An Autonomous Topology Management Framework for QoS Enabled P2P Video Streaming Systems

Ihsan Ullah, Guillaume Doyen, Grégory Bonnet, Dominique Gaïti

An autonomous topology management framework for QoS enabled P2P video streaming systems

Ihsan Ullah*, Guillaume Doyen†, Grégory Bonnet‡ and Dominique Gaïti†
*Department of Computer Science & IT
University of Balochistan Quetta, Pakistan
Email: ihsan.ullah.2012@utt.fr
†ERA / Institut Charles Delaunay – UMR 6279
Université de Technologie de Troyes, France
Email: doyen,gaiti@utt.fr
‡UMR CNRS 6072 GREYC
University of Caen Lower-Normandy, France
Email:gregory.bonnet@unicaen.fr

***Abstract*—Streaming live video over the Internet presents great challenges due to its sheer bandwidth requirements. Client/Server model suffers from scalability issues and high deployment cost to provide this service. Peer-to-Peer (P2P) approach provides an excellent alternative due to its potential scalability and ease of deployment. Nonetheless, a major limitation of P2P approach lies in its high dependency on users. Since peers relay content, which themselves are controlled by users, the behavior of the latter has a major impact on the streaming quality perceived by users. Indeed, unlike dedicated servers, peers join the system intermittently, which poses great challenges in providing QoS for operated live streaming services. In this paper, we propose an autonomous topology management framework for P2P live streaming architectures that minimizes the impact of peers' frequent departures. It consists in a stabilization strategy for push-based systems that moves unstable peers towards the outskirts of the topology. To validate our approach, we performed experiments on PlanetLab and show here the significant improvement of our contribution as compared to an existing system in terms of the global service quality.**

I. Introduction

Multimedia streaming over the Internet has attracted immense interest from both academia and industry. Numerous live video streaming solutions [1] have been proposed and several practical systems [2] have been deployed which attract users in large numbers. Due to the limited deployment of IP multicast as well as the cost and scalability issues of content delivery networks, P2P approach provides an efficient alternative. Instead of requiring routers to enable multicast or centralized servers, it relies on end-hosts called peers to relay content to other peers. This approach allows a quick and easy deployment and provides scalability.

Based on their content diffusion strategies, P2P streaming systems can be categorized into three major types: push-based, pull-based and hybrid. Push-based approaches [3], [4] organize end-hosts into well-defined parent children relationships where a child peer always receives the content from its parent peer. A parent peer pushes the content to its child peers as it receives it. Pull-based systems [5], [6] allow peers to establish connections with several other peers and pull the content from them through explicit requests for certain blocks of data. Peers advertise their buffer maps to let others know about the content they can provide. Finally, hybrid systems [7], [8] incorporate both the pull and push approaches together. In normal operation, the content is usually pushed while the missing content is pulled from other peers.

P2P approach for streaming applications faces its own challenges. The independent arrivals and departures of peers, called churn, turn the network topology highly dynamic. Management of such a network is not trivial: it explains the current barrier toward the deployment of P2P architectures in the context of SLA compliant services operated by a dedicated provider. The problem becomes further severe for live video streaming service since live content has stringent playback deadlines. Churn degrades the Quality-of-Service (QoS) resulting in video playback freezes and skips. Consequently, the users' Quality-of-Experience (QoE) is impacted.

Since each peer is controlled by a user, the behavior of the latter has a direct impact on the performance of the system. Systems incorporating a pure push strategy over a tree-based P2P network suffer the most from user behavior. The sudden departure of an upstream peer in such a network disrupts the video stream to all its descendant peers. On the other hand, systems with pull-based strategy attempt to mitigate the impact of user behavior through pulling content from several peers at the same time and using large buffers. They achieve streaming continuity at the cost of longer startup and playback delays. In the hybrid approach too, the departure of a pushing peer forces all its downstream peers to operate completely in pull mode, thus inheriting the QoS issues of pull-based approach.

To overcome these problems, user behavior needs to be considered directly in P2P topology management. Constructing user-aware streaming topologies can minimize the impact of user behavior. In this paper, we propose an adaptive control mechanism aimed at the stabilization of tree-based push systems that periodically moves the unstable peers towards the outskirts of the tree. This mechanism uses estimations of formerly proposed user behavior models to carry out control

978-3-901882-48-7

decisions. Experiments over PlanetLab show that our approach improves significantly the performance of an existing system.

The rest of this paper is organized as follows. Section II mentions works related to the integration of user behavior in P2P topology control mechanisms as well as previous work we proposed in this area and on which the contribution of this paper relies. Section III describes our autonomous topology management strategy through novel algorithms. Section IV presents the results of experiments we performed over PlanetLab that rely on an implementation prototype we developed. The latter shows the validity of our topology management proposal while also providing a comparative analysis of peers' stability estimation strategies. Finally, section V presents conclusions and future research perspectives.

II. Related Work

A major part of the research activities over user behavior in multimedia streaming applications is dedicated to measurement studies [9]. These measurements get insights into user behavior through analyzing logs and traces, collected by the service provider or by using crawlers and passive monitoring techniques. Analysis results provide foundations for modeling user behavior that enable to adapt P2P live streaming systems for improved streaming quality and better QoE.

Based on the insights from measurements, a few other works propose user behavior models. An overview of these works is given below.

