ReStream – A Replication Algorithm for Reliable and Scalable Multimedia Streaming

Shabnam Ataee* Benoît Garbinato* Fernando Pedone†

*University of Lausanne
Information Systems Institute
CH-1015 Lausanne (Switzerland)
{shabnam.ataee, benoit.garbinato}@unil.ch

†University of Lugano
Faculty of Informatics
CH-6904 Lugano (Switzerland)
fernando.pedone@usi.ch

***Abstract*—Multimedia consumption over the Internet is emerging as one of the largest sink of network resources, making scalable and reliable streaming increasingly challenging. To address this challenge, we propose ReStream, an adaptive replication algorithm that relies on replication to achieve reliable and scalable streaming in resource-constrained environments. Our algorithm dynamically adapts replica placement to maximize the number of consumers under latency and bandwidth constraints, while minimizing the number of replicas. In addition, ReStream supports partitioning, i.e., replicas can be located anywhere in the network and do not necessarily form a connected graph. This allows ReStream to yield the same performance in consumption models where consumers tend to be geographically co-located, as well as in consumption models where consumers placement is totally random.**

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

I. Introduction

In recent years, multimedia consumption over the Internet has grown dramatically, in particular due to the general trend consisting in moving multimedia content from local disks to cloud-based storage on the Internet. The expansion of audio and video streaming services, such as iTunes Match,[1] Google Play,[2] or Netflix,[3] testifies of this general trend. Streaming over the Internet in a reliable and scalable manner is however difficult, due to network bandwidth limits, as well as computer memory and processing constraints. For this reason, a recurring aspect of many streaming solutions is their adaptivity to changes in the distributed environment in which they execute. Most solutions proposed in peer-to-peer networks for instance aim at dynamically routing blocks through the path with the highest amount of resources, in order to avoid bottlenecks [1], [8], [13], [16], [18]. These solutions work well when available resources are sufficient to cope with the number of consumers in the system. However, when resources at the server fall short and no longer allow it to support the growing number of consumers, routing solutions alone cannot prevent bottlenecks.

Replication to the rescue. Here replication can be of great value. To illustrate this claim, consider the example of Figure 1(a), where a server is connected to the network via low-bandwidth links.[4] In this example, as the number of consumers grows, a routing solution that normally assumes the presence of high-bandwidth links has no way to scale and at the same time to continue delivering high quality streaming. The usual approach here is to reduce the flow of data, by sending fewer or smaller blocks, which in turn lowers the quality of the multimedia content being streamed. Figure 1(b) then shows the benefit of applying replication: by adding replicas of the media at nodes directly connected to the previous server, we increase the number of servers and thus the number of streams that can be supported, while keeping the same quality of service. In this very simple example, replicated servers can now stream more than five times what was possible for a single server. We thus believe that replication is a promising complementary approach to routing to further increase scalability and reliability of multimedia streaming.

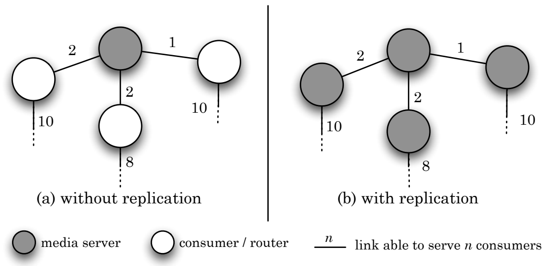


Fig. 1. Replication: motivating example

Contribution and roadmap. In Sections II and III, we formally define the system model and the problem we consider in this paper, i.e., reliable and scalable multimedia streaming. In Section IV, we then introduce ReStream, a multimedia streaming algorithm

[1]http://www.apple.com/itunes/itunes-match
[2]http://play.google.com
[3]http://www.netflix.com

[4]Links are labeled with the maximum number of streams they can support, reflecting the maximum bandwidth.

that relies on replication to support reliable and scalable streaming in resource-constrained environments. Our algorithm is specifically targeted at *video-on-demand*, where consumers can request streaming to start and to end at any point in time (as opposed to *live streaming*). Roughly speaking, our approach consists in adapting replica placement to maximize the number of consumers being served concurrently under strict latency and bandwidth constraints, while at the same time minimizing the number of replicas. Contrary to many approaches targeted at streaming, RESTREAM supports partitioning, i.e., replicas can be located anywhere in the network and do not necessarily form a connected graph. This makes it more reliable and more resource-efficient. In Section V, we evaluate the performance of RESTREAM and compare it with our previous replication algorithm that does not support partitioning [4]. Finally, Section VI discusses research related to RESTREAM, while Section VII sketches our ongoing and future work on multimedia streaming.

