Adaptive Multicast Streaming for Videoconferences on Software-Defined Networks

Christelle Al Hasrouty, Mohamed Lamine Lamali, Vincent Autefage, Cristian Olariu, Damien Magoni, John Murphy

Adaptive Multicast Streaming for Videoconferences on Software-Defined Networks

Christelle Al Hasrouty[a,b], Lamine Lamali[a], Vincent Autefage[a], Cristian Olariu[b], Damien Magoni[a], John Murphy[b]

[a] *University of Bordeaux, LaBRI, Talence, France*
[b] *University College Dublin, Lero@PEL, Dublin, Ireland*

Abstract

Real-time applications, such as video conferences, have strong Quality of Service requirements for ensuring a decent Quality of Experience. Nowadays, most of these conferences are performed over wireless devices. Thus, an appropriate management of both heterogeneous mobile devices and network dynamics is necessary. Software Defined Networking enables the use of multicasting and stream layering inside the network nodes, two techniques able to enhance the quality of live video streams. In this paper, we propose two algorithms for building and maintaining multicast sessions in a software-defined network. The first algorithm sets up the initial multicast trees for a given call. It optimally places the stream layer adaptation function inside the core network in order to minimize the bandwidth consumption. This algorithm has two versions: the first one, based on shortest path trees is minimizing the latency, while the second one, based on spanning trees is minimizing the bandwidth consumption. The second algorithm adapts the multicast trees according to the network changes occurring during a call. It does not recompute the trees, but only relocates the stream layer adaptation functions. It requires very low computation at the controller, thus making our proposal fast and highly reactive. Extensive simulation results confirm the efficiency of our solution in terms of processing time and bandwidth savings compared to existing solutions such as multiple unicast connections, Multipoint Control Unit solutions and application layer multicast.

Keywords: Software Defined Networking, Videoconference, Multicast, Scalable Video Coding, Quality of Service.

1. Introduction

For the past decade, the usage of video conference applications has considerably grown. One of the main criteria when judging a video conference call is by the quality of its video streams. These streams are usually very sensitive to the network state and have stringent Quality of Service (QoS) requirements, such as low end-to-end delay, low packet loss, high bandwidth, etc. In addition, they are also affected by the heterogeneity of the users' wireless link characteristics, which may degrade the quality of the video streams for some or all users.

Traditional networks find it complex to manage all the QoS requirements in real time. Earlier research has focused on developing new video codecs, such as Scalable Video Coding (SVC) [1], that propose to separate a video stream in several layers of different quality. Thus, these codecs are able to adjust the bitrate of the video stream more easily, simply by dropping unnecessary layers. This allows a dynamic adaptation of the video stream to the characteristics of the end-user access link. While SVC permits a better control over the stream bitrate, it only drops the unnecessary layers at the endpoints or in a Multiple Control Unit (MCU). Layers could not be adapted inside the core network.

The development of Software Defined Networking (SDN) has enabled new possibilities for the complex control of video conferences. This architecture provides a separation between the control plane and the data plane of a network. Thus, it enables a logically centralized view that allows the management of the network in a more efficient way.

In this work, we propose two algorithms that build and manage multicast trees in order to transmit a tailored video stream to each user. We propose two versions of the tree computation. The first version, called *Shortest Path Tree* (SPT), minimizes the end-to-end delay; while the second one, called *Minimizing Spanning Tree*, minimizes the bandwidth consumption in the core network. Both versions optimize the placement of the SVC dropping rules in the multicast tree.

Access bandwidth variations may occur quite often during a call. In order to avoid a complete computation of the multicast tree, that can be costly in processing time, we propose an algorithm that optimally relocates the SVC dropping rules at each bandwidth variation. The algorithm is based on tree traversal ideas and is very fast, thus offering high reactivity.

The paper is organized as follows: Section 2 discusses the state of the art of video conferencing methods, especially in the SDN context. Section 3 describes our proposed solution. Section 4 details the model and assumptions of our proposal for both static and dynamic environments. Section 5 describes the proposed algorithm for setting up a video conference in a static environment. It also discusses its complexity. Section 6 describes the proposed

adaptation algorithm that reacts to bandwidth variations. Section 7 details the performed simulations, and shows the efficiency of our algorithms compared to the state of the art. Finally, Section 8 concludes the paper.

This paper is an extended version of our two preceding conference papers [2] and [3]. Compared to these previous versions, we have restructured together the two (static and dynamic) algorithms to show their complementarity. We have also performed extended simulations to compare our solution to more existing solutions: unicast, MCU-based and application-layer multicast.

2. Related Work

Many efforts have been made on real-time applications in order to improve the QoS/QoE for users while saving network resources. However, device and access network heterogeneity has brought many challenges to the field compared to the rather predictable Plain Old Telephone Service (POTS). To address these challenges, it was necessary to adapt the video streams to each end-user. This led to the creation of a special type of control called *Multipoint Control Unit* (MCU) [4]. An MCU is a central node that connects different end-users of a video conference. The communicating users send video streams to the MCU in order to transmit it the other end-users. The MCU can adapt them if necessary, before forwarding them to the destinations. The adaptation of the stream can be done through transcoding in order to adapt the device and access network heterogeneity. In case of bad reception of any of the end-users, the latter asks the MCU to reduce the quality (via bitrate, resolution, or frame rate reduction) of the transmitted video stream. However, since the MCU is centralized, it becomes a single point of failure. In addition, as the connection between the MCU and the network requires high bandwidth capacity links, this makes this solution very costly.

Another solution for adjusting stream bitrate is shown in application supporting Real-Time Protocol (RTP). RTP supports the notion of "translators" and "mixers", which could reconstruct and mix the high bitrate streams into a single stream, then, translates the audio encoding to a lower bitrate [5].

2.1. Multicasting

An MCU-based system uses unicast to manage data traffic between the video conference participants. However, since unicast consists in sending replicated packets from a source host to each destination, the traffic load in the network becomes very high. Thus, unicast is ineffective in terms of network resources and it presents scalability issues. This led to the creation of IP-multicast as a network layer protocol and service [6].

IP-multicast is a routing technique that allows one or many sources to send IP packets to many destinations. Hence, rather than sending the same packets to each end-host, IP-multicast create a multicast tree where a single packet is sent and is replicated at the suited nodes. Thus, IP-multicast generates less traffic load in the network and increases the QoS.

The underlying algorithmic problem of multicasting is the well-known *Steiner tree problem*, i.e., computing the minimum weight tree connecting a set of nodes in a network. It is NP-complete if the weight is defined as an additive or multiplicative metric. In this case, any practical solution is based on heuristics. Here, we consider two metrics: i) the maximum end-to-end delay between the source and any destination, ii) the total (i.e., sum) bandwidth consumption overall used links of a call. Note that, since the first metric is concave, the problem is polynomial. While the problem remains NP-complete for the second metric[1].

Many protocols were built to perform multicast tree construction. Protocol Independent Multicast-Sparse Mode (PIM-SM) [7] is one of the most well-known. PIM-SM is designed to efficiently establish distribution trees across WANs. Using multicast to deal with multi-party video calls is, therefore, a natural approach.

The deployment of IP-multicast at the network layer led to some problems: high complexity and low scalability. *Application Layer Multicasting* (ALM) [8, 9] was introduced in order to overcome the hurdle of router dependence and to create efficient data delivery without modifying the network. ALM is implemented by the end-hosts instead of the routers. Unlike IP-Multicast, ALM consists of multicasting at the application layer in the end-hosts, while still using unicast at the network layer. There are many research works defining ALM protocols: ALMI [10], ZIGZAG [11], NICE [8], and OMNI [12], to take a few. While NICE reduces the network resource usage, it does not optimize the end-to-end delay, which is very important for real-time systems. Unlike NICE, OMNI and ZIGZAG minimize the average delay. However, ZIGZAG faces bandwidth load on the nodes close to the source. On the other hand, ALMI is subject to a single point of failure due to its centralized nature. While ALM solutions overcome the deployment and maintenance limitations of IP multicast, it has very few information about the network topology, compared to IP multicast.

Recently, some works started involving SDN in order to improve multicasting. This involvement is called Software Defined Multicast (SDM). The combination of SDN features with multicasting enables better deployment of new routing algorithms. SDM can overcome the limitations of traditional IP multicast. Thanks to the centralized view, the recalculation of multicast routing tables has become more flexible and reusable. Applications like live video

[1] Note that in a single path computation, delay is an additive metric and bandwidth is a concave one. However, in our specific problem about multicast trees, we consider as objective functions, the *maximum* delay between any pair of end-users, and the *sum* of the used bandwidth over all the links. This makes the first metric concave and the second one additive.

streaming, video and audio conferencing, etc., use multicasting to transmit data to multiple users simultaneously. The existing multicast scheduling (coordinating access to the available channels) algorithms are either not efficient or not scalable. The authors of [13] used SDM to create a multicast solution for fat-tree data center networks to ease the connection/disconnection of a user in a multicast group. This solution has better performances compared to the existing multicast scheduling algorithms. Another work [14] empowered users in a multicast group by giving them membership control, which does not exist in IP multicast. In addition, this work has developed a new approach for computing multicast trees. The results show a decrease in the number of flow table entries.

2.2. Video Layering

Since the video quality of a video conference call is highly dependent on the available bandwidth, video codecs able to dynamically adapt to the network characteristics have been developed. SVC is a layer-based video codec that divides a video stream into different layers of different qualities. All the layers are built upon a base layer with minimal quality. SVC allows the users to select the video layer corresponding to their capacity. Some works (ex. [15] and [16]) use these mechanisms to improve the participants' Quality of Experience (QoE). The authors of [15] used SVC to improve performance of video streaming over wireless shared channels. While the authors of [16] used the dropping mechanism to balance between the bandwidth utilization and video quality. However, the video layering adaptation in these works can only be performed at the application layer, and thus the network bandwidth consumption cannot benefit from it.

