Multicast Tree Construction with QoS Guaranties

O. Moussaoui[1], A. Ksentini[1], M. Naïmi[1], A. Gueroui[2]

[1]LICP EA 2175, Université de Cergy-Pontoise- 2 Av Adolphe Chauvin 95302
Cergy-Pontoise – France
{omar.moussaoui, adlen.ksentini, mohamed.naimi}@dept-info.u-cergy.fr

[2]PRiSM CNRS, Université de Versailles- 45, Av des Etats-Unis 78035
Versailles – France
E-mail : mogue@prism.uvsq.fr

Abstract. Multimedia applications, such as videoconferences, require an efficient management of the Quality of Service (QoS) and consist of a great number of participants which requires the use of multicast routing protocol. Unlike unicast protocol, multicast protocol handles a great number of users while minimizing both network overhead and bandwidth consumption. However, combining multicast routing and QoS guarantee is a hard challenging task, known as the delay and delay variation multicast problem. This problem is considered as an *NP-complete* problem, and is resolved only by heuristic solutions. In this paper, we propose a scalable multicast algorithm that tackle the delay and delay variation by exploiting the Hierarchic Tree construction concepts. In fact, the proposal algorithm guarantees QoS by: *(i)* reducing the network charge; *(ii)* decreasing the multicast delay variation. We compare the performance of our algorithm against the DDVCA (Delay and Delay Variation Constraint Algorithm) scheme and demonstrate lower multicast delay variation and efficient bandwidth utilization while maintaining lower time complexity.

1 Introduction

The demand for multimedia that combines audio, video and data streams over a network is quickly increasing. Among the most popular real-time interactive applications, videoconferences and games require a considerable amount of bandwidth and a great number of participants. In this context multicast is regarded as a promising solution for group multimedia applications. In fact, multicast is a bandwidth-conserving technology that reduces traffic by simultaneously delivering a single stream of information to thousands of corporate recipients or groups. Multicast delivers source traffic to multiple receivers without adding any additional burden on the source or the receivers while using the least network bandwidth of any competing technology. Multicast packets are replicated in the network through routers enabled with multicast protocol and other supporting multicast protocols resulting in the most efficient delivery of data to multiple receivers possible. All alternatives require the source to send more than one copy of the data. Some even require the source to send an individual copy to each receiver. If there are thousands of receivers, even low-bandwidth applications benefit from using multicast. High-bandwidth applications, such as H.264 video, may require a large portion of the available network bandwidth for a single stream [1]. In these applications, the only way

to send to more than one receiver simultaneously is by using multicast. Several protocols were proposed in the literature [2], [3], [4], [5] aiming to propose a multicast protocol. However, the majority of them are not scalable, which means not adaptable to the networks of great dimension like Internet. By limiting the deployment of these protocols, the Internet is becoming the unavoidable network. Furthermore, these protocols work as best effort protocols [6], so they cannot handle sensitive traffics such as video conference and real time game. Indeed, these applications require that all the destination nodes must receive the same data simultaneously; otherwise the communication may lose the feeling of an interactive face-to-face discussion. Thus it is important to sustain good QoS support while proposing a scalable multicast protocol. This constraint is related to the multicast Delay and delay-Bounded Multicast Tree (DVBMT) problem [7]. Although this problem is considered as *NP-complete* hard problem, there are some heuristics that are proposed as a possible solution [7], [8].

In this paper we tackle the DVBMT problem by proposing a novel scalable multicast algorithm, which produces multicast tree while maintaining a good sustained QoS. The main idea is to combine a hierarchic tree construction with efficient multicast grouping concepts. Firstly, like [9], we decompose the multicast group into local groups based on delay constraint and user's station capacity. This allows us to have several groups with a reduced intra group delays. Afterwards, we select a server that minimizes the delay variation with the others selected nodes from each group. From the server's set obtained, we choose a core nodes or rendezvous points. Finally, we use both the hierarchical trees and the core nodes to connect these multicast group members. Thus, we solve the DVBMT problem by decomposing the problem into two parts: *(i)* end to end delays which is solved by constructing local group with minimum delays; *(ii)* multicast delay variation solved by joining the hierarchic tree construction with the core nodes concepts.

The rest of the paper is organized as follows: In section 2, we give an overview of QoS and multicast protocols. Section 3 presents details of the proposed algorithm. Then, in section 4, we evaluate the proposed scheme by simulation model. Section 5 concludes this paper.