A. User behavior models

Tang *et al.* [10] analyze the stability of peers through measurements and observe that users with longer elapsed time in a session tend to stay longer for the remaining part of that session. Based on this information, they propose a neighbor selection strategy to improve the streaming quality. This approach prefers to choose a long-lived peer as an upstream peer. Wang *et al.* [11], [7] use a similar approach to identify stable peers for putting them in the backbone of the topology. Nevertheless, a downside of such an approach is that it tends to consider all recently arrived peers as unstable ones, which is not always the case.

Horovitz *et al.* [12] propose an SVM (Support Vector Machine) model that enables peers to actively detect the load in the uplink of source peers and alert their clients to replace their source. This approach only considers the bandwidth dynamics and does not take into account other metrics such as stability and streaming quality. Liu *et al.* [13] consider peers' resilience to minimize service disruption in P2P streaming systems. The resilience is determined with the help of peer's age. This approach too ignores other well-known influential factors such as time-of-day, type and popularity of the content.

Zheng *et al.* [14] argue that node churn is increased with degradation in service quality. They propose a peer selection strategy at short time scales, that takes into account content availability to balance at the same time connection efficiency and stability requirements. This work establishes a correlation between service quality and node churn but does not focus on an accurate estimation of the latter.

Liu *et al.* [15] take a step further and they analyze through measurements that stability and bandwidth contribution of users are influenced by several factors. For instance, they observe that a user experiencing a good initial streaming quality while watching a popular channel stays longer. They propose models to estimate the stability and bandwidth contribution of peers. However, these models do not consider all the impacting factors observed by other measurements, such as the impact of elapsed time on stability. Moreover, they do not integrate these models in real systems to evaluate the possible performance improvement.

B. Previous work

Previously, we proposed a non-contextual approach and a contextual one to model user behavior. We briefly discuss them here.

1) Non-contextual approach: The non-contextual approach [16] only considers the historic sessions of peers to estimate their stability through exponential moving average and the Bayes rule. We also proposed the first elements of a basic receiver-driver stabilization strategy that, by using estimations of these models, enables to move a stream receiving peer to a stable source before the estimated departure of its current stream provider. A limitation of this strategy is that it does not push the unstable peers away from the streaming source and hence, unstable peers might stay near to the source making it difficult for other peers to find stable providers. Moreover, the sole dependency over the sessions' history might not be able to produce accurate results since users have different interests and viewing habits [17].

2) Contextual approach: The contextual approach [18] presents a Bayesian network model that considers all the known variables involved in the user behavior. These variables have been extracted from user behavior measurements through an exhaustive synthesis. A Bayesian network model is derived from these relationships as shown in Figure 1. Nodes of this model represent variables and directed arcs show their dependency relationships. The model is a mixed one, containing both discrete and continuous variables. Discrete variable are shown through rectangles and continuous ones are depicted through ellipses.

Discrete variables include time-of-day, content type and elapsed time. Time-of-day represents the joining time of a user. This variables has been discretized into 24 states, having one for each hour. The number of states have been chosen through simulations. Content type has three states, namely, reality, fiction and sports. Reality consists of news, talk shows and music, and fiction includes movies, documentaries and serials. Elapsed time has two states. One for channel holding time up to 2 minutes and another for all other values of elapsed time. The former favors surfing mode and the latter favors viewing mode. To estimate the value of any of these variables, the state with the highest probability is taken as its estimated value.

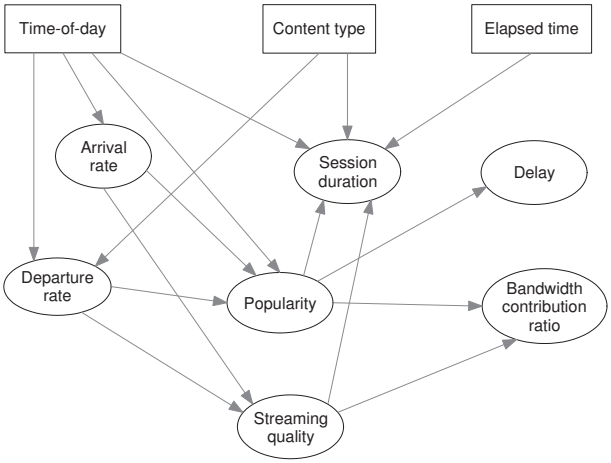


Fig. 1. Bayesian network model of user behavior

Arrival rate, departure rate, popularity, session duration, streaming quality, delay and bandwidth contribution of a peer are continuous variables. All of these variables are Gaussian distributed. The mean value of a continuous variable is taken as its estimated value at a given time.

Since Bayesian networks allow multi-way inference, this model enables to estimate the value of any required variable at a given time. Parameters of the model are learned from data, which encode the strength of dependencies among variables. Nevertheless, it takes long to get properly trained which emerges as a limit. To overcome this problem, a classification mechanism can be coupled with the Bayesian network model. To do so, a Bayesian network can be trained for a class of users and on an arrival of each user, a classifier first classifies it. For estimations, the appropriate trained Bayesian network is assigned to that user. A classifier model is out of the scope of this paper, therefore we do not discuss it in further detail.

III. A USER-AWARE TOPOLOGY MANAGEMENT FRAMEWORK

Managing a P2P system for performance purposes could be achieved in several ways but since these systems build an adhoc overlay topology, controlling it appears as one of the most straight-forward and efficient management strategy. Following this statement, our approach consists in an adaptive topology control mechanism that aims at improving the service quality of P2P live streaming systems. It relies on the estimations of user behavior models to maintain a stable topology. A user behavior model can estimate several behavior metrics but in our work we only use their estimations of current session duration. The online remaining time of a peer shows the stability which can be determined from the length of its current session and its joining time.

