Computer Networks 53 (2009) 153–167

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

A performance evaluation of scalable live video streaming with nano data centers

Jiayue He [a], Augustin Chaintreau [b,*], Christophe Diot [b]

[a] Princeton University, 35 Olden Street, Princeton, NJ, 08544, United States
[b] Thomson, 46 Quai A Le Gallo, 92648 Boulogne, France

ARTICLE INFO

Article history:
Available online 25 October 2008

Keywords:
Nano data centers
Live video streaming
Peer-to-peer technology
TCP tandem
Scalability

ABSTRACT

To improve the efficiency and the quality of a service, a network operator may consider deploying a peer-to-peer architecture among controlled peers, also called here nano data centers, which contrast with the churn and resource heterogeneity of peers in uncontrolled environments. In this paper, we consider a prevalent peer-to-peer application: live video streaming. We demonstrate how nano data centers can take advantage of the self-scaling property of a peer-to-peer architecture, while significantly improving the quality of a live video streaming service, allowing smaller delays and fast channel switching. We introduce the branching architecture for nano datacenters (BAND), where a user can "pull" content from a channel of interest, or content could be "pushed" to it for relaying to other interested users. We prove that there exists an optimal trade-off point between minimizing the number of push, or the number of relaying nodes, and maintaining a robust topology as the number of channels and users get large, which allows scalability. We analyze the performance of content dissemination as users switch between channels, creating migration of nodes in the tree, while flow control insures continuity of data transmission. We prove that this p2p architecture guarantees a throughput independently of the size of the group. Analysis and evaluation of the model demonstrate that pushing content to a small number of relay nodes can have significant performance gains in throughput, start-up time, playback lags and channel switching delays.

1. Introduction

The feasibility of live streaming over peer-to-peer (p2p) networks have been demonstrated in recent years with commercial applications such as PPLive and PPStream[1], attracting millions of users. However, recent measurements study of PPLive shows that a mesh-pull p2p streaming architecture depends on a small number of well connected participating nodes (c.f. superpeers, supernodes, amplifiers) and incurs important start-up delay, playback lags and time for channel transitions [1]. Supporting streaming service with comparable quality and latency as traditional television broadcast remains an open problem, with the exception of a few national television channels distributed by ISPs using IP-multicast and dedicated bandwidth. The performance attainable via p2p architecture for live streaming of many channels with varying popularity remains unclear.

In this paper, we show that the above issue can be addressed in the context of a controlled p2p environment: a large set of *nano data centers* (devices similar to home gateways and set-top boxes) that are deployed by ISPs as part of their connection services [2–6]. Nano data centers have several features. They are assumed to be always on regardless of the content the user is actually watching (as for home gateway today). As one option, a node may "push" content when necessary to another nano data center belonging to a user that is not currently viewing it. As ISPs have control over their own network design, one can

* Corresponding author.
E-mail address: augustin.chaintreau@thomson.net (A. Chaintreau).
[1] See www.pplive.com and www.ppstream.com.

1389-1286/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2008.10.014

expect the bandwidth available per node to be approximately homogeneous, and sufficient for delivery of a few high-quality video streams. In this controlled environment, the only system dynamics are introduced by channel switching of the users and short-term congestion from background traffic. Our goal is to adapt the p2p content delivery to the requests of users across all channels, while minimizing the load for each nano data center.

This paper presents BAND: a distributed solution to quickly reconfigure the p2p delivery trees, allowing users to browse between channels as they watch them. We focus here on a single delivery tree per channel, since we do not expect to deal with the high user churn characteristic of uncontrolled environments. Nevertheless, the technique extends naturally to the case where a channel uses multiple delivery trees. In BAND, first a delivery tree is built among all nodes, regardless of the interest of users in the specific channel. Then, users requesting the channels will populate a subtree that is going to be used for actual content delivery. The key insight is to select a certain number of nodes where content is "pushed", and who will participate in the delivery although they are not "pulling" content from the channel.

Our contributions are two-fold:

- *Design*: We analyze the trade-off between the overhead of the push strategy and the scalability of the system, by introducing a branching node approach. A node is a k-branching node if, among the subtrees rooted in all of its children, at least k contain a node requesting the channel. We prove that choosing $k = 2$ is optimal as it minimizes the amount of content push while keeping the system scalable. With this strategy, the load scales with the popularity of the channel; the savings are particularly useful in the typical situation where there are many unpopular TV channels and a few popular ones [1].
- *Performance evaluation:* Since our solution reconfigures the delivery tree quickly, one important issue is the impact of tree reconfiguration on the ability for nodes to consistently receive data. We prove that a simple buffering scheme, which uses TCP congestion control together with a local back-pressure algorithm, can ensure data continuity under node migration. Using a model of last-passage percolation, we study the performance of this scheme for a large group of nodes, allowing for channel switching and traffic dynamics. We prove scalability of this scheme: for any buffer size, throughput is lower bounded independently of group size. Then through simulation, we show that the ratio between the throughput of a large group and that of a single dedicated unicast connection is typically lower bounded by 65%. In addition, we find that the delay grows logarithmically with the number of users requesting a channel.

To the best of our knowledge, this paper is the first to characterize how node migration in the delivery tree affects continuity of data delivery and performance of peer-to-peer live streaming. These results significantly extend a recent methodology based on properties of directed graphs, that was only previously applied to static trees [7,8]. Our results prove that pushing content to a small number of well selected nodes, as made possible in controlled p2p environment, can have significant performance gains to ensure high throughput, as well as small start-up times and playback lags.

The rest of the paper is organized as follows. We place our contributions in context of the extensive literature on p2p in Section 2. We introduce the key architectural components in Section 3 and analyze the trade-off between overhead and scalability in Section 4. Section 5 describes our flow control scheme. In Section 6, a last-passage percolation model of the system is introduced, analyzed and simulated. We conclude and point to future work in Section 7.

2. Related work

Scalable live streaming delivery to large groups has been a long standing research challenge for the last fifteen years. Peer-to-peer (p2p) architectures have emerged as the leading candidate for two reasons: they do not require deploying multicast communication protocol inside routers, and their capacities grow with the number of participating users [9]. Indeed the p2p paradigm has been already widely adopted for asynchronous file sharing; the most successful architecture to date relies on exchanges of chunks of data between peers, orchestrated by an incentive mechanism usually called "swarming" [10]. It is called mesh-pull as data may be delivered using different paths following users' requests. This architecture proved more robust than trees for file sharing because it maintains connectivity under high users churn and heterogeneous network conditions, while preventing free-loaders.

Following this success, mesh-pull peer-to-peer architectures have recently been applied to live streaming [11,12], where there are now widely adopted. However, despite promising early adoption by millions of users, these latest p2p live streaming schemes appear less mature as observed in various measurement studies [1,13–15,12]. First, in contrast with file sharing, they do not implement incentive mechanisms; recent studies have shown that they rely on a relatively small number of "amplifier" nodes [1,13,15]. These well connected nodes are critical to the system and support the majority of the system workload as observed as well in [16]. Our solution, in contrast, does not rely on nodes with high capacity and applies well to a homogeneous setting. Second, the connections among peers of a mesh-pull architecture, established for a short-term via exchanges of chunks, becomes significantly more complex to orchestrate when deadlines of content delivery are involved [12,14]. In practice, this problem can be overcome but it implies necessarily two things: possible overloading of certain links and nodes which reduces the quality of the video stream, and a significant start-up delay and playback lags needed to watch the live content [12,17]. Consequently, the service is essentially focusing on a single channel and a significant time is needed to switch between different channels.

It is now subject to debate in the p2p research community how to significantly improve p2p live streaming to reach the quality required for regular TV service, while offering fast switching between multiple channels. Some argue that the better option is to improve mesh-pull (e.g. building incentives [18], a better scheduling strategy [19,20], or using additional server bandwidth wisely [15,17]). Others claim that p2p architectures where content is pushed through one or several delivery trees can be revisited to address the above concern. Some results show that the limitations of these schemes with regard to churn and heterogeneity could be circumvented [21,22]. Meanwhile, several works propose to combine push techniques with current mesh-pull schemes and show it brings significant improvement of delay and quality [12,23,24].

Our work addresses the complex issues surrounding p2p live streaming through a new approach. First, we follow the observation that a p2p live streaming service may be deployed among nodes of a controlled ISP network, similar to home gateways, as opposed to volatile hosts in the Internet. Indeed it has already been observed that ISP and p2p may cooperate for improved performance [25]. Several recent works proposed a similar approach to improve p2p architecture, for video-on-demand [3,2,4,6], or live streaming [5]. Second, since users' churn and heterogeneity between hosts are significantly reduced in the controlled environment, architectures based on delivery trees can be an attractive alternative to mesh-pull. Still, in the presence of a large number of channels with varying popularity and users switching between them, the delivery trees should adapt to users demands so as to avoid congestion in hosts. Third, to keep a p2p architecture like that one transparent to the network and collaborating with others flows, traffic should adapt using the regular TCP protocol. To avoid overflow and/or loss of quality in the content, flow between p2p nodes should be controlled and adapt as well when the tree is reconfigured. These issues, which are specific to p2p live streaming services with multiple channels, cannot be fully addressed following classical techniques and analysis of p2p architecture based on trees, because of tree reconfiguration occurring as users switch between channels. Nevertheless, this paper prove that the above issues can all be addressed by a distributed solution. This distributed technique does not require any central coordination, super-node, or tracker. We prove that it is scalable: it offers a throughput independently of the size of the network, for any buffer size. In other words, under the condition of a controlled environment to avoid churn, it reproduces at much smaller cost the quality of a traditional television broadcast service.

Alternative solutions to provide live streaming services in a controlled environment rely on either servers [17,15,26] or networks implementing IP-multicast on static trees with bandwidth reservation. These offer good quality, but are typically limited in terms of available channels. This paper establish that a controlled p2p environment is a meaningful paradigm for scalable multi-channel live streaming: the operator is responsible for maintaining the participating peers, but peers are nevertheless cooperating independently of the network or any centralized unit. It offers the ability for an ISP or a company to provide live streaming service with diverse content, without the overhead of a mesh-pull mechanism or advanced coding techniques such as [27], and without the need to scale up its server capacity with the users. Users may decide to adopt such a system via their ISP, as opposed to an uncontrolled mesh-pull system, to receive higher quality content and reduced start-up delay and jitter. ISPs may have incentive to propose such collaborative p2p services as it can improve the locality of their peer-to-peer traffic to minimize content sent over cross-ISPs links (see e.g. [5]).

