UNIVERSITÉ LIBRE DE BRUXELLES
Faculté des Sciences Appliquées
CODE

Implémentation d'une application de vidéo conférence exploitant ALM (Application Layer Multicasting)

Mémoire présenté en vue de l'obtention
du grade d'Ingénieur Civil en Informatique
Promoteur : Esteban Zimanyi
Co-promoteurs : Jean-Michel Dricot,
Shervin Shirmohammadi, Dewan Tanvir Ahmed

Jimmy BONNEY
Année académique 2006–2007

The Free University of Brussels
Faculty of Applied Sciences

Peer-to-Peer Routing Algorithms Comparison in Application Layer Multicast

Master of Science Thesis
Supervisor : Esteban Zimanyi
Co-supervisor : Jean-Michel Dricot,
Shervin Shirmohammadi, Dewan Tanvir Ahmed

Jimmy BONNEY
Academic Year 2006–2007

Contents

List of Figures

Chapter 1

Introduction

Multimedia applications are all over the web. Watching streaming video, listen to radio broadcast, playing MMORPG (Massive Multiplayer Online Role-Playing Game) is now very common and getting more and more popular. Some data need to be transmitted from one computer to many others, but not all others. It is easy to see that in a MMORPG for example, data must be transmitted in a group belonging to the same area. It would create too much traffic to send data to thousands of players if only a hundred needs the information.

Many applications now have a video content in addition of voice and text. The problem is no more to create multimedia content but how to diffuse it. Websites like *youtube* or *dailymotion* allows simple users to easily post videos. But here, it is no more live use and huge requirements for bandwidth on the server side are needed.

In order to limit bandwidth requirements, and especially to allow simple end-users to diffuse video to a certain amount of receivers, peer-to-peer solutions have been developed. Indeed many applications are already using this concept to share data (*bittorrent* [6], *emule* [8], *skype*). Therefore why not using it to share live video and create bandwidth respectful video-conference applications for example? This is the idea behind application-layer multicast (ALM) and this will be the subject of our thesis.

Our work is divided in two main parts. The first one is a theoretical introduction to our subject and is divided in three chapters. The second part presents our work and is divided in two chapters.

Chapter two begins with a small reminder about Internet and TCP/IP model. We explain how IP multicast works and why this method, even though it seems perfect, is not really used by applications. Issues with this technology lead researchers to find other solutions. One of them is application-layer multicast, using the intelligence at end-hosts to overcome the problem. We will also see some parameters used to compare results from different algorithms.

Following this, chapter three will present a partial state-of-the-art in application-layer

multicast. It was not possible to be thorough since too many solutions have been created all over the world. However, we present some well known algorithms which goal can be different. Some are focused on reliability, others on low latency or little overhead.

Last chapter of the first part will introduce the simulating environment, *OPNET*. This environment is mostly used in industry to simulate behavior of algorithms before they are implemented in a prototype. It allows to develop new model at every layer of the TCP/IP model and to simulate complex networks, worldwide spread. On top of that, many tools are present by default to study the results but user can also add their own parameters.

Chapter five present the work we did with OPNET. We explain how we modeled NICE and DS-ALM algorithms at the application-layer using finite state machine. We also compared these algorithms in order to explain and justify the parameters we used during the implementation. Finally, since OPNET comes with many libraries we had only to deal with behavior at the application-layer and did not have to bother about routing schemas at network level.

Second chapter of the second part presents our results. We simulated the simplified behavior of NICE and DS-ALM and compare them in term of end-to-end delays, number of hops and stretch. Since both applications mainly focus on low latency communications, end-to-end delays were a minimum to study. We decided to add the number of hops even if it does not really reflect delays. Finally stretch allowed to see how many times packets were duplicated at a end-host and therefore to have a comparison with direct unicast connections.

Part I

Preliminary notion

Chapter 2

Application Layer Multicast

2.1 Introduction

The Internet is the interconnection of many networks. Many applications send data from one network to one or many others. Lately, with the development of large bandwidth connections in homes or offices, applications can now easily send multimedia traffic over the Internet.

However, a problem of high bandwidth consumption can appear when one server wants to send the same packet to many hosts at the same time. Indeed, for a group of x users it would multiply the server's bandwidth requirement by x. That is the reason why a method has been developed in IPv4: IP multicast. This method allows to send one packet and to duplicate it at the network layer (by routers) while it is going through the network. All registered members will receive the packets for a low cost and the same delay as if the sending was a unicast message.

Despite this wonderful idea, many routers do not implement multicast capability. Therefore, an idea is to report the duplication of packet at end-hosts (receivers). They will then forward it to the next interested members. One can easily see that it will create higher traffic than the ideal IP multicast. It will also introduce higher latency.

In this chapter we will give a quick overview of the TCP/IP model and will focus specially on the Internet layer. We will then present IP multicast in detail and will conclude with the particularities of Application Layer Multicast.

2.2 Network models

Networks are organized in layers. Nowadays, the most used model is the OSI model but practically, TCP/IP model is most deployed. Figures 2.1 and 2.2 [19, p. 48] illustrate these models and show the layers' correspondence. It is a simplified view since layers in both model

do not stop at the same "level". However, we will only consider the TCP/IP model for the rest of the chapter since we focus on the practical and not the theoretical aspect.

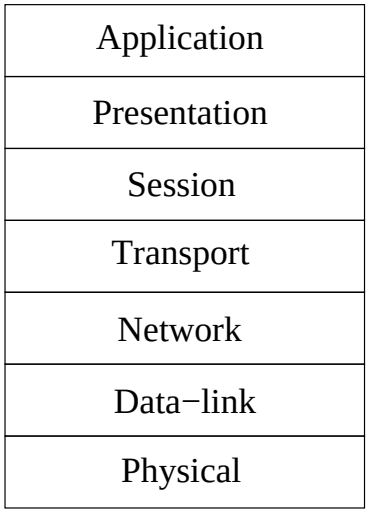

Figure 2.1: OSI model.

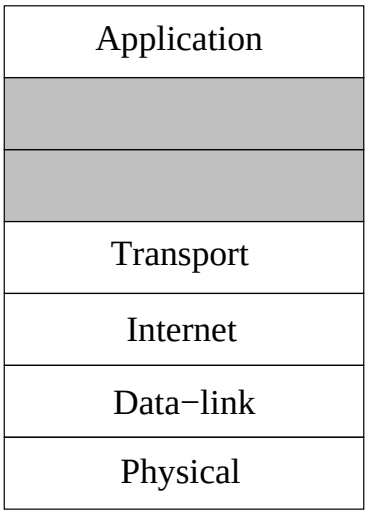

Figure 2.2: TCP/IP model

In this chapter we will focus on two layers of the TCP/IP model:

- Internet layer
- Application layer

2.2.1 Internet layer

The Internet layer connects different unconnected networks. It ensures that a message sent from one host in a network A is able to reach another host in a network B. Therefore, the technologies used in data-link or transport layers will not have any role to play in the path the message will take. This ability of achieving data through the Internet is the routing. Internet layer supports Internet Protocol, IP, which is connectionless. The path in the network is not defined a priori. When a host sends a packet to another one, the other side does not need to do anything and the path forwarded by the packet is not reserved/blocked.

To provide an efficient routing, IP uses addresses called IP addresses. Each address is a 32 bit integer unique on the network. IP addresses are divided in classes as it is shown on figure 2.3 [15, Lecture 2]:

- Class A: from 1.0.0.0 to 127.255.255.255
- Class B: from 128.0.0.0 to 191.255.255.255
- Class C: from 192.0.0.0 to 223.255.255.255

- Class D: from 224.0.0.0 to 239.255.255.255
- Class E: from 240.0.0.0 to 255.255.255.255

We will focus later on class D addresses since they are multicast addresses.