A. Design considerations

Making use of stability estimations for performance improvement depends on the underlying system. Primarily, tree-based push systems and hybrid push-pull systems can be benefited the most from these estimations. Tree-based systems are efficient in terms of timely stream delivery but they are vulnerable to stream disruptions due to the departure of peers placed at high level of the tree. These systems can be indeed made practical through building stable tree structures. Similarly, hybrid systems can use these estimations during subscription to receive stream through push strategy. Subscription to stable peers will increase the amount of stream received through push operations which ultimately improves the performance.

In this paper, we focus on the stabilization of tree-based systems which is the most challenging one. Our goal is to achieve a stable tree with the following considerations:

- The approach should be decentralized, since video streaming attracts a large number of users that can lead to scalability issues if managed in a centralized way;
- The overhead involved in the process should remain minimum;
- The approach should not impact other performance parameters negatively such as the increase of startup and playback delay or the lowering of the streaming throughput.

Besides these criteria, we do the hypothesis that peers cooperate in providing their stability information to each other in order to self-organize the topology. Security concerns of such an approach are out of the scope of this paper (e.g. a malicious peer that changes his session time estimation to bias the topology control algorithm).

A stable tree can be achieved at two stages. Firstly, at the joining stage of a peer by choosing a stable content provider. Secondly, by continuously monitoring the structure and performing stabilization after joining. Both stages can also be considered together. However, in the first case, choosing a stable provider peer at the joining time requires probing several other peers which can delay the actual stream delivery. This can potentially increase the startup delay which is not desirable and gives rise to early departures of users. Therefore, we do not consider choosing a stable provider node at the joining time since it will contradict our third criterion.

On the other hand, the stabilization of the overlay can be accomplished in several ways after new peers have joined it. For example, peers can exchange stability information periodically and promote stable peers at higher levels of the tree. Indeed, it requires continuous messages exchanges leading to a high overhead on the system. A receiver-driven strategy avoids this through enabling the stream receiving peer to change to a new stream provider peer before the departure of its current provider. However, a limitation of this approach is that it does not push unstable peers away from the source which limits the options of other peers to find stable stream providers forcing them to remain connected to unstable ones.

Therefore, we propose a novel strategy in which the unstable provider peers move themselves to the outskirts of the tree, hence allowing stable peers to stay near to the source. It ensures a stable tree which is resilient to peers' departures with low overhead involved. We discuss this approach in detail.

B. Maintaining a stable tree

We propose a cooperative strategy that ensures the building and maintenance of a stable tree. As mentioned earlier, this approach does not modify a peer's joining mechanism specified in the system. After joining, each peer records its joining time and estimates its current session duration. It schedules a swap with a child peer to be carried out at the end of the current estimated session. The child is chosen at the time of swap. The goal of the swap is to choose the most stable peer among the child peers and swap its own position with it. This process is detailed in two algorithms and we explain them through the help of Figure 2. The algorithms are formalized with a pseudo-code syntax. Moreover, we assume that each node provides (1) basic functionalities that enable the retrieval of the node's parent and its child peers and (2) protocol primitives to send and receive topology control messages. Such an API is depicted in Algorithm 1. Finally, each message exchanged between peers is composed of dedicated fields that we represent through a dot separator.

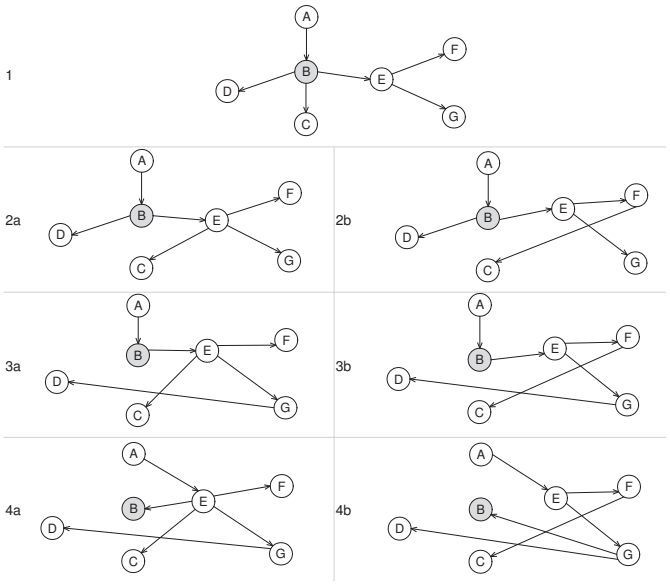


Fig. 2. Unloading a departing peer

Algorithm 1 Basic operations of a node

```
1: void: JOIN
2: Node: GETPARENT
3: List{Node}: GETCHILDREN
4: void: SEND(Message, Node)
5: Message: RECEIVE(Node)
```

1) Selecting stable nodes: Algorithm 2 is run by a peer as its estimated session duration is going to expire. We assume peer *B* as a departing peer in Figure 2. In the first step, *B* queries its child peers (if any, otherwise peer is already at the leaf) for their estimated remaining time in the session. Each child peer estimates its remaining time in the current session from its estimated current session duration and joining time and provides this information to its departing parent. All this process is shown in statements 2 to 10.