2.3. SDN Applied to Video applications

SDN architecture separates the network control from the forwarding functionalities. Thus, it enables a direct programmability of the network and moves the control to the application layer. Since SDN can provide a logically centralized view of the network topology, it may ease the management of the video delivery and improve the QoS. Nowadays, Content Providers such as Netflix and YouTube are responsible for most of the Internet video traffic. Thus, providing the best QoE has become the main challenge. Video quality degradation is mainly caused by bandwidth overload that increases playback buffering, startup delays for video streaming and exceeds latency bound for live/interactive video applications. Researchers today are applying SDN to implement less complicated techniques for bandwidth estimation, better resources utilization and rate adaptation used by real-time and streaming video providers. To enhance video delivery, several works (ex.[17], [18] and [19]) propose new routing methods over SDN that reduce packet loss rate and delay for enhanced video delivery. the work in [17] beets Dijkstra and Bellman–Ford algorithms with a genetic algorithm for finding the optimal path. It reduces the packet loss rate while improving the throughput and the peak signal–to–noise ratio (PSNR). The authors of [18] propose a routing algorithm that provides a trade–off between the delay and the packet loss while selecting non-shortest paths. It makes sure of introducing more packet losses to a path that maintains a tolerable delay. The authors of [19] minimize packet loss and latency by proposing a dynamic QoS routing algorithm that routes some flows based on a Constraint Satisfaction Problem (CSP). Some works (ex.[20] and [21]) use SDN to focus on providing the best quality bitrate to the end-user. The authors of [20] improve the quality of scalable video streaming by proposing dynamic rerouting of QoS streams with a minimum disturbance on best-effort traffic. The authors of [21] build a video control plane which enforces video quality fairness among concurrent video flows generated by heterogeneous end-users. The bitrate adaptation assistance showed the best results in terms of video quality fairness among clients. On another hand, some researchers work on improving the bandwidth utilization in an SDN environment (ex.[22] and [23]). The authors of [22] propose new APIs as an extension of OpenFlow. These APIs redirects some flows to a rate-limiters for a better use of the aggregated bandwidth. A DASH-aware networking architecture based on SDN was proposed by the authors of [23]. This proposal enables the better configuration of the network and defines how bandwidth should be shared between video traffic and other traffic types, and among video players. Several works focus on the problem of network resource management to achieve a better resource utilization in SDNs. The authors of [24] propose to monitor the QoS metrics of each flow to determine adequate parameters. By examining flow at the source and destination switches, accurate results can be obtained while minimizing the network and switch CPU overhead. In [25], the authors present a model based on queuing theory to evaluate the impact of different processing strategies. Another approach, presented in [26], takes into account device and network requirements to optimize the QoE for all video streaming devices in a network.

The authors of [27] and [28] propose that the SDN controller directly manages the multicast tree construction and the video layering inside the network. The work in [27] minimizes the bandwidth consumption while providing an acceptable video quality stream of the participants. The work in [28] reduces the packet-loss ratio while ensuring the bounded delay between all sources and destinations.

Some work combine SDN with video layering. The authors of [29] propose SDM2Cast, an SDN-based, scalable multimedia multicast streaming scheme, where each SVC video layer has its own multicast tree. The proposed scheme optimizes multimedia flow management and provides scalability and flexibility. The authors of [30] focus on how to solve the SVC video manycast problem of efficiency coming from multiple sources to multiple destinations. They designed two heuristics to achieve the optimal solutions for small-scale problems. Several works (ex.

[31] and [28]) use SDN with SVC to reduce the number of video freezes and the bandwidth consumption, respectively. However, the first one only applies to one-to-one video delivery, while the second focuses on the participants' screen size capacity and does not take into account the network access link properties. Both works keep SVC functionalities at the end users or in the MCU. Since SVC is capable of adjusting the bitrate of a video stream and of reducing its load, it would be interesting to use it in the core of the network. The work of [32] create a distributed architecture for SDN network control in order to comply with QoS constraints. However, the authors use SVC in unicast mode. More recently, the authors of [33] and [34] use SVC and SDN to propose a method that provides admission control, in-network adaptation, and supports heterogeneous devices having different display capabilities.

2.4. Dynamic aspects

The heterogeneity of participants' access link bandwidth is already an issue. Moreover, the access links are *dynamic*, i.e., subject to variations during a single call. This means that a video conference characteristics evolve continuously and that the network should adapt to the new characteristics.

There is an extensive literature on video calls under bandwidth variations. For example, the authors of [35] provide an architecture and different mechanisms to perform video streaming that adapts to congestion and bandwidth variations. It investigates both unicast and multicast methods. But the congestion control is performed at the application layer, and thus at the endpoints of the communication. The network resource consumption cannot benefit from it.

The authors of [36] investigate the effect of congestion and bandwidth variation on a Skype video call. However, the case study includes only two hosts and, again, the adaptation mechanism is performed within the application layer. Some other works focus on the reactivity to bandwidth variation. For example, the authors of [37] propose a cross-layer approach where the *Bit Error Rate* (BER), and thus the bitrate, is directly estimated on the physical layer. This can allow a quick adaptation.

More recently, the new approach of pseudo-analog video transmission, which allows a direct adaptation of the video stream quality according to the channel noise, is investigated in [38], [39], but, again, the video quality adaptation is done at the endpoints. In the case of the multicast approach, this has two consequences: i) If the video quality adaptation is performed at the sender, it will adapt to the video quality of the receiver with the lowest bandwidth. This means that receivers with higher bandwidth cannot benefit from a quality corresponding to their access channel. ii) If the adaptation is performed at the receivers, the sender has to emit the highest quality stream through the multicast tree, which implies higher bandwidth consumption in the core network.

3. Proposed Solution

The problem we are facing is more complex. After computing the multicast trees, we aim at optimally placing the SVC dropping rules in the trees. By *optimally*, we mean minimizing the bandwidth consumption while providing the maximum video stream quality that an end-user can receive. This consists of placing the SVC dropping rules as close as possible to the source.

In order to adapt the streams inside the network and minimize the core bandwidth consumption, without the need of a specific MCU server to manage the control, we propose algorithms leveraging SDN architecture that minimize the bandwidth consumption in the core network while providing the best possible video quality to each participant. We assume switches having SVC functionalities [2].

Our algorithm computes a multicast tree from each sender to all the receivers. It provides two modes: the first one minimizes the end-to-end delay, by computing the shortest path tree (hence the name SPT); the second one, called *Minimizing Spanning Tree* (MST), minimizes the bandwidth consumption and proceeds iteratively. The receivers are first ordered according to their access link bitrate. Then the shortest path between the sender and the first receiver is computed and added to the multicast tree. Afterward, each receiver is linked to the tree through the shortest path to *any node* already in the tree. Note that this algorithm is different from a classical *minimum spanning tree* algorithm. It is a heuristic since the problem is NP-complete.

Once the tree completely built, the algorithm places adaptation rules to drop the unnecessary video layers at optimal locations (on SVC switches), i.e., as close as possible to the sender.

When an access link variation occurs, one can completely recompute the multicast tree thanks to our algorithm. However, this may be costly and long, inducing possible disruptions in the call. To avoid this, we propose an algorithm that does not recompute the tree but relocates the SVC adaptation rules only when and where needed. When a variation occurs at a receiver's access link, the algorithm propagates the new bitrate upward in the tree, until there is no need to move further. This algorithm is very fast and offers high reactivity, thus avoiding possible disruptions.

Our algorithms rely on the SDN controller that provides a logically centralized view of the core network infrastructure and is aware of bandwidth variations. Therefore, the SDN controller is able to define where inside the network, some video layers should be dropped. Thanks to the video layer dropping, we are able to adapt to the bandwidth capacities of the receiving devices. However, note that we do not consider the mobility case where the

[2] A simple way to implement such functionalities is to associate each SVC layer to a specific UDP port. Thus, dropping a layer is done through suited forwarding rules.

end-users may change their access node during a video call. This is left for a future work. Note also that the end-users are not forwarders of the different streams, but the access nodes are able to perform this task.

4. System model

In this section, we describe the technical assumptions and the formal model used in this paper.

4.1. Technical assumptions

When a video conference is set between a set of users, they are called participants. A participant is, at the same time, a sender of its video stream and a receiver of video streams from all other participants. we assume that the participants are connected to any node belonging to the network, and that several participants can be connected to the same node. Participants are considered to be connected wirelessly to their node by a 4G cellular technology such as LTE. Each node of the network is supposed to provide a wireless access function (such as an eNode-B) and an SDN switching function. As the network is SDN-enabled, we assume that all nodes within the network contain OpenFlow switches and can communicate with an SDN controller that is responsible for the management of the calls. All switches are supposed to be able to adapt SVC streams by dropping unnecessary layers on the paths indicated by the controller.

SVC streams are structured in layers, all built upon a base layer. We assume the SVC layers consisting of a base layer L1, and three enhanced layers L2, L3, and L4. However, our algorithms can be easily generalized to any number of layers. In addition, a given layer can be used only if all the lower layers are also received. Each participant accepts the highest number of layers allowable by its downlink capacity. With only the base layer, the participant will receive the lowest video quality. If the participant's downlink capacity falls below what is required to receive the base layer, it can still remain in the conference if it can at least receive an audio-only stream. If any participant can not receive the audio stream, the call is rejected.

In a video-conference, the highest possible bitrate permitted by the access link bandwidth is recommended in order to achieve a better QoE. Therefore, during a video call, each participant sends the maximum bitrate stream acceptable by any one receiver, and this stream is then degraded (i.e., higher layers are dropped) to adapt to the participants with lower downlink bandwidth capacity. After discovering the network topology and channel conditions using SDN global view, it is significantly easier to identify the locations of the switches where it is necessary to degrade SVC streams.

Our algorithm, executed in the SDN controller for setting up a video conferencing call, builds a multicast tree from each participant (sender) to all the others (receivers). It relies on the following assumptions:

- All SVC layers belonging to the video stream emitted by one sender, follow the same paths. There is one tree for all layers of a given video stream. Since the adaptation are performed in the core network, some branches of that tree may carry only a subset of the layers.
- The SDN controller knows the complete topology of the network, the position of the participants as well as their available uplink and downlink access bandwidth.
- Any SDN switch can degrade (i.e., drop higher quality layers) an SVC stream.

4.2. Formal model

We model the network as a graph $G = (V, E)$, where $|V| = n$ is the number of nodes and $|E| = m$ is the number of links. The set of participants is denoted by $\mathcal{P} = \{1, \ldots, p\}$. There is p multicast trees, one for each participant as a sender. Thus, a multicast tree T_i is associated with a sender s_i and a set of receivers $R_i = \{r_{i,j} \mid i \in \mathcal{P} \wedge j \neq i\}$. For simplicity, we consider that the participants are also nodes of the trees (the receivers as leaves and the sender as the root). For any link specified by a pair of nodes (x, y), the bandwidth[3] (or capacity) is denoted by $B(x, y)$ and the bitrate (the currently used capacity in a specified tree T_i) is denoted by $b_i(x, y)$. In a tree T_i, the parent of a node x is denoted by $P_i(x)$ and the set of its children (if any) by $C_i(x)$. Note that each receiver has the same parent in all the trees, as it has only one access link. For simplification, $B(r)$ (resp $b_i(r)$) can denote the download bandwidth (resp. the bitrate in T_i) of the receiver r. The codec bitrate CB represents the bitrate levels recognized by the codec used in the system. In our case, CB is equal to the SVC levels $\{L1, L2, L3, L4\}$ and the audio layer. Table 1 summarizes the variable notations.

5. Setting a video conference call

In this section, we present our algorithm to set up a video conference call. The algorithm consists of the following general steps:

1. Collect the uplink/downlink bandwidth for each participant.
2. Compute the optimal receiving and sending bitrates for each participant.
3. If it is possible, build a multicast tree from each participant to all the others.

In step 1, the controller backs up the network state, then it retrieves the uplink/downlink bandwidth value of each participant by polling its access switch (the one through which the participant is connected to).

[3]Note that it is possible that $B(x, y) \neq B(y, x)$ if the uplink and downlink bandwidths are not the same.