2 Multicast and QoS Overview

Algorithms for the tree construction in multicast protocols can be categorized as follows: Source-Based Algorithms (SBA) and Core-Based Algorithms (CBA) [10].

SBA constructs a specific tree where the tree's root is the source node and the leaves are the multicast group's components. SBA is currently used as the tree construction algorithm for Distance Vector Multicast Routing Protocol (DVMRP) [2], Protocol Independent Multicast Dense Mode (PIM-DM) [3], and Multicast Open Shortest Path First (MOSPF) [4].

CBA is used in the context of many-to-many multicasts. Actually, the core-based algorithm selects a core node as multicast tree's root. Afterwards, a tree rooted at the core node is constructed to span all members in the multicast group. Therefore it is very important to select the best core node as much as possible. Thus messages generated at the source are sent to the core node, and they are distributed to destinations through this core node. Multicast protocols using CBA as a tree construction algorithm include Protocol Independent Multicast Sparse Mode (PIM-SM) [5] and the Core-Based Tree (CBT) protocol [11]. The core-based algorithms are highly suitable for sparse groups and scalable for large networks. They provide excellent bandwidth conservation for receivers.

In addition to the need of scalability, group based multimedia applications also demand stringent QoS requirements such as bounded end-to-end delay, multicast delay variation and the efficient use of the bandwidth. The multicast end-to-end delay stands for an upper bound of all end-to-end delays associated with the paths from the source node to each of the destination nodes. The purpose of setting this parameter is to limit the time for message transmissions in the network. If the end-to-end delay exceeds the upper bound, the message will be counted useless. The multicast delay variation is the difference of the maximum end-to-end delay and the minimum end-to-end delay among the paths from the source node to all the destination nodes has to be kept within. Enabling this parameter allows all the destination nodes to receive the same data simultaneously as much as possible. The issue first defined and discussed in [7] is to minimize multicast delay variation under multicast end-to-end delay constraint. In fact, the authors tackle the DVBMT problem by proposing a heuristic solution called Delay Variation Multicast Algorithm (DVMA). DVMA constructs at first the tree by considering only the end-to-end delay constraints. Afterwards, the tree is enhanced by considering the multicast delay variation constraint. At the end, DVMA's algorithm returns a feasible tree, which minimizes the end-to-end delays and optimizes the multicast delay variation. Nevertheless, the main weakness of DVMA is time complexity. Actually, DVMA exhibits a high time complexity about $O(plmn^4)$, where in the worst case, the maximum value that p and l can take is equal to the maximum number of paths of the tree, m is the size of the multicast group M, and n is the number of nodes in the network. Accordingly, this time complexity does not fit in modern high-speed computer network environment. Delay and Delay Variation Constraint Algorithm (DDVCA) presented in [8] aims to solve the DVBMT problem by proposing another heuristic solution with lower time complexity than DVMA. DDVCA's algorithm is based on the Core Based Tree (CBT). In fact the authors propose to build the multicast tree around one core node, which is selected as the node with the minimum delay variation with all the others nodes present in the multicast group. Thus it has been shown that DDVCA outperforms DVMA in terms of the multicast delay variation of the constructed tree. Furthermore, DDVCA shows lower time complexity than DVMA, which is equal to $O(mn^2)$. Nonetheless, if we consider the network utilization, we see that DDVCA exhibits high network charge around the core node. In fact, all the multicast session's packets transit through the core node, which leads to network congestion in the neighboring of the core node. Furthermore, when packets arrive to the core node from the sender, this last one resends these packets to the leaves using a unicast routing protocol. Therefore, DDVCA looses the benefits of using a multicast routing protocol.

To overcome these limitations, we propose an algorithm which allows efficient communication between the multicast group's members by supporting QoS constraints. The proposed algorithm solves the DVBMT problem by constructing a hierarchical tree based on delay and multicast delay variation. Firstly, we decompose the multicast members into a disjoint local groups based on their localization and their response to application QoS requirement. Thus we obtain local groups with minimum intra-group delay. Afterwards, from each group a server is selected, where the server is the node that minimizes the multicast delay variation with the others group's node. At this point we have the first level of the hierarchical tree. Secondly, we select a core node from the server sets, and this core node is the server which minimizes the multicast delay variation with the others selected server. Here we are able to construct the second level of the hierarchical tree. In fact, based on this core node we build the autonomous domain (AD). The first AD contains a set of groups, where the delay from the core node to each group's server is less than a predefined threshold. From the server sets that not belong to the first AD, we

elect another core node. This second core node is the server that minimizes the delay with the first core node. Through this second core node we create another AD. Thus we redo the procedure of AD creation until we incorporate all the local groups into different AD. At the end, we construct a hierarchical tree with core-based algorithms. However, unlike DDVCA where the tree construction is based only on one core node, we extend this construction of the tree on several core nodes. This allows us to share out the network charge around different core node, leading to minimize the bandwidth consumption. Further we limit the use of the unicast mechanism only at the intra local group communications.