Algorithm 2 Choosing a stable child for replacement

```
   Declaration:
   LRT: Integer                          ▷ Longest Remaining Time
   child, SC: Node                       ▷ Child Identifier
   RT: Integer                           ▷ Remaining Time
   CT: Integer                           ▷ Current Time
   RTQM: Message                         ▷ Remaining Time Query
   RTRM: Message                         ▷ Remaining Time Response
   CQM: Message                          ▷ Child Query
   CRM: Message                          ▷ Child Response
   CPL: List{Node}                       ▷ Child Peers
   RTRL: List{RTRM}                      ▷ Remaining Time Responses
   PPL: List{Node}                       ▷ Potential Parents
 1: function STABLENODESELECTION
 2:     CPL ← GETCHILDREN
 3:     for i ← 1, CPL.size() do
 4:         child ← CPL.get(i)
 5:         SEND(RTQM, child)
 6:     end for
 7:     while RTRL.size() < CPL.size() do
 8:         RTRM ← RECEIVE(any)
 9:         RTRL.add(RTRM)
10:     end while
11:     LRT ← 0
12:     for i ← 1 : RTRL.size() do
13:         RTRM ← RTRL.get(i)
14:         RT ← RTRM.RT
15:         if RT > LRT then
16:             LRT ← RT
17:             SC ← RTRM.source
18:         end if
19:         PPL.add(SC)
20:     end for
21:     SEND(CQM, SC)
22:     CRM ← RECEIVE(SC)
23:     PPL.add(CRM.grandChildren)
24:     return PPL
25: end function
```

In the next step, the departing parent analyzes each response and chooses the child with the longest remaining time as a stable one to replace it. This process is given in statements 11 to 20. In our case, we suppose this peer is *E*. For *B* to proceed a swap with *E*, *A* can directly replace *B* by *E*, since it does not require extra resources from *A*. However, for *B* becoming a child of *E* is not straightforward. *E* may have already filled its outgoing capacity and in this case it will be unable to accept *B* as a child peer and hence a swap would not be possible. Moreover, the other child peers of *B*, namely *C* and *D*, should not ideally remain connected to *B* since it is no more stable.

Therefore, *B* requires a list of potential parents for itself and its other children.

To get that list, *B* sends a request to its stable child *E* for its children. *E* responds with the list including *F* and *G*. Finally, *B* prepares a list of potential parents including *E*, *F* and *G*. This process is shown in lines 21 to 24. After having the list of potential parents, peer *B* is now ready to be replaced by *E*. We describe this process in Algorithm 3.

Algorithm 3 Processing the swap

Declaration:
$accepted$: Boolean ▷ Child acceptance
$parent$: Node ▷ The parent Node
RRM: Message ▷ Replacement Request
RAM: Message ▷ Replacement Acknowledgement
MM: Message ▷ Move Message (to inform a child move to another parent)
JM: Message ▷ Join Message (to subscribe to a parent)
JAM: Message ▷ Join Acknowledgement

```
1:  procedure NODESWAP(PPL: List{Node}, SC: Node)
2:      CPL ← GETCHILDREN
3:      for i ← 1, CPL.size() do
4:          MM.PPL ← PPL
5:          child ← CPL.get(i)
6:          SEND(MM, child)
7:      end for
8:      parent ← GETPARENT
9:      SEND(RRM, parent)
10:     RAM ← RECEIVE(SC)
11:     CPL.remove(RAM.source)
12:     if SAM.status = false then
13:         accepted ← false
14:         PPL.remove(RAM.source)
15:         for i ← 1, PPL.size() do
16:             PP ← PPL.get(i)
17:             SEND(JM, PP)
18:             JAM ← RECEIVE(PP)
19:             if JAM.status = true then
20:                 parent ← JAM.source
21:                 accepted ← true
22:             end if
23:         end for
24:         if accepted = false then
25:             JOIN
26:         end if
27:     else
28:         parent ← RAM.source
29:     end if
30: end procedure
```

2) Processing the replacement: In the first step, the departing peer (*B*) informs its remaining child peers (*C* and *D*) to choose a new parent among the potential parents as shown in lines 2 to 7. Each child node iteratively requests the potential parent to join, unless one of them accepts it as a child. If the list is exhausted and none of the potential parents accepts a child, then it performs a random rejoin. A peer should accept a child if it has the available capacity. These steps are shown in Figure 2 (2a, 2b, 3a and 3b). Then, it sends a replacement request to its parent *A* (line 9), asking to replace it by peer *E*. Parent *A* replaces child *B* by *E* and informs *E* that it has replaced peer *B*. Peer *E* updates its parent and checks if it can add *B* as a child. If it has the available capacity to add a new child, it adds *B* as a child, otherwise it does not add *B*. In both cases, it sends an acknowledgement message to *B*. These two cases are shown in Figure 2.4a and Figure 2.4b respectively.

After receiving the replacement acknowledgement response from *E*, *B* removes *E* from its children list as shown in lines 10 and 11. It then checks whether it is accepted as a child or not by *E*. In the former case, it just adds its previous child as a parent. In the latter case, first it removes this peer (*E*) from the potential parents list and then it iteratively requests each peer of this list, unless one of them accepts it as a child. In case of failure, it performs a new join. All this process is given in lines 12 to 29. In both these cases, one can notice that peer *B* has no child peer now (Figure 2.4a and Figure 2.4b) and if it departs, it will not disrupt the stream to other peers, consequently improving the quality of streaming.