3. P2P-enabled nano data centers

In this section, we first outline the features of nano data centers that guided the design decisions for the BAND architecture. We then give a brief overview of BAND.

3.1. Nano data center features

Nano data centers is an emergent example of a controlled environment and it has several characteristic features.

- *System dynamics:* In the Internet, users of a live streaming service are assumed to be "on" only when watching a particular content stream, therefore they arrive and depart frequently and unpredictably. Nano data centers are assumed to be on regardless of whether the user is watching content. There may be minor off times due to equipment failure, but such cases are extremely rare. We assume users do not turn off their set-top boxes manually, because they rely on it for other services (e.g., VoIP phone). In this controlled environment, the only system dynamics are introduced by channel switching and short-term congestion from background traffic.
- *Cooperation model:* Traditionally, users are assumed to use the "pull" model, i.e., they will only relay content that they are currently watching. In nano data centers, there is the option to "push" to a nano data center content that is not currently viewed by that user, if it serves the purpose of the service. We wish to push content as little as possible, as each of these operations adds load for this user which is not directly related with its request. Later, we prove that pushing content cannot be avoided in such systems but that the amount of contents that needs to be pushed remains small. In order for users to choose a cooperative service, as opposed to a non-cooperative one, a service provider can propose a discount to access the service. As an example, some french ISPs propose free wireless upgrade to any user that joins the FON network.[2]
- *Network resources:* Unlike the Internet, nano data centers tend to be quite *homogeneous* in their capabilities. As ISPs have control over their own network design, there should be *sufficient* bandwidth provisioned in the sys-

[2] See www.fon.com.

tem for delivery of a few high-quality video streams. Note that high-quality video streams use more bandwidth, hence we cannot assume that some nodes (c.f. superpeers) relay a large number of streams.

The key features of the nano data centers significantly shape the choices made when designing an architecture to stream live video in this controlled environment, which we describe in the next subsection.

3.2. Architecture overview

Branching architecture over nano data centers (BAND) consists of the following components: initial tree construction, tree reconfiguration as users switch between content streams, and flow control to handle temporary network congestion (from background traffic) and continuity of data delivery. The first component is described in the rest of this section. The second component creates an "active" tree on top of the initial tree, using content push, it is described below and in more details in Section 4. The third component combines TCP, together with finite buffers implemented in each peer, and a local back-pressure rule to ensure content delivery under temporarily network congestion and node migration. It is described in Section 6.

BAND constructs multiple overlay multicast trees, one per channel; every user belongs to all the trees, but is only "pulling" content from one tree at any given time. In general, even mesh-pull based approaches implicitly construct delivery trees, but at a much finer granularity: per slice, per block or per packet [28]. Fine-grained delivery trees are necessary to maintain connectivity in the face of high user churn, but at the cost of continuous tree reconstruction. Since nano data centers have low user churn, constructing trees per channel is both sufficient for connectivity and much more efficient than finer granularity options. The initial tree construction mechanism can be any one of a number of scalable tree construction mechanism, already well studied, e.g., [29–31,22,12]. The overlay hop-lengths in the tree should reflect real network properties such as delay or IP routing hops. BAND avoids wasting sending bandwidth of the leave nodes by having each tree rooted in a different node, so leaves nodes in one tree are internal nodes in another tree (see for instance [21]).

On top of each tree, an "active" tree containing all the nodes which are "pulling" content is then constructed, the tree can also contain nodes which act as relays between the root node and "pulling" nodes. In a traditional overlay multicast tree, if a node on the path between source and destination is not pulling data, it still acts as a relay that forwards the data. In other words, the content is pushed automatically to this node. Asking nodes to relay small amounts of data is quite reasonable, and suitable for low data rate applications such as message boards. For a streaming application, however, where we expect high data rates, asking a node to forward data when they are not pulling becomes a much bigger strain on network and node resources [32]. In the next section, we study how to reduce this amount of push content and how it impacts the topology of the p2p network.

4. Reconfiguration of trees with minimum overhead

As users decide with time to watch different channels, the peers pulling a data stream change. We show in this section how BAND handles such events with limited overhead. As already described, we assume that all peers belong to one rooted tree whether they actually pull the data from this channel or not. A peer that pulls data from this channel is referred as a pulling peer, or pulling node. All pulling peers may participate in the delivery tree, and, in addition, we may choose to push the content to some other peers as well. We wish to determine efficiently a subtree to deliver the content to all pulling nodes while using the smallest number of other nodes.

4.1. Branching nodes

For any $k \geqslant 1$, we say that a node u is a k-branching node if, among all the subtrees that are rooted in immediate children of u, at least k contains pulling nodes. Any k-branching node is a $(k-1)$-branching node and so on. The largest value of k such that this property holds for u is called the branching factor of u, it is obviously less than its out-degree in the original tree. Note that the 1-branching nodes are exactly all the ancestors of pulling nodes. One may immediately check that a node is a k-branching node if and only if k of its immediate children are the ancestor of a pulling node (i.e. k of its children are 1-branching nodes). Following this observation it is easy for nodes to compute their branching factor, starting from the leaves of the tree up to the root.

For any value of k, and any subset of pulling nodes we define the delivery tree $\mathbb{T}_k$ as follows: it contains all the pulling nodes and the k-branching nodes, and each node in $\mathbb{T}_k$ is directly connected with its closest ancestor in the tree that is also in $\mathbb{T}_k$. An example of how to construct such a delivery tree when $k=2$ is shown in Fig. 1 (to make the presentation easier to read, a node with branching factor 0 is called "inactive", a node with branching factor 1 is called "semi-active", while all 2-branching nodes and all pulling nodes are all called "active" nodes).

We may summarize these procedures as follows: each node keeps a *state*: "inactive", "semi-active" or "active", maintained using the messages received from its children in the tree. In Figs. 2 and 3, white nodes are inactive, black nodes are active, and half-black nodes are semi-active. Only the active nodes participate in the dissemination tree. As seen in Fig. 2, when a node s starts to pull a data stream, it finds its closest ancestor t that is not inactive, and connects to it. If t is semi-active, then it also has to become active and join the dissemination tree, along with s. The detailed connection scenarios are described in pseudocode in Fig. 1a and correspond to the three scenarios in Fig. 2. As seen in Fig. 3, when a node s stops pulling data, it is able to remove itself from the delivery tree unless two of its children are semi-active/active. When s decides to stops pulling, it is possible that the node it was receiving from also leaves the tree if it only has two active/semi-active children. The detailed connection scenarios are described in Fig. 1b and correspond to the scenarios in Fig. 3.

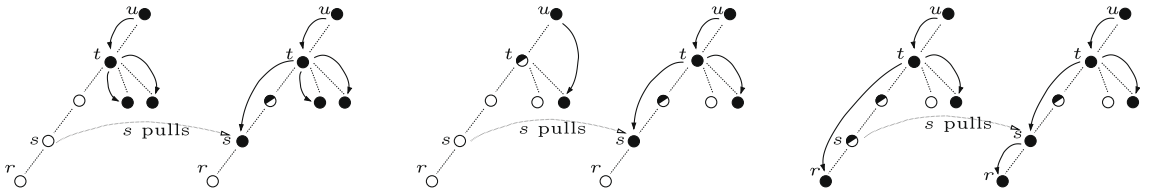
```
/* s starts unconnected */
/* s wants to pull */
if (state(s) == "inactive"){
 t = parent(s);
 while (state(t) == "inactive") {
  state(t) = "semi-active"; single-child(t) = s;
  t = parent(t);
 }
 connect s to t;
 if (state(t) == "semi-active"){
  state(t) = "active";
  let r = single-child(t);
  let u = closest active ancestor of t;
  disconnect r from u;
  connect r to t; connect t to u;
  }
}elseif (state(s) == "semi-active"){
 let t = closest active ancestor of s;
 connect s to t;
 let r = single-child(s)
 disconnect r from t;
 connect r to s;
}
state(s) = "active";
```

(a) start-pull

```
/* s starts connected to t */
/* s wants to stop pulling */
if (s serving no children) {
 disconnect s from t;
 state(s) = "inactive";
 c = parent(s);
 while (state(c) == "semi-active") {
  state(c) = "inactive";
  c = parent(c);
 }
 if (t has two children including s){
  let u = closest semi-active/active ancestor of t;
  /* r and s are active children of t */
  disconnect r from t;
  connect r to u;
  state(t) = "semi-active"; single-child(t) = r ;
 }
}elseif (s serving one child){
 /* r is the active child of s */
 disconnect r from s;
 connect r to t;
 disconnect s from t;
 state(s) = "semi-active"; single-child(s) = r;
}
```

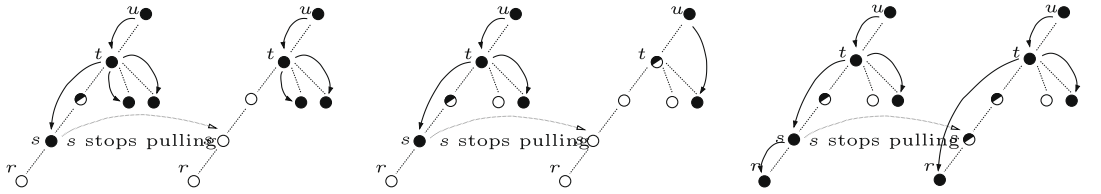
(b) stop-pull

Fig. 1. Pseudocode describing processes to start and stop pulling ($k = 2$).



(a) s inactive and t is active (b) s inactive and t is semi-active (c) s is semi-active

Fig. 2. Three possible scenarios when a node s starts pulling ($k = 2$).



(a) s has no children, t at least two other children (b) s has no children, t only one other child (c) s has one child

Fig. 3. Three possible scenarios when a node s stops pulling ($k = 2$).