3 Description of the Proposed Algorithm

To illustrate the proposed algorithm, we will use directed graph $G = (V, E)$ to denote a network, where V is a set of nodes or routers and E is a set of directed links, respectively. Each link $(i, j) \in E$ is associated with delay d_{ij}. The delay of a link is the sum of the perceived queuing delay, transmission delay, and propagation delay over that link. Here we note a path as sequence of nodes $u, i, j, ..., k, v$, where $(u, i), (i, j), ..., (k, v) \in E$. Let $P(u, v) = \{(u, i), (i, j), ..., (k, v)\}$ denotes the path from node u to node v, so a simple path is a path where all its elements are distinct. At this point a multicast group $M \subseteq G$ is constituted by m processes (participants) distributed geographically on the Internet network. Moreover, these processes participate at the same multimedia application such as videoconferencing. Note that, communication between two processes m_i and m_j can take different path.

3.1 Local Groups' Construction

Considering the high number of participants in multicast group M, it appears essential to divide them into local groups according to their concentration in the various areas of the Internet. This decomposition allows to: *(i)* efficiently use the bandwidth; *(ii)* reduce the consumption of resources; *(iii)* optimize the delay; *(iiii)* ensure communications between processes.

At first we begin by constructing the neighboring sets NS_i for each process m_i according to two conditions: the round-trip delay and packets Time to Live (TTL). Actually, each m_i sends a request packet along its multicast groups by using the Internet Protocol (IP) multicast addresses. In the current IP multicast architecture, a globally unique class D is allocated to each group to identify the set of destination hosts belonging to the group. Through the responses obtained from the multicast group, m_i selects the process m_j to include into the NS_i by checking if the TTL_{ij} from m_i to m_j (decremented hop by hop) is not null and the Round-Trip Delay RTD_{ij} between m_i and m_j is less than a given threshold of delay $SupD$ (1). In other word, the path from m_i to m_j is the shortest in terms of delay. According to this selection, we divide the multicast group onto a set of NS_i with low intra-communication delays.

$$NS_i = \{m_j \in M / RTD_{ij} \prec SupD \;\&\; \&TTL_{ij} \succ 0\} \tag{1}$$

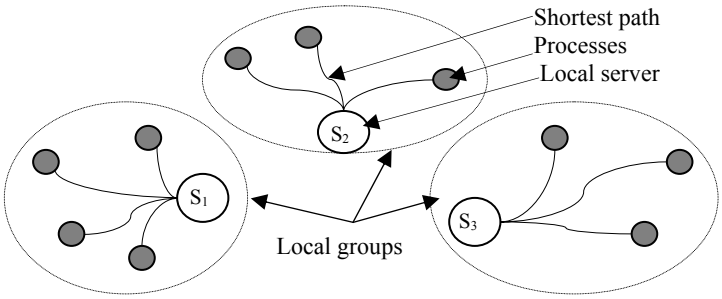

Fig. 1. Construction of local groups

At this point, each m_i has its own NS_i. However, this is insufficient if we consider the different process' capacities in terms of media unit processing time and the generated rate of media units, noted mpt_i and σ_i, respectively. In this context we must refine the NS_i in order to take the process's capacities into account. To this end, each m_i carries out evaluations on its capabilities of processing and buffering of media units, in order to choose the m_j which will constitute its new enhanced NS_i. Here we note this enhanced NS_i, by the transit group TG_i. Thus each m_i elects the m_j process that composes its transit group according to both (2) and (3). In fact these two constraints allow that m_i selects the other process in respect to its capabilities to handle data flow coming from its neighbors. Thus a process m_i must be able to: *(i)* process all media units generated in the same time by its neighbors in a time duration not exceeding the time allocated to the processing of media units; *(ii)* store all media units coming on different paths between m_i and its neighboring processes.

$$\left[\sum_{m_j \in NS_i} \sigma_j \right] * mpt_i < 1 \tag{2}$$

$$\sum_{m_j \in NS_i} B_{ij}(C_{ij}) < B_i \tag{3}$$

Where B_i represents the maximum buffer space available at the process m_i, and B_{ij} denotes the buffer size at m_i and C_{ij} is the path between m_i and m_j. Therefore, since m_i founds that neither (2) nor (3) are feasible when adding another m_j to the TG_i, the process stops and the TG_i is finalized.

