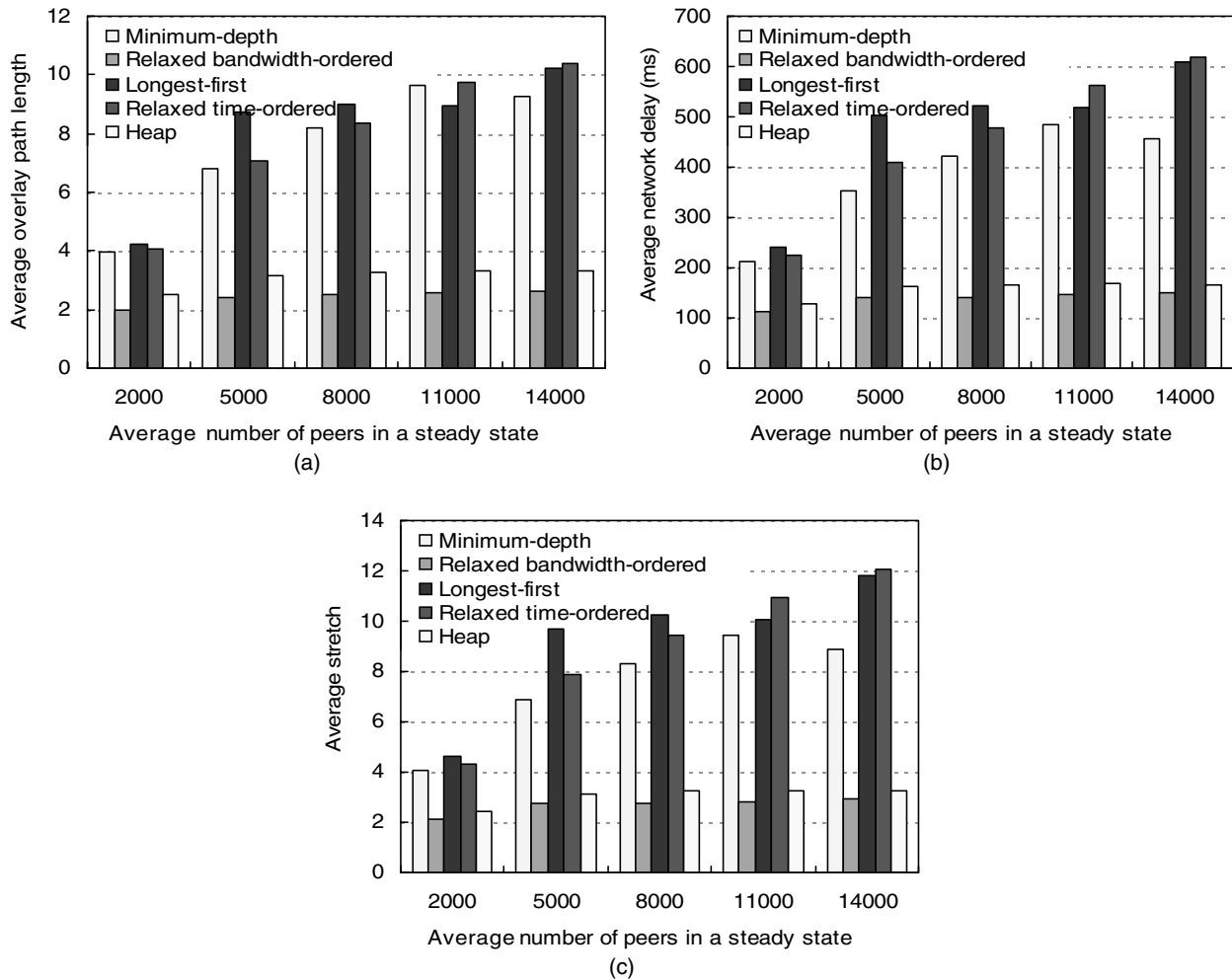


Figure 6. Comparison of service delays

defined as the ratio of the path delay from the server to a peer in the overlay to the delay along the direct unicast path in the underlying network [1, 3].

Figure 6 plots the results obtained under different network scales. 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 in terms of all three metrics except 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 in all metrics of 10% to 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 suboptimal bandwidth layout.

8.3 Protocol Overhead

Both bandwidth ordering and time ordering require reestablishment 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 reconnections imposed on a single peer during its lifetime. Figure 7 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 very small values only for convenience of observations.