Notation	Definition
$B(r_{i,j})$	Downlink bandwidth of the receiver $r_{i,j}$
$B(s_i)$	Uplink bandwidth of the sender s_i
$B(x,y)$	Bandwidth of the link (x,y)
$b(s_i)$	Uplink bitrate of the sender s_i
$b_i(r_{i,j})$	Downlink bitrate of the receiver $r_{i,j}$ in the tree T_i
$b_i(x,y)$	Bitrate of the link (x,y) in the tree T_i
$C_i(r)$	Set of children of a node r in the multicast tree T_i
CB	Set of SVC bitrates
m	Number of links in the network
n	Number of nodes in the network
$\mathcal{P}$	Set of participants
p	Number of participants
$P_i(r)$	Parent of a node r in the multicast tree T_i
R_i	Set of receivers in the tree T_i
$r_{i,j}$	Receiver j in the tree T_i
s_i	Sender i
T_i	Multicast tree associated to the sender s_i
$path$	Shortest path connecting a receiver to a sender or to the closest node of the tree
k	Node where dropping layer is needed

Table 1: Model notations

In step 2, the receiving bitrate of each participant $b_i(r_{i,j})$ is calculated by first dividing its downlink bandwidth $B(r_{i,j})$ by the number of participants excluding itself $(p-1)$, and then by picking the highest bitrate level lower or equal to this computed value. The receiving bitrate represents the highest bitrate which a receiver is able to receive from any sender. The sending bitrate $b(s_i)$ is then defined as the maximum received bitrate level $b_i(r_{i,j})$ found among all the receivers. The bitrate $b(s_i)$ should be lower or equal to its uplink bandwidth $B(s_i)$. As we assume that the uplink bandwidth is always higher than the highest bitrate level, the sending bitrate is thus the same for each participant. However, the sending bitrate may be lower than the highest quality bitrate if no receiver can receive it. Eq.(1) summarizes the computation of the receiving/sending bitrates.

$$\begin{aligned} b_i(r_{i,j}) &\leftarrow \max_{k \in CB} \left\{ k, k \leq \frac{B(r_{i,j})}{p-1} \right\} \\ b(s_i) &\leftarrow \min \left\{ B(s_i), \max_{r_{i,j} \in R_i} b_i(r_{i,j}) \right\} \end{aligned} \quad (1)$$

In step 3, the controller builds a multicast tree from each sender to all the receivers.

5.1. The algorithm

Algorithm 1 builds multicast distribution trees and sets rules for dropping video layers at optimal locations, after computing the best receiving/sending bitrate of each participant.

First, Algorithm 1 takes as input participant s_i and builds its diffusion tree T_i (lines 1-3). The function

Algorithm 1: Setup of the Video conference Call

```
1  Input: a sender s_i with i ∈ P
2  Output: a multicast tree T_i
3  T_i ← {s_i}
4  R_i ← SortReceivers(R_i, s_i)
5  higestBitrate ← max(b_i(r_{i,j})) with j ∈ P/{i}
6  foreach r_{i,j} ∈ R_i do
7      path ← BuildShortestPathToTree(r_{i,j}, T_i) if mode = MST
         // Builds the shortest path from the receiver the closest
         node in T_i
8      path ← BuildShortestPathToSender(r_{i,j}, s_i) if
         mode = SPT // Builds the shortest path from the
         sender s_i
9      if path ≠ {∅} then
10         if r_{i,j} == R_i[0] ∧ b_i(r_{i,j}) == higestBitrate then
11             foreach edge ∈ path do
12                 AddEdge(T_i, edge, b(s_i))
13         else
14             if b_i(r_{i,j}) == b(s_i) then
15                 foreach edge ∈ path do
16                     AddEdge(T_i, edge, b(s_i)) if edge ∉ T_i
17             else
18                 foreach edge ∈ path do
19                     AddEdge(T_i, edge, b_i(r_{i,j})) if edge ∉ T_i
20                     k = edge[1] // k is the intersection node
                         of the added edge and the tree
21                     Rule(k, r_{i,j}) ← (b_i(P_i(k), k), b_i(r_{i,j})))
```

`SortReceivers()` (line 4) sorts the receivers R_i first, from the highest receiving bitrate to the lowest, then from the closest (to the sender) to the farthest. For each receiver $r_{i,j}$, the shortest path is built depending on the mode (lines 7-8). Two modes are defined :

- **Minimizing Spanning Tree (MST)**: in this mode, *BuildShortestPathToTree()* builds a shortest path up to the closest node already belonging to the tree. Thus, MST builds a tree that minimizes bandwidth usage in the network (line 7).
- **Shortest Path Tree (SPT)**: in this mode, *BuildShortestPathToSender()* builds a shortest path to the sender, potentially stopping at the first node of the tree. This mode builds a shortest path tree that minimizes latency between participants (line 8).

Starting from the sender s_i, the first shortest path of the tree is built to the first receiver in the list of sorted receivers, regardless the mode. If no path is found, due to saturated links, the call is rejected. If a path is found, three cases are considered:

- $r_{i,j}$ is the closest receiver to the sender with the highest bitrate. Thus, the computed path is the first in the tree. We simply add the *path*'s edges to the tree T_i with a bitrate equal to the sender bitrate $b(s_i)$. To add an edge between two nodes we use the *AddEdge(tree, (node1, node2), bitrate)* function that takes as parameters the tree to which we are adding

the edge, the edge which is a couple of two nodes and the bitrate associated to this edge (lines 10-12).

- $r_{i,j}$'s receiving bitrate $b_i(r_{i,j})$ is equal to the sender bitrate $b(s_i)$; knowing that *path*, in this case, is not the first shortest path in the tree, its edges might overlap with others in the tree. Thus, before adding new edges, we must check if they already exist in the tree. The bitrate associated to each of the new edges is equal to the sender bitrate $b(s_i)$ which is the maximum bitrate that can be sent (lines 14-16).
- $r_{i,j}$'s receiving bitrate $b_i(r_{i,j})$ is lower than the sender bitrate $b(s_i)$; the edges are added from the receiver up to the sender stopping on the first overlapping edge. In this case, the bitrate associated to each of the new edges is equal to the receiver's receiving bitrate $b_i(r_{i,j})$. Knowing that the existing edges of the tree carry bitrate equal or greater than the current receiving bitrate $b_i(r_{i,j})$, we may need to drop some bitrate layers on the intersecting node k. Thus, using *Rule()*, node k will drop some layers on k to attend the receiver bitrate $b_i(r_{i,j})$ (lines 18-22).

The goal of this algorithm is to minimize the consumed bandwidth. To do so, it places rules that drop layers from a higher quality stream as close as possible to the sender, thus saving the bandwidth in the core network. `Rule()` takes as input a node k and a receiver $r_{i,j}$ and specifies that some bitrate layers will be dropped at node k to reach the acceptable receiving bitrate $b_i(r_{i,j})$ (line 21).

Complexity: As the problem of generating an optimal multicast tree (also known as the Steiner tree problem) is NP-complete [40], we use a single source (sender) approach and the two modes mentioned above for defining a non-optimal heuristic that builds the tree backward from each receiver to a given sender. We then iterate the procedure for every sender. We use Dijkstra algorithm for all nodes in the network, in order to obtain the shortest paths needed by our algorithm, The complexity of an all-pairs Dijkstra's algorithm on a sparse network of size n, with a binary heap implementation, is $O(n^2 \times \log n)$. This can be done upfront. For a video conference call with p participants and k video layers, on a network of diameter D, the time/computation complexity of our algorithm is $O(p^2 \times k \times D)$. Given that k and D are usually constant during a video call, and that $p << n$, the time complexity of our algorithm is very small.

6. Dynamic adaptation

In this section, we present an algorithm that replaces optimally the adaptation rules when a bandwidth variation occurs. As recomputing the trees is costly, we opt for keeping them unchanged. The basic principle of our algorithm is to always guarantee that each participant receives the highest video quality allowed by its bandwidth but with minimum computation. Thus, when a bandwidth variation occurs, the bitrates are recomputed. If there is a bitrate change at a receiver, this change is propagated and the adaptation rules are pushed upward the tree. In the same way, if a bitrate change occurs at the sender, the propagation is done downward. This algorithm is expected to be performed by the SDN controller, and it takes as input the global model of the network in the controller. The forwarding/adaptation rules are then propagated to the SDN switches.

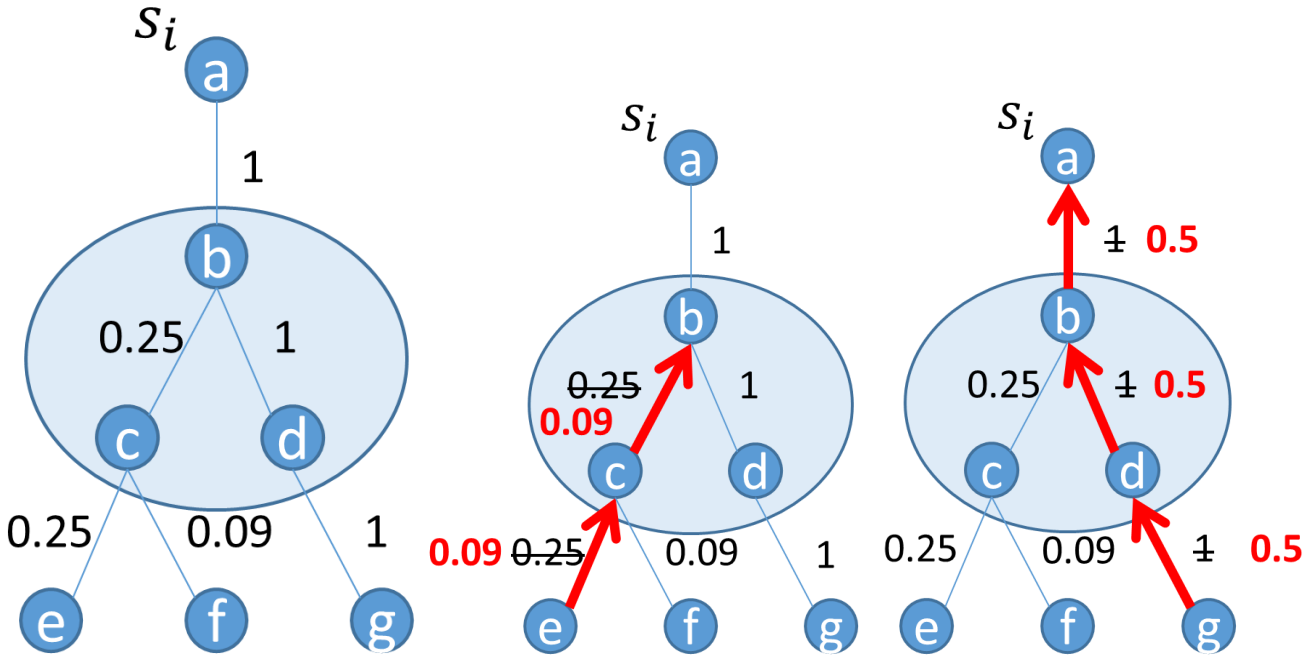
6.1. Change at a receiver

We aim to illustrate with Figure 1 the propagation mechanism when a receiving bitrate changes. Figure 1a depicts a multicast tree where node a is the sender and nodes e, f, and g are the receivers. The other nodes are SDN switches. The bitrate of each link is given beside it. In Figure 1b, because of a bandwidth change, the receiving bitrate of (c, e) is no longer 250Kbps but 90Kbps. While neither e nor f need a stream of 250Kbps, the layer corresponding to this rate is dropped at b, the parent of c. In Figure 1c, the bitrate of (d, g) changes from 1Mbps to 0.5Mbps. This change is propagated until b, but neither c nor d require 1Mbps, thus the bitrate of a is also adapted. This method adapts the bitrate upward from a receiver, but it is infrequent that it reaches the sender. It happens only in the case where the bitrate of the receiver and the sender are the same, and a change of the first one affects the second one. The goal of Algorithm2 is to minimize the computation task while minimizing the consumed bandwidth. To do so, it places the adaptation rules that drop higher quality streams as close as possible to the sender, thus saving the bandwidth in the core network.

