ScaleStream – An Adaptive Replication Algorithm for Scalable Multimedia Streaming

Shabnam Ataee* Benoît Garbinato* Mouna Allani† Fernando Pedone‡

*University of Lausanne {shabnam.ataee,benoit.garbinato}@unil.ch
†Imperial College London m.allani@imperial.ac.uk
‡University of Lugano fernando.pedone@usi.ch

***Abstract*—In this paper, we propose ScaleStream, a new algorithm to support scalable multimedia streaming in peer-to-peer networks, under strict resources constraints. With the growth of multimedia consumption over the Internet, achieving scalability in a resource-constrained environment is indeed becoming a critical requirement. Intuitively, our approach consists in dynamically replicating the multimedia content being streamed, based on bandwidth and memory constraints. Our replication management strategy maximizes the number of consumers being served concurrently, while minimizing memory usage, under strict bandwidth constraints.**

Keywords: large-scale systems, adaptive replica placement, multimedia streaming.

I. Introduction

Multimedia streaming is emerging as one of the largest sink of network resources. The expansion of video-on-demand services, such as Netflix [14], offers a good example of this growth. Delivering multimedia content in a scalable manner is a challenge for any streaming protocol because networks have inherent bandwidth limits and computers have inherent memory and processing constraints. To ensure scalable streaming, protocols thus have to consider these constraints. In peer-to-peer networks, most of the proposed solutions focus on routing to ensure scalability and adaptiveness [1], [5], [8], [9].

In scenarios where resources at the media server are not sufficient to support a large set of consumers however, no routing solution can prevent bottleneck issues. When it comes to replication, a considerable amount of research has been published in the field of *data replication*, a well-known approach to increase availability and ensure scalability [7], [11], [16]. Traditionally however, replication-based approaches have mainly focused on *read/write accesses*, whereas multimedia streaming implies a *continuous flow* of data from servers to consumers. We advocate that replication is a promising complementary approach to routing to further increase the scalability of multimedia streaming.

Contributions. After formally defining our model and the *scalable multimedia streaming* problem in Sections II and III, we propose our novel replication solution in Section IV, followed by its performance evaluation in Section V. Our approach is targeted at *video-on-demand* streaming, where consumers can request streaming to start and to end at any point in time (contrary to *live streaming*). The *adaptiveness* of our approach lies in the fact that the replication scheme will *expand* and *contract* according to consumer needs and to resources constraints. It is *scalable* in the sense that it maximizes the number of consumers being served, while minimizing memory usage and taking bandwidth constraints into account. Finally, our approach is *decentralized* because it relies on purely local strategies and hence induces a reasonable overhead in managing the replication scheme. We describe researches related to ours in Section VI and sketch future work in Section VII.

II. Modeling multimedia streaming

We consider a distributed system composed of processes (nodes) that communicate by message passing. We model the system topology as a connected graph $G = (\Pi, \Lambda)$, where $\Pi = \{p_1, p_2, ..., p_n\}$ is a set of n processes ($|\Pi| = n$) and $\Lambda = \{l_1, l_2, ...\} \subseteq \Pi \times \Pi$ is a set of bidirectional *reliable* communication links. That is, we have $V(G) = \Pi$ and $E(G) = \Lambda$. A link from p_i to p_j is denoted by $l_{i,j}$. We define a $path$ as a combination of links and intermediate processes through which a message transits from a source node to a destination node.

Overlay tree. We assume that (1) some tree overlay T covers the system graph G and (2) nodes only communicate via the links in T. Links are assumed to be *reliable*, i.e., they are typically implemented on top of TCP, which is the case of most streaming systems today. The set of links present in T is noted Λ_T, so we have $\Lambda_T \subseteq \Lambda$. If $l_{i,j} \in \Lambda_T$ and $i \neq j$, we say that p_j is a neighbor of p_i. The set of neighbors of p_i in T is denoted by $neighbors(p_i)$. We then denote $path(p, q)$ the subset of links of Λ_T connecting processes p and q. We assume that T is created by some underlying communication layer based on a

criteria that makes sense for upper layers, as suggested in [12], i.e., *geographical proximity* between nodes in our case.