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 communication links. That is, we have $V(G) = \Pi$ and $E(G) = \Lambda$. Nodes and links are assumed to be *unreliable*. In this system, we consider *crash-recovery* failure model. In this model, if a node or a link crashes, it recovers after a while.

Overlay network. Regarding message passing, we assume that (1) there are some tree overlay(s) $T_1, T_2, ..., T_n$ in the system, i.e. $\bigcup_{i=1}^{n} T_i$ covers the system graph G, (2) each node is *at least* in the membership of a tree T_i and (3) nodes only communicate via the links in $T_1, T_2, ..., T_n$. The set of links present in T_i is noted Λ_{T_i}, so we have $\Lambda_{T_i} \subseteq \Lambda$. Links are typically implemented on top of TCP, which is the case of most streaming systems today [12]. If $l_{i,j} \in \Lambda_{T_i}$ and $i \neq j$, we say that p_j is a neighbor of p_i in T_i. The set of neighbors of p_i in $\bigcup_{i=1}^{n} T_i$ is denoted by $neighbors(p_i)$. Note that this paper does not address the question of how to build tree overlay(s) $T_1, T_2, ..., T_n$. Rather, we assume that $T_1, T_2, ..., T_n$ were created by the underlying communication layer based on a criteria that makes sense for upper layers, i.e., *geographical proximity* between nodes in our case. This architecture based on two-layers is inspired by [20].

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. There is at least one *seed* in the network. The seed is a node $p_i \in \Pi$ which *always* has a copy of media M. We assume that each *seed* p_i is the root of a tree overlay T_i, and also the root of each tree overlay T_i is a *seed* (p_i). Therefore, the number of seeds in the network is equal to the number of the tree overlays. In addition, we assume that each T_i is a *directed tree* i.e., each process $p \in \Pi_{T_i}$ knows the direction to the respective seed p_i of T_i. In other words, in directed tree T_i each node knows its *parent*.

Each process $p \in \Pi$ can become a consumer. Whenever p wishes to receive media content M as a stream, it sends a request to the seed of a tree T_i to which it belongs. This request is routed along the path to the seed until it reaches the *closest replica* which has a copy of media M. This replica eventually accepts p's request and starts streaming M block-by-block to p. From then on, either p consumes the whole multimedia content or it sends a request to the seed to stop streaming M.

The *replication scheme* of media M, denoted R, is then defined as a set of nodes holding a copy of M ($R \subseteq \Pi$). Contrary to our previous work [4], the replication scheme of RESTREAM can be *partitioned*, i.e., it does not necessarily form a connected graph. We also define $\bar{R}$ as the set of nodes of the network which are out of the replication scheme ($\bar{R} = \Pi - R$). Now, we can divide the set of the neighbors of any node p to two parts: neighbors inside the replication scheme, denoted $neighbors_R(p)$, and neighbors outside the replication scheme, denoted $neighbors_{\bar{R}}(p)$. Similarly, in each directed tree T_i, the set of the children of any node p is composed of children inside the replication scheme, denoted $children_{R_i}(p)$, and of children outside the replication scheme, denoted $children_{\bar{R}_i}(p)$.

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 for simplicity, hereafter all bandwidths are expressed 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 *consumption models*. 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, when evaluating the performance of our adaptive replication approach.

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 whole media content M, it immediately starts receiving the media M again from the beginning. 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 models where consumers change over time: *random consumption* and *geo-dependent consumption*. In the *random consumption model*, each process has a certain independent probability to be a consumer at any point in time. We assign values to these probabilities in Section V, when we evaluate the performance of our algorithm in the random consumption model. In the *geo-dependent consumption model*, the probability for a process to be a consumer is no longer independent but a function of the number of consumers in its neighborhood. This model accounts for scenarios where the media consumption is submitted to geographical constraints, typically time zones, and only makes sense when overlay trees are built based on geographical proximity.

Resource limitation. To account for resource limitations, we assume that each node in the network has a limited *upload* and *download* bandwidths. Contrary to our previous work [4] and in order to account for the bandwidth heterogeneity of the Internet, upload and download bandwidths are not necessarily the same for all nodes. Hereafter, we denote the upload and download bandwidth capacity of node p as $\Delta_u(p)$ and $\Delta_d(p)$ respectively. As suggested earlier, we express bandwidth in terms of the number of distinct streams of M that can be supported. The *upload* and *download* bandwidth actually used by node p is then denoted $bandwidth_u(p)$ and $bandwidth_d(p)$ respectively. With respect to directed tree T_i, the download bandwidth of process $p \in T_i$ is relevant when p is receiving blocks from its *parent* in T_i and its upload bandwidth is relevant when p is sending blocks to its *children* in T_i.