Algorithm 2: Relocate Up

```
1 Input: A tree T_i and a node r
2 b_max ← max_{1≤j≤k}(b_i(P_i(r), c_{i,j}) with c_{i,j} ∈ C_i(P_i(r))
3 b_min ← min(b_max, b(s_i))
4 if (b_min ≠ b_i(P_i(P_i(r)), P_i(r)) then
5     b_i(P_i(P_i(r)), P_i(r)) ← b_min
6     AdaptRule(P_i(r))
7     if P_i(P_i(r)) ≠ s_i then
8         RelocateUp(T_i, P_i(r))
9 AdaptRule(P_i(r))
```

Algorithm 2 formalizes this method. It takes as input the receiver where a change occurs, then propagates the change recursively. At each step, it checks if all the children of a node $P_i(r)$ receives less than their parent (line 3). If it is the case, $P_i(r)$ does not need to receive its current bitrate from $P_i(P_i(r))$. The latter bitrate is adapted (line 5). With *AdaptRule*() (line 6), the rules at $P_i(r)$ are updated or deleted in a way that the bitrate received by the node $P_i(r)$ will be the maximum bitrate sent by this node. *AdaptRule*() is further described in Algorithm 4 in the Section 6.3. The algorithm operates recursively (line 8) until it reaches the sender (line 7) or



(a) Tree before bitrate change. (b) Bitrate change propagated. (c) Bitrate change propagated to the sender.

Figure 1: Bitrate change propagation

there is no need to go further, i.e., the bitrate received by $P_i(r)$ has not to be changed.

Complexity: In Algorithm 2, the number of recursive calls is bounded by the height of the multicast tree, which is itself bounded by the diameter of the network. The diameter is in $O(n)$ in the worst case, however, it is much smaller in random and realistic networks. It is well known that the diameter of an Erdős-Rényi[4] graph $G(n, \phi)$ is "almost constant[5]" when ϕ is fixed, and the diameter of a scale-free graph is $\sim \log n / \log \log n$ [41]. At each recursive call, Algorithm 2 checks the bandwidth between $P(r)$ and its children. This suggests that this operation is in $O(p)$ (because the number of children of any node in the tree is smaller than the number of participants). The same argument applies to the complexity of Algorithm 4 (AdaptRule). Assuming that the diameter of the network is in $O(\log n)$, the complexity of Algorithm 2 is in $O(p \log n)$.

6.2. Change at the sender

When the bitrates are recomputed and the value of $b(s_i)$ is modified, the method used is almost the same as the previous section, except that the propagation is performed from the sender to the receivers. Again, the goal is to keep the adaptation rules as close as possible to the sender. The main difference between this case and the previous one is that the propagation is not only done on a branch of the tree but can occur on several branches down to the leaves (receivers).

Algorithm 3: Relocate Down

```
1 Input: A tree T_i and node r
2 foreach c_{i,j} ∈ C_i(r) do
3     if b_i(r, c_{i,j}) ≠ b_i(P_i(r), r) then
4         b_i(r, c_{i,j}) ← b_i(P_i(r), r)
5         AdaptRule(P_i(r))
6         RelocateDown(c_{i,j})
```

Algorithm 3 is performed when the sender's bitrate decreases. It starts at each sender's child. From each one of these nodes, if their own children receive more than the sender's bitrate (line 3), the children bitrates should be changed (line 4) and the rules updated (line 5)using *AdaptRule() further described in 6.3*. The algorithm performs recursively (line 6) until reaching a leaf. Unlike Algorithm 2 that starts from a leaf but infrequently reaches the sender, Algorithm 3 always reaches at least one receiver. Because there is always a receiver that receives the same bitrate as the one emitted by the sender (otherwise the sender could send a lower bitrate), and this receiver's bitrate must be updated.

Complexity: the alg. 3 browses all the tree in the worst case. In the worst case, the size of the tree is n, giving a complexity of $O(n)$. However, in the more realistic case where the tree height is in $O(\log n)$, the size of the tree is at most $O(p \log n)$, which is also the complexity of alg. 3.

[4]Erdős-Rényi graphs are usually denoted by $G(n, p)$ where p is a probability to exist for a link. We replaced it by ϕ in order to avoid confusion with the number of participants.

[5]The diameter of an Erdős-Rényi graph where ϕ is fixed tends toward 2 when $n \to +\infty$.

6.3. Adapting the SVC Downsizing Rules

When incoming and outgoing bitrates are changed at a node, the adaptation rules should be updated. Algorithm 4 takes as input a node r and, for each one of its children, deletes the old rule if it exists and replaces it by the correct one, i.e., the pair (incoming bitrate into r, outgoing bitrate from r to its child).

Algorithm 4: Adapt rules

```
1 Input: A tree T_i and node r
2 foreach c_{i,j} ∈ C_i(r) do
3     if b_i(r, c_{i,j}) = b_i(P_i(r), r) then
4         Delete rule if exists
5     else
6         Rule(r, c_{i,j}) ← (b_i(P_i(r), r), b_i(r, c_{i,j}))
```

Complexity: On any node, there is at worst one rule per child. The number of children of any node is smaller than the number of participants, this gives a complexity of $O(p)$.

6.3.1. The general algorithm

Algorithm 5 is performed at the SDN controller. The call is first established by using Algorithm 1 with either MST mode or SPT mode of Section 5. Then at each event (bandwidth variation on the access links), the controller reacts and adapts the multicast trees. We consider four events:

- $B(r)\downarrow$: The downlink bandwidth of a node r decreases. This may lead to a change of $b_i(r)$ and $b(s_i)$ for each tree T_i according to the formulas in (1). In this case, the bitrates are recomputed and the new bitrate of r is propagated in all the trees (except the one where it is the sender) using Algorithm 2. Thus the complexity of processing this event is in $O(p^2 \log n)$.
- $B(s_i)\downarrow$: The Uplink bandwidth of a node s_i decreases. This can affect the bitrate of s_i and those of the receivers, but only in the tree T_i where s_i is the sender. The change is propagated using Algorithm 3 on the access node of s_i, which is its only child. The complexity is that of Algorithm 3, i.e., $O(p \log n)$.
- $B(s_i)\uparrow$: The uplink bandwidth of a node s_i increases. Again, this can impact the bitrates of s_i and the receivers $r_{i,j}$. After recomputation, if the receivers' bitrates do not change, there is no need to propagate the new bitrate of s_i, because no receiver can get a higher bitrate. Otherwise, the new bitrates are propagated in T_i using Algorithm 2. Likewise the case $B(r)\downarrow$, the complexity is in $O(p^2 \log n)$.
- $B(r)\uparrow$: The downlink bandwidth of a node r increases. This case is more complex since, according to formulas 1, it can impact all the bitrates in all the trees (except the one where r is the sender). For each tree, there are two possible cases. i) The sender's bitrate is not affected, in this case, the new downlink bitrate of r is propagated upward. ii) The sender's bitrate is affected, in this case, it can, in turn, affect the other receivers' bitrates. This case is similar to the previous one where the sending bitrate increases. The complexity of processing this event is in $O(p^3 \log n)$.

Complexity: Note that the number of participants is much smaller than the network size, i.e., $p \ll n$. If we consider the network size as a parameter, Algorithm 5 processes each event in $O(n)$ if the diameter is linear and in $O(\log n)$ in the more realistic case where the diameter is logarithmic. This complexity is much lower than the complexity of MST/SPT that is in $O(n^3)$ if the network is dense and $O(n^2 \log n)$ if the network is sparse. A lower complexity when processing an event implies less consumed resources in the controller and better reactivity for the participants.

Algorithm 5: The general algorithm

```
1  Perform MST or SPT to create a video conference
2  while an event occurs do
3      if Downlink bandwidth of a receiver r decreases (B(r)↓) then
4          Recompute all the bitrates
5          foreach Multicast tree T_i do
6              RelocateUp(T_i, r)
7      if Uplink Bandwidth of a sender decreases (B(s_i)↓) then
8          Recompute the uplink bitrate of s_i
9          RelocateDown(c) where {c} = C(s_i)
10     if Uplink Bandwidth of a sender increases (B(s_i)↑) then
11         Recompute the uplink bitrate of s_i
12         Recompute the downlink bitrates of each r_{i,j}
13         foreach receiver r_{i,j} of T_i do
14             if the bitrate of r_{i,j} increased after recomputation then
15                 RelocateUP(T_i, r_{i,j})
16     if Downlink bandwidth of a receiver r increases (B(r)↑) then
17         Recompute all the bitrates
18         foreach Tree T_i do
19             if the bitrate of s_i increases after recomputation then
20                 foreach receiver r_{i,j} of T_i do
21                     if the bitrate of r_{i,j} increased after recomputation then
22                         RelocateUp(T_i, r_{i,j})
23             else
24                 RelocateUp(T_i, r)
```

7. Evaluation

In this section, we evaluate the efficiency of our static and dynamic algorithms through extended simulations. We compare them to the main state of the art algorithms: unicast, MCU and ALM. More precisely, we study the performance of the different algorithms in terms of processing time, bandwidth saving, maximum delay and network call capacity. The implementation used for MCU and ALM is detailed in Section 7.1.1. All the algorithms were implemented in Python 2.7.10, using the NetworkX package [6].

7.1. Simulation of setting up a video call

In this section, we evaluate the different algorithms for setting up a video call. We detail the simulation methodology and parameters, then we show the results assessing the efficiency of our algorithms.

7.1.1. Simulation methodology and parameters

We evaluate the different algorithms on random topologies according to two models:

- Erdős-Rényi model (ER) [42].
- Magoni-Pansiot model (MP) [43]; where the degree distribution follows a power law (as on the Internet and in large communication networks). It is based on an algorithm that performs a sampling on measured Internet maps.

Using two different models allows us to evaluate the effect of the topologies on our algorithms efficiency.

We have implemented Algorithm 1 in two different modes (MST, STP) and compared them to the unicast, MCU and ALM algorithms.