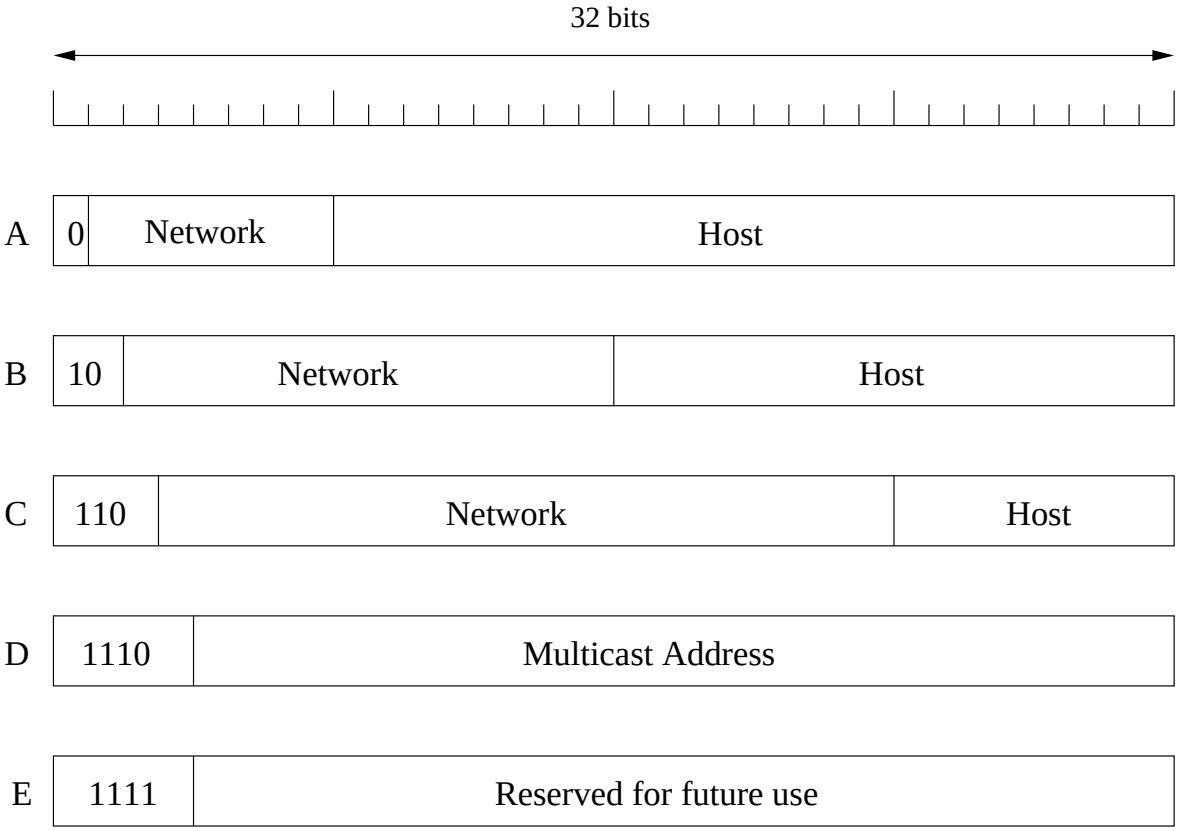

Figure 2.3: IP address format.

In addition to the routing, this layer also segments/desegments data from the upper layer but do not offer any reliability (except a checksum for the header). It can suffer from:

- Data corruption
- Out of order arrival
- Duplicate arrival
- Lost or dropped/discarded

The segmentation consists in encapsulating data from the upper layer in one or more packet(s). These packets will have an IP header which format is given in figure 2.4 [19, p. 467]. One can easily see that IP addresses are added to the header, but a lot of other

elements are present too. For the details of these different fields, see [19, p. 467-471] or/and [3, Section 3-7].

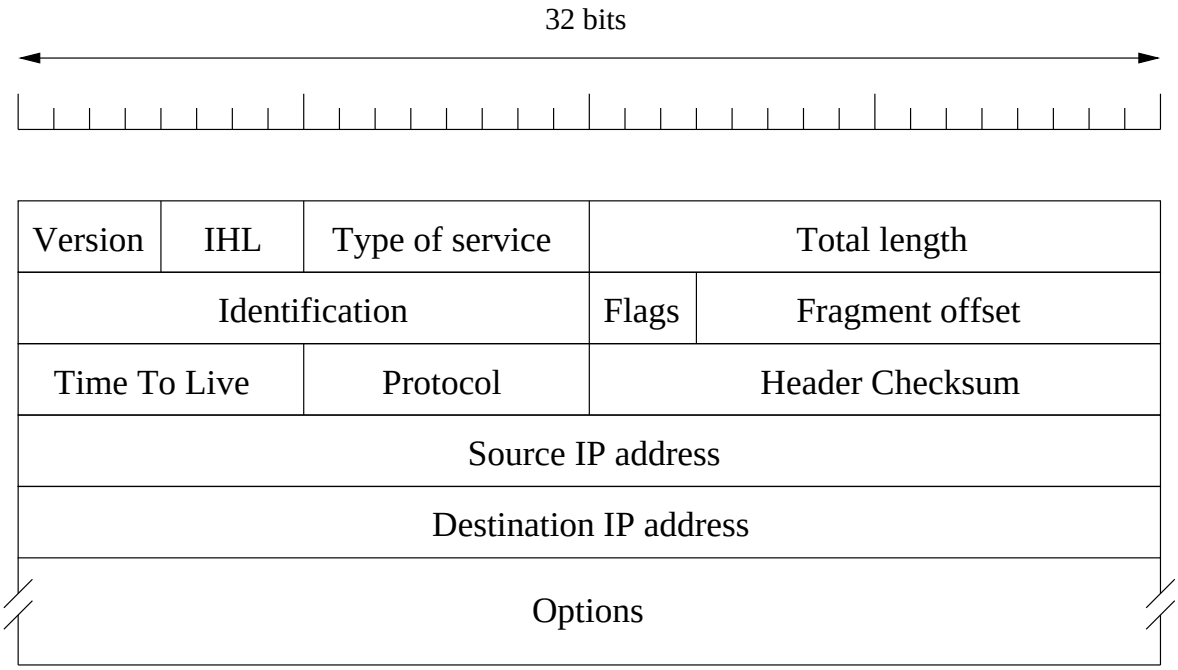

Figure 2.4: IP packet format.

2.2.2 Application layer

The application layer is just above the transport layer. It provides services for the end user. Some well known applications are TELNET, FTP, HTTP, SMTP, TLS/SSL, DNS. All these applications rely on the services offered by the transport layer.

2.3 Network Layer Multicast

2.3.1 Network layer addressing methods

Different ways are possible to address another host on the Internet. We begin to give a short description of all of them as well as an illustration:

- Unicast (figure 2.5). It refers to a direct connection between two hosts: a sender and a receiver. Usually each network device possesses a unicast address.
- Broadcast (figure 2.6). It allows a sender to communicate with all other possible hosts by sending only one message.

- Multicast (figure 2.7). It allows sender to communicate with members of a group of interested hosts by sending only one message. The packets are duplicated by the routers which will then send the message to the register hosts.

- Anycast (figure 2.8). By sending only one packet, the sender can reach multiple hosts, the ones that are closest to him according to the routers.

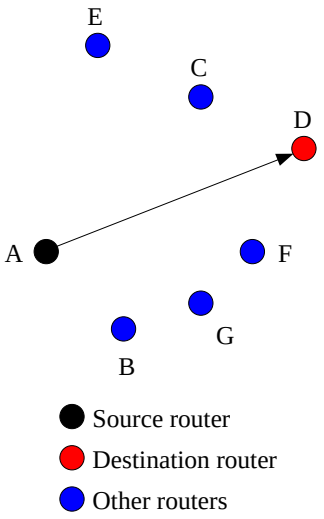

Figure 2.5: Unicast.

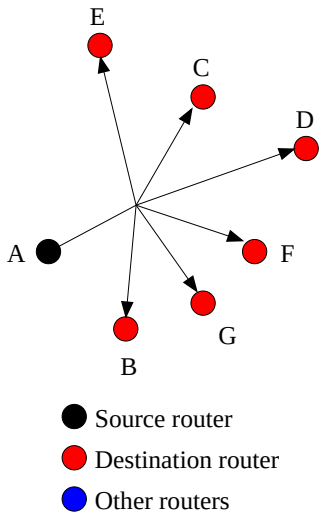

Figure 2.6: Broadcast.

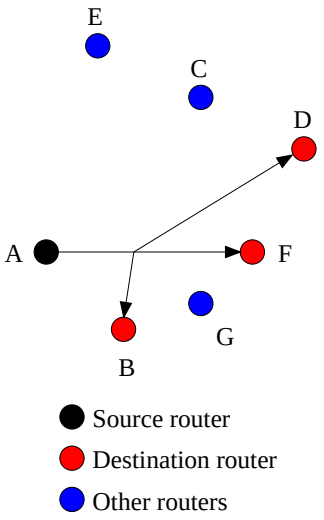

Figure 2.7: Multicast.

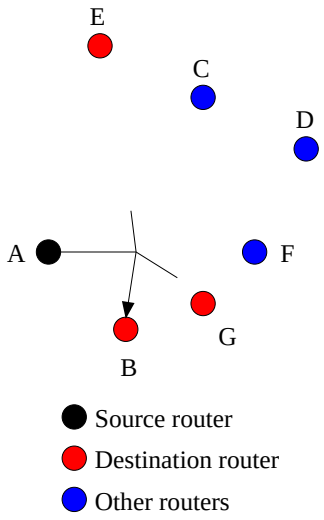

Figure 2.8: Anycast.

In the following section, we will develop the multicast addressing method.

2.3.2 IP multicast

As described above, multicasting allows registered users to receive data they are interested in. IP multicast solve a problem in which one sender wants to send a packet to many receivers (who can be or not in the same network) but do not have a bandwidth high enough to provide all receivers with the data stream through unicast connections. One can easily see that this problem can happen often nowadays with our media oriented society. Indeed, numbers of videos or music tracks are now accessible on the web.