IV. EXPERIMENTAL EVALUATION

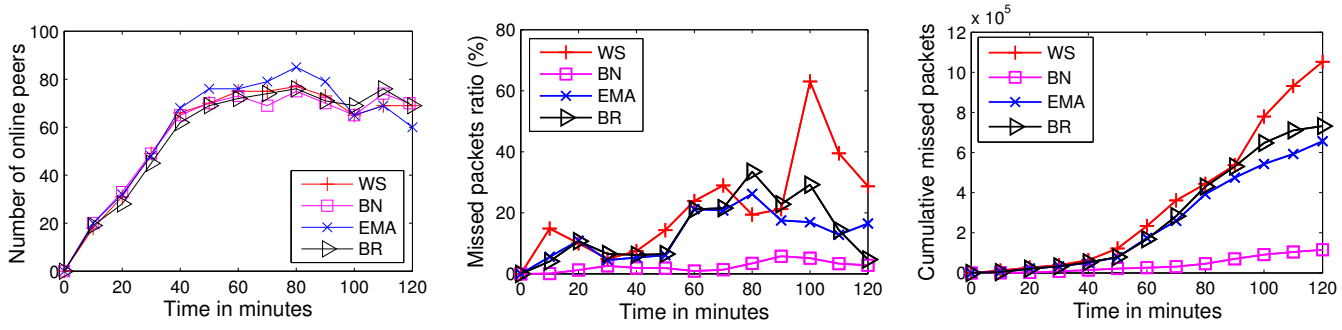
In order to evaluate the cost and benefits of our user-aware topology management strategy and show under which conditions it enables the improvement of the perceived Quality of Service, we performed wide-area experiments over PlanetLab [19] through a Java-based prototype implementation. PlanetLab[1] is a globally distributed testbed for networking and distributed systems' research that provides a realistic environment for the evaluation of Internet-scale applications. The objectives of our evaluation are twofold. They mainly consist in validating the proposal of this paper, but also enables us to compare different strategies that can be considered to estimate the session duration of a user which is crucial for the efficiency of any subsequent control mechanism. We discuss them in the following of this section.

A. Experimental framework

Our prototype implementation relies on the FreePastry[2] open source implementation of Scribe [20]. We choose this system for the assessment of our approaches, because while being the most efficient stream delivery approach, its single tree structure presents serious challenges towards the provision of QoS-enabled live streaming services. Moreover, its open source availability enables us to integrate our approach into it.

We add a five seconds smoothing buffer to Scribe and analyze its performance over PlanetLab. We use 60 nodes, and if the online population of users exceeds this number, we launch more than one peer on a same PlanetLab node, uniformly distributed over all nodes in the slice. To deploy, execute and monitor our application, we use PlanetLab Experiment

[1] www.planet-lab.org
[2] http://www.freepastry.org/



(a) (b) (c)

Fig. 3. (a) Peers' population; (b) Missed packets during a 10 minutes slot; (c) Cumulative missed packets

Manager (PIMan[3]). Dummy packets with a bit rate of 64 Kbps are broadcasted over the tree. The streaming source is fixed throughout each experiment, behaving as a server instead of a peer controlled by a user. Other peers remain online during their specified life times in the traces. The out-degree of each peer is limited to three.

In total, we perform four distinct experiments, one Without Swap (WS) in which we use Scribe in its original form. Second, we use the stabilization strategy based on the estimations of our Bayesian network (denoted as BN) model coupled with a classifier. For the remaining two experiments we utilize the estimations of two non-contextual models presented in section II-B. We shortly denote them by EMA (Exponential Moving Average) and BR (Bayes' Rule) respectively. These experiments enable us to compare the performance gain of our stabilization strategy over the existing system according to different session duration estimation strategies.

We choose to replay the two hours traces generated through a semi-Markovian model [21]. Starting from the first minute, a peer is entered into the system each time its joining time corresponds to the current one. It remains online for a period equal to its actual session duration, while its estimated session duration through one of the three estimators is used by the stabilization strategy.

B. Results analysis

Throughout each experiment, we measure missed packets, initial streaming quality, buffering delay, average delay, played packets and the number of control messages added by the stabilization strategy. These latter are to measure the overhead involved due to our strategy. All other elements are important QoS metrics which impact users' QoE. The ratio of missed packets is an important metric in these experiments since pure push-based systems are efficient in terms of other metrics such as buffering and playback delays. We include the analysis of these latter metrics to investigate if an improvement in packet losses negatively impact other performance parameters, which is the case in pure pull-based systems.

Before discussing these results, we plot the online number of users during each experiment in Figure 3.a. These plots clearly show that all experiments are performed in similar conditions, since all the curves present similar shapes. Small variations occur due to the underlying unstable platform, which may lead to peers' failures. It might be the case even on an actually deployed P2P streaming system over the Internet.

1) Missed Packets: Missed packets are those which do not meet the playback deadline. At startup each peer waits for its buffer to be filled and then starts picking packets at the source's transmission rate. The playing position is initialized at the first packet a peer receives. In the meanwhile, a packet not present in the buffer is counted as a missed one. This metric indicates the video playback quality experienced by a user.