When it comes to memory, on the contrary, we assume no strict limits, since practically any multimedia content can be cached onto stable storage thanks to the ubiquity of high-density hard disks. In addition, all computers today have enough RAM to perform stream buffering while serving or consuming multimedia content.[5] 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*.

[5]In practice, ReStream includes a straightforward buffering scheme to smooth out small but significant bandwidth fluctuations and thus avoid potential latency issues due to those fluctuations.

III. Scalable and reliable streaming

The main objective 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. In addition, our adaptive replication algorithm should be *reliable*, i.e., it continues to work even when nodes or links crash. 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: 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. So, in such settings, fully replicating all multimedia contents is simply not possible.

Scalability as an 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 and latency threshold*, 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 *non-static* 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. As discussed in Section V, this convergence property of our algorithm results in significant performance gain for dynamic consumption scenarios as well.

So, given a static consumption model, Expression 1 formally expresses the optimization problem we are addressing. Expression 1(a) captures our optimization objective, whereas Expressions 1(b) to 1(e) capture the constraints under which an optimal solution 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 the upload bandwidth usage of each node p cannot exceed the upload bandwidth capacity $(\Delta_u(p))$ of node p, while Constraint 1(d) states that the download bandwidth usage of each node p cannot exceed the download bandwidth capacity $(\Delta_d(p))$ of p. Finally, Constraint 1(e) expresses that the latency a consumer should expect before starting to receive the multimedia content must not exceed Δ_l. Maximum latency Δ_l is expressed as the distance

$$\left.\begin{array}{lll} \textbf{\textit{minimize}} & |R| & (a) \\ \textbf{\textit{subject to}} & \forall p \in C_M : p \in S_M & (b) \\ & \forall p \in \Pi : bandwidth_u(p) \leq \Delta_u(p) & (c) \\ & \forall p \in \Pi : bandwidth_d(p) \leq \Delta_d(p) & (d) \\ & \forall p \in C_M : distance(p, R) \leq \Delta_l & (e) \end{array}\right\} \quad (1)$$

in number of hops between the consumer and the replication scheme R.

Scalability, reliability and decentralization. Expression 1 formalizes the optimization goal towards which our replication algorithm should converge. Achieving this goal is however insufficient for our algorithm to actually scale: it needs in addition to be *reliable* and *decentralized* in its operation.

Reliability is a key issue in multimedia streaming concept: replicas should continue streaming to consumers in the presence of nodes and links failure. In terms of *decentralization*, every node $p \in \Pi$ 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 upload and download bandwidth usage of node p itself.

IV. The ReStream Algorithm

The ReStream algorithm essentially performs two tasks: (1) *serving consumers* and (2) *adapting replication scheme R*. Sections IV-A and IV-B describe these two basic tasks and how they are combined when R needs to be adapted while serving consumers. Finally, Section IV-C describes how ReStream algorithm copes with node and link failures.

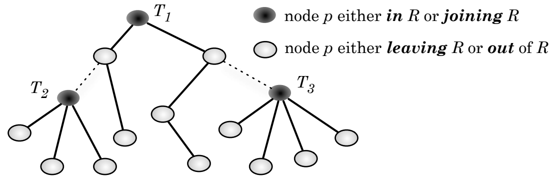
As the tasks performed by each seed and its tree are completely independent of other seeds and their respective trees, in this section for simplicity we focus our presentation on one seed p_s and the corresponding overlay tree T rooted at p_s. When multiple trees for distinct media contents are used, the system shares the upload and download bandwidth for streaming distinct media contents. In this case, the usage rate of upload and download bandwidth of different nodes increases, which might trigger the expansion of replication scheme for each media, more often and sooner.

In this section, as suggested in Section II, we also assume that each process p has access to the underlying communication layer that created tree overlay T. This communication layer is also responsible for routing request messages (for media M) from each consumer to the closest replica in the path to the seed, and for routing media blocks from that replica back to the consumer.

In ReStream, a process p can be in one of four possible states with respect to replication scheme R: ***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 is also aware of the state of its direct neighbors in T.

A. Serving consumers

Each node p in T can become a consumer and ask for media M. To understand how *new* consumers are served, let us consider the graph we obtain when corresponding to each replica p_r either ***in*** R or ***joining*** R, we remove its link which connects p_r to its *parent* p_p in T.[6] As illustrated in Figure 2, we end up with a *forest F*, where each replica either ***in*** R or ***joining*** R is the root of exactly one subtree T_i, and each process p either ***leaving*** R or ***out*** of R is contained in exactly one subtree. So when it comes to streaming the content of M, each replica either ***in*** R or ***joining*** R is simply responsible for serving new consumers located in its subtree T_i.