The procedures we have presented to start and stop pulling information have low overhead. The storage overhead for a node is constant; it contains the state of the node (inactive,semi-active,active), the node it is receiving from, the nodes it is sending to (bounded by the out-degree), and its unique child when it is semi-active. The overhead of message passing is linear in the depth of the tree or $O(\log N)$ for N nodes in the tree. This is because in the worst case, a node will need to visit every ancestor (including the root). Hence, we can expect to complete each of these procedures very quickly.

4.2. Analysis of topological properties

The delivery tree $\mathbb{T}_k$, for different values of k, is such that only pulling nodes and k-branching nodes are included in the tree. In BAND we choose to set $k = 2$, and the results of this section along with the evaluation in Section 6.4.3 justifies why this is the best possible choice.

First, note that choosing different values of k allows describing a spectrum of intermediate solutions between two extreme strategies: When k is equal to 1, then a node relays data if and only if it is ancestor of a pulling node.

Therefore, all nodes on a path between the source and a pulling node are participating. This strategy is generally the default option for multicast overlay, we call it "traditional overlay multicast". As k increases, the number of nodes that are forced to push content reduces. In particular, once k is increased above the max out-degree of the tree, only pulling nodes participate in data delivery, as the branching factor of a node is always less than its out-degree in the original tree. We call this strategy "selfish" because it requires no content pushing.

Increasing k always reduces the amount of nodes in the delivery tree. It may appear at first that setting k arbitrarily large is optimal. The following result proves that this is not scalable: when the system grows, the out-degree of some nodes may become arbitrarily large. This phenomenon occurs as soon as $k > 2$ and is prominent for non-popular groups containing only a few pulling nodes. In contrast, choosing a value of $k \leqslant 2$ guarantees a bounded out-degree for arbitrary large trees and arbitrary groups.

Proposition 1. *We assume that the underlying tree has homogeneous out-degree D and depth H. We consider a subset of pulling nodes and its associated delivery tree $\mathbb{T}_k$, constructed using the k-branching rule.*

- *If $k > D$ and each node may pull independently with probability $p \leqslant 1 - \frac{1}{D}$, then the expected out-degree in $\mathbb{T}_k$ of any node diverges as the system grows.*
- *If $k > 2$, for any node u with depth i, there exist subsets of pulling nodes such that the out-degree of u in $\mathbb{T}_k$ is greater than $(k-1)^{H-i}$.*
- *If $k \leqslant 2$, then the out-degree of any node in $\mathbb{T}_k$ is less than D.*

Proof. Let us consider a node u with depth i, and denote by d_i the average number of immediate children of u in $\mathbb{T}_k$, provided that u is in $\mathbb{T}_k$. Since $k > D$ only pulling nodes are included in T_k, hence d_i satisfies:

$$d_H = 0 \quad \text{and} \quad d_i = D(p \cdot 1 + (1-p) \cdot d_{i+1}).$$

Hence, we have $d_i = Dp(1 + \ldots + (D(1-p))^{H-i-1})$ which becomes arbitrarily large as H grows when $1 - p \geqslant 1/D$.

To prove the second assertion, consider a partial tree rooted in u, with degree $k-1$ at each level. This partial tree has height $H - i - 1$. Assume that the pulling nodes are exactly all the leaves nodes of this partial tree, and any immediate child of u does not belong to the partial tree. Then no interior node of this partial tree is a k-branching node except u. The leaves of this partial tree are then all connected to u which proves the result.

If $k = 1$, a node is connected in the subtree with exactly all its immediate children that are ancestor of a pulling node, such that there are at most D of them. We now assume that $k = 2$. For two nodes chosen in a tree, there exists in this tree a 2-branching node that is ancestor of both nodes. One can use this fact recursively to show that for any collection of nodes there exists a 2-branching node that is an ancestor of all nodes in the collection.

Let us consider a node u and the D subtrees rooted in all of its immediate children. We may apply the fact above to each subtree and obtain that there exists a 2-branching node that is ancestor of all pulling nodes in this subtree, except if this subtree contains at most 1 pulling node. The tree $\mathbb{T}_k$ for $k = 2$ contains all the 2-branching nodes. Therefore, at most one node from each of the D subtrees is connected with u in $\mathbb{T}_k$, which proves the result. □

The previous result shows that, as our system should handle large depth and maintain bounded degree, choices are limited to $k = 1$ or $k = 2$, as any other degree may diverge for sparse group. Note that $k = 2$ seems always optimal as it involves less nodes in the delivery tree $\mathbb{T}_k$. However it is not obvious a priori whether that brings a significant improvement. The following result demonstrates that it is significant: the number of nodes active when $k = 2$ is directly proportional to the number of pulling nodes, whereas it may be arbitrary large when $k = 1$. As an example, we prove in a random model where we assume that nodes pull independently with a small probability, system load is divided by the degree of the tree.

Proposition 2. *Under the same assumption as Proposition 1, let us denote the number of pulling nodes by n*

- *If $k = 1$, for a fixed n the size of $\mathbb{T}_k$ may diverge as H grows.*
- *If $k = 2$, then the size of $\mathbb{T}_k$ is at most $2 \times n$.*

Let us now assume that every node decides to pull with probability p independently of others. Then for small p we have as H grows $\frac{\mathbb{E}[|\mathbb{T}_2|]}{\mathbb{E}[|\mathbb{T}_1|]} \sim 1/D$.

Proof. When $k = 1$ every ancestor of a pulling node should be included in $\mathbb{T}_1$, a tree with large depth may then contain only n pulling nodes but any arbitrary number of nodes in $\mathbb{T}_1$.

The proof for $k = 2$ may be shown by induction. The results holds trivially if $n = 0$ since the tree $\mathbb{T}_2$ does not contain any node. When a new node starts pulling, it adds at most one other node in $\mathbb{T}_2$ after reconfiguration, as shown in the scenarios illustrated in Fig. 2.

The proof of the last assertion may be found in Appendix A. We calculate the expected number of active nodes for both the BAND and traditional overlay multicast architecture. Then L'Hôpital's Rule is applied as $p \to 0$. □

The topological properties of delivery trees based on the k-branching rule prove that the value $k = 2$ represents the best tradeoff. It is the sole choice for k which combines two properties holding for system of any size: a bound on the degree and on the number of nodes used to deliver content. Interestingly, the savings seem to be most important for the case of unpopular channels.

5. Flow control coping with node migration

From the previous section, we have seen that the branching rule allows efficient delivery of data to the subset of pulling nodes. The delivery tree constructed using the branching nodes creates connections between nodes that are not necessarily neighbors in the underlying tree. As a consequence, this process of node migration may impact delivery of content. In this section, we describe the

mechanisms implemented via local buffers in peers to ensure continuity of data transmission and adapts the rate.

5.1. Back-pressure mechanism and buffers

There are several ways to deliver packets when link loads change over time in the system. We choose a simple solution: combine existing TCP with a blocking back-pressure mechanism in the output buffer. We first review the mechanisms and associated buffers for a static tree.

In a multicast tree, all active nodes need to provision buffers for the incoming and outgoing TCP connections. At a single node, there is one input buffer for the incoming TCP connection that corresponds to the receiver window size of the said connection. There are also several output buffers, one corresponding to each downstream TCP connection. In addition, a backup buffer stores copies of packets removed from the input buffer to the output buffers. These backup buffers can be used when TCP connections are re-established for the children nodes after their parent node fails.

If the buffers are included as part of the TCP connection, there can be three different types of packet losses. Loss that occurs on the path of the TCP connection is handled by the TCP acknowledgment and retransmit mechanisms. Loss due to input buffer overflow will not occur due to flow control of the receiver window in TCP. A blocking back-pressure mechanism can avoid losses due to output buffer overflow [7]. It ensures that a packet is removed from the input buffer only when it can be copied to all output buffers. So if a single output buffer is full, the copying process will be blocked and will only resume once there is at least one space in all output buffers. Such a system does not create deadlock, as will be proved in the next section.

5.2. Impact of nodal dynamics on buffers

Three types of nodal dynamics related to changes in pull may occur in a multicast tree:

- Creating a new connection from an active node to an inactive node, as in Fig. 2a.
- Removing a connection to a node that becomes inactive, as in Fig. 3a.
- Migrating the content delivery from receiving from one node to receiving from another node, as in Fig. 2b and c, Fig. 3b and c.

The buffer structure in Fig. 4 can take care of all the nodal dynamics with a storage buffer at each active node. Since a node may receive data from two different TCP connections simultaneously for a short period of time, there are two input buffers, although each packet will only be sent on one connection. Back-pressure is then implemented in two ways: packets are only allowed to leave an input buffer if the space for this packet is available in the storage buffer of this node, *and* packets are only allowed to leave the storage buffer when sufficient memory is available for active output buffers.

In general, a single incoming TCP connection is sufficient to handle reconfiguration. Two TCP connections are required simultaneously when changing $r \to s$ to $r \to t$, t is an ancestor of s, see Figs. 3b and c. Let the last packet sent from s be m and the last packet sent by t be $m + W_m$, where W_m denotes the window size for packet m. r must get the packets up to $m + W_m$ from s to avoid delaying the other nodes receiving from t, so two simultaneous TCP connections are required. For a brief period, r will receive from both s (up to packet $m + W_m$) and t starting with packet $m + W_m + 1$. s just removes itself from the connection after it receives the ACK for packet $m + W_m$. The other children of t will not experience any delay unless there is back-pressure from the storage buffer of r.

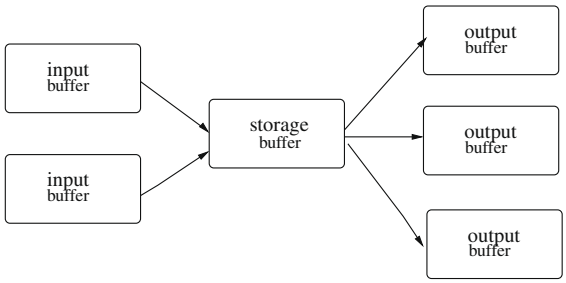


Fig. 4. Buffers required by BAND in a single node.