Once transit groups are built, each process knows the members of its own set and the paths that connect it to them. However, some processes can belong to several transit groups at the same time. To solve this problem, a mechanism must take place to remove useless connections. Each process must broadcast its transit group to its neighbors. Once these packets are received, each receiver m_i selects the maximum of the processes existing simultaneously in these groups of transit ($LG_i = \bigcap_{j \in TG_i} TG_j$). If a process belongs to several local groups, then this process is placed into the smallest LG_i in size. This allows to equally balancing the process number in these groups.

Thus, the multicast group M is divided into local sub-groups (Figure 1). Each element of M belongs to only one local group. After that a local server and a secondary server are elected to represent each local group. In other terms, a process communicates with the

other participants of the multicast group only through the server of its local group. The local and secondary servers are the processes that minimize the multicast delays with the others process, meanwhile these servers must have the maximum processing capacity. Note that, the principal role of the secondary server is to replace the local server if this last leaves the multicast group or it crashes (break down). For completeness, we draw in Figure 2 the local group algorithm construction.

Let $G=(V, E)$ a computer network and $M=\{m_1, m_2, ..., m_m\}$ a set of participants in group multicast M.

Begin

1. for each process $m_i \in M$ do //construct a neighboring set NS_i of m_i
2. $NS_i = \{m_j \in M / RTD_{ij} \prec SupD \;\&\&\; TTL_{ij} \succ 0\}$
3. end of for each $m_i \in M$ loop
4. for each process $m_i \in M$ do //construct a transit group TG_i of m_i
5. $\sigma = 0, B = 0$ //σ and B are temporaries variables
6. for each process $m_j \in NS_i$ do
7. $\sigma = \sigma + \sigma_j, B = B + B_{ij}$
8. if $\sigma * mpt_i \prec 1$ and $B \prec B_i$ then $TG_i = TG_i \cup \{m_j\}$
9. end of for each $m_j \in NS_i$ loop
10. end of for each $m_i \in M$ loop
11. for each process $m_i \in M$ do // construction of local groups
12. $LG_i = \bigcap_{j \in TG_i} TG_j$
13. end of for each $m_i \in M$ loop
14. for each local group LG_i do // election of local servers
15. elect local server S_i and secondary server SS_i
16. $\Gamma = \Gamma \cup \{S_i\}$ and $k=k+1$
17. end of for each local group LG_i loop
18. for each local server $S_i \in \Gamma$ do
19. for each $m_j \in LG_i$ do
20. S_i joins m_j by the path which has a minimum round-trip delay.
21. end of for each $m_j \in LG_i$ Loop
22. end of for each local server $S_i \in \Gamma$

End of the algorithm

Fig. 2. Algorithm of local groups' construction

3.2 Multicast Tree Construction between Servers

Let us consider k local groups are built and each group has its local server (Figure 1). Here $\Gamma = \{S_1, S_2, ..., S_k\}$ is the set of these local servers distributed in different networks. It is important to note that the number of these servers can be very high and several

sources can belong to the same multicast group. Accordingly, it is necessary to build a multicast tree which links these servers while reducing the multicast group's participants. In this context, like DDVCA, we propose the use of core-based tree. However, we based the tree construction on several core nodes instead of one, aiming to avoid congestion problem. These core nodes are selected from Γ, by considering the servers which minimize the multicast delay variation with the others Γ's servers.

Table 1. Packets and data structures employed

Type	Arguments	function
INIT	$adrS_i$: address of the local server S_i *TTL:* Time to Live *adrM:* address of the group multicast	initialization packet
ACK	$adrS_j$: address of the sender S_j $adrS_i$: address of the receiver S_i	acknowledgement packet
SUCC	$adrS_i$: address of the new indicated core node S_i $adrCN_j$: address of the core node predecessor of S_i *$*: set of local servers which do not yet belong to any autonomous domains of core nodes already created *MAT*: matrix allowing to store the minimum delay between the elements of $ and core nodes in the multicast tree	successor packet

Initially, each local server broadcasts hop by hop an initialization packet *INIT* (Table1) at the multicast group *M* address. All members of *M* that are not servers delete the received packet, while the other nodes (server S_i) carry out the following operation:

- Response to the sender server S_j with an acknowledge packet *ACK*.
- Compute the minimum delay $d_{\min}(S_i, S_j)$ between it and S_j by using Dijkstra's algorithm [12].
- Memorize the address of S_j and the value of delay $d_{\min}(S_i, S_j)$.