T_1
T_2
T_3
node p either ***in*** R or ***joining*** R
node p either ***leaving*** R or ***out*** of R

Fig. 2. Serving consumers with ReStream

B. Adapting replication scheme R

When it comes to adapting 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* or *growing*, and under what condition a replica leaves R, in which case we say R is *contracting* or *shrinking*.

Expanding R. Intuitively, R expands as soon as a risk of bandwidth shortage or locating too away from R is detected by a node in the network. Because of the *decentralized* nature of our algorithm, the expansion is detected locally by a node in the network. Expanding R can occur in one of the following situations.

Let p be some node in T and $p_h \in neighbors(p)$ either ***leaving*** R or ***out*** of R, and let $openStreams(p)$ be the set of open streams routed via p. Node p, upon the reception of an open-stream message via p_h, triggers the expansion of R towards p_h if the *upload bandwidth shortage* is detected at p and therefore Expression 2 becomes true.

$$|openStreams(p)| \geq \Delta_u(p) - |children_{\bar{R}}(p)| \quad (2)$$

[6]The seed is the root of T and it does not have any parent.

To trigger expansion, node p sends a join message to p_h, asking it to join R. Node p also sends the open-stream message received via p_h back to p_h, because node p does not have enough upload bandwidth to serve this specific consumer.

Since the expansion condition is typically detected upon the reception of an open-stream message by p, the bandwidth headroom theoretically needs only to be of one block for each child p_h ***out*** of R. Indeed, as soon as p_h goes into ***joining*** state, it stores all blocks it receives. In addition, it asks for the media from the first block to fill in the gap. As a consequence, p_h is able to serve new open-stream messages.

Node $p \in \bar{R}$, which did not detect an upload bandwidth threshold upon the reception of an open-stream message, adds this new request to the list of its $openStreams(p)$. Node p attempts an expansion of R towards itself when it detects *download bandwidth shortage* expressed in Expression 3.

$$|openStreams(p)| \geq \Delta_d(p) \tag{3}$$

To trigger expansion, node p goes in ***joining*** state, and it accepts the received open-stream message.

Latency is another reason for expanding replication scheme R. Let p be a consumer ***out*** of R. Upon reception of the first block of media M, p checks how far it is from the nearest replica in the path to the seed, using information piggybacked on block messages. If this distance is greater than a *latency threshold* Δ_l (Expression 4), consumer p joins R.

$$distance(p, R) > \Delta_l \tag{4}$$

In all these situations, new replica p_r in ***joining*** mode continues to route media blocks to consumers downstream of it, but now stores them locally. As soon as p_r can take over the streaming to a consumer, it does so and asks its parent p_p to stop sending blocks to that specific consumer. If process p_p is not in R, it will forward this close-stream request further to its parent.

Finally, if replica p_r in ***joining*** mode receives a close-stream message from a consumer but it does not serve this consumer, it stops routing media blocks to that consumer but in order to speed up the procedure of fulling in its copy of media M, it does not forward this close-stream message further to its parent.

Contracting R. Intuitively, a contraction of R will be attempted when the risk of bandwidth shortage disappears at one of its replicas p_r (excluding the seed). That is, replica p_r either ***in*** R or ***joining*** R periodically attempts a contraction of R towards itself when Expression 5 becomes true.

$$|openStreams(p_r)| < \Delta_d(p_r) \tag{5}$$

If Expression 5 becomes true, replica p_r switches to ***leaving*** state. During the time that replica p_r is ***leaving***, it does not accept new open-stream requests and just forwards them towards the seed to its parent. But, it should continue streaming to its current consumers. As soon as it finished serving its current consumers, it can leave R and change its state to ***out***. Finally, the partitioned nature of ReStream implies that each replica can decide to leave R *independently*. For this reason a problematic situation can occur i.e., all replicas might leave R, in which replication scheme R would then remains empty. To avoid this special situation, we assume that the *seed* cannot leave R and all the nodes always know the direction to the seed.

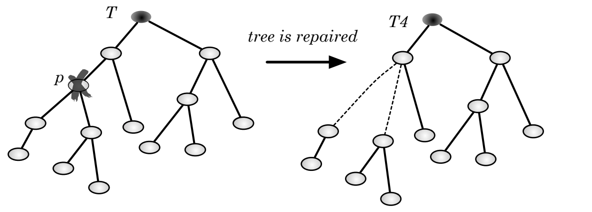
Note that in practice, it might not be a good idea to start contracting R *immediately* after detecting the contraction condition. In some cases indeed, ReStream might enter into a series of wasteful contraction-expansion cycles. For this reason, when a replica detects the contraction condition, it delays the contraction attempts and rechecks the condition a while later. Contraction will actually be triggered only if the condition remains true after that delay. Note also that with ReStream, if replica p_r is also a *consumer*, it is not allowed to leave R. This introduces a certain level of fairness into the system: while replica p_r is a consumer and is being served by other replicas, it cannot leave R and must serve other consumers.