We will now consider the changes required in output buffers due to branching nodes. In [7], all output buffers were dimensioned to be the same size for each TCP connection for a static tree. This is a reasonable model when all connections span one overlay hop in the multicast tree. With BAND, the overlay hop-length of the TCP connection vary amongst nodes receiving from a sending node. To compensate for this, BAND dimensions output buffers to be proportional to the number of overlay hops between the sending and receiving nodes. Note that we assume the overlay hop-length in a multicast tree is closely correlated to the RTT and this can be easily adapted to reflect RTT instead of overlay hop-length.

6. Performance analysis

In this section, we focus on the performance scalability of BAND in the face of link load dynamics and content switching. First, we introduce a mathematical model of queuing and packet flow, we prove the throughput of this model is lower bounded independent of system size and content switching using an argument of last-passage percolation. Finally, we present some evaluation of large controlled peer-to-peer networks, based on the models we introduced.

6.1. Queuing model of BAND

In this subsection and the next, we build an analytical model to capture packet flows through queues in the system. In our model, we focus on capturing the additive increase, multiplicative decrease of the TCP window evolution. We leave the technical details associated with the retransmission of packets due to packet loss to Appendix A.2. We do not capture timeouts since we expect most losses will be detected by triple-ACKs rather than timeouts for online video

streaming (which have long-lived flows). In [7], the authors find through evaluation that these details do not impact the analytical scalability result for static trees. We do not capture user joins and leaves because they happen at a much longer timescale compared to link load dynamics and content switching. We also assume there is no contention among the packet flows leaving the same node.

In this subsection, we focus on introducing systematic labels for all possible queues in the system. This involves labeling all nodes and all possible TCP connections between any pair of nodes. In addition, we introduce routers to model the nodes in the underlying network as individual queues. We also tune the state parameter introduced in Section 4.1 to precisely define when a connection will exist. This will form the basis for modeling packet flow in the next subsection.

Our notation is similar to [7]. Each node in the tree is labeled as (k,l), the first index k represent its distance to the root node, the second index l numbers the end-systems at the same level. An ancestor i overlay hops above in the multicast tree can be labeled as $(k-i, \mathtt{m}_i(k,l))$. Let each packet be labeled by $1,2,\ldots,m,\ldots$. The fan-out at each node is bounded by a constant D and let the receiving nodes be labeled $(k+i,l',i)$ where $l' \in \mathtt{d}_i(k,l)$. A TCP connection from an ancestor i overlay hops above in the multicast tree to an end system (k,l) is labeled (k,l,i). The routers and buffers in the tree can be labeled as follows:

- Routers of connection (k,l,i) are labeled as index $h = 1,2,\ldots,H_{(k,l,i)}$. The buffer for router h of connection (k,l,i) can be denoted by (k,l,i,h).
- The root node is denoted by $(0,0)$. The storage buffer of the root is denoted by $(0,0,\mathtt{home})$.
- The buffers of node (k,l) are labeled as follows: let $(k,l,i,\mathtt{beg})$ denote the output buffer of node $(k-i,\mathtt{m}_i(k,l))$ and the start of connection (k,l,i); $(k,l,i,\mathtt{end})$ denotes the input buffer of node (k,l) and the end of connection (k,l,i); $(k,l,\mathtt{home})$ denotes the storage buffer of node (k,l). Let the sizes input, output and storage buffers be $B^{(k,l)}_{\mathtt{IN}}$, $B^{(k,l,i)}_{\mathtt{OUT}}$ and $B^{(k,l)}_{\mathtt{HOME}}$ respectively for connection (k,l,i).

Let $s(k,l,m)$ denote the state (as defined in Section 4.1) of node (k,l), for packet m. A node may either be inactive ($s=0$), semi-active ($s=1$) or active ($s=2$). Note that the node with index (k,l) only receives a data packet m when $s(k,l,m)=2$. We assume $s(0,0,m)=2$ for all m.

Definition 1. The connection (k,l,i) is said active for packet m if and only if we have $s(k,l,m)=2$ and $i = \min\{j | s(k-j, \mathtt{m}_j(k,l), m) = 2\}$.

Two cases are possible for each packet m, a node may be receiving packets $s(k,l,m) \geqslant 2$ or not receiving packets $s(k,l,m) < 2$. For the nodes that are receiving packets, each router in the system is modeled as a single server queue containing only packets from the reference connection. The service times for these packets in this queue are random variables describing the impact of cross traffic. Let the service for packet m through router h in connection (k,l,i) be denoted by $\sigma_m^{(k,l,i,h)}$. For a node not receiving packets, the service time is null.

6.2. Last-passage percolation on a graph

In this section, we model packet flows through queues as last-passage percolation on a randomly weighted graph. It may be seen as an extension to max-plus algebra, although this result is self-contained and does not require any familiarity with max-plus algebra. More details about the background of this model may be found in [8].

The model works as follows: there is a graph where each vertex represents a task that needs to be completed and each edge represents preceding constraints between tasks. We generally assume that a task starts as soon as all its preceding tasks have been completed. Here, a task corresponds to the service of a customer in a queue (either a data packet or an acknowledgement going through one of the buffers defined above). Therefore, a task is indexed by the index of the queue and the packet number denoted by m.

We consider the graph $(\mathscr{V},\mathscr{E})$ defined with the following set of vertices:

$$\begin{aligned}\mathscr{V} = {} & \{(k,l,\mathtt{home},m) \mid k \geqslant 0\} \cup \{(k,l,i,\mathtt{beg},m),(k,l,i,\mathtt{end},m) \\ & \mid k \geqslant 1\}, \\ & \cup \{(k,l,i,h,m),(k,l,i,h,m)' \mid k \geqslant 1, 1 \leqslant h \leqslant H_{k,l,i}\}.\end{aligned}$$

where, $0 \leqslant l \leqslant 2^k - 1, 1 \leqslant i \leqslant k, m \in \mathbb{Z}$. Each vertex represents a queue at a router or buffer. The weight of each vertex represents the processing time of packet m in a router or buffer. For intermediate routers, (k,l,i,h,m) has weight $\sigma_m^{(k,l,i,h)}$ and it represents the queue when it is part of an active connection. When the connection is inactive, packets flow through vertex $(k,l,i,h,m)'$ with a weight of 0.

The edge of this graph are given by $\mathscr{E} = \mathscr{E}_0 \cup \mathscr{E}_1 \cup \mathscr{E}_2 \cup \mathscr{E}_3 \cup \mathscr{E}_4 \cup \mathscr{E}_5$:

- $\mathscr{E}_0$: edges representing the movement of packet m along routers, each router must wait for it to be processed at the previous router.
- $\mathscr{E}_1$: edges representing the movement of packets through a specific router, each packet cannot be process before the previous packet is processed.

$$\begin{aligned}\mathscr{E}_0 = {} & \{(k,l,i,\mathtt{beg},m) \to (k-i,\mathtt{m}_i(k,l),\mathtt{home},m), \\ & (k,l,\mathtt{home},m) \to (k,l,i,\mathtt{end},m),(k,l,i,\mathtt{end},m) \\ & \to (k,l,i,H_{(k,l,i)},m), \\ & (k,l,i,1,m) \to (k,l,i,\mathtt{beg},m) \mid k \geqslant 1\} \\ & \cup \{(k,l,i,h,m) \to (k,l,i,h-1,m)' \mid k \geqslant 1, 2 \leqslant h \leqslant H_{(k,l,i)}\} \\ & \cup \{(k,l,i,h,m)' \to (k,l,i,h,m) \mid \text{ if } (k,l,i) \\ & \quad \text{is active for packet } m, \\ & \quad k \geqslant 1, 1 \leqslant h \leqslant H_{(k,l,i)}\}, \\ \mathscr{E}_1 = {} & \{(k,l,\mathtt{home},m) \to (k,l,\mathtt{home},m-1) \mid k \geqslant 0\} \\ & \cup \{(k,l,i,\mathtt{end},m) \to (k,l,i,\mathtt{end},m-1), \\ & \quad (k,l,i,\mathtt{beg},m) \to (k,l,i,\mathtt{beg},m-1) \mid k \geqslant 1\} \\ & \cup \{(k,l,i,h,m)' \to (k,l,i,h,m-1)', \\ & \quad (k,l,i,h,m) \to (k,l,i,h,m-1)' \mid k \geqslant 1, 1 \leqslant h \leqslant H_{(k,l,i)}\},\end{aligned}$$

where $0 \leqslant l \leqslant D^k - 1, 1 \leqslant i \leqslant k, m \in \mathbb{Z}$.

In Fig. 5 we show the different types of horizontal and vertical edges connected to a router. Note in particular that a path may visit the working vertex (k,l,i,h,m) only if connection (k,l,i) is active for that packet. Otherwise, it can only visit the vertex $(k,l,i,h,m)'$, indicated in white, which has a null weight.

- $\mathscr{E}_2$: edges representing congestion window control, a packet is only released from the output buffer of the sending node if an ACK for a previous packet has been received.
- $\mathscr{E}_3$: edges representing receiver window control, a packet is only released from the output buffer of the sending node if there is enough space in the input buffer of the receiving node.
- $\mathscr{E}_4$ and $\mathscr{E}_5$: edges representing blocking back-pressure mechanism: a packet is only released from the input buffer of sending node if there is enough space in the storage buffer ($\mathscr{E}_4$) and a packet is only released from the storage buffer if there is space in all active output buffers of the sending node ($\mathscr{E}_5$).

$$\mathscr{E}_2 = \{(k,l,i,\mathtt{beg},m) \to (k,l,i,H_{k,l,i},m - W_m^{(k,l,i)})'\},$$
$$\mathscr{E}_3 = \{(k,l,i,\mathtt{beg},m) \to (k,l,\mathtt{home},m - B_{\mathtt{IN}}^{(k,l)})\},$$
$$\mathscr{E}_4 = \{(k,l,i,\mathtt{end},m) \to (k,l,\mathtt{home},m - B_{\mathtt{HOME}}^{(k,l)})\},$$

where $k \geqslant 1, 0 \leqslant l \leqslant D^k - 1, 1 \leqslant i \leqslant k, m \in \mathbb{Z}$.

$$\mathscr{E}_5 = \{(k,l,\mathtt{home},m) \to (k+i,l',i,H_{k+i,l',i},m - B_{\mathtt{OUT}}^{(k+i,l',i)})', \mid \text{if } (k+i,l',i) \text{ is active for packet } m, \; l' \in \mathtt{d}_i(k,l), i \geqslant 1\},$$

where $k \geqslant 0, 0 \leqslant l \leqslant D^k - 1, m \in \mathbb{Z}$.