Multimedia replication. We define M as the multimedia content to replicate and $size(M)$ denotes its size in number of blocks of some predefined size. The *replication scheme* of media M, denoted R, is then a subtree of T composed of nodes holding a copy of M, which implies that the replication scheme is always connected. A process $p \in R$ is sometimes called a *replica*. In addition, we define $\bar{R}$ as the part of the network that is outside the replication scheme ($\bar{R} = T - R$). The set of the neighbors of any node p is composed of neighbors inside the replication scheme, denoted $neighbors_R(p)$, and of neighbors outside the replication scheme, denoted $neighbors_{\bar{R}}(p)$. Furthermore, the subset of the nodes in R that have at least one neighbor outside R is denoted by B_R (for *Border*), while the subset of the nodes in R that has *only* one neighbor inside R is denoted L_R (for *Leaf*).

Whenever some process $p \in \bar{R}$ wishes to receive media content M as a stream, it sends a request to the closest replica in R; if process p is in R, it simply reads its local copy of M. Since processes communicate exclusively through T, process p only needs to know to which of its neighbors it must send its request. The latter is either in R or it forwards the request towards the replication scheme via one of its neighbors. So, the closest replica eventually receives p's request and can start streaming M block-by-block to p. From then on, either p consumes the whole multimedia content or it sends a request stop streaming M.

In the following, we assume that the stream rate required to receive multimedia content M without lagging is of one block per time quantum Δ_t. While the specific size of a block and the specific duration of Δ_t actually depends on the codec and on the network, we are only concerned with their ratio here. In network terms, this ratio of one block per Δ_t simply expresses the bandwidth needed to stream M to one consumer. So we express bandwidth in terms of the number of distinct streams of M that can be supported.

Consumption models. When and how processes become consumers, and how long they remain consumers, is modeled by a *consumption model*. In this paper, we consider three consumption models: (1) *static consumption*, (2) *random consumption* and (3) geographically-dependent consumption, or *geo-dependent consumption* for short. These models are used in Section V for our performance evaluation.

When studying the convergence of our replication algorithm towards an optimal use of resources, formally defined in Section III, we assume a *static consumption model*. In such a model, some processes are randomly chosen to be consumers at time $t = 0$ and remain consumers *forever*. That is, after a consumer received the last block of M, it immediately receives the first block again, and all subsequent blocks of M. The goal of this assumption is to study the convergence of our replication algorithm under stable conditions.

When studying the dynamic behavior of our algorithm, we consider two dynamic consumption models: *random consumption* and *geo-dependent consumption*. In the random consumption model, each process has a certain independent probability to be a consumer at any time. In the geo-dependent consumption model, the probability for a process to be a consumer is a function of the number of consumers in its neighborhood within T. This model accounts for scenarios where media consumption is submitted to geographical constraints, typically time zones.

Resource limitation. We assume that each link in the network has a limited bandwidth capacity, which for simplicity we assume to be the same for all links and denote by Δ_B. This assumption does not compromise the generality of our approach. The bandwidth actually used by link l is then denoted $bandwidth(l)$.

When it comes to memory, on the contrary, we assume no strict limits, since practically any multimedia content can be cached onto stable storage today, thanks to the ubiquity of high-density hard disks. In addition, all computers today have enough RAM to perform stream buffering while consuming multimedia content. Nevertheless, as explained in next section, our adaptive replication scheme aims at serving as many consumers as possible in spite of bandwidth constraints, while at the same time *minimizing the number of replicas*.

III. Achieving scalable streaming

The goal of our adaptive replication algorithm is to support *scalable* multimedia streaming, i.e., it should dynamically adapt the membership of replication scheme R, in order to serve as many consumers as possible. Since we assume no memory constraints, a naive solution consists in simply replicating M everywhere. This is however a bad idea, based on the following observation. Unless all or at least a large portion of processes are indeed consumers, fully replicating a large multimedia content would initially eat up a lot of bandwidth for very little gain. In addition, although our presentation focuses on a single multimedia stream, in real settings several multimedia contents are managed concurrently, each one being consumed by a subset of all processes.