The results show that the relaxed time-ordered algorithm yields the highest overhead, and the heap algorithm is comparable with the relaxed bandwidth-ordered algorithm. Besides which, the relaxed bandwidth-ordered algorithm and the heap algorithm both require fewer than two recon-

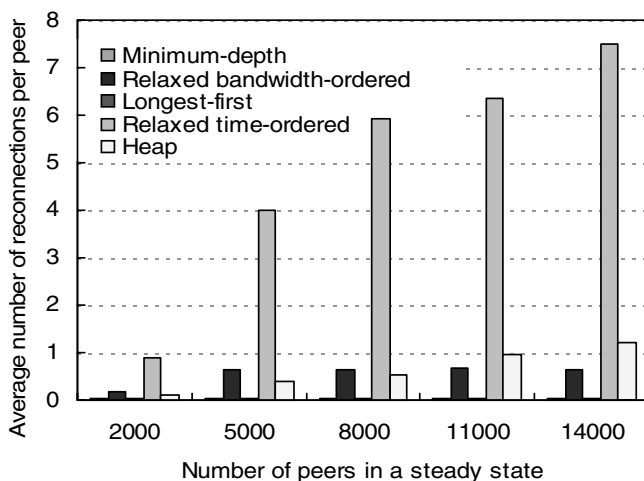


Figure 7. Comparison of protocol overheads

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

8.4 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 the overhead on the tree manager. To show how the sift-up frequency affects the service delay, Figure 8 plots the average network delays, changing over a time interval of 16.7 hours with different sift-up frequencies. The network scale 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 on average. The protocol overheads corresponding to the 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.

9. Conclusions and Future Work

This article presents a theoretic analysis for the performance of overlays constructed by several algorithms for P2P live media streaming. Three performance aspects 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 is evaluated.

Based on this, a new algorithm, called the heap algorithm, is devised with the objective to take 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.

Future work includes the development of a distributed version of the heap algorithm, which will allow the scheduling task to be relocated from the root to the distributed peers and thus improve system scalability.

10. Appendix A: Proof of Theorem 2

According to the property of conditional probability,

$$\begin{aligned}
 f_{X|a < X < b}(x) &= \frac{f_X(x)}{\Pr\{a < x < b\}} \\
 &= \frac{f_X(x)}{F_X(b) - F_X(a)}, a < x < b.
 \end{aligned}$$

Hence,

$$E[X | a < X < b] = \frac{1}{F_X(b) - F_X(a)} \int_a^b f_X(x)x dx.$$

For any X_i , the probability of falling in the range (a, b) is $F_X(b) - F_X(a)$, so by the law of large numbers, when $n \rightarrow \infty$, the number of elements in S , $|S| \rightarrow \infty$. By the strong law of large numbers, \bar{S} converges to $E[X | a < X < b]$

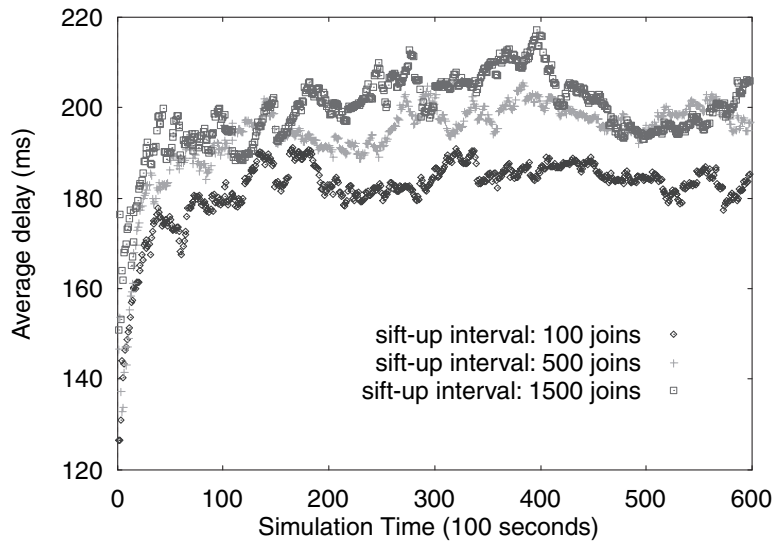


Figure 8. Average service delays changing over time under different sift-up frequencies

with probability 1 as $n \rightarrow \infty$; that is,

$$\lim_{n \rightarrow \infty} \Pr\{\bar{S} \rightarrow E[X \mid a < X < b]\} = 1.$$

This establishes the theorem.