In Fig. 6, we show examples from each edge set for an end system (k,l) which contains input buffers, storage buffer and output buffers. To keep the figure simple, we only show one input buffer and one output buffer. We also omit some of the possible horizontal and vertical edges for simplicity. Note that the edge representing a blocking back-pressure mechanism only exists when connection (k,l,i) is active.

When all buffers are initially empty for $m = 0$, we see that the time at which a task (k,l,h,m) is completed follows the last-passage percolation time, defined as follows:

$$x_m^{(k,l,i,h)} = \max\{\mathtt{Weight}(\pi)\}, \tag{1}$$

where π is a path in $(\mathscr{V},\mathscr{E}), \pi : (k,l,i,h) \rightsquigarrow (0,0,\mathtt{home},0)$. This represents the packet flow from the queue at the storage buffer of the root node to any queue in the system according to the mechanisms used in BAND.

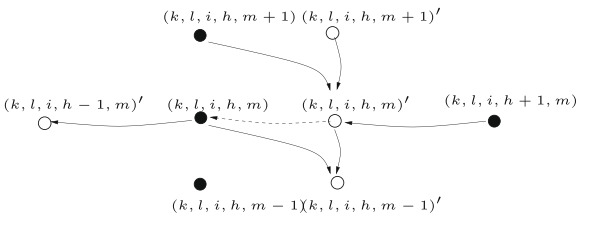


Fig. 5. Sample of horizontal edges ($\mathscr{E}_0$) and vertical edges ($\mathscr{E}_1$) for router (k,l,i,h), the dotted arrows indicate an edge which only exists if connection (k,l,i) is active.

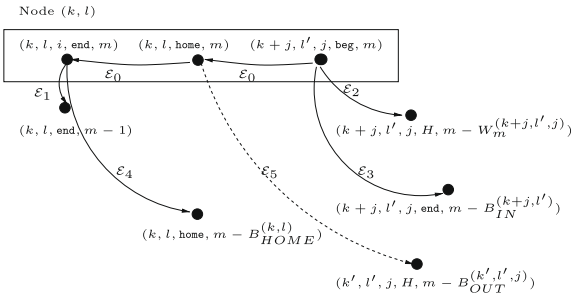


Fig. 6. Possible edges for end system (k,l). We omit the subscripts of H for legibility.

6.3. Scalability analysis

In order to study the performance scalability of BAND, we are interested in seeing how the system behaves as the number of queues goes to infinity. We are limited to analytical techniques as simulators cannot evaluate such a large configuration. In a system of infinite queues, to show that the throughput is lower bounded is not an easy task. We focus on the following simplified model:

- The multicast tree has bounded degree D.
- There is an upper bound on the number of routers between any two nodes: $H_{k,l,i} \leqslant H$. The DHT structure will ensure the overlay hop length is related to the overlay hop-length of the underlying topology.
- Output buffers are dimensioned to be of size $B_{\mathtt{OUT}}^{(k,l,i)} \geqslant B^* i$ for connection (k,l,i) as explained in Section 5.2.
- The packet loss process is independent and identically distributed in all connections, with probability p. While this assumption is not necessary for the scalability result, we chose it for simplicity.
- Aggregated service time represents the time a packet waits to be served in the queue once the previous packet from the same connection has been served. It includes the impact of packets from other flows. Aggregated service times are independent and identically distributed in all routers, according to process σ that is light-tailed (implies finite mean and variance).
- The start/stop pull occurs according to a known process that is fixed independently of the service time (i.e. the set of pulling nodes for each packet m follows a process independent of the rest).

Let the random variable $x_m^{(k,l,\mathtt{home})}$ be the maximum weight of a path drawn from $(k,l,\mathtt{home},m)$ to $(0,0,\mathtt{home},0)$ in the graph $(\mathscr{V},\mathscr{E})$.

Theorem 1. *Consider an overlay multicast tree with infinite height. Under the assumption that σ is light-tailed, then: uniformly in (k,l).*

$$\lim_{m\to\inf} \inf \frac{m}{x_m^{(k,l,\mathtt{home})}} \geqslant const(H,D) > 0 a.s. \tag{2}$$

Proof. (*Outline of the Proof*). The essential ingredient of the proof is to embed the evolution of all the connections that may be used by the tree in a single graph. For each packet, the start/stop pull process reconfigures locally a finite number of edges of this graph, which remains locally finite. We introduce a function $\phi(v)$ that is an upper bound on the number of vertices visited in a path from the root to v. We then apply Markov's inequality to find a bound for the weight of each vertex, as made possible by the light-tailed assumption. Finally, we apply the Borel–Cantelli to prove that almost surely $x_m^{(k,l,\mathrm{home})}$ remains under a linear bound on m. See Appendix A.1 for the detailed proof. □

6.4. Performance evaluation

In this section, we evaluate BAND with a custom max-plus simulator built by representing the model for packet flow (Sections 6.1 and 6.2) in matrix form, see [7,33] for the mechanics. The simulator is then implemented in C. The purpose of our simulations is two-fold: we would like to verify the analytical results presented earlier and study additional properties such as delay. In order to focus on scalability, we must test BAND on large systems, therefore do not use ns-2 or PlanetLab [34]. Similar to analysis, we focus on modeling the start/stop pull process, the service time process and the evolution of TCP windows.

6.4.1. Experimental set-up

To start with, we assume there is a single multicast tree with a fan-out of 2, though the trends shown are independent of the outdegree chosen. We vary the depth of the tree from 1 to 14, so the total number of nodes (pulling or not) in the system lies between 1 and $2^{14} - 2 = 16{,}382$. In this tree, nodes may pull independently of each other, following a Markov process. We assume, between the transfer of any two consecutive packets, a non-pulling node may start pulling with a probability p_{up} and a pulling node may stop pulling with a probability p_{down}. All nodes start in the steady state, i.e. a random set of nodes is already pulling at the beginning of the experiment. We consider two cases:

- *Sparse*: with $p_{\mathrm{up}} = 0.000005$ and $p_{\mathrm{down}} = 0.001$, we maintain the average number of pulling nodes at 4.7%.
- *Dense*: with $p_{\mathrm{up}} = p_{\mathrm{down}} = 0.001$, we maintain the average number of pulling nodes at 50%.

The sparse case represents the case where a small percentage of nodes in a tree are pulling given content. This is typical when tree construction happens much more slowly than content switching. The dense case represents when tree construction happens on a similar timescale as content switching.

We assume that the congestion window implemented on each connection follows a Markov process similar to the one of TCP new Reno: each connection maintains a window size. A coin is tossed between two successive packets, and with probability $1 - p$, the window size is increased according to:

$$W_{m+1} = \min(W_{\max}, W_m + 1/(2 \cdot W_m)).$$

Similar to [7], we only increase the window size by 1/2 to compensate for the delayed acknowledgment mechanism implemented in TCP. With a probability p, a congestion event will reduce the window size according to:

$$W_{m+1} = \max(1, \lfloor W_m/2 \rfloor),$$

and set $\mathtt{ssthresh} = W_{m+1}$. The maximum window size is set to 40 packets and p is set to 0.001 for our experiments. We choose buffer sizes to accommodate the maximum window size. Buffers in an active node are sized as follows: $B_{\mathrm{IN}} = 40$ packets, $B_{\mathrm{HOME}} = 60$ and $B_{\mathrm{OUT}} = i^* 40$ packets, where i is the number of overlay hops separating the sources and the destination.

Each connection in the multicast is composed of a sequence of 10 routers. The first router is shared by all out-going flows of the same node. We model the queuing delay for a packet at each router as a random process. We consider two distinct cases: *light-tailed* case where service time follows an exponential distribution; and *heavy-tailed* case where service time follows a Pareto distribution with coefficient 2.1. We chose an average service time of 10 ms, though the trends shown are independent of the chosen value. In other words, we assume that flows are served on a link with a total capacity of 100 Pkts/s (i.e. 150 kBps (1.2 Mbps)). All out-going flows from the same node go through the same first router, hence this capacity is divided among them (i.e., their aggregated service time is multiplied by their number).

6.4.2. Evaluation of overhead scalability

In this section, we compare the out-degree distribution and the ratio between active node and pulling nodes for three schemes: BAND ($k = 2$), traditional overlay multicast ($k = 1$) and selfish ($k > D$). We focus on the *sparse* case where the three schemes differ the most since in the *dense* case, the three schemes behave similarly. As predicted by Proposition 1, the outdegree for the selfish architecture grows with the size of the tree. For example, in a group of 600 pulling nodes, on average 10 have degree of more than 80. In the next section, we will show the selfish architecture does not scale.

We summarize the out-degree distribution of BAND and traditional overlay multicast in Table 1 for a tree of depth 10, the trend is similar for all tree sizes. In traditional overlay multicast, about 55% of active nodes have degree 1. Among those nodes, the ones that are not pulling are exactly those which are removed in BAND. Indeed we observed that only a small portion of them were pulling nodes, as BAND only includes 5.5% percent of nodes with degree 1.

In Fig. 7, we plot the ratio between the number of active nodes and the number of pulling nodes for BAND and traditional overlay multicast. The error bars indicate the 10th and 90th percentile for the outdegree distribution of all ac-

Table 1
Out-degree of active nodes for BAND and the traditional overlay multicast.