Scalability as optimization function. Intuitively, our replication algorithm aims at dynamically adapting replication scheme R in order to *maximize the number of consumers being concurrently served without*

exceeding the bandwidth capacity, while at the same time *minimizing the size of* R. When considering dynamic consumption scenarios however, such as the random consumption model and the geo-dependent consumption model, the translation of this intuitive optimization goal into a *time-dependent* optimization function can be quite challenging.

For this reason, we formally define the optimization goal pursued by our adaptive replication algorithm *for the static consumption model only*. This allows us to show that our algorithm converges towards an optimal usage of resources, as soon as the consumption stabilizes for long enough. This convergence property of our algorithm results in significant performance gain for dynamic consumption scenarios as well.

$$\left.\begin{array}{ll} \textbf{\textit{minimize}} & |R| \quad (a) \\ \textbf{\textit{subject to}} & \forall p \in C_M : p \in S_M \quad (b) \\ & \forall l \in \Lambda_T : bandwidth(l) < \Delta_B \quad (c) \\ & \forall p, q \in \Pi_R : path(p,q) \subseteq \Lambda_R \quad (d) \end{array}\right\} \quad (1)$$

Given a static consumption model, Equation 1 formally expresses the optimization problem we are trying to solve. Expression 1(a) captures our optimization objective, whereas Expressions 1(b) to 1(d) capture the constraints under which an optimal must be found. In Constraint 1(b), the term C_M denotes the set of processes that have requested to be a consumer at some point or another, while the term S_M denotes the set of consumers that will eventually be served forever. So, Constraint 1(b) simply states that eventually all consumers must be concurrently served forever. Constraint 1(c) then states that bandwidth usage cannot exceed the bandwidth capacity Δ_B, while Constraint 1(d) states that R must always be connected.

Scalability and decentralization. Equation 1 formalizes the optimization goal of our replication algorithm. Achieving this goal is however not sufficient for our algorithm to actually scale: it needs in addition to be *decentralized* in its operation. That is, every replica p in R should be able to *locally* adapt R based solely on what it observes *in its direct neighborhood*. Purely local observations typically consist of the number of consumers requesting to be served by p and the bandwidth usage of links directly connected to p.

IV. The ScaleStream Algorithm

The ScaleStream algorithm performs two tasks: (1) *serving consumers* and (2) *adapting replication scheme* R. For Task 1, simply note that each replica is the root of exactly one subtree and each consumer is part of exactly one subtree. So each replica is responsible for serving consumers located in its subtree.

When it comes to adapt replication scheme R, some process p can be in one of four possible states: ***in*** R, ***joining*** R, ***leaving*** R, or ***out*** of R. Apart from the obvious ***in*** and ***out***, the ***joining*** state means that p is already serving new open-stream requests but does not yet hold a full copy of media M, whereas the ***leaving*** state means that p is willing to leave R but has not yet been cleared to do so. In addition to its own state, each process $p \in R$ is also aware of the states of its direct neighbors in T. So when it comes to adapt R, the key questions are when and how a process moves from one state to another, i.e., under what condition a process joins R, in which case we say R is *expanding*, and under what condition a replica leaves R, in which case we say R is *contracting*. Periodically, each replica first checks whether expansion is needed and if not, it then checks whether contraction is possible.

Expanding R. Intuitively, R expands as soon as a risk of bandwidth shortage is detected at its border. For this detection, our algorithm uses a bandwidth threshold parameter, denoted Δ_b, which is strictly smaller than the actual bandwidth capacity Δ_B. For the sake of simplicity, we assume that all links have the same bandwidth capacity Δ_B, which implies that all replicas can use the same bandwidth threshold Δ_b.