Finally, each server S_i calculates its multicast delay variation through:

$$\delta_{S_i} = \max\left\{\left|d_{\min}(S_i, S_j) - d_{\min}(S_i, S_k)\right| / \forall S_j, S_k \in \Gamma, j \neq k\right\} \quad (4)$$

From this point each server exchanges the multicast delays variations values with the other local servers. Thus the server that minimizes the multicast delay variation will be elected as the first core node and noted CN_1 ($\delta_{CN_1} = \min\{\delta_{S_i} / S_i \in \Gamma\}$).

Afterwards, the first core node CN_1 builds its Autonomous Domain AD_1 by selecting all local servers of Γ that are accessible through a delay time lower than a given threshold of delay *D* (the threshold *D* is selected by taking account of the network extent). These allow us to build an Autonomous Domain with a minimum intra-delay communications.

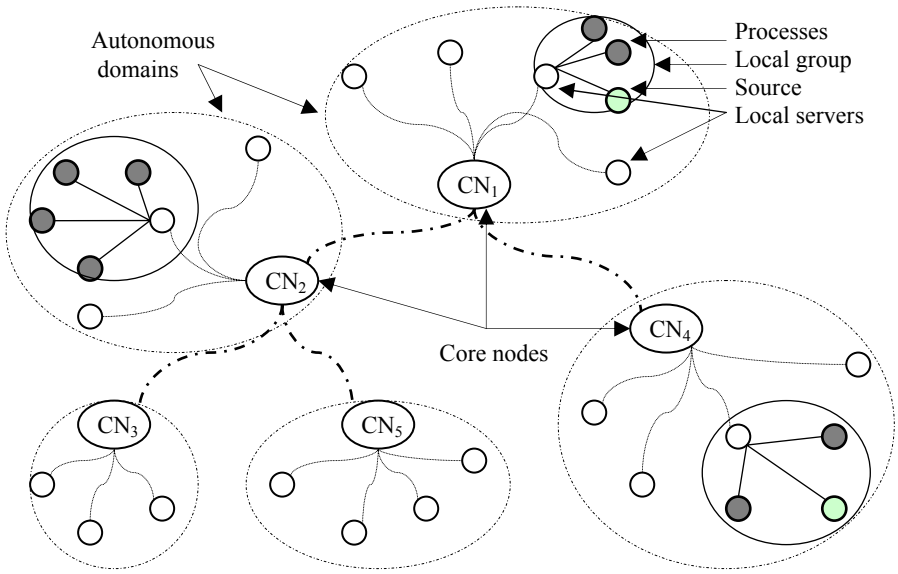

Fig. 3. Hierarchical structure of multicast group

Table 2. Delays between core nodes and servers of $

	S_i	S_j	…
CN_1	$d_{\min}(CN_1, S_i)$	$d_{\min}(CN_1, S_j)$	
CN_2	$d_{\min}(CN_2, S_i)$	$d_{\min}(CN_k, S_i)$	
…			

Here, all the other servers that are not in CN_1's domain will be stored in $ such as AD_1: $ = $\Gamma \setminus AD_1$. Then, CN_1 stores the minimum delay between it and all servers of $ in the matrix *MAT* (Table 2). After that CN_1's AD is built, and CN_1 gives the relay to another server in order to build the next AD. This is made by sending a *SUCC* packet to the nearest server S_i belonging to $. In other words, S_i is the server which minimizes the delay with CN_1 ($d_{\min}(CN_1, S_i)$). Finally, CN_1 joins the members of its autonomous domain AD_1 via the shortest path tree while the root is CN_1.

Meanwhile, when a local server S_i receives a *SUCC* packet from CN_1, this means that S_i is promoted as new core node noted CN_2. Accordingly S_i carries out the following operations:

- Select all servers of $ (the set of servers which are not in the domain of CN_1) having a delay smaller or equal to the threshold D and put them into its autonomous domain AD_2: $AD_2 = \{S_i \in \$ / d_{\min}(CN_2, S_i) \prec D\}$.
- Remove from $ the selected servers: $= $\setminus AD_2$.
- Remove the columns of the matrix *MAT* corresponding to the elements of its domain.
- Add a line into *MAT* to store the minimum delay between it and all servers of $: which do not belong to any domain.
- Search the smallest value of delays in *MAT* and take the pair (CN_p, S_q) corresponding to this value. In other terms, CN_2 selects the server *Sq* which minimizes the delay with already created core nodes (CN_1 or CN_2).