C. Coping with failures

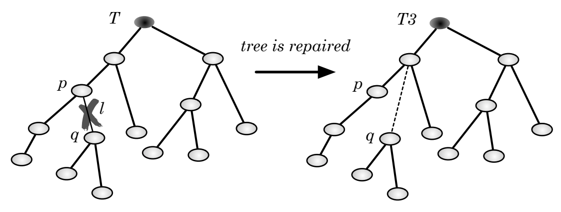
When it comes to reliability, we consider crash-recovery failure model. When a node or link fails (permanently or temporarily), tree overlay T is partitioned into two or more subtrees and the first step consists in repairing it. This is the responsibility of the underlying communication layer, which is also in charge of detecting failures and notifying the upper layer, in our case ReStream, of changes in T's structure. Among the subtrees resulting from the failure, only one remains connected to the seed.[7] So repairing T simply consists in reconnecting all subtrees disconnected from the seed to the node immediately after the failed node or link, in the direction of the seed. This is illustrated in Figure 3.

As soon as T is repaired, all nodes whose neighborhood changed due to the failure are notified by the underlying communication layer. Observe here that whether the failure impacted some node p, or some link l that connected node p in the direction of the seed, all disconnected subtrees will be reconnected to p's parent (see Figure 3). So, if p was not a replica, its parent simply resumes streaming to all consumers in the reconnected subtrees, either as replica or simply as routing node. If p was a replica however, its parent must either take over its role, if it is a replica itself, or send open-stream requests upstream towards to seed,

[7]We assume that seeds never fail; in practice, this means that they are hosted by some robust server infrastructure, typically managed by the stream provider, e.g., Netflix, Apple, etc.



(a) node failure



(b) link failure

Fig. 3. Coping with failures

in order to resume streaming to consumers in the reconnected subtrees. In addition, these changes might of course trigger further expansion or contraction of R in subsequent steps of the ReStream algorithm.

When a node recovers after a failure, it will be reconnected as a leaf to one of its geographical neighbors by the underlying communication layer. When a link recovers, it will not be reused in the structure of current recovered tree immediately but may be used later, when recovering from further failures.

V. Performance evaluation

In order to assess the performace of our ReStream algorithm, 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 consider only one *seed* in the network and perform experiments with 100 to 1000 processes on a *geographical tree network* rooted at the seed, using the Sinalgo simulation framework.[8]

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 (see Section II): in each step at most one block is transmitted to each consumer. We also assume when a consumer starts to get the first block, it should receive the entire media in a predefined period of time. If during this period, it does not receive the whole media, it asks to stop streaming. This assumption is necessary to evaluate the performance of the algorithm precisely. The simulation continues until all the consumers have received the entire movie or have stopped watching it by sending a request-leave 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 our simulation, we assume that the latency threshold is 3 hops ($\Delta_l = 3$ *hops*). In addition, to simplify the interpretation of our results and make it comparable with our previous work, we consider that processes and links are reliable, but this time processes could have different download and upload bandwidth capacities ($\Delta_d(p)$, $\Delta_u(p)$). Different download and upload bandwidth capacities offered in our network are shown in Table I. These bandwidth capacities are typically offered by Internet Service Providers (ISPs) in Western Europe. We assume that the seed has the maximum download and upload bandwidth capacity.

$down/up$ ($kbits/sec$)	$down/up$ ($blocks/step$)
5,000/500	5/1
25,000/2,500	25/3
50,000/5,000	50/5
100,000/7,000	100/7

TABLE I
AVAILABLE DOWNLOAD/UPLOAD BANDWIDTHS

A. Convergence to the optimal replication scheme

We now show that ReStream tends to converge 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 replica (the seed) and after a while, the number of replicas stabilizes. To show that the resulting replication scheme tends towards the optimal, our results are compared to a *centralized algorithm* that finds the optimal replication scheme for the same set of static consumers. Indeed, Expression 1 is a rather simple optimization problem when it comes to solve it in a centralized manner.

Table II summarizes the results of comparing ReStream with the optimal centralized algorithm for networks of different sizes. Each experiment was repeated 10 times, i.e., for each network size we created 10 different networks and executed both algorithms. Although in these executions ReStream does not end up with a replication scheme containing exactly the same number of replicas as the centralized algorithm, it always ends up with the number of replicas about 1.5 times of the replicas in the centralized algorithm.