So, let $p_r \in B_R$ be some replica either ***in*** R or ***joining*** R and $p_h \in neighbors_{\bar{R}}(p_r)$, and let $consumers_{p_r}(p_h)$ be the set of consumers served by p_r through p_h. Replica p_r triggers the expansion of R towards p_h as soon as the following *expansion condition* holds: $\Delta_B - |consumers_{p_r}(p_h)| \leq \Delta_b$.

To trigger expansion, replica p_r actually sends a join message to p_h, asking it to join R. From then on, p_r continues to serve media blocks to consumers downstream of p_h, but the latter now stores them.

Since the expansion condition is typically detected upon the reception of an open-stream message by p_r, the bandwidth headroom theoretically needs only to be of one block, i.e., $min(\Delta_B - \Delta_b) = 1$. Indeed, as soon as p_h goes in ***joining*** state, it stores all blocks it receives, including the first block for the new consumer whose open-stream messages triggered the expansion. So p_h is able to serve new open-stream messages.

Contracting R. Intuitively, a contraction of R will be attempted when the risk of bandwidth shortage disappears at one of its leaves, provided that leaf replica did not detect a need for expansion. More precisely, let $p_r \in L_R$ be some replica ***in*** R and p'_r its parent in R, and let $consumers_{p_r}$ be the set of all consumers being served by p_r (including itself if it is consuming as well). Replica p_r, which did not detect a need for expansion, attempts a contraction of R towards p'_r when the following *contraction condition* holds: $\Delta_B - |consumers_{p_r}| > \Delta_b$.

To attempt a contraction, p_r sends a request-leave message to p'_r (its parent in R). The latter immediately authorizes p_r to leave, unless it is itself trying to

leave R. Note that this can only occur if p_r and p'_r are the only two replicas left in R. We must nevertheless consider this special case, otherwise both replicas might leave R, which would then remain empty.

So, if p'_r is itself in ***leaving*** state when receiving the request-leave message from p_r, it must apply an additional deterministic criterion to decide whether to let p_r go or not. Assuming that each process p in the system is given a unique numerical identifier, denoted $id(p)$, a suitable deterministic criterion could simply be $id(p'_r) > id(p_r)$.

As a result of this decision process, replica p'_r sends a reply-leave message containing its decision back to p_r. If p_r is authorized to leave R, it checks whether new open-stream messages were received while waiting for a reply from p'_r. If such is the case, p_r must check the contraction condition again and send a cancel-leave message back to p'_r if the condition no longer holds. Otherwise, p_r sends p'_r a confirm-leave message and gets rid of M. As soon as p'_r receives the confirm-leave message, it takes over all streaming activities performed by p_r. For this to work, we simply have to assume that the confirm-leave message contains information about the next block expected by each consumer in $consumers_{p_r}$ prior to the contraction. In practice, contraction is actually triggered only if the condition holds for some time.

V. Performance evaluation

To assess the performance of ScaleStream, we evaluate it against the three different consumption models introduced in Section II, namely *static consumption, random consumption,* and *geo-dependent consumption.* For all of these models, we performed experiments with 100 to 1000 processes on a *geographical tree network*, using the Sinalgo simulation framework. To implement this geographical tree, we first build a ring network and we then additionally link each node to its two closest indirect neighbors (one in each direction). The resulting ring-mesh topology is thus a ring where each node is transitively connected to its four closest neighbors, which we can interpret as being geographically located north, south, east and west from that node. Finally, on top of this network, a random tree is built, i.e., each node has *at most* four neighbors.

For our simulations, we assume that the multimedia content consists in a movie lasting two hours (7200 *seconds*) and that each block contains 10 seconds of movie. Therefore, the entire movie is made of 720 blocks. The simulation proceeds in *steps* of Δ_t duration: in each step at most one block is transmitted to each consumer. The simulation continues until all the consumers received the entire movie or stopped watching it by sending a close-stream message to R.

As in each step at most one block of the movie is transmitted to each consumer, it takes at least 720 steps to transmit the entire movie to a specific consumer. In addition, to simplify the interpretation of our results, we consider that processes and links are reliable, and that all links have the same bandwidth capacity (Δ_B) and all processes use the same bandwidth threshold (Δ_b). More precisely, we have $\Delta_B = 6$ *blocks/step* and $\Delta_b = 2$ *blocks/step*, which means that only part of the total bandwidth capacity can be used for streaming.