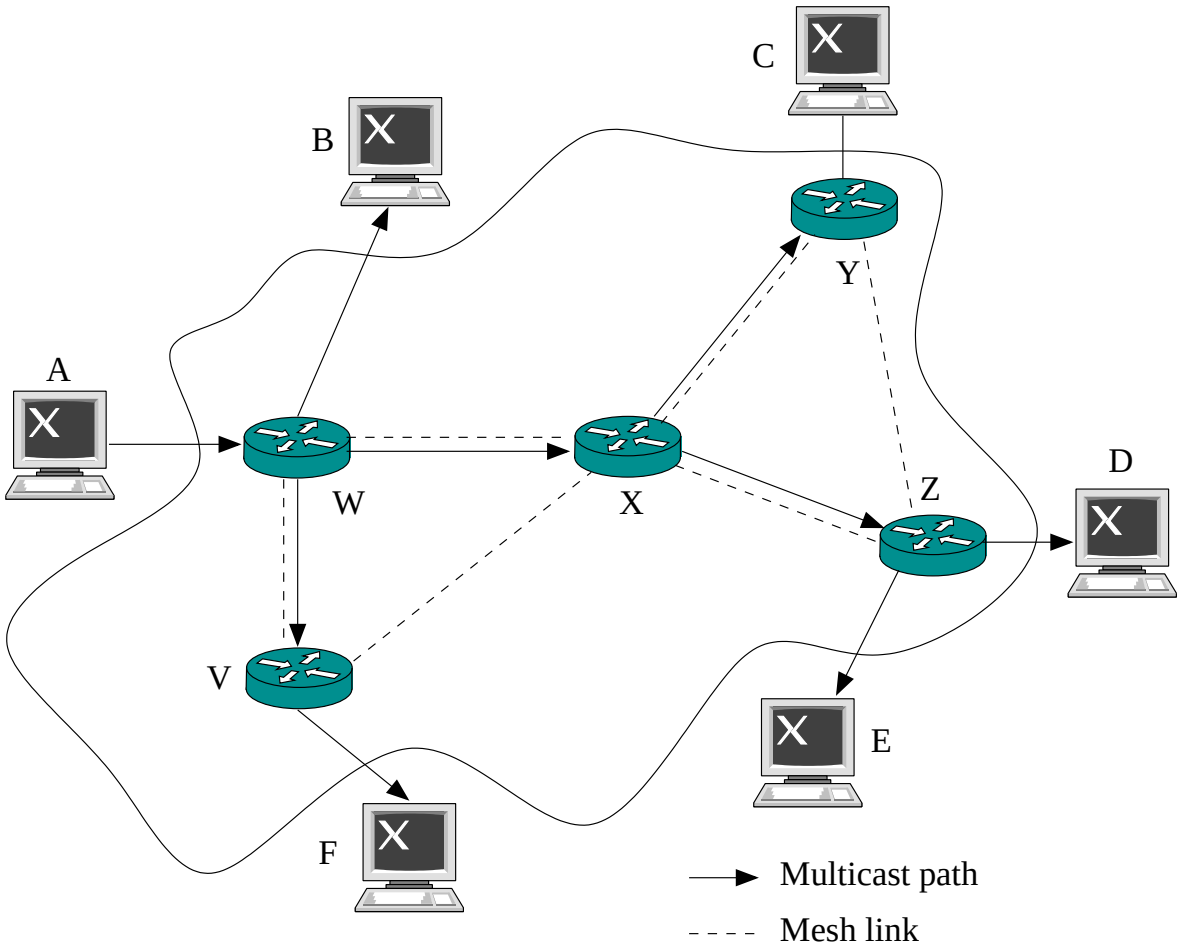

Figure 2.9: IP multicast illustration.

As we explained above in section 2.2.1, IP provides class D addresses as multicast addresses. The first four bits (1110) identify IP multicast address while the twenty-eight other bits are used to identify the multicast group [7]. Addresses 224.0.0.0 to 224.0.0.255 are reserved for local purpose and addresses 239.0.0.0 to 239.255.255.255 are reserved for administrative purpose. The remaining addresses offer $2^{28} - 2^{24} - 256 \simeq 251.657 \cdot 10^6$ possible groups.

Despite this big amount of groups and the beautiful idea behind IP multicast some hosts are not able to provide this service, or are partially able to do it. Three levels of conformance exist [10]:

Level 0: No support for IP multicast. It is not mandatory in IPv4 protocol to provide support for IP multicast. Therefore many hosts do not have this functionality. If they receive IP multicast packet, they are just ignored.

Level 1: Support for sending IP multicast packets but not for receiving. In addition, it is not mandatory to join a multicast group to send packets.

Level 2: Full support for IP multicast. Hosts are able to send and receive multicast packet but they also know the procedure to join and leave a multicast group.

We did not talk about the transport layer earlier in order to concentrate on the network and application layers. However, one should know that two main protocols are present in the transport layer: TCP (Transmission Control Protocol) which is connection-oriented and UDP (User Datagram Protocol) which is connectionless. To summarize, purpose and capabilities of both protocols are described in figure 2.10 [19].

TCP	**UDP**
Reliable	Unreliable
Ordered delivery	Unordered delivery
Stream transmission	Datagram transmission
Heavyweight	Lightweight

Figure 2.10: TCP/UDP proprieties

Based on that, one can easily deduce that IP multicast will use UDP as a transport protocol since there are no connection establishment between the sender and the receiver. However, lots of protocols have been developed for multicast but it would be too long to develop all of them. [10] gives some examples: RTP (Real Time Protocol), SRM (Scalable Reliable Multicast), MFTP (Multicast File Transfer Protocol).

As we explained earlier, one of the main problems of IP multicast is that some hosts are not able to use this service. The worst problem is that if only one router in the middle of the network does not support IP multicast then all potential members behind this router will not be able to participate in the group. To avoid this problem, a solution has been developed: MBone or Multicast Backbone. It consists of "virtual multicast network based on multicast islands connected by multicast tunnels" [10]. To develop the idea, MBone is a network on

top of the Internet connecting multicast-available island with unicast connection in order to avoid the problem of non-multicast routers. This requires special hosts called mrouter. A mrouter has two functionalities: being able to contact other mrouter by establishing a tunnel (i.e. a unicast connection through the Internet) and being IP-multicast capable. Packets for a multicast group are encapsulated in a traditional IP unicast packet while traveling through non-capable router and are decapsulated while reaching the destination mrouter. Figure 2.11 illustrates a mBone topology, where island A to F contain IP multicast enable hosts. G is also an IP multicast island but not yet attached to the MBone.

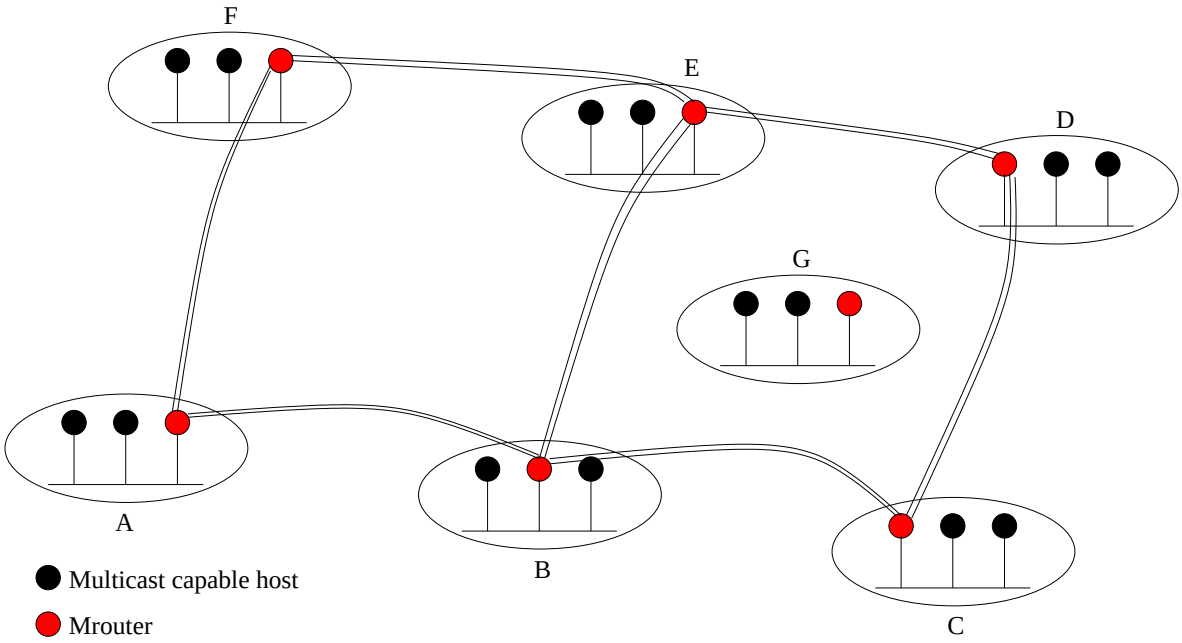

Figure 2.11: MBone illustration.

To join an IP multicast group, a host sends a message to the router it is linked to. This router will forward the message to all other multicast router it is linked to so that they know they have to forward the message to this new router.

2.4 Application Layer Multicast

Since a lot of routers are not able nowadays to deal with IP multicast and since multi-diffusion is getting more and more present in the Internet, some people suggested relay the multicast to a higher level, the application level.

Of course the problem of reporting the multicast to the application layer is that the IP multicast becomes useless. Therefore the sender will not be able to send only one packet to

access all group members. There will be many unicast packets. The idea will be to divide the sending load between all multicast members. Figure 2.12 has to be compare with figure 2.9. We can see that the duplication process is no more the responsibility of the routers but of the end-hosts.