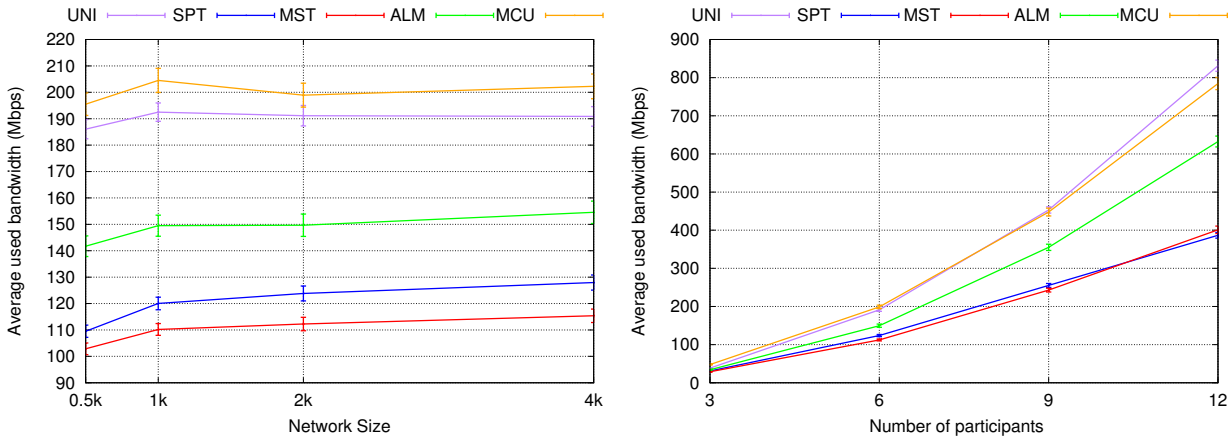
- ALM: This solution is usually based upon a hierarchical clustering of the application-layer multicast peers. However, since our call sessions contain few participants (up to 12), the creation of the clusters is not necessary. Thus, we have implemented a simple form of ALM that creates a multicast tree by connecting each participant to the closest existent participant in the tree. When the tree is built, the sender emits multicast streams to the other participants. The bitrate of the emitted multicast stream corresponds to the sender's capacity. Once the stream is received, each participant adapts it with respect to his own capacity.
- MCU: All the participants are connected to the MCU node. To build the tree of each sender, the algorithm first builds the shortest path from the sender to the MCU (sender-MCU). Secondly, it builds the shortest paths from the MCU to each receiver (MCU-receiver(s)). Then, it connects the first path (sender-MCU) to the others (MCU-receiver(s)) to get the final tree (sender-MCU-receiver(s)). A sender emits a stream with a certain bitrate to the MCU. Knowing the capacity of each receiver, the MCU adapts this stream by choosing the minimum bitrate between the initially sent stream and the receiver's capacity. We note that while implementing MCU, the algorithm finds the shortest path from MCU to all participants and stores it in memory, which may affect the processing time of the algorithm.

To build our simulation system, we consider the following parameters:

- For p participants, p multicast trees will be built.
- We assume the SDN network to be WAN-sized. The network topology sizes implemented are 500, 1000, 2000 and 4000 nodes, which allows us to evaluate the impact of the size of the network on the efficiency of the algorithms.
- Access downlink bandwidths are chosen from a range of plausible 4G data rates for each participant. They may vary from a call to the other. Thus, the access downlink capacity for each participant is set to a value in the range from 4 Mbps to 14Mbps and its uplink bandwidth is set to 1.5Mbps.
- Core links are supposed to all have the same bandwidth dedicated for this type of A/V traffic. This means that links may have different bandwidth capacities but they all reserve the same amount of bandwidth for the video conferencing calls. The core link bandwidth is set to 1Gbps.
- Access links delay is set to 30 ms (a typical average value as shown in [44]) and core link delay is set to 10 ms (as observed in [45]). The maximum latency authorized over any path is set to 250ms.
- SVC layers (including network headers' overhead and audio streams) are set to:
  - Audio-only, 32kbps.
  - Layer 1: Scalable Constrained Baseline, Level 1, 90kbps.
  - Layer 2: Scalable Baseline, Level 1.1, 250kbps.
  - Layer 3: Scalable Constrained High, Level 1.2, 0.5Mbps.
  - Layer 4: Scalable High, Level 1.3, 1Mbps.

  The audio-only profile enables receivers with very low access link bandwidth to participate to the call.

[6] `https://networkx.github.io/`



(a) (b)

Figure 2: Average bandwidth usage according to the network size and the number of participants on topology type MP (with 6 participants per call for (a) and a network size of 2k node for (b)).

To obtain our results, we first start by creating 20 random topologies of each type (ER and MP) and of each network size (500, 1000, 2000 and 4000). Then, for each one of these topologies, we connect randomly the clients with different access capacities. This operation is repeated 20 times to obtain 20 different instances. Thus, each point on the following plots is the average of the values obtained from 400 simulations.

7.1.2. Simulation results

Figures 2a, 2b show consecutively the impact of the network size and the number of participants on the average bandwidth usage obtained for MP topologies.

Bandwidth saving:

Figure 2a shows the influence of the network size (from 500 to 4000 nodes) on the bandwidth usage of a call with 6 participants. In Figure 2a every solution exhibits a flat nearly linear relationship between the network size and the bandwidth consumption. Our MST and STP solutions are better in saving bandwidth than the other solutions. As expected, the MST mode is the least bandwidth consuming, closely followed by the SPT mode, due to MST nature that finds the shortest path to the closest node existing in the tree, unlike SPT which creates the shortest path up to the tree root. ALM mode indicates higher bandwidth consumption since it creates the tree by connecting the new participants to the closest existent participant in the tree. The MCU mode, on the other hand, consumes the highest bandwidth usage since the stream needs to attend the MCU before getting to the receivers. Similarly, Unicast mode has high bandwidth consumption due to the redundancy of the streams on the shared links. With 4k network size, MST mode roughly consumes 56.7% of the bandwidth used by the MCU mode and 60% of the one used by unicast mode. The ratio drops to 26% for ALM. We notice the network size affects the average bandwidth usage less than the number of participants does.

Figure 2b shows the influence of the number of participants $(3, 6, 9, 12)$ on the bandwidth usage of a call with a network size of 2000 nodes. Obviously, the bandwidth consumption increases according to the number of participants. With 3 participants, the bandwidth usage of all modes is very close to each other due to the small size of the created trees. As expected and noticed in Figure 2a, MST and SPT modes have the lowest results. On another hand, with unicast and MCU, the bandwidth usage is more affected by the participants' number.

Figures 3a, 3b show consecutively the impact of the network size and the number of participants on the average bandwidth usage obtained for ER topologies. The results are similar to those obtained for MP topologies. MST and SPT are still the most bandwidth saving. But we can notice in figure 3a that the network size in ER has more effect on the average bandwidth usage than the one in MP. This is due to the sparse nature of MP topologies.

Processing time:

Figures 4a, 4b show consecutively the impact of the network size and the number of participants on the processing time obtained for MP topologies.

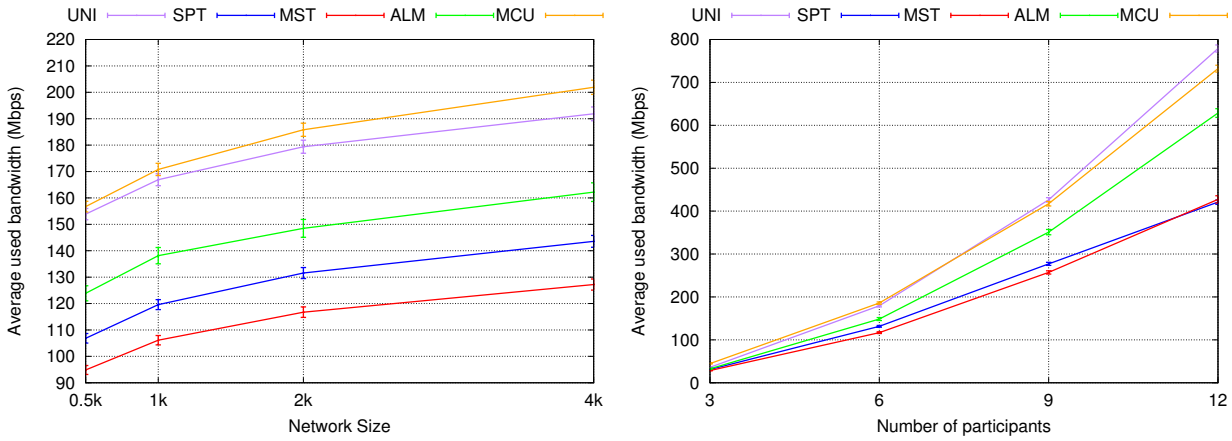
On Figure 4a, we observe the average processing time of the different algorithms per call of 6 participants.

MST follows ALM, UNI and STP with very close processing time and have the similar results as they all use shortest paths to a specific point in the tree.

The impact of the number of participants on the processing time is showed in 4b. The processing time of MST grows faster than the others'. This is due to its complexity as explained in Section 5.1. However, it remains very applicable since it requires only 0.4s to establish a video call with 12 participants.

Figure 5 shows the same results on ER topologies.

Figures 6a, 6b show consecutively the impact of the network size and number of participants, on the maximum latency obtained for MP topologies.



(a) (b)

Figure 3: Average bandwidth usage depending on the network size or the number of participants on topology type ER (with 6 participants per call for (a) and a network size of 2k node for (b)).

Figure 6a shows the influence of the network size on the maximum latency of a call. The maximum latency of a call is defined as the maximum latency measured over all paths between all participant pairs. It takes into account access and core link delays. We observe that the network size does not have a big influence on the latency since MP topologies have a small diameter. As expected, ALM shows the highest latency: it reaches 255ms ($\simeq$19.5$\times$core link delays of 10ms + 2$\times$access link delay of 30ms) with 4k network node. MCU has also a high latency (215ms with 4k network node) due to the fact that all the streams need to reach the MCU before reaching the destination. MST mode shows a maximum latency of 208ms since MST, unlike SPT, is designed to save the bandwidth without taking into consideration the latency. The latency is exactly the same for the SPT and unicast modes, as expected, and decreases with the increase of the network size. This is also expected as MP topologies are power-law type graphs where average distance and diameter do not (much) increase when the network size increases.