Convergence to the optimal. ScaleStream converges to an *optimal replication scheme* satisfying the optimization problem of Equation 1, when facing a static consumption model. An optimal replication scheme is one containing the minimum number of replicas required to transmit one block to each consumer in each step.

To show this, 20% of the nodes of the network are randomly selected to be *static consumers*, i.e., $|C_M| \cong 0.2 \times |\Pi|$. The simulation starts with just one randomly placed replica and after a while, the number of replicas stabilizes. To show that the resulting replication scheme is optimal, our results are compared to a *centralized algorithm* that finds the optimal replication scheme for the same set of static consumers.

$\|\Pi\|$	$\|R\|$	*steps*
100	24.4	262.4
300	168.8	574.4
500	318	861.1
1000	724	1604.3

(a) summary

$\|\Pi\|$	$\|R\|$	*steps*
1000	672	1586
1000	704	1621
1000	783	1598
1000	876	1586

(b) details

TABLE I
Convergence towards the optimal

Table I(a) shows how many steps ScaleStream takes to converge towards a replication scheme comparable to the one computed by the optimal centralized algorithm, for networks of 100, 300, 500, and 1000 nodes. That is, in all executions, ScaleStream ends up with a replication scheme containing the same number of replicas as the centralized algorithm, which shows that it does indeed converge to the optimal. Each experiment was repeated 10 times, i.e., for each network size we created 10 different networks and executed both algorithms. Table I(b) then details the results of various experiments with a network of 1000 nodes. We can see that, as the positions of consumers vary from one simulation to the other, our algorithm converges to replication schemes with various numbers of replicas.

Dynamic consumption models. To evaluate the dynamic behavior of our algorithm, we consider two models where consumers change over time (*random consumption* and *geo-dependent consumption*). For both models, and in order to evaluate our algorithm in

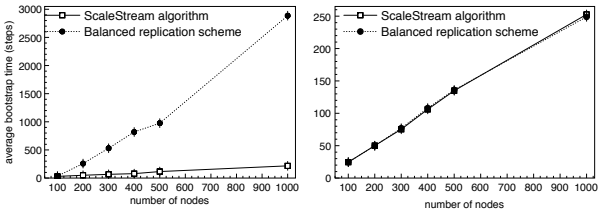
a fair manner, our results are compared to a balanced replication scheme. A *balanced replication scheme* is a static replication scheme that has better performance than other static connected replication schemes. Intuitively, a balanced replication scheme is computed statically and tries to minimize the distance to its furthest consumers. As a result, it is roughly located in the middle of the network. It should be noted that the balanced replication scheme is different from the optimal replication scheme presented in Section V.

In our simulations, ScaleStream always starts with a replication scheme consisting of just one replica, while the balanced replication scheme has more than one replica right from the start. This fact counts against our ScaleStream algorithm but even then ScaleStream outperforms the balanced replication scheme.

1) Geo-dependent consumption model: Our algorithm is able to take advantage of potential geographical correlations in dynamic consumption patterns, in order to improve its performance. Such geographical correlations account for what happens in reality when streaming the same multimedia content across multiple timezones, e.g., in a continent-wide video-on-demand scenario. We assume that the geographical tree network is divided into five time zones and that the number of nodes in all these timezones is approximately equal. Connected timezones have a one hour difference and 20% of the nodes are selected as *eventual consumers* at the beginning of the simulation. Furthermore, we assume that each consumer starts consuming the movie stream between 7:45pm and 8:15pm in its timezone; this is a typical video-on-demand scenario: people in a timezone start watching some newly released movie at roughly the same time. Formally, in one timezone after the other, eventual consumers become *actual consumers* for 30 *minutes* (180 *steps*), with gaussian distribution ($\mu = 8pm$, $\sigma^2 = (\frac{180}{4})^2$). We also assume that consumers stop watching it with probability 0.5 after receiving half of the movie (by sending a close-stream message to R).