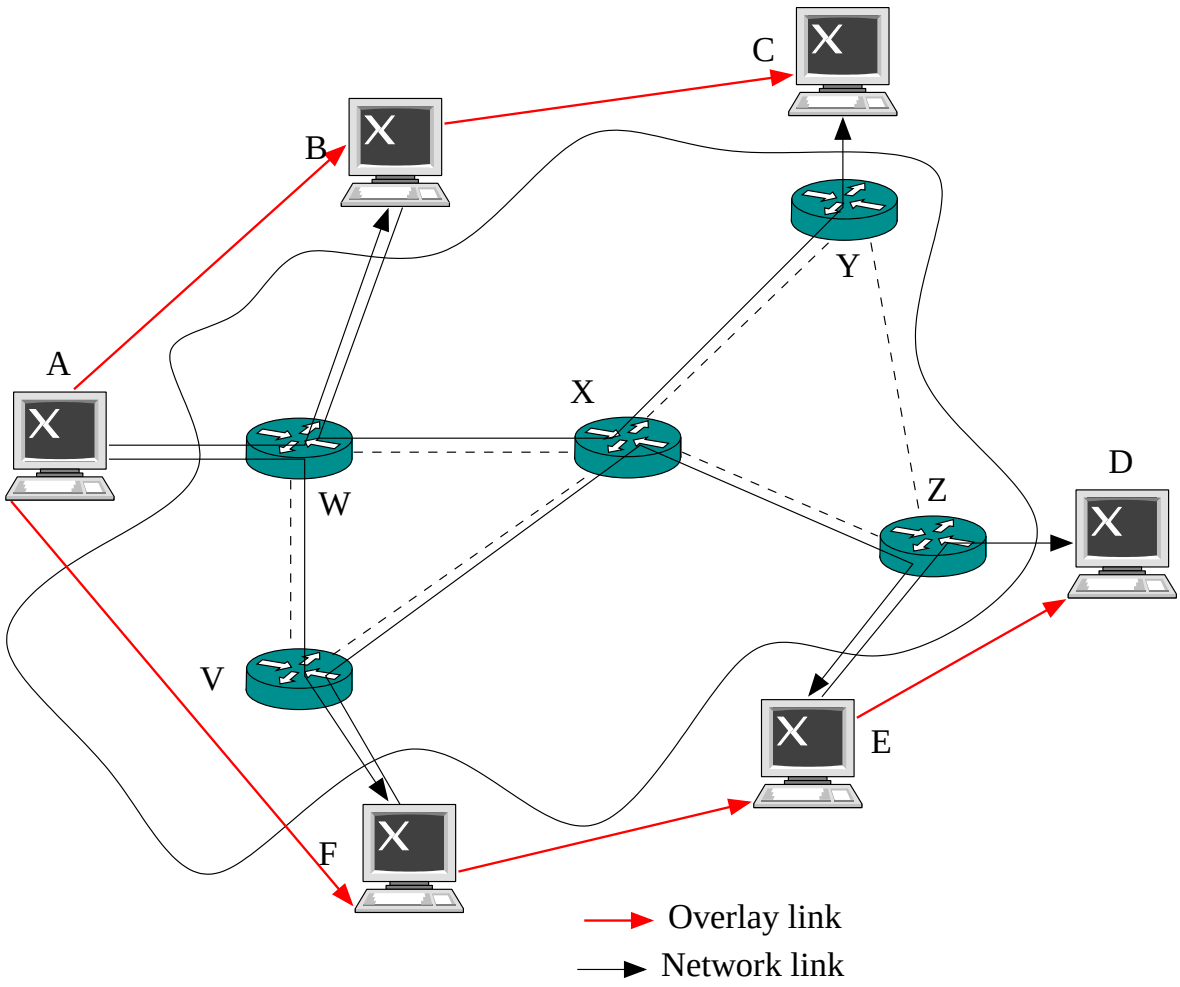

Figure 2.12: ALM illustration.

Of course this method will introduce duplicate messages. On figure 2.12, one can see that two messages are initially sent by A (instead of one with IP multicast). One can observe as well that end-host F receives a packet from router V before sending the same packet to router V. Therefore link $F - V$ will see the same message twice.

Another aspect is the latency introduced by the forwarding process. Indeed, with IP multicast, messages were sent directly to the receivers. On the other hand, on can see on figure 2.12 that D receives the messages only when F and E have forwarded it.

Different parameters permit to study the behavior of Application Layer Multicast proto-

cols. Indeed, many protocols exist and we need some parameters to be able to compare them. The traditional ones are:

Link stress. This is the number of duplicate packets over a physical link. For example, the link $F - V$ in figure 2.12 has a stress value of 2. The lower is the stress value, the better. Indeed that means that we are getting closer to IP multicast.

Node degree. This is the number of unicast connection each host is maintaining with its neighbors. A has two neighbors: B and F and thus a degree value of two.

Hops. Two possible definitions exist. One considers the overlay topology, i.e. above the Internet topology and thus it just considers end-hosts as possible hops. The other one considers the Internet topology i.e. both end-hosts and routers are considered as possible hops. The difference is shown in figure 2.12. If we consider the first definition, the number of hops between A and D is 3:

$$A \rightarrow F \rightarrow E \rightarrow D$$

The second definition for the same path will be 9:

$$A \rightarrow W \rightarrow V \rightarrow F \rightarrow V \rightarrow X \rightarrow Z \rightarrow E \rightarrow Z \rightarrow D$$

Stretch. The stretch measures the ratio of ALM end-to-end delay over unicast end-to-end delay. If t_i is the time taken to send a packet from A to D on the overlay network and t_d is the time to send the same packet with a unicast connection from A to D, then the stretch is given by

$$s = \frac{t_i}{t_d}$$

End-To-End delay. As the name mentions, we consider the delay between end-systems. The most interesting is the maximum end-to-end delay for the whole system since it gives the worst case scenario for two end-hosts trying to reach each other. As many of the parameters above, the lower is the maximum end-to-end delay, the better.

Bandwidth. The overlay network requires message exchanges. A lot of messages are sent by group members. Indeed, many messages are usually sent to join the diffusion group, others are sent as heartbeat messages, others for the maintenance of the network, etc. These messages are not used for the application by itself and require some additional bandwidth. It has to be taken in consideration for each host when it is joining the group.

Robustness. The ALM protocol has to ensure that it will not involve too many packet loss due to the structure of the overlay network or a too high bandwidth consumption for example.

All these parameters will be influenced by the diffusion schema. Application Layer Multicast protocols have to choose how members are connected. Some build a tree, others prefer mesh and some do not want to make a choice and use both. Next chapter will illustrate different ALM protocols to give the reader some ideas.

2.5 Conclusion

TCP/IP model through its Internet layer and its class D addresses provides a perfect way to address the one-to-many sending problem. Unfortunately it is not mandatory for routers to implement an IP multicast solution. Therefore a single router can stop the propagation chain and stop the diffusion. A solution has been developed consisting on IP multicast islands connected together by tunnels, i.e. unicast link.

Another solution, easier to implement for end-users, is to transfer the replication and forwarding processes in the application layer. This is the assumption of Application Layer Multicast. In this case, end-hosts set up an overlay network. The problem of this solution is that it introduces higher latency. It also introduces duplicate packets on the same physical links leading to bigger bandwidth requirements. Next chapter presents solutions to address these problems.

Chapter 3

State of the art

3.1 Introduction

Many ALM protocols have been developed during the last few years. We can not make a thorough list in this report, neither can we explained all of them in detail. Therefore, we had to make a selection. We will analyze the basics of eight protocols. Some of them focus on a single-source – multi-receivers [18, 20] while other are developed for multi-sources – multi-receivers [5, 22, 16]. Some favor reliability while others favor shortest paths. Anyway, we decided to explain what are the processes used to build the overlay network without concentrating on how it is maintained. Therefore, we usually describe the joining process and the overlay topology and illustrate a multicast diffusion. We do not explain how protocols deal with early departure or late comers, some of them even not having these functionalities.

3.2 NICE: NICE is the Internet Cooperative Environment

NICE [4, 5] protocol has been developed by the University of Maryland (USA). It suggests a tree-based algorithm to establish a hierarchy among clients while they gradually connect to the diffusion group.

The construction of the tree is based on the minimum end-to-end latency between the new member and existing ones. NICE hierarchy is based on different layers (the lowest layer is layer zero and denoted L_0). Each layer is then divided in clusters which size vary between k and $3k - 1$ where k is a constant of the protocol. Each cluster will contain hosts that are closed to each other. In each cluster is elected a leader which is the center of this cluster (it has the minimum maximum distance to all other members of the cluster). The root of the tree will be the leader of the highest cluster.

All hosts belong to the lowest layer L_0. These hosts are separated in clusters by a specific

algorithm. A leader is elected for each cluster and responsible of the cluster splitting operation. All leaders of layer L_i then join layer L_{i+1}. This is illustrated in figure 3.1 where $k = 3$. One can see that the leader of clusters in layer 0 are C, F and M. Therefore, those nodes are members of layer 1 and belong to the same cluster which leader is here F. By an iterative process, since F is leader of the cluster in layer 1, it also belongs to layer 2 and since it is alone in layer 2, F will be the root of the tree.