Maximum latency:

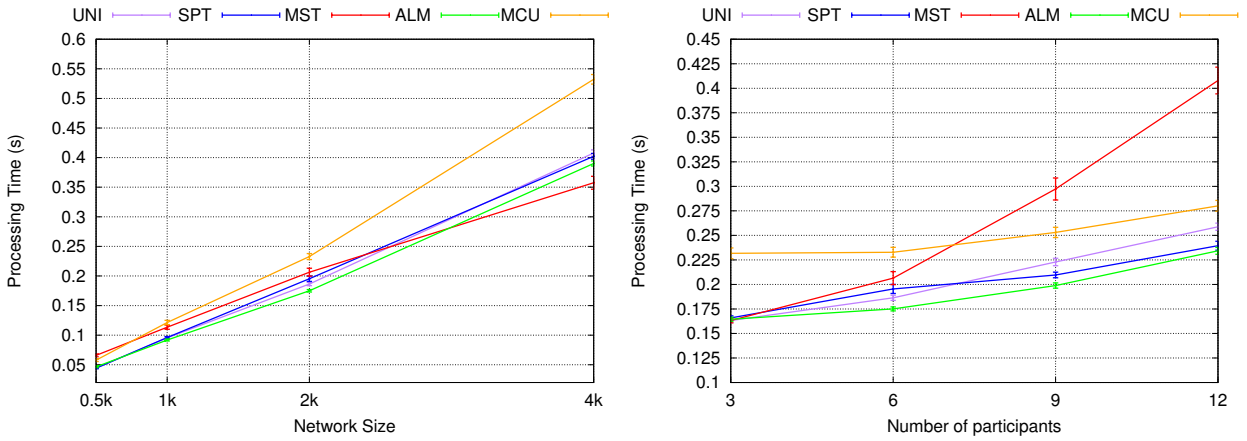
Figure 6a shows the influence of the network size on the maximum latency of a call. The maximum latency of a call is defined as the maximum latency measured over all paths between all participant pairs. It takes into account access and core link delays. We observe that the network size does not have a big influence on the latency since MP topologies have a small diameter. As expected, ALM shows the highest latency: it reaches 255ms ($\simeq$19.5$\times$core link delays of 10ms + 2$\times$access link delay of 30ms) with 4k network node. MCU has also a high latency (215ms with 4k network node) due to the fact that all the streams need to reach the MCU before reaching the destination. MST mode shows a maximum latency of 208ms since MST, unlike SPT, is designed to save the bandwidth without taking into consideration the latency. The latency is exactly the same for the SPT and unicast modes, as expected, and decreases with the increase of the network size. This is also expected as MP topologies are power-law type graphs where average distance and diameter do not (much) increase when the network size increases.

Figure 6b shows the influence of the number of participants on the maximum latency of a call. The number of participants affects mostly ALM mode since it based on finding the closest participant to create the tree. MST mode exhibits a higher latency than MCU with more than 9 participants. However, MST latency values remain acceptable (<250ms) on MP. The latency in MCU, SPT and unicast is barely affected by the number of participants as they create the shortest paths to a specific point randomly selected. The SPT and unicast modes yield the same and lowest results regardless of the number of participants, which is expected as they both use shortest paths to the source of the tree, unlike MCU mode which uses the shortest path to the MCU node.

Figures 7a and 7b show similar behavior on ER to the one on MP topology. However, the latency on ER is higher than the one on MP and it exceeds 250ms with MST, ALM mode with a number of participants of 12 participants.

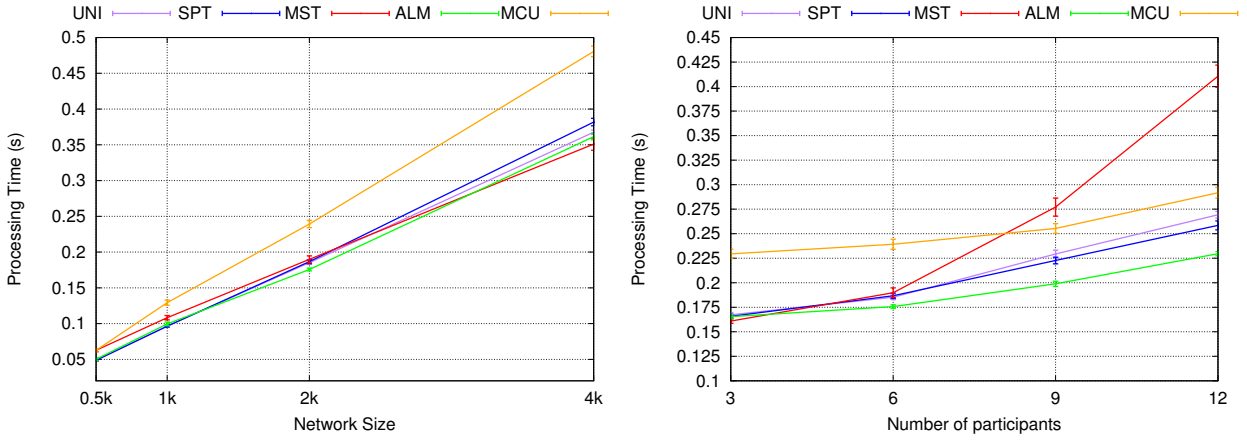
Network call capacity:

Figure 8 shows the impact of the network size on the network call capacity. The network call capacity is determined as follows: the call arrival model is a Poisson distribution with a mean arrival time depending on the expected number of simultaneous calls in a network with specific core link bandwidth capacity. We assume that the call duration model follows an exponential distribution with an average of 23 minutes and the call arrival model is a Poisson distribution. At some point, the $BuildShortestPath()$ function defined in Algorithm 1 will return false, indicating that the call could not be setup because one or more links involved in the call were saturated (i.e., filled at the maximum of their bandwidth capacity). In this case, the call is rejected. After 100 rejected calls, the network call capac-



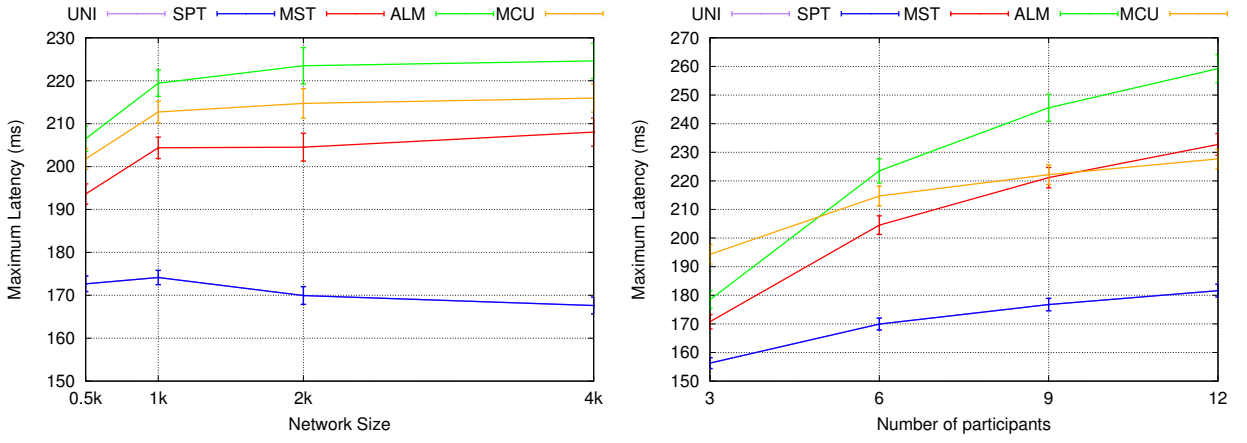
(a) (b)

Figure 4: Average processing time according to the network size and the number of participants on MP (with 6 participants per call for (a) and a network size of 2k node for (b)).



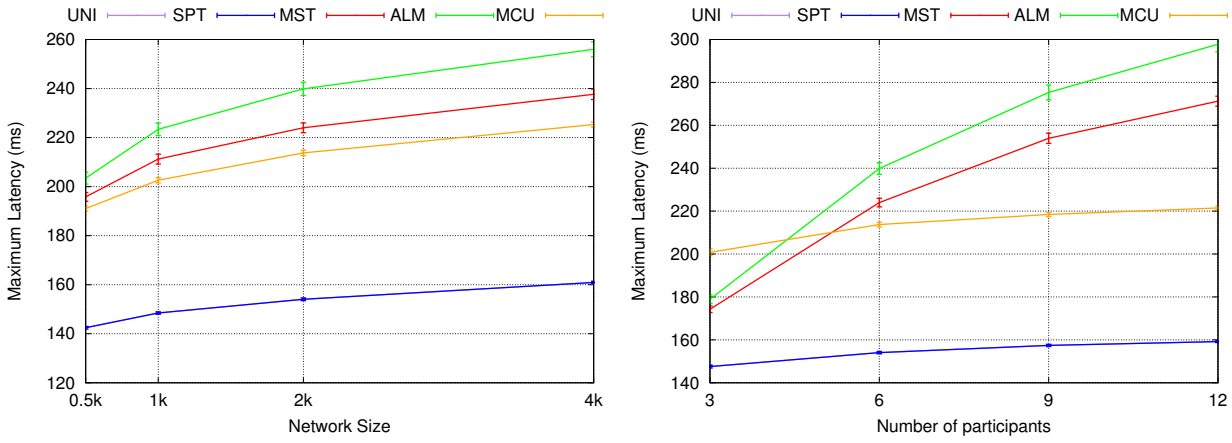
(a) (b)

Figure 5: Average processing time according to the network size and the number of participants on ER (with 6 participants per call for (a) and a network size of 2k node for (b)).



(a) (b)

Figure 6: Maximum latency according to the network size and the number of participants on MP (with 6 participants per call for (a) and a network size of 2k node for (b)).



(a) (b)

Figure 7: Maximum latency according to the network size and the number of participants on ER (with 6 participants per call for (a) and a network size of 2k node for (b)).

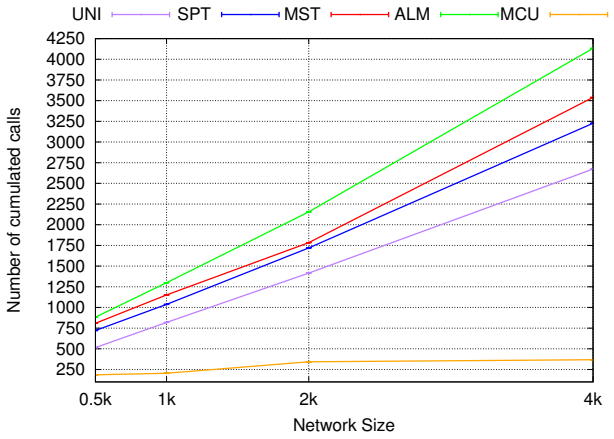


Figure 8: Number of calls supported by the network usage depending on the network size on MP.

ity is considered reached. The number of supported calls depends on the sequence of construction of all the calls which are generated according to our model. Therefore, for each experiment, the network capacity will slightly vary. We have performed 400 experiments on MP networks with core link bandwidth set to 1Gbps. In Figure 8, for all the algorithms except MCU, the call capacity quickly increases with the network size. ALM mode exhibits the best results with the highest number of supported calls (4200 with 4k node), as the trees in ALM are not optimized, and the paths are more distributed among the topology which lead to a slower saturation of the network, thus more supported calls. MST and SPT modes show as well good results followed by unicast. On the other hand, MCU does not seem to be affected by the network capacity. That is due to the fact that all the calls need to pass through the MCU which may saturate some links and cause a quick call rejection.

As a conclusion, our solutions, MST and SPT modes, save a lot of bandwidth comparing to the other modes. MST shows better performance than SPT for bandwidth usage on both ER and MP topologies. In term of latency, both SPT and MST modes have good performance and do not exceed the maximum allowed latency on MP topology. However, with ER topology, MST exceeds the latency limits when the number of participants increases. With ALM mode the network can support the highest number of simultaneous calls, followed by MST and STP mode. However, since ALM has a very high latency, our solutions SPT and MST seem more suitable than the other algorithms.