To understand why ReStream algorithm does not completely converge to optimal R, first note that initially, all consumers tend to receive the media from

[8] *http://dcg.ethz.ch/projects/sinalgo*

$\lvert\Pi\rvert$	$\lvert R_{opt}\rvert$	$\lvert R\rvert$	$\lvert R\rvert/\lvert R_{opt}\rvert$	*steps to converge*
100	8.9	14.4	1.62	1176.1
200	17.2	24.4	1.42	1373.2
300	25.1	37.6	1.50	1183.3
400	33.9	50.0	1.47	1180.8
500	40.5	61.0	1.51	1400.2
1000	80.5	115.8	1.44	1382.8
2000	160.4	232.5	1.47	1469.3

TABLE II
AVERAGED RESULTS FOR VARIOUS NETWORK SIZES

seed but, as bandwidth is limited, some intermediate nodes will decide to join R. These new replicas then start to serve other replicas and consumers. Among these consumers, some will in turn detect a potential latency problem and therefore decide to join R as well.

However, having any consumer with a latency problem become a replica is not the best choice to minimize the size of R. Rather, consumers joining R for a latency problem should be located as close as possible to the seed, because they can serve more consumers and therefore minimize the size of R. But for consumers with latency problems to agree on who should join R, we need some kind of agreement protocol, which would then compromise the fully distributed nature of RESTREAM.

Finally, in Table III we compare RESTREAM with SCALESTREAM, an earlier replication algorithm we proposed for streaming which does not support partitioning [4]. In comparison of these two algorithms it is shown that as the number of nodes in the network increases, the size of R in our previous algorithm (SCALESTREAM) increases sharply, which is not the case in RESTREAM algorithm. This benefit happens because of the *partition nature* of our replication scheme and this shows the *scalability* of RESTREAM algorithm. The scalability of our algorithm is also visible in the number of steps to converge to optimal R. As it is shown in Table III, independent of the number of nodes in the network, the replication scheme always converges to optimal one in steps around 2 times of the size of media M (in number of blocks).

	SCALESTREAM		RESTREAM	
$\lvert\Pi\rvert$	$\lvert R\rvert$	*steps*	$\lvert R\rvert$	*steps*
100	24.4	262.4	14.4	1176.1
300	168.8	574.4	37.6	1183.3
500	318	861.1	61.0	1400.2
1000	724	1604.3	115.8	1382.8

TABLE III
COMPARISON OF RESULTS OBTAINED IN SCALESTREAM AND RESTREAM ALGORITHMS

B. Dynamic consumption models

In order 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, in order to evaluate our algorithm in a fair manner, our results are compared to two types of static but disconnected replication schemes named *random replication scheme* and *consumers-centric replication scheme*. A *random replication scheme* is a static and disconnected replication scheme in which replicas are selected randomly, whereas a *consumers-centric replication scheme* is a type of random replication scheme in which replicas are chosen randomly among consumers.

These two static replication schemes are *best effort*: when a replica receives a request from a consumer p, it starts streaming to the consumer and sends one block in each step to p. Each block is routed in the path to the destination and it arrives at p if all the intermediate nodes in this path have enough download and upload bandwidth to route this block. Otherwise, the block is dropped.

Geo-dependent consumption model. To implement this consumption model, as explained earlier, 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. Each zone is connected to its direct neighbor zones (each zone has *at most* two direct neighbor zones).

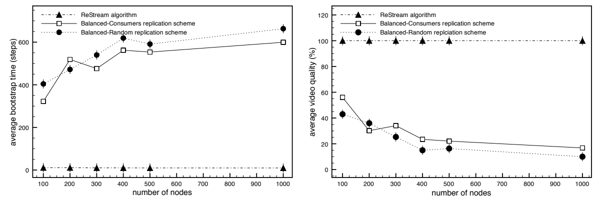
Connected timezones have a one hour difference and 20% of the nodes are selected as *eventual consumers* at the beginning of the simulation. Furthermore, we assume that each consumer starts consuming the movie stream between 7:45pm and 8:15pm in its timezone.[9] Formaly, in one timezone after the other, 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 each node consumes the movie at most once and that only half of the consumers watch the movie until the end. That is, after receiving half of the movie, consumers stop watching it with probability 0.5 (by sending a request-leave message to R).