- Send a *SUCC* packet to the new core node S_q noted CN_3.

Finally, CN_2 joins its domain members and the predecessor core node CN_1 via the shortest path tree where the root is CN_2. Once the new core node CN_3 receives the packet *SUCC*, it makes the same thing as CN_2. Note that the AD building process is ended when \$ is empty or in other words, until all servers are connected to the multicast tree (Figure 3). For completeness we draw in Figure 4 the multicast tree construction algorithm.

Let $\Gamma = \{S_1, S_2, ..., S_k\}$ be a set of elected local servers.

Begin

1. for each local server $S_i \in \Gamma$ do
2. $d_{\min}(S_i, S_j)$ = the minimum delay between S_i and S_j, where $S_j \in \Gamma$ (the minimum delay is computed by Dijikstra' Algorithm)
3. end of for each local server $S_i \in \Gamma$ loop
4. for each local server $S_i \in \Gamma$ do //calculate the multicast delay variation of S_i
5. $\delta_{S_i} = \max\left\{d_{\min}(S_i, S_j) - d_{\min}(S_i, S_k) \right| / \forall S_j, S_k \in \Gamma, j \neq k\}$
6. end of for each local server $S_i \in \Gamma$ loop
7. *NewCN* ← S_i, where $\delta_{S_i} = \min\{\delta_{S_j} / S_j \in \Gamma\}$ // the server which has a minimal multicast delay variation represents the first core node
8. \$ ← Γ, *Predecessor* ← *NewCN* and *SetCN* ← Ø
9. while \$ ≠ Ø do
10. CN_i ← *NewCN*, *SetCN* ← *SetCN* U {CN_i}, *Pred*(CN_i) ← *Predecessor*
11. $AD_i = \{S_j \in \Gamma / d_{\min}(CN_i, S_j) \prec D\}$ //CN_i builds its autonomous domain
12. CN_i joins member of its domain AD_i and its predecessor *Pred*(CN_i) by the shortest path tree which root is CN_i
13. \$ ← \$ \ AD_i, removes AD_i from columns of *MAT* and adds line corresponding to $\{d_{\min}(CN_i, S_i) / S_i \in \$\}$ in *MAT*
14. *NewCN* ← S_q and *Predecessor* ← CN_p, where $d_{\min}(CN_p, S_q) = \min\{MAT(CN_j, S_k) / S_k \in \$, CN_j \in SetCN\}$
15. end of while loop

End of the algorithm

Fig. 4. Algorithm for multicast tree construction between local servers

3.3 The Time Complexity of the Algorithm

In order to determine the complexity of the proposed algorithm, we consider the following lemma:

Lemma. The worst case complexity of the algorithm is $O[(k+1)n^2]$, where k is the number of selected local servers and n is the number of nodes in the network.

Proof. The proposed algorithm time complexity is the sum of the time complexity of building the local groups (Figure 2) and the time complexity of building the multicast tree between the servers (Figure 4). On the one hand, the time complexity of building the local groups is in the worst case $O(m^2)$, where m is the participants' number in the group multicast M. In fact, the time complexity of constructing a neighbouring set NS_i of process m_i (Figure 2, line 2) is $O(m)$. Given that our algorithm executes the loop from line 1 to line 3 once for each process m_i belonging to M. ($m_i \in M$), the time complexity of lines 1-3, therefore, is $O(m^2)$. Here, the transit groups are constructed from line 4 to line

10, so during one iteration of the outer loop (4-10), the lines 7 and 8 are executed at most m times ($\forall m_i \in M, \left|NS_i\right| \prec m$). Accordingly, the time complexity of lines 4-10 is $O(m^2)$. Further the loop from line 11 to line 13 constructs local groups. Since the time $O(m)$ is required at most in line 12 ($|TG_i| << m$), then the time complexity of lines 11-13 is $O(m^2)$. From this point the number of iterations required to elect local and secondary servers for each local group LG_i is $|LG_i|$. If one consider that $m= |LG_1| + |LG_2| + \ldots + |LG_k|$, then the time complexity of lines 14-17 is $O(m)$. Finally, the time complexity of loop from line 18 to line 22 is $O(km)$. Since m is much higher than k ($m>>k$), then the time complexity of constructing local groups is $3*O(m^2) + O(km) + O(m)= O(m^2)$.