11. Appendix B: Proof of Corollary 1

By the definition of $f_X(x)$,

$$F_X(x) = \frac{1 - (u/x)^\alpha}{1 - (u/v)^\alpha},$$

so

$$\frac{1}{F_X(b) - F_X(a)} = \frac{1 - (u/v)^\alpha}{(u/a)^\alpha - (u/b)^\alpha}.$$

Furthermore,

$$\int_a^b f_X(x)x \, dx = \frac{u^\alpha}{1 - (u/v)^\alpha} \frac{\alpha}{\alpha - 1} \left(\frac{1}{a^{\alpha-1}} - \frac{1}{b^{\alpha-1}} \right).$$

Hence, by theorem 2, as $n \rightarrow \infty$,

$$\begin{aligned} \bar{S} &\rightarrow \frac{1 - (u/v)^\alpha}{(u/a)^\alpha - (u/b)^\alpha} \frac{u^\alpha}{1 - (u/v)^\alpha} \frac{\alpha}{\alpha - 1} \\ &\quad \left(\frac{1}{a^{\alpha-1}} - \frac{1}{b^{\alpha-1}} \right) \\ &= \frac{\alpha}{\alpha - 1} \frac{ab^\alpha - a^\alpha b}{b^\alpha - a^\alpha} \end{aligned}$$

with probability 1.

12. Appendix C: Proof of Theorem 3

Consider the sequence $p_1, p_2, \dots, p_i, \dots, p_j, \dots, p_m$, in which $i < j$ and $p_i > p_j$. Let $\omega = \prod_{k=1}^{i-1} (1 - p_k)$, $E_0 = \sum_{k=1}^{i-1} p_k D(k) \prod_{j=1}^{k-1} (1 - p_j)$, and $E_1 = \sum_{k=j+1}^m p_k D(k) \prod_{j=1}^{k-1} (1 - p_j)$, and then equation (10) can be rewritten as

$$\begin{aligned} E[R] &= \\ &E_0 + E_1 + \omega p_i D(i) + (1 - p_i) \omega p_{i+1} D(i+1) + \dots \\ &+ (1 - p_i)(1 - p_{i+1}) \dots (1 - p_{j-2}) \omega p_{j-1} D(j-1) \\ &+ (1 - p_i)(1 - p_{i+1}) \dots (1 - p_{j-1}) \omega p_j D(j). \end{aligned}$$

Now exchange the positions of p_i and p_j in the sequence such that peer i is mapped to node j and peer j is mapped to node i , which gives

$$\begin{aligned} E'[R] &= \\ &E_0 + E_1 + \omega p_j D(i) + (1 - p_j) \omega p_{i+1} D(i-1) + \dots \\ &+ (1 - p_j)(1 - p_{i+1}) \dots (1 - p_{j-2}) \omega p_{j-1} D(j-1) \\ &+ (1 - p_j)(1 - p_{i+1}) \dots (1 - p_{j-1}) \omega p_i D(j). \end{aligned}$$

Consequently,

$$\begin{aligned} E[R] - E'[R] &= \\ &\omega(p_i - p_j) D(i) - \omega(p_i - p_j) [p_{i+1} D(i+1) \\ &+ \dots + (1 - p_{i+1}) \dots (1 - p_{j-2}) p_{j-1} D(j-1)] \\ &- \omega(p_i - p_j) (1 - p_{i+1}) \dots (1 - p_{j-1}) D(j) \\ &> \omega(p_i - p_j) D(i) - \omega(p_i - p_j) [p_{i+1} \end{aligned}$$

$$\begin{aligned}
& + \cdots + (1 - p_{i+1}) \cdots (1 - p_{j-2}) p_{j-1}] D(i) \\
& - \omega(p_i - p_j)(1 - p_{i+1}) \cdots (1 - p_{j-1}) D(j) \\
& = \omega(p_i - p_j)(1 - p_{i+1}) \cdots (1 - p_{j-1}) D(i) \\
& - \omega(p_i - p_j)(1 - p_{i+1}) \cdots (1 - p_{j-1}) D(j) \\
& = \omega(p_i - p_j)(1 - p_{i+1}) \cdots (1 - p_{j-1}) [D(i) - D(j)] \\
& > 0.
\end{aligned}$$

This indicates that for any pair p_i and p_j in the sequence, if $i < j$ and $p_i > p_j$, then by exchanging their positions such that $p_i < p_j$, the $E[R]$ can always be changed smaller. This process can continue until $p_1 \leq p_2 \leq \cdots \leq p_m$, when no such pairs can be found, and $E[R]$ reaches its minimum.