	0 (%)	1 (%)	2 (%)
Traditional overlay multicast	22.5	54.9	22.5
BAND	47.2	5.5	47.2

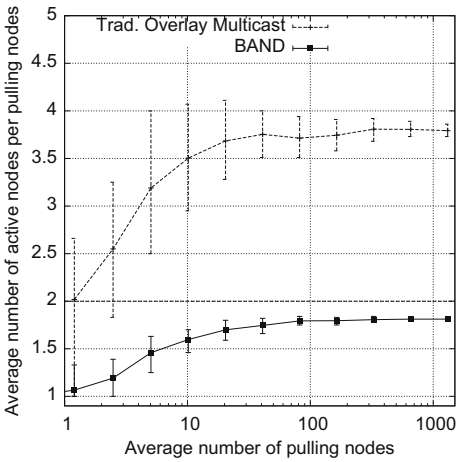


Fig. 7. Ratio between the number of active nodes and the number of pulling nodes.

tive nodes in the tree. As shown by Proposition 2, BAND never requires more than *two* active nodes per pulling node. On the other hand, traditional overlay multicast can require close to *four* active nodes per pulling node, roughly twice more than we can expect from Proposition 2. BAND should save 1–1/2 (since $D = 2$ for this case) of the nodes from being active and this is indeed the case. Similar trends are observed for larger degree trees.

6.4.3. Evaluation of performance scalability

Studying performance scalability for a single tree, we observe empirically a lower bound on the group throughput (as seen in Theorem 1), which is quickly reached by BAND in a system of moderate scale. In addition, we show that the delay grows logarithmically with the average number of pulling nodes.

We plot throughput and delay of three schemes versus the average number of pulling nodes in Figs. 8 and 9. Initially, all buffers are empty, therefore, we need to warm up the system before recording to measure the steady-state. We consider the system to be warmed up when delays no longer increase with each packet sent. For our experiments, a total of 500,000 packets were sent, and we only calculate throughput and delay for the last 250,000 packets.

For the throughput experiments, we consider a saturated source. For packets in a given range, we define the throughput as the number of packets in that range divided by the time difference between receiving the last packet and the first packet averaged over all pulling nodes. We calculate a moving average of the throughput with a window size of 50,000 packets, for each point on the curve, we show the confidence intervals at 10th and 90th percentile. In Fig. 8a and b, we observe that for light-tailed service time distribution, the throughput quickly stabilized to stay above 65% of a single connection dedicated connection for BAND and traditional overlay multicast, but does not scale for the selfish architecture. In particular, the throughput obtained for 600 pulling nodes up to an average of 8000 are almost identical, confirming the existence of a positive throughput independent of the size as proved by Theorem 1. Fig. 8a illustrates that BAND performs better than traditional overlay multicast when there is significant timescale separation between tree construction and content switching. Otherwise, BAND performs no worse than traditional overlay multicast, as in Fig. 8b.

Next, we test what happens when the assumption of light-tailed service time distribution is removed from Theorem 1. For the heavy-tail service time distribution shown in Fig. 8c, a lower bound does not appear to exist, unlike the light-tailed case. Still, the rate of decrease is slow and the throughput remains above 50% of a single dedicated connection for average pulling group with size 500.

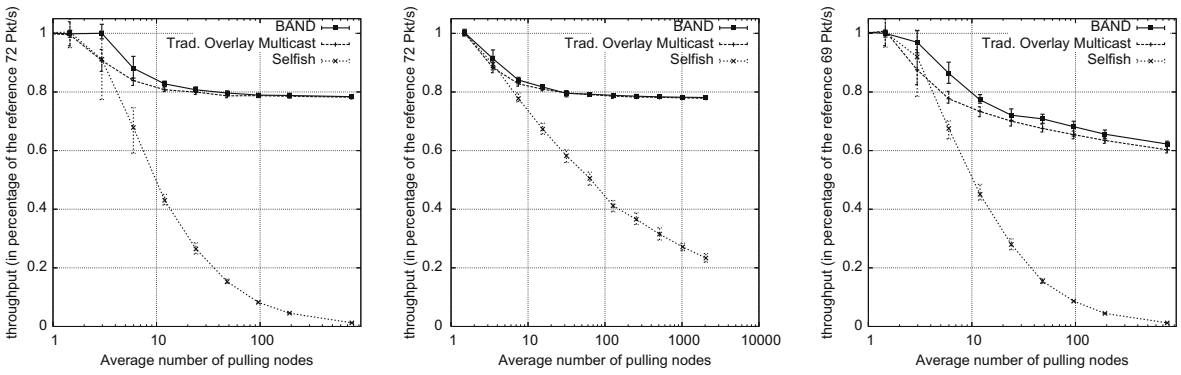
This indicates that service time distribution matters and that networks with heavy-tailed service time distributions are less scalable. The actual service time distribution will depend on many factors in a network, which is why it is important to compare the performance of a scheme under different conditions. Since packets are bounded by MTU in size, and a minimum bandwidth is given for each TCP connection at all times, it is reasonable to assume that the tail of a service time decreases quickly after some value (as for a light-tailed distribution).

While we have no analytical results for delay, it is a crucial performance metric in online streaming. For these experiments, we consider a root node sending periodically at a rate of 50 packets per second and a tree of depth 7. Since the selfish architecture cannot sustain an acceptable throughput for large trees, we omit characterizing its delay properties. We calculate the delay for each packet as the difference between the time when it is created and the time when it is received by all pulling nodes. For each point on the curve, we show the confidence intervals at the 10th and 90th percentile. From Fig. 9a, we observe that the average delays grows logarithmically with the number of pulling nodes (or linearly with the depth of the tree). We plot the cumulative distribution of the delay values in Fig. 9b. We observe for BAND, 99% of the packets are arriving within 2.5 s, and 99.9% of the packets arrive within 3s, so by having a playout buffer of 3 s, then you will only lose 0.1% of your packets. In both Fig. 9a and b, we observe that the BAND outperforms traditional overlay multicast. This is not surprising since fewer relay nodes in the overlay translates into lower delay.

6.4.4. Impact of multiple groups

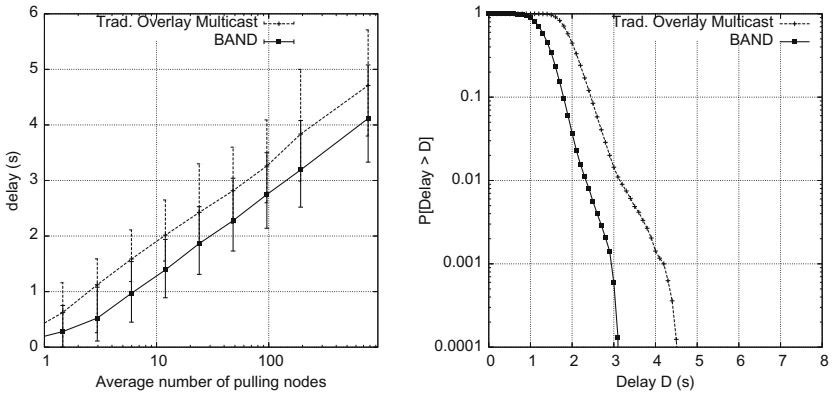
In our next set of experiments, we consider multiple groups that use the same population of nodes. Each content stream is independent and therefore the trees are also independent from each other. Each node can participate in multiple groups, either as pulling nodes or relays. To model the impact of multiple groups at the same node, we define the *load* of a node as the number of simultaneous TCP connections which are departing from it. We assume that the aggregated service times of the router closest to the node on each of these TCP connections follows an exponential distribution with mean depending linearly to the load of the node. To show the trends versus number of groups, we show only the results for a tree of height 9, though the trends hold for other tree sizes.

In Fig. 10a, we plot average throughput achieved versus the number of groups. We see that selfish is simply not scal-



(a)Light-tailed, sparse (b)Light-tailed, dense (c)Heavy-tailed, sparse

Fig. 8. Throughput of three schemes versus average number of pulling nodes.



(a)Average delay, (b)Delay distribution

Fig. 9. Delay properties of BAND and traditional overlay multicast for light-tailed services times and sparse distribution.

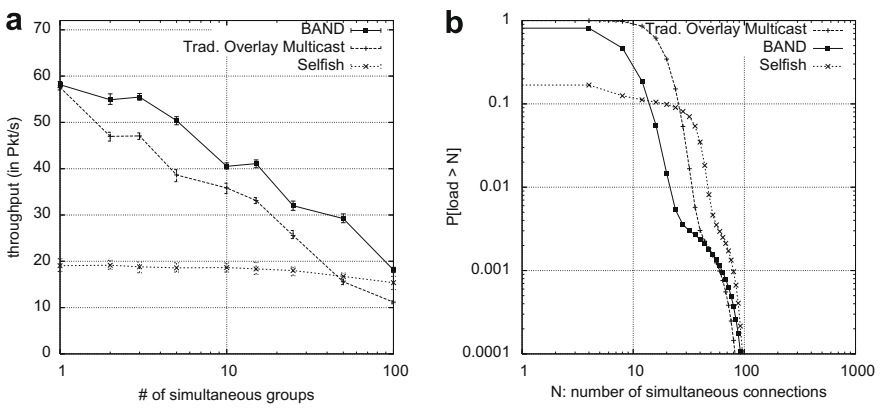


Fig. 10. (a) Throughput for multiple simultaneous groups, and (b) cumulative distribution function for the number of connections at each node, for 100 groups.

able. For the same throughput requirement, BAND can support more groups than traditional overlay multicast. If the desired throughput is 400kb per second and each packet is 1500 bytes (MTU for Ethernet), then we need to send at a rate of 33 packets per second. BAND may then support 25 groups, compared with 15 for traditional multicast.

Fig. 10b shows how many connections are maintained by a node when there are 100 groups. Selfish has the most

number of nodes with 30 or more connections, which precisely contributes to its inability to scale. Traditional multicast, on the other hand, has a large number of total connections (found by looking at the area under the curve) and therefore wastes network resources. BAND strikes the correct balance by reducing the number of total connections, without having a large number of nodes with 30 or more connections.

Our simulations show that BAND requires less overhead *and* has better (or at least equal) performance compared to traditional overlay multicast and clearly outperforms selfish networks where the out-degree can grow without bound.