In order to compare the performance of RESTREAM with the two static replication schemes, we only consider executions where the *average memory cost* for all these three algorithms is roughly equal (the average memory cost is the average number of replicas across an execution). In the following, we compare RESTREAM with the random and the consumers-centric replication schemes in terms of *bootstrap time (latency)* and average *video quality* for networks of 100, 200, 300, 400, 500, and 1000 nodes. Here again, each experiment was repeated 10 times. The *bootstrap time* shows the latency and it 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 *video quality* is the percentage of the number of blocks that are effectively received by a

[9]This is a typical video-on-demand scenario: people in a timezone start watching some newly released movie at roughly the same time.

consumer over the number of blocks that should be received.

Figures 4(a) and 4(b) show our algorithm always exhibits smaller average bootstrap time and more average video quality than two static replication schemes. This advantage of ReStream grows as the number of nodes in the network increases, which shows the *scalability* of our approach.



(a) Average bootstrap time (b) Average video quality

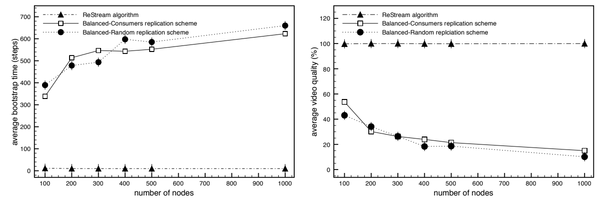
Fig. 4. Results in geo-dependent consumption model

Compared to our previous work, both ReStream and ScaleStream reach good performance when facing geo-dependent consumption. The good performance of our previous system, in spite of its connected nature, is due to the fact that *open-stream* requests come in waves from one region after the other, as consumers spread across timezones. Thus, its connected replication scheme has enough time to expand towards these regions, while contracting from other regions, resulting on average in more video quality and in shorter bootstrap time.

Random consumption model. Evaluating our algorithm on random consumption model gives us a measure to how adaptive ReStream 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.

ReStream is again compared to the random and the consumers-centric replication schemes in terms of bootstrap time (latency) and average video quality, only for executions where the average memory cost for all these algorithms is roughly equal. Results are presented in Figures 5(a) and 5(b) and show that ReStream reaches almost the same results as in geo-dependent consumption model, contrary to our previous system.

This achievement shows that although consumers are selected completely randomly and thus end up all over the network, the replication scheme R can quickly adapt to this new situation. As a consequence, achieving the good performance in ReStream (whether in geo-dependent or random consumption) is *independent* of the nature of consumption models and this good achievement comes thus due to the *partitioned* nature of the ReStream replication scheme. This aspect of ReStream is quite precious in today's globalization of multimedia consumption over the Internet.



(a) Average bootstrap time (b) Average video quality

Fig. 5. Results in random consumption model

VI. Related works

To support the growth of media streaming, research efforts have mainly focused on routing algorithms. Typically, a routing solution consists in building and maintaining an *overlay* through which the stream blocks are routed. These solutions are known as *overlay-based* solutions. *Tree* and *mesh* are two widely studied overlay topologies in P2P networks. A single tree is an acyclic graph that leads to loopless routing. This simple routing scenario causes some of the overlay-based solutions rely on a tree topology for their routing process [13], [16], [2], [8], [17], [6], [9], [21]. But tree-based data topology cannot fairly use the network resources because the bandwidth of leaf peers is not utilized in this topology [7].

To overcome this limitation, *multiple tree-based data topologies* were proposed. CoopNet, SplitStream, and MutualCast are examples of these systems [22], [8], [14]. In CoopNet and SplitStream, content is partitioned into multiple stripes, each distributed over a different tree. That is, the leaf peers of one tree are made intermediate peers of another tree to construct more efficient system that could potentially take advantage of all resources in the P2P network [7].

In tree-based algorithms, routing is done globally, while in *mesh-based* systems, delivery decisions are made locally at each peer [7]. This property causes that mesh-based systems are being widely deployed in many popular P2P applications like Bit-Torrent[10], PPLive[11] and UUSee[12]. Our ReStream algorithm is also complementary to mesh-based systems.

These routing solutions work well when available resources are sufficient to serve all the consumers in the system. However, when resources at the multimedia server fall short and no longer allow it to support the growing number of consumers, routing solutions alone

[10]http://www.bittorrent.com
[11]http://www.pplive.com
[12]http://www.uusee.com

cannot prevent bottlenecks. P2P *caching* and *replication* are solutions to cope with increasing number of consumers in P2P networks [10], [23], [24], [5]. P2P caching and replication reduce origin server load, network bandwidth usage, and also client-side latency. These benefits lead to improve scalability, reliability and performance of the system with lower cost [7].