On the other hand, it is easily observed that the execution time of the multicast tree construction algorithm (Figure 4) is mainly spent on the loop between lines 1 and 3, namely on calculating the minimum delay between local servers. The time complexity of computing these minimum delays by Dijkstra's Algorithm is $O(n^2)$ [12]. Given that the proposed algorithm executes the loop from line 1 to line 3 once for each local server, the time complexity of lines 1-3, therefore, is $O(kn^2)$, where k is the number of elected local servers. Line 4 through line 6 compute the multicast delay variation for each local server $S_i \in \Gamma$, so if we consider that the time $O(k)$ is required in line 5, then the time complexity of lines 4-6 is $O(k^2)$. The design of the local server which has a minimal multicast delay variation as a first core node is in line 7. Thus, the time complexity required for execute line 7 is $O(k)$. Here, the loop from line 9 to line 15 connects all the local servers to the multicast tree. Line 11 requires k iteration to construct an autonomous domain AD_i for each selected core node CN_i. In line 12 the selected core node CN_i joins members of its domain AD_i and its core node predecessor $Pred(CN_i)$, within a number of iterations less than k ($|AD_i| + 1 <k$). Further line 14 selects the new core node from the set of servers which are not belonging to any created autonomous domain through k^2 times. Thereby, since the loop between lines 9 and 15 is executed at most k times, then overall time complexity of lines 9-15 is $O(\mathrm{k}^3)$. Finally, the time complexity of constructing multicast tree between servers is $O(kn^2) + O(k^2) + O(k) + O(k^3) = O(kn^2)$, because n is much higher than k ($n>>k$).

At this point by considering that $O(m^2) + O(k\ n^2) = O[(k+1)n^2]$, because the number of destinations nodes m is lower than the number of nodes in the network n, the overall time complexity of the proposed algorithm is $O[(k+1)n^2]$. Since $k<<m$, then our algorithm shows lower time complexity than DDVCA, which is equal to $O(mn^2)$.

4 Simulations and Analysis

In order to evaluate the advantages of the proposed scheme, we have constructed a set of simulation using ns-2 (Network Simulator) [13]. We compare the proposed algorithm with DDVCA. The simulations focus on the protocols' abilities to maintain low multicast delay variation while minimizing the bandwidth consumption. During the simulation, we deliberately change the network topology by changing the network's size (500 and 1000 stations) in order to evaluate the ability of the proposed scheme to suit different network configuration. Further, the destination nodes in the multicast group represent 50% of the network size, while 10 source nodes are chosen randomly. Each run consists of 100 second, where each source generates 20 (media units/s). Note that the media unit size is 500 bytes. For completeness the scheduling of multimedia flows' diffusion is shown in Table 3.

Table 3. Scheduling of diffusion multimedia stream

Sources	1	2	3	4	5	6	7	8	9	10
Begin (s)	0	10	20	30	40	50	60	70	80	90
End (s)	100	100	100	100	100	100	100	100	100	100

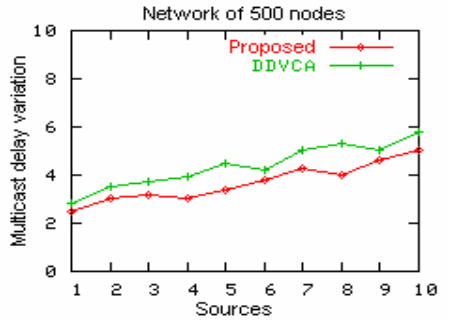

Fig. 5. Multicast delay variation's average 500 stations

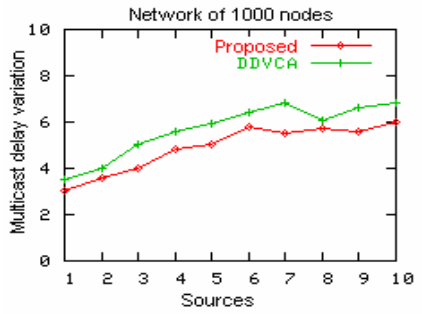

Fig. 6. Multicast delay variation's average 1000 stations

Figures 5 and 6 represent the multicast delay variation average when using 500 and 1000 stations, respectively. It's clearly seen that our mechanism outperforms the DDVCA mechanism in both situations. This is expected as our algorithm constructs both the local group and the autonomous domain according to the multicast delay variation constraint. In contrast DDVCA takes this constraint into account only through the choice of the core node. Furthermore, it is important to note that our scheme's gain in multicast delay variation over DDVCA is roughly 1 sec, which is very high if we consider the case where the flows represent a video-based session. In fact, this translates into a high jitter entailing devastating consequences on the perceived video quality at the receiver.