13. References

- [1] Chu, Y., S. Rao, and H. Zhang. 2000. A case for end system multicast. In *Proceedings of ACM SIGMETRICS*, June.
- [2] Pendarakis, D., S. Shi, D. Verma, and M. Waldvogel. 2001. ALMI: An application level multicast infrastructure. In *Proceedings of the 3rd Usenix Symposium on Internet Technologies and Systems*, March.
- [3] Banerjee, S., B. Bhattacharjee, and C. Kommareddy. 2002. Scalable application layer multicast. In *Proceedings of ACM SIGCOMM 2002*, August.
- [4] Chawathe, Y. 2000. Scattercast: An architecture for Internet broadcast distribution as an infrastructure service. Ph.D. diss., University of California, Berkeley.
- [5] Chu, Y., A. Ganjam, T. S. E. Ng, S. G. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang. 2004. Early experience with an Internet broadcast system based on overlay multicast. In *Proceedings of USENIX 2004 Annual Technical Conference*.
- [6] Padmanabhan, V. N., H. J. Wang, P. A. Chou, and K. Sripanidkulchai. 2002. Distributing streaming media content using cooperative networking. In *ACM NOSSDAV*, May.
- [7] Sripanidkulchai, K., A. Ganjam, B. Maggs, and H. Zhang. 2004. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. In *Proceedings of ACM SIGCOMM*, Portland, OR.
- [8] Tran, D. A., K. A. Hua, and T. T. Do. 2004. A peer-to-peer architecture for media streaming. *IEEE Journal on Selected Areas in Communications* 22 (1): 121-33.
- [9] Guo, M., and M. Ammar. 2004. Scalable live video streaming to cooperative clients using time shifting and video patching. In *Proceedings of INFOCOM 2004*.
- [10] Veloso, E., V. Almeida, W. Meira, A. Bestavros, and S. Jin. 2004. A hierarchical characterization of a live streaming media workload. *IEEE/ACM Transactions on Networking* 14 (1): 133-146.
- [11] Sen, S., and J. Wang. 2004. Analyzing peer-to-peer traffic across large networks. *IEEE/ACM Transactions on Networking* 12 (2): 219-32.
- [12] Padmanabhan, V. N., H. J. Wang, and P. A. Chou. 2003. Resilient peer-to-peer streaming. In *11th IEEE International Conference on Network Protocols (ICNP)*.
- [13] Sripanidkulchai, K., B. Maggs, and H. Zhang. 2004. An analysis of live streaming workloads on the Internet. In *Proceedings of the 4th ACM SIGCOMM IMC*, October, Italy.
- [14] Kulkarni, V. G. 1996. *Modeling and analysis of stochastic systems*. London: Chapman & Hall.
- [15] David, H. A., and H. N. Nagaraja. 2003. *Order statistics*. 3rd ed. New York: Wiley-Interscience.
- [16] Saroiu, S., P. Gummadi, and S. Gribble. 2002. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking (MMCN)*.
- [17] Tan, G., S. A. Jarvis, D. P. Spooner, and G. R. Nudd. 2005. On efficient and robust overlay construction for large-scale P2P live media streaming. Report TR-2005-02, Department of Computer Science, University of Warwick, UK.
- [18] Ooi, W. T. 2005. Dagster: Contributor-aware end-host multicast for media streaming in heterogeneous environment. In *Proceedings of Multimedia Computing and Networking (MMCN)*.
- [19] Chu, Y., S. G. Rao, S. Seshan, and H. Zhang. 2001. Enabling conferencing applications on the Internet using an overlay multicast architecture. In *Proceedings of ACM SIGCOMM 2001*.
- [20] Zegura, E. W., K. Calvert, and S. Bhattacharjee. 1996. How to model an internetwork. In *Proceedings of IEEE INFOCOM '96*, San Francisco.

Guang Tan is a Ph.D. student in the Department of Computer Science at the University of Warwick, Coventry, United Kingdom. His research interests include distributed computing, networking and multimedia systems. He is a student member of the IEEE and the IEEE Computer Society.

Stephen A. Jarvis is a senior lecturer in the High Performance Systems Group at the Department of Computer Science, University of Warwick, UK. He has authored over 100 refereed publications (including three books) in the areas of high performance computing and distributed systems. While previously at the Oxford University Computing Laboratory, he worked on the development of performance tools with Oxford Parallel, Synchron Let., and Microsoft Research at Cambridge. He has close research ties with IBM, including current projects with the IBM T. J. Watson Research Center in New York and with IBM Hursley Park in the UK.

Xinuo Chen is a Ph.D. student in the Department of Computer Science at the University of Warwick, Coventry, United Kingdom. His primary research interests are high performance computing and peer-to-peer systems.

Daniel P. Spooner is a lecturer in the Department of Computer Science at the University of Warwick, UK, and a member of the High Performance Systems Group. His primary research interest is in the application of performance prediction techniques to improve resource scheduling in grid environments. Other interests include distributed network management architectures and high performance Web services.