In a replication solution, the replica placement depends on some cost function to optimize. For example, in [26] the replication scheme adapts to the read-write pattern. In [3], the objective is to adapt to the read-write pattern in unreliable environment. The approach defined in [15], [19] also takes into consideration storage costs and node capacity to serve requests. In [11], the replication aims at balancing load in terms of CPU and disk utilization in order to increase the system throughput. In [25], the replica placement strategy aims at minimizing the average read and write latency and the bandwidth usage to enforce consistency between replicas. In SCALESTREAM, the replica placement strategy aims at bandwidth optimization [4]. Similar to these previous replication solutions, RESTREAM can be considered as a type of adaptive replica placement that specifically aims at bandwidth and latency optimization.

VII. CONCLUDING REMARKS

In this paper, we proposed RESTREAM, a scalable and reliable algorithm based on replication to support multimedia streaming. We illustrated its benefits by first showing that it converges towards an optimal replica placement in static consumption models and by then showing that its partitioned nature helps to cope with failures and to achieve good performance in different types of dynamic consumption models. In terms of reliability, we could of course go further: for example, rather than fixing the latency threshold, we could adapt it based on the reliability of nodes and links along the path from the consumer to the replication scheme. Our next steps will also consist in investigating how underlying layer can be built to maximize performance and reliability. In addition, we are working on real implementation of RESTREAM to test it in real streaming scenarios. To achieve this goal, we are implementing an experimental testbed based on RESTREAM on top of PlanetLab Europe.[13]

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. *NCA'09*.

[2] M. Allani, B. Garbinato, F. Pedone, and M. Stamenkovic. A gambling approach to scalable resource-aware streaming. *SRDS'07*.

[3] M. Allani ans B. Garbinato, A. Malekpour, and F. Pedone. Reliable communication infrastructure for adaptive data replication. *DOA'09*.

[4] S. Ataee, B. Garbinato, M. Allani, and F. Pedone. ScaleStream: An adaptive replication algorithm for scalable multimedia streaming. *NCA'11*.

[5] et al. B. Zhao. Tapestry: a resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 2004.

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

[7] J.F. Buford, H. Yu, and E. Keong Lua. P2P networking and applications. *Elsevier Inc.*, 2009.

[8] M. Castro, P. Druschel, A. M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. *SOSP'03*.

[9] Y. Cui, B. Li, and K. Nahrstedt. oStream: asynchronous streaming multicast in application-layer overlay networks. *IEEE Journal on Selected Areas in Communications*, 2004.

[10] F. Dabek, E. Brunskill, F. Kaashoek, and D. Karger. Building peer-to-peer systems with chord. *HotOS'01*.

[11] S. Elnikety, S. G. Dropsho, and W. Zwaenepoel. Tashkent+: memory-aware load balancing and update filtering in replicated databases. *EuroSys'07*.

[12] B. Plattner J. Yan, W. Mhlbauer. Analytical framework for streaming over tcp. *TIK Report, No. 333*, 2010.

[13] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and Jr. J. W. O'Toole. Overcast: reliable multicasting with on overlay network. *OSDI'00*.

[14] P.A.Chou andC.Zhang J.Li. Mutualcast: an efficient mechanism for content distribution a p2p network. *SIGCOMM'05*.

[15] K. Kalpakis, K. Dasgupta, and O. Wolfson. Optimal placement of replicas in trees with read, write, and storage costs. *IEEE Transactions on Parallel Distributed Systems*, 2001.

[16] D. Kostić, Rodriguez A., J Albrecht, and A. Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. *SOSP'03*.

[17] F. Liu, X. Lu, Y. Peng, and J. Huang. An efficient distributed algorithm for constructing delay and degree-bounded application-level multicast tree. *ISPAN'05*.

[18] B. Garbinato M. Allani, J. Leito and L. Rodrigues. Rasm: Reliable algorithm for scalable streaming. *PDP'10*.

[19] J. MacCormick, N. Murphy, V. Ramasubramanian, U. Wieder, J. Yang, and L. Zhou. Kinesis: A new approach to replica placement in distributed storage systems. *TOS'08*.

[20] A. Malekpour, F. Pedone, M. Allani, and B. Garbinato. Streamline: An architecture for overlay multicast. *NCA'09*.

[21] L. Mathy, R. Canonico, and D. Hutchison. An overlay tree building control protocol. *NGC'01*.

[22] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. *NOSSDAV'02*.

[23] S. Ratnasamy, P. Fancis, M. Handley, and R. Karp. A scalable content-addressable network. *SIGCOMM'01*.

[24] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *Middleware'01*.

[25] S. Sivasubramanian, G. Alonso, G. Pierre, and M. van Steen. GlobeDB: Autonomic data replication for web applications. *WWW'05*.

[26] O. Wolfson, S. Jajodia, and Y. Huang. An adaptive data replication algorithm. *ACM Transactions on Database Systems*, 1997.

[13]http://www.planet-lab.eu/