Once the topology is established, all members of a cluster are aware of the cluster members. Therefore on the example of figure 3.1, all members of cluster [ABCD] know each other and it is the same for each cluster on every layer. The data path for sending messages can now be established. In order to avoid implementing duplicate packet detection, and suppression mechanism the delivery path for multicast message is a tree.

When a member of the diffusion group wants to send a message, it will send it to all members of all clusters it belongs to. When receiving the message, each member will forward it to all members of the clusters it belongs to, except in the layer the message came from. It permits avoiding duplicate messages while ensuring that every member of the group will receive the message. This is illustrated in figure 3.2. A is the initial sender of the packet. When C receives it, it forwards it to the cluster it belongs to in layer 1 (since it receives it in layer 0). Then F and M will forward the packet to the cluster they belong to in layer 0 since they receive it from cluster in layer 1.

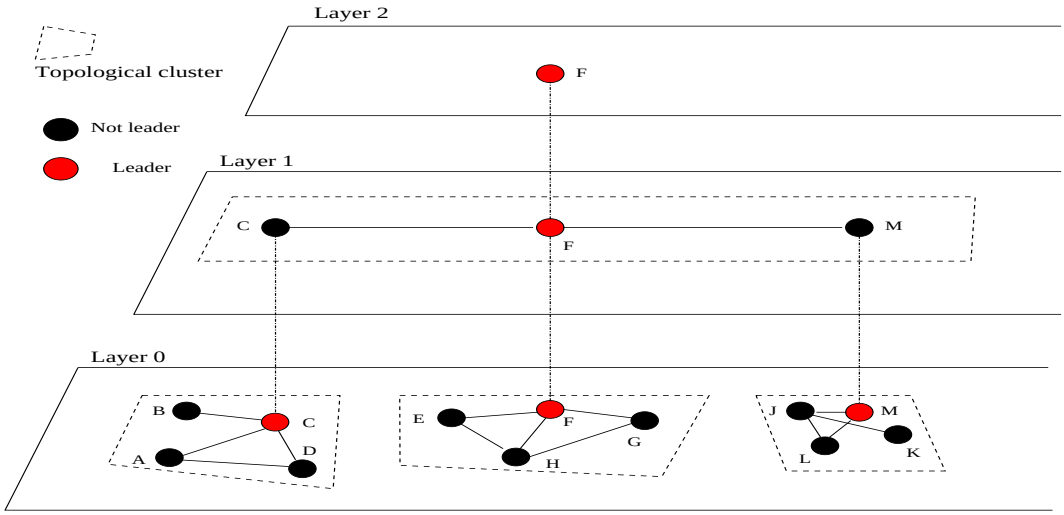

Figure 3.1: NICE. Hierarchical topology.

The joining process is quite simple. When a new host wants to join the diffusion group, it contacts the Rendezvous Point (RP) which address is well known by all potential clients. The RP replies by sending the root of the tree. Then a iterative process takes place so that

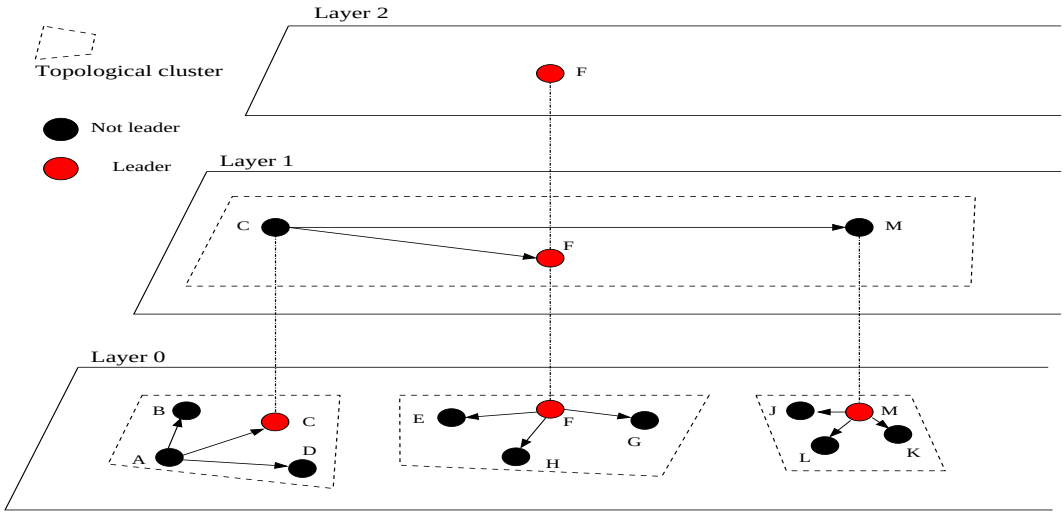

Figure 3.2: NICE. Multicast illustration.

the new member finds its place in the tree:

1. New member contacts all hosts of the cluster in layer L_i and waits for replies.
2. New member chooses the closest host (in term of end-to-end delay) and ask him for members in its cluster in layer L_{i-1}.

The process ends when the new member knows in which cluster of L_0 it will be. Once he knows its new cluster's members, he sends them a joining message.

This joining process is illustrated in figure 3.3. The new member will begin to contact the RP. Then it will contact the root of the tree (i.e. the member in the highest layer). When receiving the answer from the root (the response are not shown in the figure) it will contact the member of the cluster in L_1 (C, F and M) in which the root is the leader. It will select the closest host (C) and ask him for the member of the cluster he is leader of in layer L_0 (A, B, C and D) to finally send to all of them a joining request.

3.3 Reliable ALM

Rong et al. [14] have developed a reliable multicast model based on peer lifetime. It suggests a tree topology which root will be the client connected for the longest time.

The model is based on the fact that peer's lifetime does not follow the traditional exponential distribution but a heavy-tailed distribution mathematically expressed as:

$$P\{X > T\} \sim T^{-\alpha}, \text{ as } T \to \infty, 0 < \alpha < 2$$

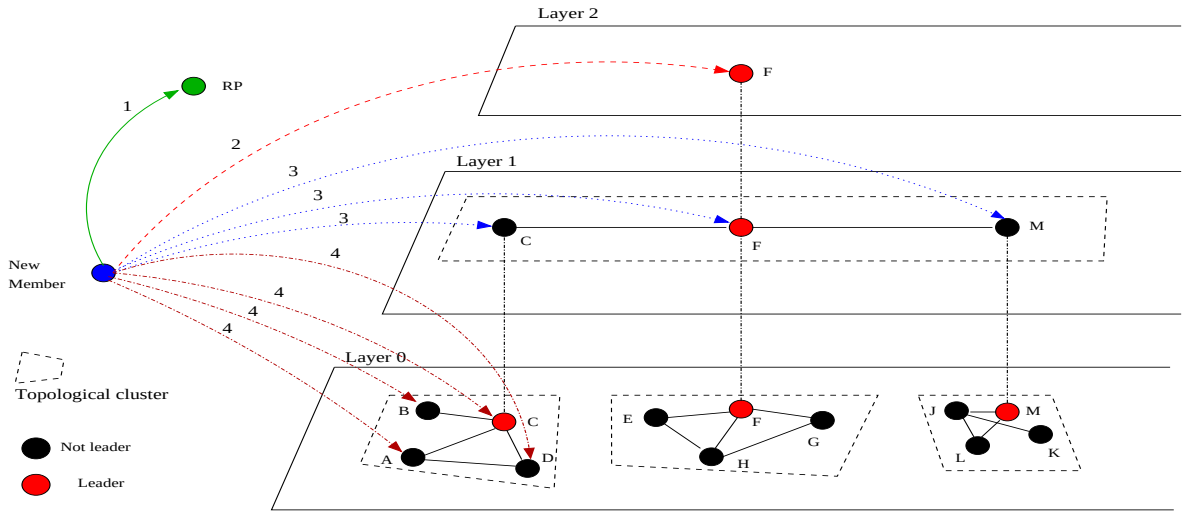

Figure 3.3: NICE. Joining procedure.

where X is a random variable representing expected lifetime of a peer and T is the time. Since lifetime of peers is following this distribution, only a few of them will be active the entire session. In addition, in opposition to exponential distribution, the system has a memory. Therefore, the remaining time of a peer is linked to the time already spent in the system. The more satisfied a peer is, the longer it will stay.

The main purpose of the method is to promote reliability. The creators proposed a logical hierarchy added on top of a multicast tree. The later is simply a shortest path tree or minimum spanning tree minimizing the network delay. The logical hierarchy will determine which nodes are the most reliable. To do so, each node will be attributed a rank depending on the time spent in the system. This rank will lead to a layer and each new joining peer will start in layer 0 (L_0) and will be attached to an "older" parent. This is illustrated in figure 3.4 and 3.5 where the arrival time of each host are $t_A < t_B < t_C < \ldots < t_F$. Member A is in the highest layer (with a rank of 2) and is considered as the source of the multicast diffusion group. It is the most stable node. Under him, layer 1 contains stable node and finally layer 0 contains the most unstable ones.