7. Conclusion

This paper studies the feasibility of providing peer-to-peer live streaming application with quality and delay comparable to a traditional television broadcast using nano data centers. As a service provided by ISPs, nano data centers form a controlled environment that mitigates peer churn and heterogeneity. The main challenge we address then is how to allow users to switch quickly between thousands of channels, possibly more. Our work proves that in this environment, a small amount of content push, when properly chosen, brings great benefit when compared with a pure pull system. We show that such a system scales with the popularity of the channel, allowing a large number of unpopular channels to be offered. The pull-push scheme we present may operate among peers in a simple distributed way that does not require supernodes or tracker, or large buffers maintained in peers. It handles continuity of data delivery with flow control, and we prove analytically that its throughput is guaranteed independently of group size. Compared with alternative solutions managed by an ISP (IP-multicast and CDNs), our solution delivers live streaming service of a larger number of channels, with significant cost savings. Our results indicate that p2p schemes can and should take advantage of a controlled environment to improve their efficiency, while keeping the ease of deployment and scalability of p2p architectures.

We see several directions to follow up on these results. First, we could not include a full fledged validation of the system with a standard discrete event simulator, or based on experimental test beds. The assumptions we made in the model have been tested previously (see [7] and references therein), so we expect to see the same performance trend. Nevertheless, further empirical analysis would allow testing the efficiency of our scheme in comparison with a mesh-pull approach, which is hard to judge analytically. Second, our analysis can be extended to prove the logarithmic increase of the delay we observe numerically, or to study the load per user as a function of the channels popularity. Last, heterogeneity of nodes may be handled by using multiple delivery trees [21,22], and possibly different values of k among them. Some of our results might then be relevant for ISP networks with different types of nodes, or even non-controlled environments. This approach seems natural as a few recent works reach the same conclusion that a small amount of content push may be desirable to improve mesh-pull schemes [24,23]. Similarly, some works demonstrated the efficiency of having peers cooperating among different overlays [19]. At the present time, proving delay and throughput scalability in the uncontrolled environment remains an open challenge.

Acknowledgements

We would like to thank Christoph Neumann and Laurent Massoulié from Thomson for discussions on this paper. We would also like to thank Jennifer Rexford of Princeton University, Jim Kurose, Don Towsley of U. Mass and Jon Crowcroft from U. Cambridge for their input.

Appendix A. Proof of Proposition 2

Let p_i denote the probability that a node in level i is semi-active and let $q_i = 1 - p_i$. For a node at level $i+1$ to be semi-active, it need only to contain one or more pulling nodes in its sub-tree of height $H - i - 1$. Therefore:

$$p_{i+1} = 1 - (1-p)^{\frac{D^{H-i}-1}{D-1}} \quad \text{and} \quad q_{i+1} = q^{\frac{D^{H-i}-1}{D-1}}.$$

Let $m_B(i)$ (respectively, $m_S(i)$) be the expected number of active nodes in level i using BAND (respectively, traditional multicast).

$$m_B(i) = D^i(p + (1-p)[1 - q_{i+1}^D - Dp_{i+1} \cdot (q_{i+1})^{D-1}]),$$
$$m_S(i) = D^i(p + (1-p)(1 - q_{i+1}^D)),$$
$$m_S(i) - m_B(i) = D^{i+1}p_{i+1}(q_{i+1})^{D-1}(1-p).$$

Let m_B, m_s be the total expected number of active nodes for these both schemes.

$$\frac{m_S - m_B}{m_S} = \frac{\sum_{i=0,\ldots,H-1} D^{i+1}p_{i+1}q_{i+1}^{D-1}(1-p)}{\sum_{i=0,\ldots,H-1} D^i(1-(1-p)q_{i+1}^D)},$$
$$= \frac{\sum_{i=0,\ldots,H-1} D^{i+1}(q^{D^{H-i}} - q^{\frac{D^{(H-i+1)}-1}{D-1}})}{\sum_{i=0,\ldots,H-1} D^i\left(1 - q^{\frac{D^{(H-i+1)}-1}{D-1}}\right)}.$$

Taking the limit as $q \to 1$ and then applying L'Hôpital's Rule:

$$\lim_{q\to 1}\frac{m_S - m_B}{m_S} = \frac{\sum_{i=0}^{H-1} D^{i+1}\left(\frac{D^{(H-i+1)}-1}{D-1} - D^{H-i}\right)}{\sum_{i=0}^{H-1} D^i\left(\frac{D^{(H-i+1)}-1}{D-1}\right)},$$
$$= 1 - \frac{HD^{H+1} - D^H}{HD^{H+2} - D^{H+1} - D^H + 1} \geqslant 1 - \frac{1}{D}.$$

A.1. Proof of Theorem 1

Proof. We define the function $\phi : \mathscr{V} \to \mathbb{Z}$ with values as follows:

$$\begin{cases} a(k-1) + b\cdot m + 1 & \text{for } v = (k,l,i,\texttt{beg},m), \\ a(k-1) + b\cdot m + 2h & \text{for } v = (k,l,i,h,m), \\ a(k-1) + b\cdot m + 2h + 1 & \text{for } v = (k,l,i,h,m)', \\ a(k-1) + b\cdot m + 2H + 2 & \text{for } v = (k,l,i,\texttt{end},m), \\ a\cdot k + b\cdot m & \text{for } v = (k,l,\texttt{home},m), \end{cases}$$

where $a = 2H + 3$, and $b = \max\left(a, \lceil \frac{a}{B} \rceil\right)$.

For any edge $u \to v$ in $\mathscr{E}$, we have $\phi(v) < \phi(u)$. As a consequence, a path included in the definition of $x_m^{k,l,\mathtt{home}}$ contains at most $a \cdot k + b \cdot m$ vertices. □

Let us first prove the following lemma:

Lemma 1

For any edge $u \to v$ in $\mathscr{E}$, we have $\phi(v) < \phi(u)$.

The result is clear for any edge in $\mathscr{E}_0$ and $\mathscr{E}_1$. When $u \to v$ is in $\mathscr{E}_2$, $\phi(u) - \phi(v)$ may be written as follows:

$$\begin{aligned} &a \cdot k + b \cdot m + 1 - \left(a \cdot k + b(m - W_m^{k,l,i}) + 2H_{k,l,i} + 1\right), \\ &= b \cdot W_m^{k,l,i} - 2H_{k,l,i}, \\ &\geqslant 1, \quad \text{as} \quad b \geqslant 2H + 3 \quad \text{and} \quad W_m^{k,l,i} \geqslant 1. \end{aligned}$$

A similar proof can be made for any edge chosen in $\mathscr{E}_3$ and $\mathscr{E}_4$. Last, when the edge is in $\mathscr{E}_5$, we have, for an $i \geqslant 1$.

$$\begin{aligned} &\phi(u) - \phi(v) = -a \cdot i + b \cdot B_{\mathtt{OUT}}^{(k,l,i)} + 2, \\ &\geqslant 2, \quad \text{as} \quad B_{\mathtt{OUT}}^{(k,l,i)} \geqslant i \cdot B \quad \text{and} \quad b \geqslant \left\lceil \frac{a}{B} \right\rceil. \end{aligned}$$

This ends the proof of Lemma 1. As a consequence, any path π going from $(k,l,\mathtt{home},m)$ to $(0,0,\mathtt{home},0)$ contains at most $a \cdot k + b \cdot m$ vertices. Following the light tail assumption on the distribution of service time, there exists $t > 0$ such that $\mathbb{E}[e^{t \cdot \sigma}] = A < \infty$. Hence, by Markov's inequality

$$\begin{aligned} &\mathbb{P}[\mathtt{Weight}(\pi) \geqslant xm] \leqslant \mathbb{E}[e^{t \cdot \mathtt{Weight}(\pi)}] / \exp(t \cdot x \cdot m), \\ &\leqslant A^{ak+bm} / \exp(t \cdot x \cdot m). \end{aligned}$$

We then make the following observation: in the definition of $\mathscr{E}_0$, one can remove the edges $(k,l,\mathtt{home},m) \to (k,l,\mathtt{end},m)$ when (k,l,i) is not active for m, as it does not impact the last-passage percolation time in this vertex. Indeed the only vertices that are reachable from $(k,l,\mathtt{end},m)$ and $(k,l,H_{(k,l,i)},m)$ are all reachable from $(k,l,\mathtt{home},m-1)$. With this modification, the number of edges leaving vertex $(k,l,\mathtt{home},m)$ is less than $D+2$ (as each corresponds to a unique connection that is active for packet m and incident to node (k,l)). The number of such connections is bounded by $D+1$ thanks to Proposition 1 (D connection downstream and one upstream). This is also obviously verified for other vertices of the graph.

The result is challenging because $x_m^{(k,l,\mathtt{home})}$ is the maximum weight of a path drawn on a graph that randomly changes with the pulling process and the window.

Let us assume first that the start/stop pull process is deterministically fixed, as well as the evolution of the window in each connection. The graph $(\mathscr{V},\mathscr{E})$ is then a deterministic graph with a outdegree bounded by $D+2$. In this case, we can consider the deterministic set of all paths leading from $(k,l,\mathtt{home},m)$ to $(0,0,\mathtt{home},0)$. Given that we know these paths have a length of at most $a \cdot k + b \cdot m$, there are at most $(D+2)^{a \cdot k + b \cdot m}$ paths in this collection. We can then deduce, from the above inequality,

$$\mathbb{P}[x_m^{(k,l,\mathtt{home})} \geqslant xm \mid (\mathscr{V},\mathscr{E}) \text{fixed}] \leqslant (D+3)^{ak+bm} \frac{A^{ak+bm}}{e^{txm}}.$$

Note that the left hand side depends on the static graph, but that the bound on the right-hand side is a deterministic constant. Because the weights of the vertices are all independent from the process used to build the edges, the same inequality holds for a random process defining the edges, after conditioning on the value of this process. The right-hand side above is therefore an upper bound for the following conditioning sum

$$\sum_{(\mathscr{V},\mathscr{E})} \mathbb{P}[x_m^{(k,l,\mathtt{home})} \geqslant xm \mid (\mathscr{V},\mathscr{E})] \mathbb{P}[(\mathscr{V},\mathscr{E})].$$

Choosing x large enough, we deduce that the series $\sum_m \mathbb{P}[x_m^{(k,l,\mathtt{home})} \geqslant xm]$ converges by an application of the Borel–Cantelli Lemma. This proves Theorem 1.