Figure 3.b depicts the ratio of missed packets to played ones, averaged over each 10 minutes period during all experiments. One can notice that Scribe without swap (WS) misses a larger number of packets than other approaches at most of the instances. By contrast, BN shows the least packet losses. To further elaborate these results, we also depict the cumulative packet losses in Figure 3.c. From these results, it is obvious that BN outperforms other approaches in terms of packet loss. Overall, BN reduces the packet loss of Scribe without swap by 89%, EMA by 38% and BR by 30%. This is a significant improvement over the existing system, showing the efficacy of swap and user behavior estimators, especially BN.

Another interesting aspect is the sudden increase in the number of missed packets around 45 minutes time. It means that once the long lived peers start departing, they impact the system more severely, since they most probably occupy higher positions in the tree. It also contradicts the older stable principle used in a few approaches such as [11], [7], [10].

2) Control overhead: The overhead of our swap mechanism lies in the number of messages sent by all peers to stabilize the tree. We measure the number of control messages added by the stabilization strategy. The total number of messages sent by all peers grouped by 10 minutes period are shown in Figure 4.a. Similarly, Figure 4.b depicts the cumulative control messages during each experiment.

BN appears to incur a larger overhead among the others but the amount of this overhead is negligible considering the number of messages peers exchange for the overlay management.

[3]http://www.cs.washington.edu/research/networking/cplane/

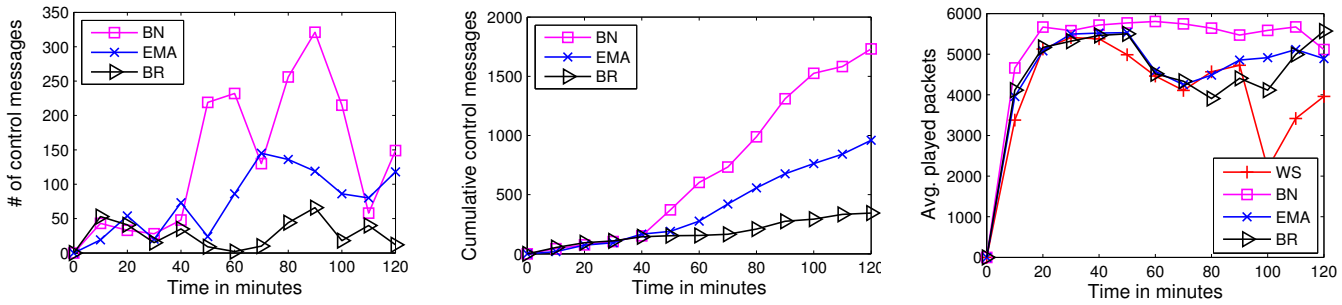


Fig. 4. (a) Control messages over a 10 minutes slot, (b) Cumulative control messages, (c) Packets meeting the playback deadline

For instance, an average of 14.4 messages per minute have been sent by all peers present in the system, which represent a very small overhead.

3) Playback delay: Playback delay is the time elapsed from the creation of a packet until it is picked by the player. Here, we remind that a five seconds buffer is used to smooth the playback, therefore a minimum of 5 seconds delay becomes inevitable.

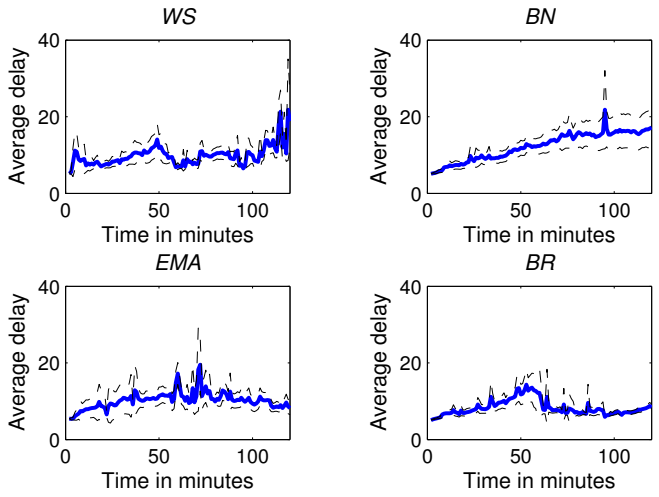


Fig. 5. Comparison of average playback delay (in seconds)

Average playback delays of each approach are depicted in Figure 5 with their 95% confidence intervals. Since pure push-based systems are efficient in timely delivery of content, one can notice that in all approaches mostly the average delay remains lower than 15 seconds. However, comparing these approaches indicate that delays with BN are slightly longer than with other approaches in the second hour.

To explain, this we plot the total number of packets that met the playback deadline at each peer during one minute time in Figure 4.c. It is noticeable that with BN, consistently a larger number of packets were received in time during the second hour. However, due to churn, delays were increased. By contrast, other approaches simply missed those packets. Therefore, we conclude that BN does not increase delays considerably, while reducing packet losses.

4) Initial streaming quality: Liu *et al.* [15] measure the initial streaming quality from the initial buffer level. We use the same metric in our experiments. The buffer level of each peer is checked after five seconds time since joining. Figure 6 depicts a scatter plot of initial streaming quality of all peers.