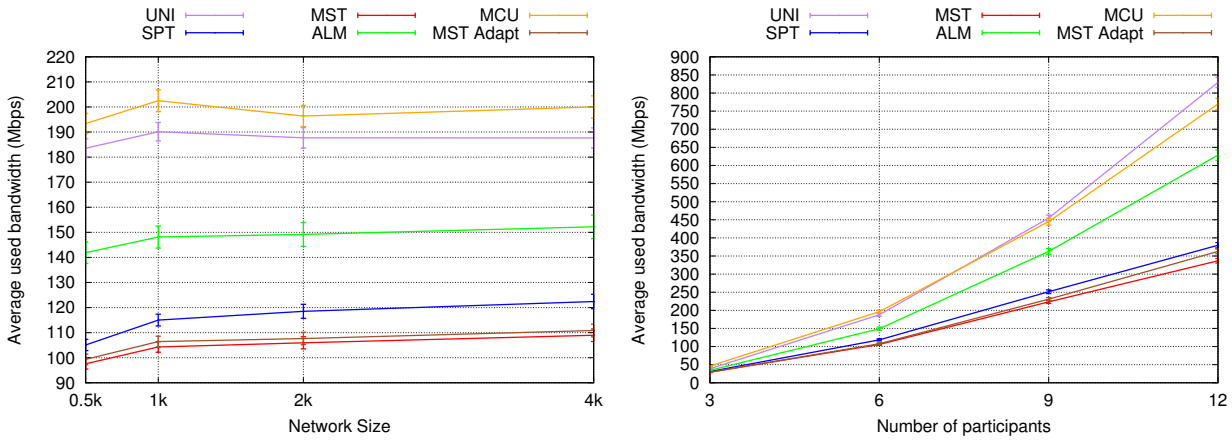
7.2. Adaptation to bandwidth variations during a call

In this section, we aim to evaluate our adaptation algorithms of section 6 under a dynamic system where access link bandwidth varies during the call.

7.2.1. Simulation parameters

For the dynamic implementation, the topology type and size are built in the same way as the topologies presented in the parameter of the section 7.1.1. As well as the access/core links capacities and delays requirements. The only difference occurs within the access links bandwidth; they may vary over the duration of the call. In our simulation, the bandwidth variation is reproduced by choosing, a each time, a random value from the interval of [4Mbps,14Mbps] for downlink access capacity and form the interval of [500kbps,1.5Mbps] for uplink access capacity. To adapt to those variations, we either recompute the trees or adapt to the changes using the algorithm described in Section 6. We compare it to unicast, ALM and MCU in a dynamic environment. We use the following approaches:

- UNI or Uni-recomputation approach: it represents the unicast approach in a dynamic environment which recomputes the shortest paths between each pair of participants at each access link bandwidth variation.



(a) (b)

Figure 9: Average bandwidth usage according to the network size and the number of participants on MP (with 6 participants per call for (a) and a network size of 2k node for (b)).

- MST or MST-recomputation approach: it represents our MST mode in a dynamic environment which recomputes all the trees at each access link bandwidth variation.
- SPT or SPT-recomputation approach: it represents our SPT mode in a dynamic environment which recomputes all the trees at each access link bandwidth variation.
- ALM approach: we have implemented ALMI as an ALM approach for dynamic environment. ALMI is an Audio/video conferencing centralized protocol, where the forwarding responsibility is given to the end hosts. In a dynamic environment, ALMI recomputes all the trees to adapt to the bandwidth change.
- MCU approach: In a dynamic environment, the path *end host - MCU* or *MCU - end host* is recomputed, depending on the type of access change.
- MST-Adapt approach: it represents our MST mode in a dynamic environment which uses the algorithms of adaptation defined in Section 6. It adapts the bitrates on the trees at each access links bandwidth variation. We have implemented our solution only for MST since the results of MST in the static environment where more efficient than SPT.

7.2.2. Simulation results

Bandwidth saving:

Figures 9a/ 9b and Figures 10a/ 10b show the impact of the network size/participants number per call on the average bandwidth usage for MP and ER topologies, respectively.

The results are almost the same as in a static environment. It is important to notice that MST-adapt is almost as efficient as MST-recompute for bandwidth saving. Recomputing the bitrates and moving the adaptation rules is enough to perform very good bandwidth saving, without the necessity of recomputing the multicast trees at each variation. It appears that the bandwidth variations at an access link are very small compared to the bandwidth scale in the core links. They are too small to induce a change in the multicast trees.

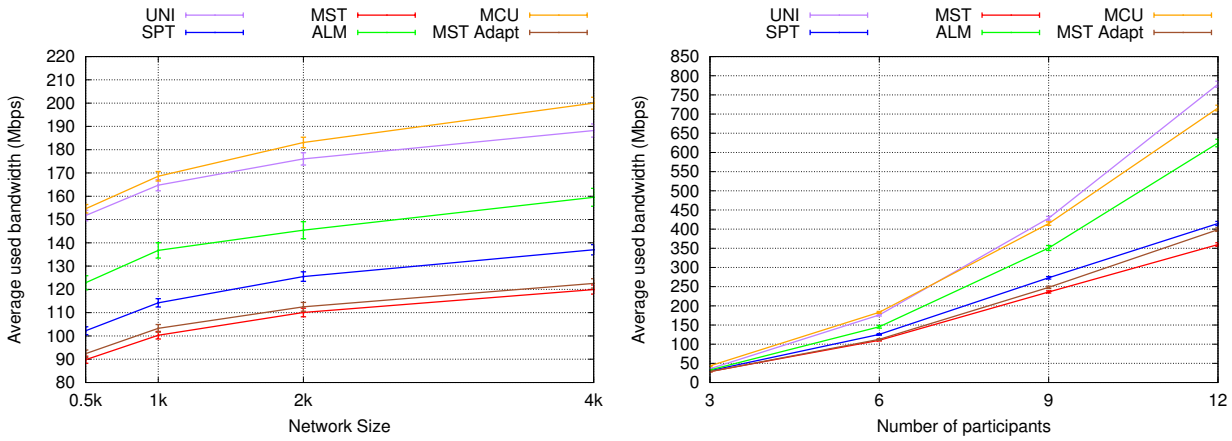
Processing time:

Figures 11a/ 11b and Figures 12a/ 12b show the impact of the network size/participants number per call on the processing time for MP and ER topologies, respectively.

As expected, the results show that MST-adapt is much faster than the other algorithms (in an order of several hundred times compared to the other algorithms – except MCU). This is due to the fact that it does not recompute the trees but just moves upward and downward the adaptation rules. This very short processing time offers high reactivity to network changes, less computation at the SVN controller, and smaller probability of call disruption.

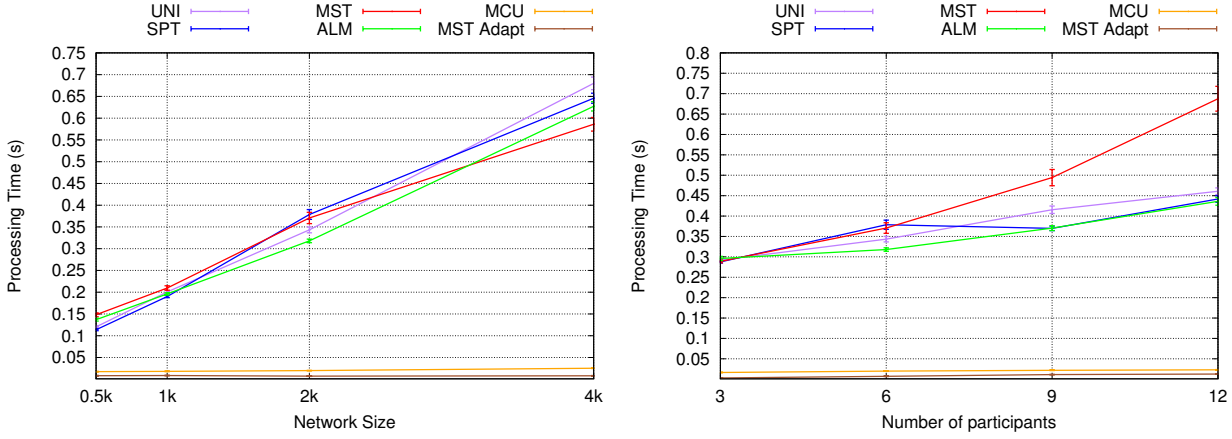
MCU is also very fast (although slower than MST-adapt) due to the fact that it recomputes only a single path in the tree at each variation.

However, as seen in the previous sections, MCU suffers from a very low network call capacity and a poor bandwidth saving. While MST-adapt is faster, more efficient and achieves almost the same bandwidth saving as MST without recomputing the multicast trees.



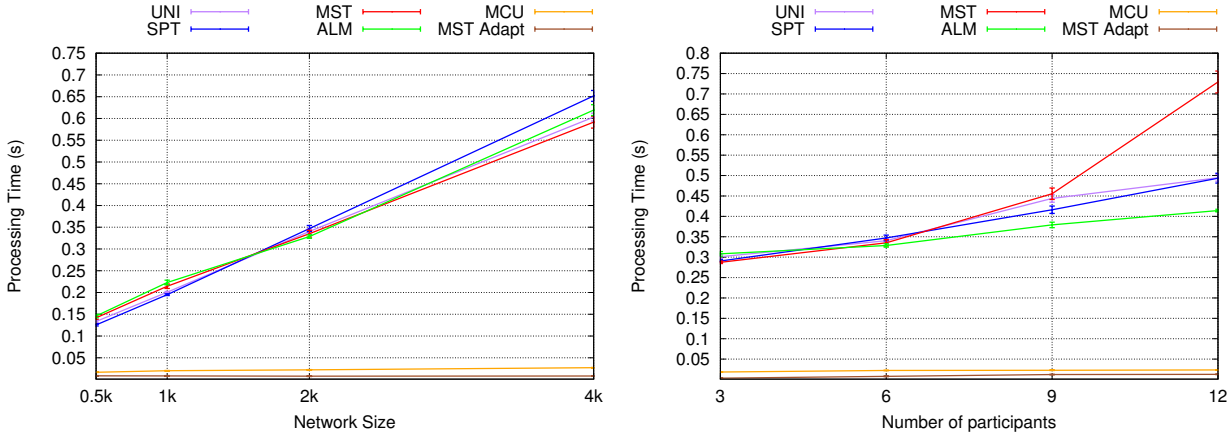
(a) (b)

Figure 10: Average bandwidth usage according to the network size or the number of participants on ER (with 6 participants per call for (a) and a network size of 2k node for (b)).



(a) (b)

Figure 11: Average processing time according to the network size and the number of participants on MP (with 6 participants per call for (a) and a network size of 2k node for (b)).



(a) (b)

Figure 12: Average processing time according to the network size and the number of participants on ER (with 6 participants per call for (a) and a network size of 2k node for (b)).