A.2. The loss and re-sequencing model

In this subsection, we describe how to explicitly handle packet losses in the graph presented in Section 6.2. As shown in [7], when packet m is lost, the following window evolution:

- The window is set to $\max((W_m - 1)/2,1)$ for $m+1, m+2, \ldots, m + W_m + \max((W_m - 1)/2,1) - 1$.
- The additive increasing evolution of the window is resumed from packet $m + W_m + \max((W_m - 1)/2,1)$ onwards.

is conservative in that the real system will have a larger window size at all times and hence better throughput.

In the random graph associated with the loss model, we add a third vertex $(k,l,i,h,m)''$, for all $k \geqslant 1$ i,l,h and m, which represents the potential retransmission of a packet sent between packets m and $m+1$. Consequently, vertices of the graph associated with the index m will refer either to packet m itself, or to a retransmitted packet that was sent after packet m and before packet $m+1$.

We also add the following edges to link this vertex to the vertical and horizontal structure introduced in Section 6.2:

- Horizontal edges: $(k,l,i,h,m)'' \to (k,l,i,h-1,m)''$ for $h = 2,\ldots,H$, $(k,l,i,1,m)'' \to (k,l,i,\mathtt{beg},m)$;
- Vertical edges: $(k,l,i,h,m)'' \to (k,l,i,h,m)$ for $h = 1,\ldots,H$.

Note that with no further additional edges, these complementary vertices play no role in the graph.

In order to represent the effect of the loss and the retransmission of packet m on the TCP connection (k,l):

- Edge $\mathscr{E}_6$: $(k,l,\mathtt{home},m) \to (k,l,i,H_{k,l},m + W_m - 1)''$ in order to represent the re-sequencing of packets $m, m+1, \ldots, m + W_m - 1$.
- Edges $\mathscr{E}_7$: $(k,l,i,h,m'' + 1) \to (k,l,i,h,m'')''$ for all $h = 1,\ldots,H_{k,l,i}$ and $m'' = m,\ldots,m + W_m$ to represent the retransmission of packet m (as the extra packet in between indices $m + W_m - 1$ and $m + W_m$) which delays the following packets.

Note that the proof of Theorem 1 can be modified to include the case when retransmission due to packet losses are explicit. This only requires a small modification to the ϕ function in Lemma 1.

References

[1] X. Hei, C. Liang, J. Liang, Y. Liu, K.W. Ross, A measurement study of a large-scale P2P iptv system, IEEE Transactions on Multimedia 9 (8) (2007) 1672–1687.
[2] V. Janardhan, H. Schulzrinne, Peer assisted VoD for set-top box based IP network, in: Proceedings of the ACM SIGCOMM Workshop on P2P-TV, August 2007.
[3] K. Suh, C. Diot, J. Kurose, L. Massoulie, C. Neumann, D. Towsley, M. Varvello, Push-to-peer video-on-demand system: design and evaluation, IEEE JSAC 25 (9) (2007) 1706–1716.
[4] S. Narayanan, D. Braun, J. Buford, R. Fish, A. Gelman, A. Kaplan, R. Khandelwal, E. Shim, H. Yu, Peer-to-peer streaming for networked consumer electronics, IEEE Communications Magazine 45 (6) (2007).
[5] M. Cha, P. Rodriguez, S. Moon, J. Crowcroft, On next-generation telco-managed P2P TV architectures, in: Proceedings of the IPTPS, 2008.
[6] N. Laoutaris, P. Rodriguez, L. Massoulie, Echos: edge capacity hosting overlays of nano data centers, SIGCOMM Computer Communication Review 38 (1) (2008) 51–54.
[7] F. Baccelli, A. Chaintreau, Z. Liu, A. Riabov, The one-to-many TCP overlay: a scalable and reliable multicast architecture, in: Proceedings of the IEEE INFOCOM, 2005.
[8] A. Chaintreau, Processes of interaction in data networks. Ph.D. Thesis, Ecole Normale Supérieure, 2006.
[9] Y. Chu, S.G. Rao, H. Zhang, A case for end system multicast (keynote address), Proceedings of the ACM SIGMETRICS, ACM Press, 2000. pp. 1–12.
[10] B. Cohen, Incentives build robustness in bittorrent, in: Proceedings of the First Workshop on Economics of Peer-to-Peer Systems, 2003.
[11] X. Zhang, J. Liu, B. Li, T.-S.P. Yum, DONET/CoolStreaming: a data-driven overlay network for live media streaming, in: Proceedings of the IEEE INFOCOM, 2005.
[12] M. Zhang, Q. Zhang, L. Sun, S. Yang, Understanding the power of pull-based streaming protocol: can we do better?, IEEE JSAC 25 (2007) 1678–1694.
[13] T. Silverston, O. Fourmaux, Measuring P2P iptv systems, in: Proceedings of the 17th NOSSDAV Workshop, 2007.
[14] A. Sentinelli, G. Marfia, M. Gerla, L. Kleinrock, S. Tewari, Will iptv ride the peer-to-peer stream?, IEEE Communications Magazine 45 (6) (2007).
[15] B. Li, S. Xie, G.Y. Keung, J. Liu, I. Stoica, H. Zhang, X. Zhang, An empirical study of the coolstreaming + system, IEEE JSAC 25 (2007) 1627–1639.
[16] S. Ali, A. Mathur, H. Zhang, Measurement of commercial peer-to-peer live video streaming, in: Proceedings of the Workshop in Recent Advances in Peer-to-Peer Streaming (WRAIPS), 2006.
[17] Saurabh Tewari, L. Kleinrock, Analytical model for bittorrent-based live video streaming, in: Proceedings of the IEEE Consumer Communications and Networking Conference, 2007.
[18] F. Pianese, D. Perino, Resource and locality awareness in an incentive-based P2P live streaming system, in: P2P-TV '07: Proceedings of the 2007 Workshop on Peer-to-Peer Streaming and IP-TV, 2007, pp. 317–322.
[19] X. Liao, H. Jin, Y. Liu, L. Ni, D. Deng, Anysee: peer-to-peer live streaming, in: Proceedings of the IEEE INFOCOM, 2006.
[20] X. Hei, Y. Liu, K. Ross, Inferring network-wide quality in P2P live streaming systems, IEEE JSAC 25 (2007) 1640–1654.
[21] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, A. Singh, SplitStream: high-bandwidth multicast in a cooperative environment, in: Proceedings of the SOSP, October 2003.
[22] J. Venkataraman, P. Francis, Chunkyspread: multi-tree unstructured peer-to-peer multicast, in: Proceedings of the International Workshop on Peer-to-Peer Systems, February 2006.
[23] T. Locher, R. Meier, S. Schmid, R. Wattenhofer, Push-to-pull peer-to-peer live streaming, in: Proceedings of the DISC, 2007.
[24] F. Wang, Y. Xiong, J. Liu, Mtreebone: a hybrid tree/mesh overlay for application-layer live video multicast, in: Proceedings of the IEEE ICDCS, 2007.
[25] V. Aggarwal, A. Feldmann, C. Scheideler, Can ISPs and P2P users cooperate for improved performance?, SIGCOMM Computer Communication Review 37 (3) (2007) 29–40.
[26] L. Kontothanassis, R. Sitaraman, J. Wein, D. Hong, R. Kleinberg, B. Mancuso, D. Shaw, D. Stodolsky, A transport layer for live streaming in a content delivery network, IEEE Special Issue on Evolution of Internet Technologies 92 (2004) 1408–1419.
[27] J.W. Byers, J. Considine, M. Mitzenmacher, S. Rost, Informed content delivery across adaptive overlay networks, IEEE/ACM ToN 12 (5) (2004) 767–780.
[28] N. Magharei, R. Rejaie, Y. Guo, Mesh or multiple tree: a comparative study of live P2P streaming approaches, in: Proceedings of the IEEE INFOCOM, 2007.
[29] M. Castro, P. Druschel, A.-M. Kermarrec, A. Rowstron, Scribe: a large-scale and decentralized application-level multicast infrastructure, IEEE JSAC 20 (2002).
[30] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, in: Proceedings of the ACM SIGCOMM, ACM Press, 2001, pp. 161–172.
[31] R. Melamed, I. Keidar, Araneola: a scalable reliable multicast system for dynamic environments, in: Proceedings of the International Symposium on Network Computing and Applications, 2004.
[32] S. Birrer, F.E. Bustamante, The feasibility of DHT-based streaming multicast, in: Proceedings of the International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, IEEE Computer Society, 2005, pp. 288–298.
[33] G. Urvoy-Keller, E.W. Biersack, A congestion control model for multicast overlay networks and its performance, in: Proceedings of the International Workshop on Networked Group Communication, 10, 2002.
[34] "PlanetLab." <www.planet-lab.org>.

Jiayue He received her B.A.Sc. (Hon.) in Engineering Science from University of Toronto in 2004. She received her M.A. and Ph.D. from Princeton University in 2006 and 2008, respectively. Her thesis work was primarily focused on traffic management. Her Ph.D. is partially funded by the Gordon Wu Fellowship at Princeton University and the graduate fellowship from National Science and Engineering Research Council of Canada. She interned at Thomson Research Labs in summer 2006.

Augustin Chaintreau joined the Thomson Research lab soon after graduating in 2006 from Ecole Normale Superieure de Paris, working at INRIA under the supervision of Francois Bacelli. During his Ph.D. he worked in collaboration with Alcatel Bell, as well as the IBM Watson T.J. Research Center in New York. He have spent a year visiting Intel Research Cambridge. His research interests focus on the analysis of emerging communication architectures, opportunistic mobile networking, peer-to-peer systems and wireless networks.

Christophe Diot received a Ph.D. degree in Computer Science from INP Grenoble in 1991. He was with INRIA Sophia-Antipolis (1993–1998), Sprint (1998–2003), and Intel Research in Cambridge, UK (2003–2005). He joined Thomson in October 2005 to start and manage the Paris Research Lab. Diot has been a pioneer in multicast communication, DiffServ, Internet measurements, and more recently Pocket Switched Networks. Diot's research activities now focus on advanced P2P communication services. Diot is the Thomson Corporate Research CTO and an ACM fellow.