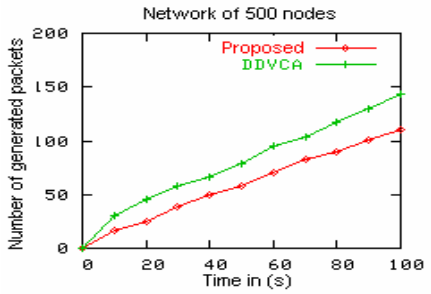

Fig. 7. Bandwidth consumption 500 stations

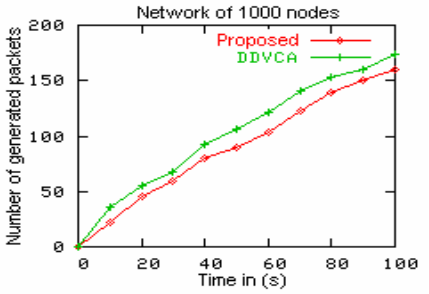

Fig. 8. Bandwidth consumption 1000 stations

Figures 7 and 8 show the bandwidth consumption of both the proposed scheme and DDVCA. Note that these graphs are obtained by computing periodically (10 s) the packets' number generated by both the proposed scheme and DDVCA. Actually, DDVCA uses more bandwidth than the proposed scheme. This is caused by the fact that DDVCA avoids using the unicast packets only between the senders and the core node. In contrary, the proposed scheme uses the unicast packets only in the intra-local group communication. Indeed, the unicast packets increase the bandwidth consumption.

Additionally, to see the influence of the multicast group's size, we have increased the number of nodes in the network (1000 stations). As indicated, it is easy to notice that the proposed algorithm is always better than the DDVCA.

5 Conclusion

In this paper we have proposed a novel multicast tree construction in order to solve the so called delay and multicast delay variation problem. The proposed scheme offers an improved ability to minimize the multicast delay variation as well as the bandwidth consumption while having a lower time complexity.

Simulations have shown that the proposed mechanism achieves numerous performance gains over the DDVCA. In addition to minimize the multicast delay variation, the proposed scheme improves efficiently the bandwidth utilization by minimizing the packets' number in the network. Furthermore, the proposed algorithm exhibits a time complexity lower than DDVCA, $O(mn^2)$ and $O[(k+1)n^2]$ respectively $(k << m)$. Our future works will focus on the implementation of our algorithm in real network configuration. Furthermore, we will also extend our proposal with a QoS management-based such as Diffserv.

References

1. A. Ksentini et al. "Novel Architecture for reliable H.26L video transmission over IEEE 802.11e", *Proc. IEEE PIMRC'05*, Barcelona, Spain.
2. D. Weitzman, C. Partridge, "Distance Vector Multicast Routing Protocol", *RFC 1075*, November 1998.
3. S. Deering et al., "Protocol Independent Multicast-Dense Mode (PIM-DM): Protocol Specification", *RFC 2365*, July 1998.
4. J. Moy, "Multicast Extension to OSPF", *RFC 1584*, Mars 1994.
5. D. Estrin et al., "Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification", *IETF RFC 2362*, June 1998.
6. A. Striegel and G. Manimaran, "Survey of QoS Multicasting Issues", *IEEE Communications Magazine*, June 2002, pp. 82-87.
7. G. N. Rouskas, I. Baldine, "Multicast routing with end-to-end delay and delay variation constraints", *IEEE JSAC*, April 1997, pp. 346-356.
8. P.-R Sheu, S.-T. Chen, "A fast and efficient heuristic algorithm for the delay and delay variation bound multicast tree problem", *Information Networking, Proc. ICOIN-15*, January 2001, pp. 611-618.
9. A. Benslimane, O. Moussaoui, "A scalable Multicast Protocol with QoS guarantees", *Proc. of IEEE/IFIP Net-Con'2003*. Muscat, Oman. October 2003.
10. B. Wang and J. C. Hou, "Multicast Routing and its QoS Extension: Problems, Algorithms, and Protocols", *IEEE Networks*, January/ February 2000.
11. A. Ballardie, "Core Based Trees (CBT Version 2) Multicast Routing: protocol specification," *IETF RFC 2189*, September 1997.
12. E. W. Dijkstra, A note on two problems in connection with graphs, *Numeric Mathematic*, vol. 1, 1959, pp. 269-271.
13. Network Simulator 2, ns-2, http://www.isi.edu/nsnam.