Ranks are calculated depending on the time spent in the system. "Time is divided into logarithmic sized bins":

- bin 1: 0 – 1
- bin 2: 1 – 3
- bin 3: 3 – 7

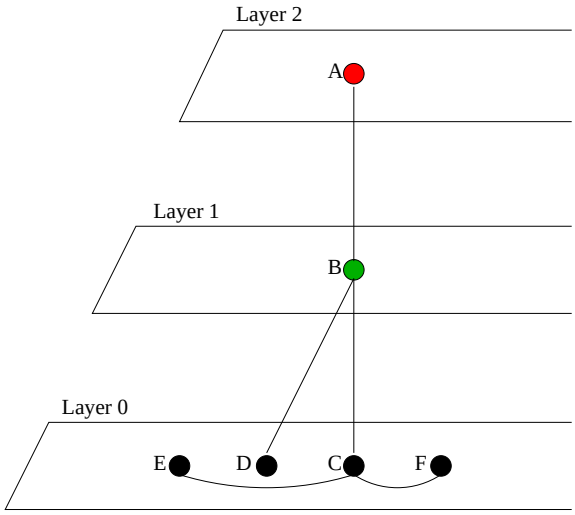

Figure 3.4: Reliable ALM. Logical topology.

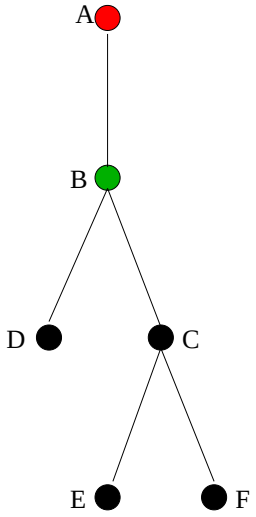

Figure 3.5: Reliable ALM. Hierarchical topology.

- ...
- bin i: $2^{i-1} - 1 - 2^i - 1$

Each bin correspond to a layer and a host belongs to layer i if its age is higher than $2^i - 1$. Therefore, each host periodically checks its rank and its time spent in the group. If necessary, it will join an upper layer and once done will leave the layer it was into.

When a new member wants to join the diffusion group, it will obtain a list of potential peer to connect to. This list can be provided by a Rendezvous-Point (RP) and then the peer will contact the members of the list and find the closest one. Since this algorithm focus on the reliable aspect of the tree, the distance metric (minimum number of hops, minimum end-to-end delay) will depend on the secondary purpose of the implementation. This step is represented in figure 3.6 where G wants to join the diffusion group. Once G has elected the closest neighbor (among the most reliable ones) it adds itself to L_0 and to the multicast tree (figure 3.7).

3.4 HOMP: Hybrid Overlay Multicast Protocol

HOMP [22] developed by Nankai University focus on good delay performance and scalability. As the name suggests, the protocol is both based on a mesh and tree networks.

HOMP creates two families of members: core members and attached members. Core members form the mesh network. To each member of the mesh can be attached a tree (which

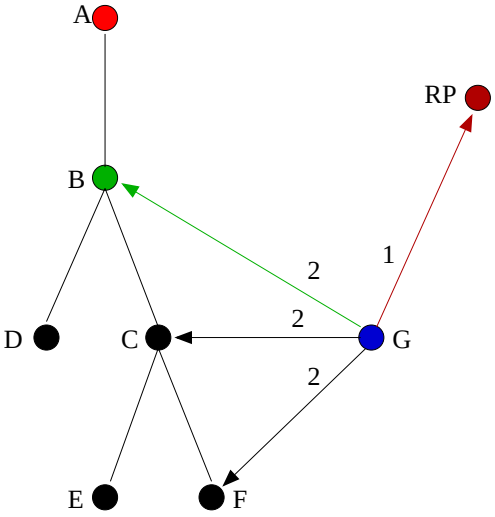

Figure 3.6: Reliable ALM. Joining procedure.

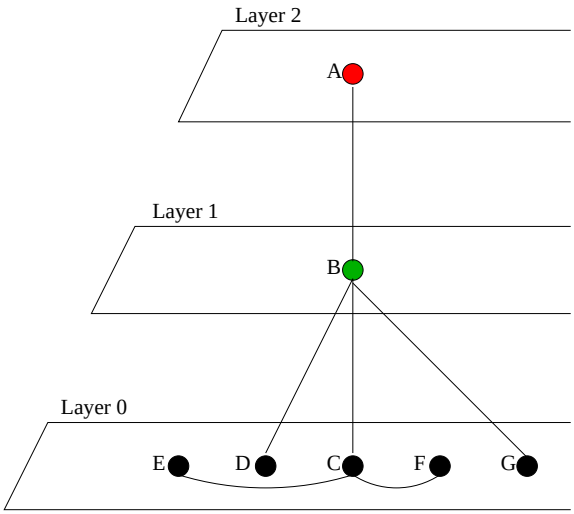

Figure 3.7: Reliable ALM. New classification.

root is the core member) containing attached members. An attached member can only be attached to one tree. This is illustrated in figure 3.8 where A, B, C, D, and E are core members and all other nodes are attached members.

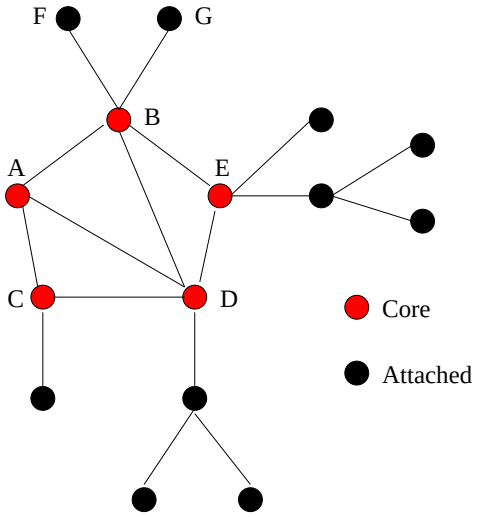

Figure 3.8: HOMP. Hierarchical topology..

Each member is limited by its bandwidth. Therefore the number of neighbors will be function of this bandwidth. It is obvious that the data path will depend of the source. Indeed, the core mesh network allows different path for the same data. Since the protocol

focuses on minimum delay, the shortest end-to-end delay path will be chosen. Figure 3.9 illustrates data path when a core member (D) is the source. In this case, all core members are directly connected to D which ensure minimum delay to each root of every tree. In figure 3.10, the source is an attached member (F). In this case, all members of F's tree, will receive the data directly from F (if they are its parent or children) or will receive it after the root (B) forwards it (if they are its cousins).

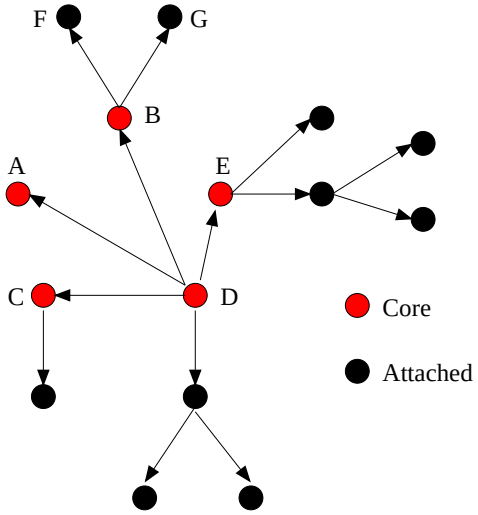

Figure 3.9: HOMP. Multicast illustration (for a core member).

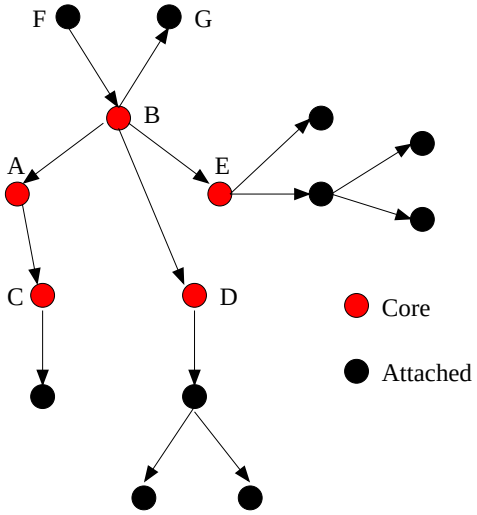

Figure 3.10: HOMP. Multicast illustration (for an attached member).

When a new member (NM) wants to join the diffusion group, it first contacts a Rendezvous Point (RP). The RP replies with the address of one of the members, x. NM contacts x to obtain a list of all core members. If x is a core member, it can replies immediately otherwise it will reply with the address of the root (y) of the tree it belongs to and NM can ask y for the list. Once the list is obtained, NM can proceed to check whether or not he is able to become a core member. To do so, the number of core members must be less than the maximum authorized by the protocol and the bandwidth of NM must be large enough to fill the requirements of being a core member. Two scenarios are possible:

1. NM becomes a core member.

2. NM becomes an attached member.

The joining procedure depending on this two cases is explained in detail in [22].

3.5 OMTP: Overlay Multicast Tree Protocol

OMTP [11] has been created by the University of Hong-Kong. The main purpose of the protocol is to speed up the joining process but it also takes into account the bandwidth of clients and tries to minimize end-to-end delay.