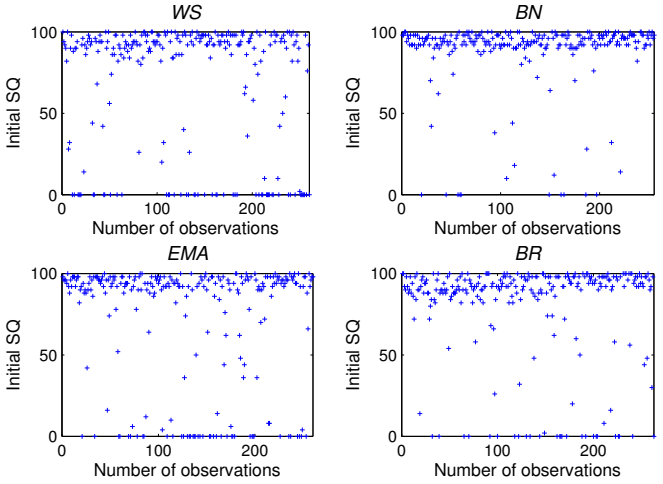


Fig. 6. Initial streaming quality in terms of buffer level

At a first look, all approaches seem to produce similar results once again due to the push strategy of the system. However, a detailed analysis shows that BN achieves better streaming quality than all other approaches. For instance, we estimate the ratio of peers with initial buffer level of 90% or more to the total number of peers. We find that with BN, 84.7% of peers achieve this level, which is the highest among all. Concerning other approaches, Scribe without swap achieves this level for 58.8%, EMA for 65% and BR for 66% of peers. It shows that a stable overlay not only reduces packet losses but also improves the initial streaming quality.

5) Buffering delay: Finally, we measure the time period required by each peer to fill its buffer after joining. This estimate is important for the quality of streaming perceived by a user since a user has to wait during buffering before the playback starts. We show a scatter plot of buffering delays of all approaches in Figure 7.

Here, again with BN, delays remain lower in most of the

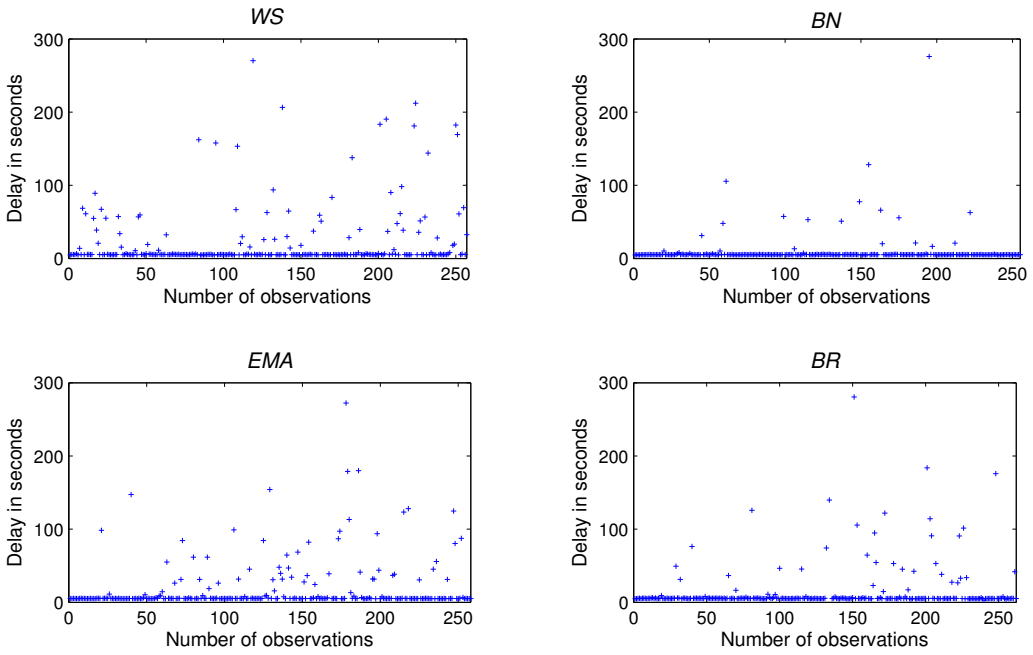


Fig. 7. Buffering delay

cases. As an example, we analyze the cases in which buffering delays remaind under six seconds. BN fulfills this condition in about 90% of cases. Scribe without swap achieves the same for 69%, EMA for 71% and BR for 79% of peers, emerging BN as a clear winner. Once again, this shows the positive impact of stabilization over the quality of streaming.

In our wide-area experimentations, we analyzed the performance gain of our strategy. We evaluated the effectiveness of our swap strategy over a contextual and two non-contextual estimators. We measured the stream disruption that occurs due to the departure of stream provider peers. We noticed that our strategy with contextual model (BN) reduces the impact of departures significantly. Moreover, it induces an acceptable overhead. On the other hand, this approach does not impact QoS and users' QoE parameters negatively. All these results validate the concept of user behavior integration in P2P streaming systems for quality improvement.

V. Conclusion and Future Work

P2P systems highly depend on the behavior of end-users and their integration in control and management algorithms is a crucial step toward the design of systems able to deal with SLA and that can be deployed in legally operated contexts. In this paper, we focused on the integration of user behavior models in P2P live streaming systems in order to adapt them for improved streaming quality. More precisely, in the context of push-based systems, we presented an autonomous topology management strategy that constantly moves unstable nodes towards the leaves of the tree, reducing the impact of their departures. We evaluated the performance gain of this strategy relying on the estimations of a Bayesian network coupled with a classifier and two non-contextual models. We also developed a prototype implementation of our proposal and performed experiments over PlanetLab, which show that the stabilization strategy improves the overall streaming quality of the system by reducing the packet losses due to churn. Particularly, relying on BN, it produces better results than non-contextual approaches. Experimental results indicate that considering user behavior improves the streaming quality and contextual approach of user behavior modeling can better estimate the behavior.