8. Conclusion

Video conferences create a heavy load on the access network due to the large size of video data. Improving video conferencing systems can lead to large bandwidth savings for network operators and can increase their ability to serve a larger number of users as well as improving their QoE. The emergence of SDN is enabling new solutions for better video management.

In order to minimize the bandwidth consumption in the core network while providing the best possible video quality to the users, we have proposed in this paper several algorithms for creating a video conferencing system leveraging SDN-enabled networks. Our solution transmits video streams using multicast trees while applying SVC layer adjustments inside the network to adapt these streams to the users' access network characteristics. Our results show that our solution saves more bandwidth compared to previous existing systems and increases the number of simultaneous calls supported by the network.

We also have designed our algorithms to dynamically adapt the video streams to the bandwidth variations of the users' access channels. For this matter, we have proposed adaptation algorithms, based on tree traversal ideas, that do not require recomputing the multicast trees (i.e., connections) of a call. These algorithms show fast adaptation due to their low complexity which allows high reactivity to network changes and low resource consumption.

Our algorithms and our network simulator are currently both implemented in Python. They are open source and available for use[7]. As a future work, we plan to port our algorithms in a Python SDN controller such as Ryu, in order to perform experiments on a virtualized network build by Mininet. This will enable us to confirm the current results at a medium scale but using real network software such as OpenVswitch. A next step would then be to deploy it on a real network.

References

[1] H. Schwarz, D. Marpe, T. Wiegand, Overview of the scalable video coding extension of the H. 264/AVC standard, IEEE Transactions on Circuits and Systems for Video Technology (TCSVT) 17 (9) (2007) 1103–1120.

[2] C. Al Hasrouty, C. Olariu, V. Autefage, D. Magoni, J. Murphy, SVC Videoconferencing Call Adaptation and Bandwidth Usage in SDN Networks, in: IEEE Global Communications Conference (GLOBECOM) (2017), pp. 1–7.

[3] C. Al Hasrouty, M. L. Lamali, D. Magoni, J. Murphy, SDN-enabled Adaptation of Videoconference Streams to Network Dynamics, in: IEEE Global Communications Conference (ICC) (2017), pp. 1–7.

[4] M. Willebeek-LeMair, D. D. Kandlur, Z.-Y. Shae, On multipoint control units for videoconferencing, in: 19th Conference on Local Computer Networks (LCN) (1994), pp. 356–364.

[5] N. W. Group, et al., Rfc 1889-rtp: a transport protocol for real time applications (1996).

[6] S. E. Deering, D. R. Cheriton, Multicast routing in datagram internetworks and extended lans, ACM Transactions on Computer Systems (TOCS) 8 (2) (1990) 85–110.

[7] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, L. Wei, Protocol independent multicast-sparse mode (pim-sm): Protocol specification, RFC 2362 (1998).

[8] S. Banerjee, B. Bhattacharjee, C. Kommareddy, Scalable application layer multicast, in: Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (2002), 2002, pp. 205–217.

[9] M. Hosseini, D. T. Ahmed, S. Shirmohammadi, N. D. Georganas, A survey of application-layer multicast protocols, IEEE Communications Surveys & Tutorials 9 (3) (2007) 58–74.

[10] D. Pendarakis, S. Shi, D. Verma, M. Waldvogel, ALMI: An Application Level Multicast Infrastructure, in: 3rd Conference on USENIX Symposium on Internet Technologies and Systems (2001), pp. 49–60.

[11] D. A. Tran, K. A. Hua, T. T. Do, A peer-to-peer architecture for media streaming, IEEE Journal on Selected Areas in Communications (JSAC) 22 (1) (2004) 121–133.

[12] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, S. Khuller, Construction of an efficient overlay multicast infrastructure for real-time applications, in: 22nd Annual Joint Conference of the IEEE Computer and Communications (ICCC) (2003), Vol. 2, pp. 1521–1531.

[13] F. Fan, B. Hu, K. L. Yeung, Distributed and dynamic multicast scheduling in fat-tree data center networks, in: IEEE International Conference on Communications (ICC) (2016), pp. 1–6.

[14] T. Humernbrum, B. Hagedorn, S. Gorlatch, Towards efficient multicast communication in software-defined networks, in: Distributed Computing Systems Workshops (ICDCSW), 2016 IEEE 36th International Conference on, IEEE, 2016, pp. 106–113.

[15] G. Liebl, T. Schierl, T. Wiegand, T. Stockhammer, Advanced wireless multiuser video streaming using the scalable video coding extensions of H.264/MPEG4-AVC, in: IEEE International Conference on Multimedia and Expo (ICME) (2006), pp. 625–628.

[16] J.-C. Chiang, H.-F. Lo, W.-T. Lee, Scalable video coding of H. 264/AVC video streaming with qos-based active dropping in 802.16 e networks, in: 22nd International Conference on Advanced Information Networking and Applications-Workshops (AINA) (2008), pp. 1450–1455.

[17] Y.-S. Yu, C.-H. Ke, Genetic algorithm-based routing method for enhanced video delivery over software defined networks, International Journal of Communication Systems (IJCS) 31 (1) (2018) e3391.

[18] S. Civanlar, M. Parlakisik, A. M. Tekalp, B. Gorkemli, B. Kaytaz, E. Onem, A qos-enabled openflow environment for scalable video streaming, in: IEEE Global Communications Conference (GLOBECOM) Workshops (2010), pp. 351–356.

[19] H. Egilmez, T. Dane, T. Bagci, A. M. Tekalp, OpenQoS: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks, in: Signal & Information processing association annual summit and conference (APSIPA ASC) (2012), pp. 1–8.

[20] H. Egilmez, S. Civanlar, M. Tekalp, An optimization framework for qos-enabled adaptive video streaming over openflow networks, IEEE Transactions on Multimedia 15 (3) (2013) 710–715.

[21] G. Cofano, L. De Cicco, T. Zinner, A. Nguyen-Ngoc, P. Tran-Gia, S. Mascolo, Design and experimental evaluation of network-assisted strategies for http adaptive streaming, in: Proceedings of the 7th International Conference on Multimedia Systems (2016), p. 3.

[22] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S.-J. Lee, P. Yalagandula, Automated and scalable qos control for network convergence., Internet Network Management Workshop/ Workshop on Research on Enterprise Networking (INM/WREN) 10 (1) (2010) 1–1.

[7] http://www.labri.fr/perso/magoni/cemeqacs

[23] K. Jan Willem, C. Sergio, C. Pablo, Delivering stable high-quality video: an sdn architecture with dash assisting network elements, in: ACM Multimedia Systems Conference (MMSys) (2016).
[24] N. L. Van Adrichem, C. Doerr, F. A. Kuipers, Opennetmon: Network monitoring in openflow software-defined networks, in: Network Operations and Management Symposium (NOMS) (2014), pp. 1–8.
[25] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, P. Tran-Gia, Modeling and performance evaluation of an openflow architecture, in: Proceedings of the 23rd international teletraffic congress (ITC) (2011), pp. 1–7.
[26] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, N. Race, Towards network-wide qoe fairness using openflow-assisted adaptive video streaming, in: Proceedings of the 2013 ACM SIGCOMM workshop on Future human-centric multimedia networking (2013), pp. 15–20.
[27] M. Zhao, B. Jia, M. Wu, H. Yu, Y. Xu, Software defined network-enabled multicast for multi-party video conferencing systems, in: IEEE International Conference on Communications (ICC) (2014), pp. 1729–1735.
[28] E.-Z. Yang, L.-K. Zhang, Z. Yao, J. Yang, A video conferencing system based on SDN-enabled SVC multicast, Frontiers of Information Technology & Electronic Engineering 17 (7) (2016) 672–681.
[29] J. Yang, E. Yang, Y. Ran, S. Chen, Sdmˆ 2 cast an openflow-based, software-defined scalable multimedia multicast streaming framework, IEEE Internet Computing 19 (4) (2015) 36–44.
[30] N. Xue, X. Chen, L. Gong, S. Li, D. Hu, Z. Zhu, Demonstration of openflow-controlled network orchestration for adaptive svc video manycast, IEEE Transactions on Multimedia 17 (9) (2015) 1617–1629.
[31] S. Laga, T. Van Cleemput, F. Van Raemdonck, F. Vanhoutte, N. Bouten, M. Claeys, F. De Turck, Optimizing scalable video delivery through openflow layer-based routing, in: Network Operations and Management Symposium (NOMS) (2014), pp. 1–4.
[32] H. Egilmez, M. Tekalp, Distributed qos architectures for multimedia streaming over software defined networks, IEEE Transactions on Multimedia 16 (6) (2014) 1597–1609.
[33] J. Yang, K. Zhu, Y. Ran, W. Cai, E. Yang, Joint admission control and routing via approximate dynamic programming for streaming video over software-defined networking, IEEE Transactions on Multimedia 19 (3) (2017) 619–631.
[34] J. Yang, E. Yang, Y. Ran, Y. Bi, J. Wang, Controllable multicast for adaptive scalable video streaming in software-defined networks, IEEE Transactions on Multimedia 20 (5) (2018) 1260–1274.
[35] D. Wu, Y. T. Hou, W. Zhu, Y.-Q. Zhang, J. M. Peha, Streaming video over the internet: approaches and directions, IEEE Transactions on circuits and systems for video technology (TCSVT) 11 (3) (2001) 282–300.
[36] L. De Cicco, S. Mascolo, V. Palmisano, Skype video responsiveness to bandwidth variations, in: 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV) (2008), ACM, pp. 81–86.
[37] M. Vutukuru, H. Balakrishnan, K. Jamieson, Cross-layer wireless bit rate adaptation, ACM SIGCOMM Computer Communication Review 39 (4) (2009) 3–14.
[38] Z. Ding, J. Wu, W. Yu, Y. Han, X. Chen, Pseudo analog video transmission based on LTE physical layer, in: IEEE International Conference on Communications in China (CIC) (2016), pp. 1–6.
[39] D. He, C. Lan, C. Luo, E. Chen, F. Wu, W. Zeng, Progressive pseudo-analog transmission for mobile video streaming, IEEE Transactions on Multimedia 19 (8) (2017) 1894–1907.
[40] P. Winter, Steiner problem in networks: a survey, Networks 17 (2) (1987) 129–167.
[41] B. Bollobás, O. Riordan, The diameter of a scale-free random graph, Combinatorica 24 (1) (2004) 5–34.
[42] P. Erdös, A. Rényi, On random graphs i., Publicationes Mathematicae 6 (1959) 290–297.
[43] D. Magoni, J.-J. Pansiot, Internet topology modeler based on map sampling, in: 7th IEEE Symposium on Computers and Communications (ISCC) (2002), pp. 1021–1027.
[44] M. Laner, P. Svoboda, M. Rupp, Latency analysis of 3g network components, in: 18th European Wireless Conference (EW) (2012), pp. 1–8.
[45] M. Hoerdt, D. Magoni, Cartographie distribuée du coeur de l'internet, Ann. des Télécom. 60 (5-6) (2005) 558–587.