The basic idea of the protocol is to match IP addresses with a common prefix in the same cluster in order to minimize the end-to-end delay. In the same time, since the number of messages exchanged is minimal this will minimize the join procedure time.

When a new host wants to join the tree, it will send a request to the Rendezvous Point (RP) which has a table containing the couple {IP address, Prefix} of all active hosts belonging to the diffusion group. According to this table, the RP will try to match the address of the incoming member with the longest prefix it can find in the table and will send the answer to the new member. The new member will then send a request to its potential parent in order to obtain the address of its children and will send join request to all of them to select its future parent. The parent will be selected based on the minimum end-to-end delay between it and the new member. In order to respect the bandwidth, each member informs its parent if it can handle more connection(s). Therefore, only member with sufficient bandwidth will be able to receive new child(ren).

Host	IP address/Subnet portion
A	203.69.12.22/24
B	202.14.181.3/23
C	218.102.23.16/27
D	143.89.14.34/20
E	212.227.34.2/24
F	143.89.40.2/20
G	147.8.181.31/22
H	147.8.182.33/22
I	147.8.182.92/22

Figure 3.11: OMTP. IP addresses and subnet of member hosts.

Figure 3.12 illustrates the joining process where J, with the IP address 147.8.181.12) wants to join the tree. J begins to ask the RP for a potential parent, i.e. member of the tree which IP address has the maximum length match prefix. RP checks its table (figure 3.11) and will reply with G's address. J will then ask G for its children. In the final step of the joining process, J will send a request to G, H and I and will finally send a join message to I since it is the closest member.

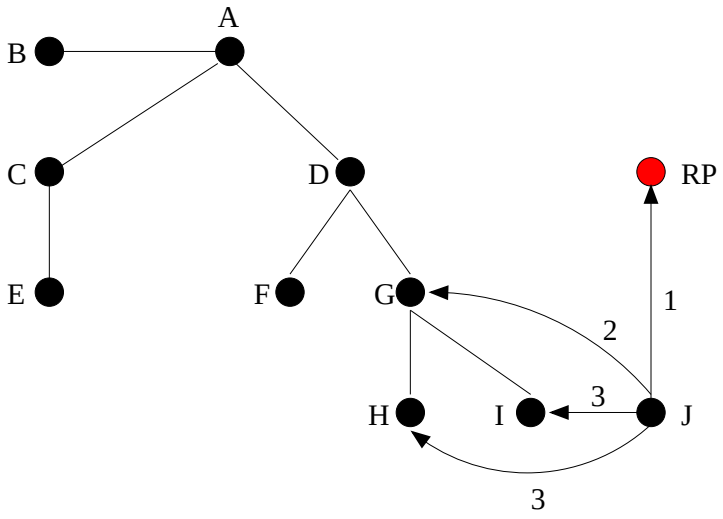

Figure 3.12: OMTP. Joining procedure.

3.6 VRING

VRing [16] has been developed by the University of Illinois in collaboration with the National Center for Supercomputing Applications. It suggests to organize the overlay multicast group in a ring rather than a tree.

Ring approach allows to reduce average node degree and to consume less bandwidth than traditional tree. In addition, a ring provides a better mechanism to deal with early departure and therefore maintains the cohesion in the group more easily. VRing is based on connected components. A connected component can be:

- an isolated group member
- a group with at least two members connected in a ring

Each connected component has a unique leader (itself for a single group member or a member of the ring in the other case). Each node also needs to register its neighbors' identity and its leader.

Each leader has to register to the Rendezvous Point (RP). Once done, the RP answer with the list of all other leaders so that it will be possible to join the different connected components. If leader u receives the list, it selects the closest [1] other leader, v, and send him a join message containing its neighbors. Basically, four different cases can occur depending on the leader type:

[1] The distance metric is not defined by the protocol therefore the programmer can choose the one that fits better his needs.

- Node-Node: u and v are isolated group members (figure3.13).
- Node-Ring: u is an isolated group member while v is leader of a ring (figure3.14).
- Ring-Node: u is leader of a ring while v is an isolated group member (figure3.15).
- Ring-Ring: u and v are leader of rings (figure3.16).

Figure 3.13: VRING. Node-Node leaders connection.

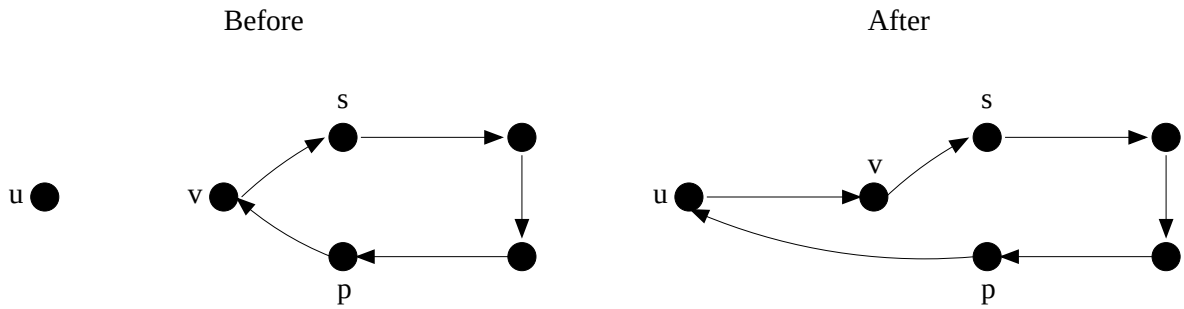

Figure 3.14: VRING. Node-Ring leaders connection.

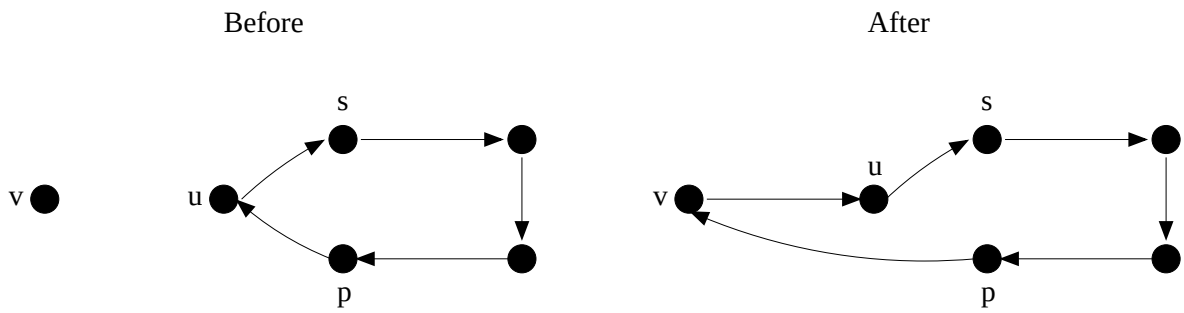

Figure 3.15: VRING. Ring-Node leaders connection.

Another key aspect of the leader member is that each leader can only communicate with only one other leader. Consequently, u and v can communicate together only if they are both free when u sends its request to v. The final ring is finally created when only one leader remains.

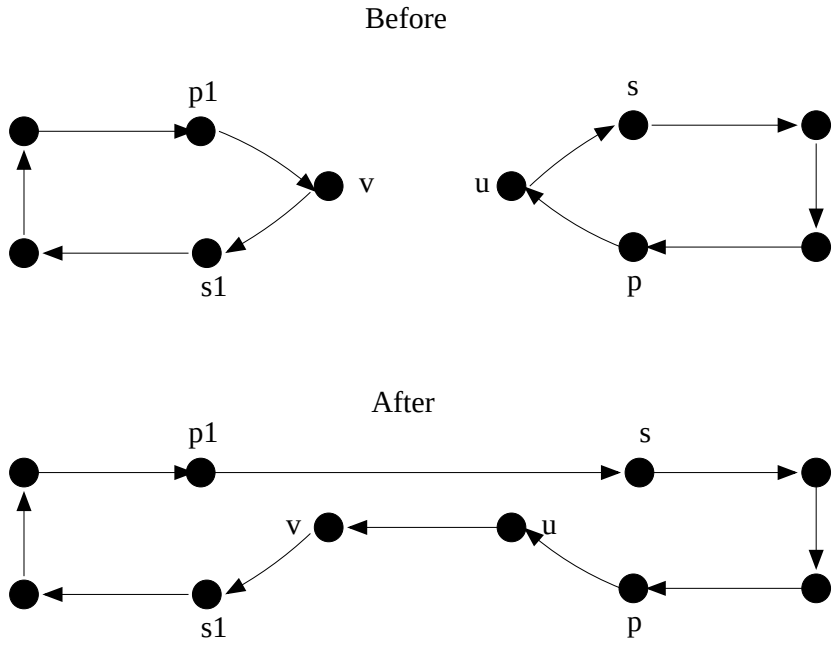

Figure 3.16: VRING. Ring-Ring leaders connection.

In order to reduce the overlay delay, the algorithm creates a spare ring in addition of the previous ring. For a ring containing N members (member 0 being the leader), spare links are created between node i and node $(i + \lceil\sqrt{N}\rceil) mod N$ as shown in figure 3.17 for $N = 8$. The original ring is $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 0$ and the spare ring is $0 \rightarrow 3 \rightarrow 6 \rightarrow 1 \rightarrow 4 \rightarrow 7 \rightarrow 2 \rightarrow 5 \rightarrow 0$.