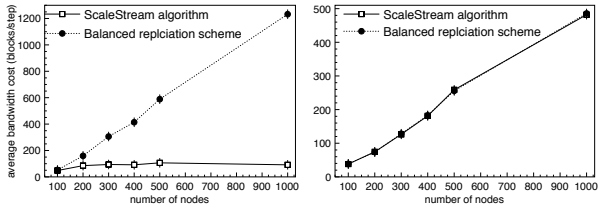
To compare the performance of ScaleStream and the performance of the balanced replication scheme, we only consider executions where the *average memory cost* for both these algorithms is roughly equal (the average memory cost is the average number of replicas across an execution). We compare ScaleStream with the balanced replication scheme in terms of *bootstrap time* and average *bandwidth cost* for networks of 100, 200, 300, 400, 500, and 1000 nodes. Here again, each experiment was repeated 10 times. The bootstrap time is the number of steps that a consumer must wait from the time it sends its open-stream request until it receives the first block of the movie. The average bandwidth cost is the average number of blocks that are transmitted in the entire network in one step.

Figures 1(a) and 2(a) show that our algorithm always exhibits smaller average bootstrap time and bandwidth cost than the balanced replication scheme. Furthermore, the advantage of ScaleStream grows as the number of nodes in the network increases, which shows the *scalability* of our approach.



(a) geo-dependent consumption (b) random consumption

Fig. 1. Average bootstrap time



(a) geo-dependent consumption (b) random consumption

Fig. 2. Average bandwidth cost

The good performance of ScaleStream when facing the geo-dependent consumption is due to the fact that *open-stream* requests come in waves from one region after the other, as consumers spread across timezones. The replication scheme has thus enough time to expand towards these regions, while contracting from other regions, resulting on average in less bandwidth usage and in shorter bootstrap time.

2) Random consumption model: We now evaluate the performance of ScaleStream when facing a completely random consumption model. This gives us a measure to how adaptive ScaleStream is in a *completely unpredictable* streaming scenario. As for the geo-dependent consumption model, 20% of the nodes are selected as eventual consumers at the beginning of the simulation. Then, for 2.5 *hours* (900 *steps*), with gaussian distribution ($\mu = 8pm$, $\sigma^2 = (\frac{900}{4})^2$), eventual consumers become actual consumers but this time they are selected purely randomly, as expressed in Section II. Here also, actual consumers might stop watching the movie with probability 0.5 after receiving half of the blocks.

ScaleStream is again compared to the balanced replication scheme in terms of bootstrap time and average bandwidth cost, only for executions where the average memory cost for both these algorithms is roughly equal. Results are presented in Figures 1(b) and 2(b)

and show that when facing identical conditions, both algorithms reach almost the same results (the curves overlap). This is not surprising, since consumers are selected completely randomly and thus end up all over the network. As a consequence, few links reach the bandwidth threshold. In other words, ScaleStream does not have to dramatically adapt the replication scheme, which then remains almost static and is practically identical to the balanced replication scheme.

VI. Related works

To support the growth of media streaming, research efforts mainly focused on routing algorithms. Typically, a routing solution consists in building and maintaining an overlay through which stream blocks are routed. The majority of the overlay-based solutions rely on a tree topology for their routing process [8], [9], [2], [5], [10], [4], [6], [13], and many solutions [8], [5], [9] aim at optimizing bandwidth. They however do not handle cases where a path is overloaded and no alternative routing path is available. Other research efforts focused on distributed adaptive data replication. In [16] the replication scheme adapts to the read-write pattern, whereas in [3] the objective is to adapt to the read-write pattern while coping with an unreliable environment. Similar to our solution, these approaches define the replication scheme as a connected subtree. Besides read-write patterns, other cost functions were defined to dictate the replication strategy. In [15] for instance, the replica placement strategy aims at minimizing a cost function taking into account the average read latency, the average write latency and the amount of bandwidth used to enforce consistency between replicas. Similar to these previous replication solutions, ScaleStream can be considered as a type of adaptive replica placement.