Research directions opened by this work are numerous. Concerning short term future work, the use of our model can be extended to include other metrics such as bandwidth contribution and streaming quality of a peer in forming the stream delivery topology. As an example case, a tree structure that progressively promotes stable peers with a good contributing ratio and streaming quality to higher levels will improve the overall throughput of the system. Similar approach can be applied to hybrid push-pull systems in which the amount of pushed stream can be increased through letting peers subscribe to stable upstream peers with sufficient available bandwidth and good streaming quality. Concerning pull-based systems, our model can be used in decision making for chunk scheduling. For instance, pulling chunks from highly contributing peers with enough available upload bandwidth can improve the performance. Concerning long term perspective, our work opens a way toward the design of user-aware management systems that directly integrate the user behavior in control algorithms. The presented use-case focused on topology management but it can be extended to several management functional areas such as capacity planning.

REFERENCES

[1] Y. Liu, Y. Guo, and C. Liang, "A survey on peer-to-peer video streaming systems," *Peer-to-Peer Networking and Applications*, vol. 1, pp. 18–28, 2008.

[2] T. Silverston, O. Fourmaux, A. Botta, A. Dainotti, A. Pescapé, G. Ventre, and K. Salamatian, "Traffic analysis of peer-to-peer IPTV communities," *Comput. Netw.*, vol. 53, no. 4, pp. 470–484, 2009.

[3] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," *IEEE Journal on Selected Areas in Communication*, vol. 20, no. 8, pp. 1456–1471, 2002.

[4] M. Bawa, H. Deshpande, and H. Garcia-Molina, "Transience of peers & streaming media," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 107–112, 2003.

[5] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "CoolStreaming/DONet: A data-driven overlay network for Peer-to-Peer live media streaming," in *INFOCOM*. IEEE, 2005, pp. 2102–2111.

[6] N. Magharei and R. Rejaie, "PRIME: Peer-to-Peer Receiver-drIven MEsh-based streaming," in *INFOCOM*. IEEE, 2007, pp. 1415–1423.

[7] F. Wang, Y. Xiong, and J. Liu, "mTreebone: A collaborative tree-mesh overlay network for multicast video streaming," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, pp. 379–392, 2010.

[8] S. Xie, B. Li, G. Y. Keung, and X. Zhang, "Coolstreaming: Design, Theory, and Practice," *IEEE Transactions on Multimedia*, vol. 9, no. 8, pp. 1661–1671, 2007.

[9] I. Ullah, G. Doyen, G. Bonnet, and D. Gaïti, "A survey and synthesis of user behavior measurements in p2p streaming systems," *IEEE Communications Surveys & Tutorials*, 2011.

[10] Y. Tang, L. Sun, J.-G. Luo, and Y. Zhong, "Characterizing user behavior to improve quality of streaming service over P2P networks," in *PCM*, 2006, pp. 175–184.

[11] F. Wang, J. Liu, and Y. Xiong, "Stable peers: Existence, importance, and application in Peer-to-Peer live video streaming," in *INFOCOM*. IEEE, 2008, pp. 1364–1372.

[12] S. Horovitz and D. Dolev, "Collabrium: Active traffic pattern prediction for boosting P2P collaboration," in *WETICE*. IEEE, 2009, pp. 116–121.

[13] B. Liu, Y. Cao, Y. Cui, Y. Xue, F. Qiu, and Y. Lu, "Minimizing service disruption in peer-to-peer streaming," in *IEEE CCNC*, 2011, pp. 1066–1071.

[14] Q. Zheng, Y. Long, T. Qin, and L. Yang, "Lifetime characteristics measurement of a P2P streaming system: Focusing on snapshots of the overlay," in *WCICA*, 2011, pp. 805–810.

[15] Z. Liu, C. Wu, B. Li, and S. Zhao, "Distilling superior peers in large-scale P2P streaming systems," in *INFOCOM*. IEEE, 2009, pp. 82–90.

[16] I. Ullah, G. Bonnet, G. Doyen, and D. Gaïti, "Improving performance of ALM systems with Bayesian estimation of peers dynamics," in *MMNS*. Springer, 2009, pp. 157–169.

[17] A. Rudstrom and M. Sjolinder, "Capturing TV user behaviour in fictional character descriptions," Swedish Institute of Computer Science, Tech. Rep., October 2008.

[18] I. Ullah, G. Doyen, G. Bonnet, and D. Gaïti, "A bayesian approach for user aware peer-to-peer video streaming systems," *Signal Processing: Image Communication*, 2012.

[19] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: an overlay testbed for broad-coverage services," *SIGCOMM Comput. Commun. Rev.*, vol. 33, pp. 3–12, 2003.

[20] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "Scribe: A large-scale and decentralised application-level multicast infrastructure," *IEEE Journal on Selected Areas in Communication*, vol. 20, no. 8, 2002.

[21] G. Bonnet, I. Ullah, G. Doyen, L. Fillatre, D. Gaïti, and I. Nikiforov, "A Semi-Markovian individual model of users for P2P video streaming applications," in *NTMS*. IEEE, 2011, pp. 1–5.