VII. Future work

Several simplifying assumptions were made throughout the paper. Future work will revisit some of these assumptions in order to make our approach more robust and general. For example, the always-connected nature of the replication scheme induces many idle replicas, which would somehow defeat our goal to minimize the memory footprint. Considering processes and links reliable is a strong assumption. In [3] we proposed a communication infrastructure for supporting data replication in an environment where processes and links fail probabilistically. We plan to study ways to adapt the results in [3] to ScaleStream.

Acknowledgment.. This research is funded by the Swiss National Science Foundation, in the context of Project number 200021-127352.

References

[1] M. Allani, B. Garbinato, A. Malekpour, and F. Pedone. Quocast: A resource-aware algorithm for reliable peer-to-peer multicast. In *Proceedings of The 8th IEEE International Symposium on Networking Computing and Applications, NCA*, 2009.

[2] M. Allani, B. Garbinato, F. Pedone, and M. Stamenkovic. A gambling approach to scalable resource-aware streaming. In *Proceedings of 26th IEEE Symposium on Reliable Distributed Systems (SRDS)*, pages 288 – 297, October 2007.

[3] M. Allani ans B. Garbinato, A. Malekpour, and F. Pedone. Reliable communication infrastructure for adaptive data replication. In *Proceeding of the 11th International Symposium on Distributed Objects, Middleware, and Applications (DOA)*, 2009.

[4] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. Omni: An efficient overlay multicast infrastructure for real-time applications. *Computer Networks*, 50(6):826–841, 2006.

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

[6] Y. Cui, B. Li, and K. Nahrstedt. oStream: asynchronous streaming multicast in application-layer overlay networks. *Selected Areas in Communications, IEEE Journal on*, 22(1):91–106, 2004.

[7] S. Elnikety, S. G. Dropsho, and W. Zwaenepoel. Tashkent+: memory-aware load balancing and update filtering in replicated databases. In *EuroSys*, pages 399–412, 2007.

[8] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and Jr. J. W. O'Toole. Overcast: reliable multicasting with on overlay network. In *OSDI'00: Proceedings of the 4th conference on Symposium on Operating System Design Implementation*, Berkeley, CA, USA, 2000. USENIX Association.

[9] D. Kostić, A. Rodriguez, J Albrecht, and A. Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. In *Proceedings of the nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, 2003.

[10] F. Liu, X. Lu, Y. Peng, and J. Huang. An efficient distributed algorithm for constructing delay and degree-bounded application-level multicast tree. In *ISPAN '05: Proceedings of the 8th International Symposium on Parallel Architectures,Algorithms and Networks*, pages 72–77, Washington, DC, USA, 2005. IEEE Computer Society.

[11] J. MacCormick, N. Murphy, V. Ramasubramanian, U. Wieder, J. Yang, and L. Zhou. Kinesis: A new approach to replica placement in distributed storage systems. *ACM Transactions on Storage (TOS), to appear*.

[12] A. Malekpour, F. Pedone, M. Allani, and B. Garbinato. Streamline: An architecture for overlay multicast. In *Proceedings of the 8th IEEE International Symposium on Network Computing and Applications (NCA'09)*, pages 44 –51, July 2009.

[13] L. Mathy, R. Canonico, and D. Hutchison. An overlay tree building control protocol. In *NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication*, pages 76–87, London, UK, 2001. Springer-Verlag.

[14] J. Meisner. Netflix Wades Deeper Into On-Demand Stream With CBS, Disney Deals. *E-Commerce Times (online edition)*, September 2008.

[15] S. Sivasubramanian, G. Alonso, G. Pierre, and M. van Steen. GlobeDB: Autonomic data replication for web applications. In *Proc. of the 14th International World-Wide Web Conference*, pages 33–42, Chiba, Japan, may 2005.

[16] O. Wolfson, S. Jajodia, and Y. Huang. An adaptive data replication algorithm. *ACM Transactions on Database Systems*, 22(2):255–314, June 1997.