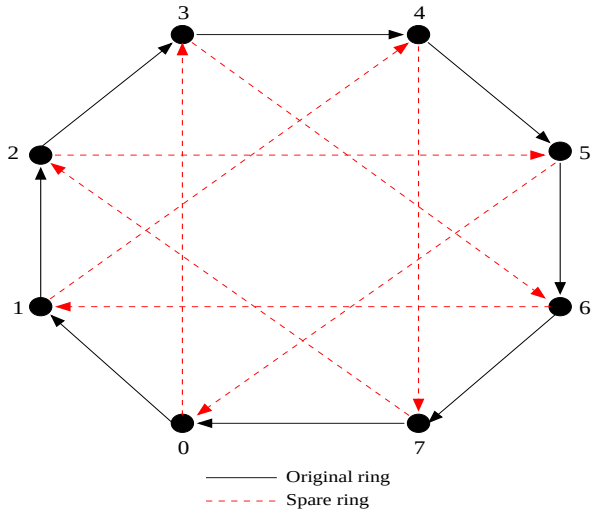

Figure 3.17: VRING. Original ring and spare ring.

When a source sends a message, it sends it on both the regular ring and the spare ring. When a member of the ring receives a packet, it first checks from which ring it comes from. If it comes from the regular ring, the node forwards the packet on both rings. On the contrary, if it comes from the spare ring, it forwards it only on the spare ring. This is illustrated on figure 3.18 where node 0 is the source. As one can see, some nodes (1, 2 and 3) receive the same message more than once. Therefore there is a need to implement a duplicate verification method which detail is given in [16].

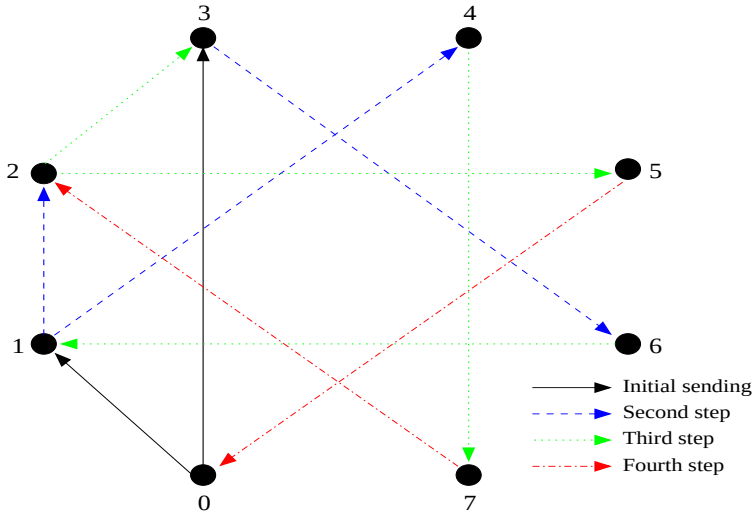

Figure 3.18: VRING. Multicast illustration.

3.7 ZIGZAG

ZIGZAG [20] is a single-source media streaming protocol developed by University of Central Florida. Member of the session are organized in a tree which root is the media server.

The common structure looks like NICE (section 3.2). Indeed, members are organized in clusters and leaders of clusters are then promoted to the higher layer. However, the similarities stop here. The highest level in ZIGZAG has one cluster instead of one single host and other differences appear as we will see later. The basic hierarchy is illustrated in figure 3.19 where we can see that the highest layer contains the source S which will obviously be a leader in all other layers.

All members belong to layer 0, L_0, and all peers in all layers are separated in cluster which size vary between k and $3k$ (where $k > 3$). The highest layer, L_{H-1}, is the only exception with a number of host between 2 and $3k$.

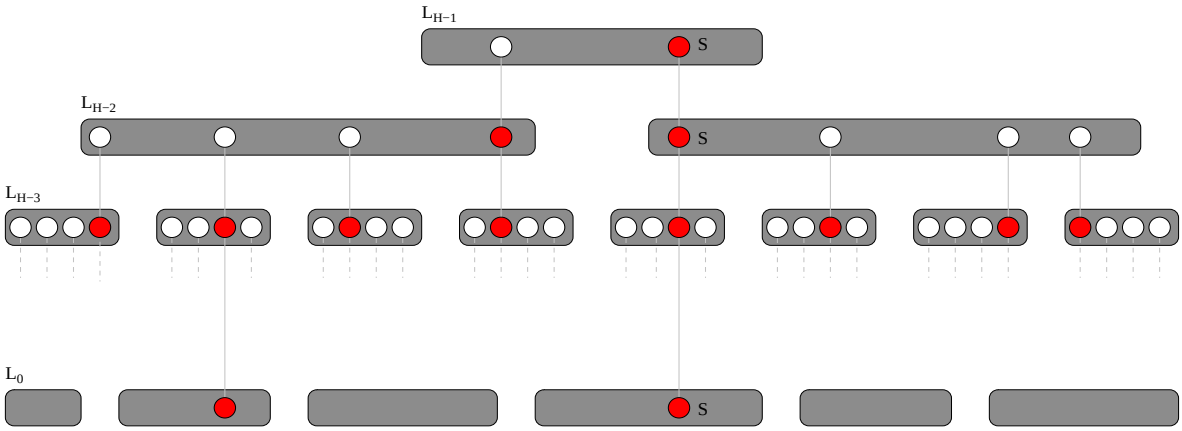

Figure 3.19: ZIGZAG. Hierarchical topology.

Based on this hierarchy, the protocol establishes some rules to define the data path. These rules are the following [20]:

Rule 1. "A peer, when not at its highest layer, cannot have any link to or from any other peer."

Rule 2. "A peer, when at its highest layer, can only link to its foreign subordinates."

Rule 3. "At layer $j < H - 1$: since non-[leader] members of a cluster cannot get the content from their [leader], they must get it somehow. In [ZIGZAG] multicast tree, they get the content directly from a foreign [leader]."

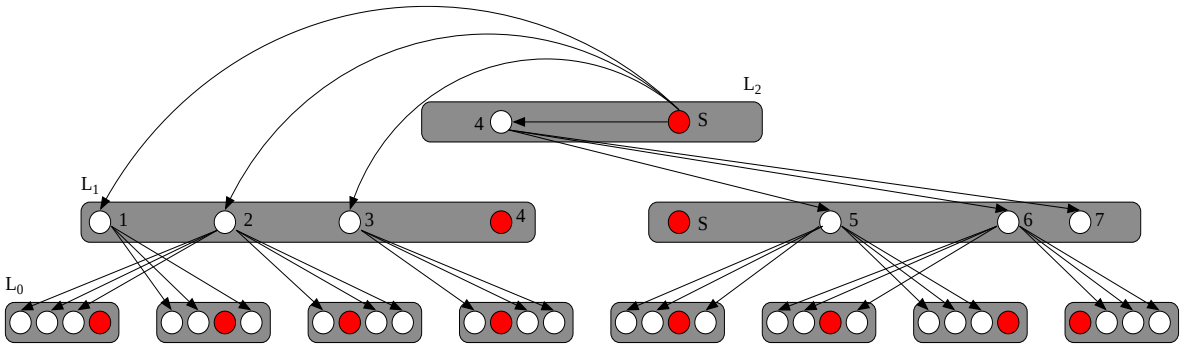

Figure 3.20: ZIGZAG. Multicast illustration.

Figure 3.20 illustrates the different rules. In layer 0 and 1, peer 4 does not have any link to nor from him (rule 1). Peer 4 in layer 2 is sending message to peers 5, 6 and 7 which do

not belong to its cluster in layer 1 (rule 2). Finally, in layer 0, all peers receive messages from a leader of another cluster in layer 0 (rule 3).

When a new member, NM, wants to join the diffusion group, it has to be directed to the correct cluster in layer 0, i.e. the one which will minimize the end-to-end delay between the new member and the server. To do so, ZIGZAG defines two variables for a member X of the multicast tree:

- Reachable(X): if it exist a path between X and a layer 0 peer.
- Addable(X): if it exist a path between X and a layer 0 peer whose cluster's size is in the interval $[k, 3k-1]$.

if X is a leaf **then**
  Add P to the only cluster of X
  Make P a new child of the parent of X
else
  if $Addable(X)$ **then**
    Select a child $Y|Addable(Y)$ and $D(Y) + d(Y, NM)$ is min
    Forward the join request to Y
  else
    Select a child $Y|Reachable(Y)$ and $D(Y) + d(Y, NM)$ is min
    Forward the join request to Y
  end if
end if

Figure 3.21: ZIGZAG. Joining algorithm.

The joining procedure is given by the pseudo-code of figure 3.21 [20] and is illustrated in figure 3.22. NM begins to send a message to the server S. Then it will send a request to all children of S, i.e. members 1, 2, 3 and 4 and will select among them the closest one (in this case 3). Finally it will send a message to the children of 3 (leaves of the multicast tree) and will be added in the cluster which leader is 4.

3.8 MeshTree

MeshTree [18] has been created by the University of Kent. As the name suggests, the algorithm uses both mesh and tree technology.

The overlay network is a degree-bounded mesh based on two structures:

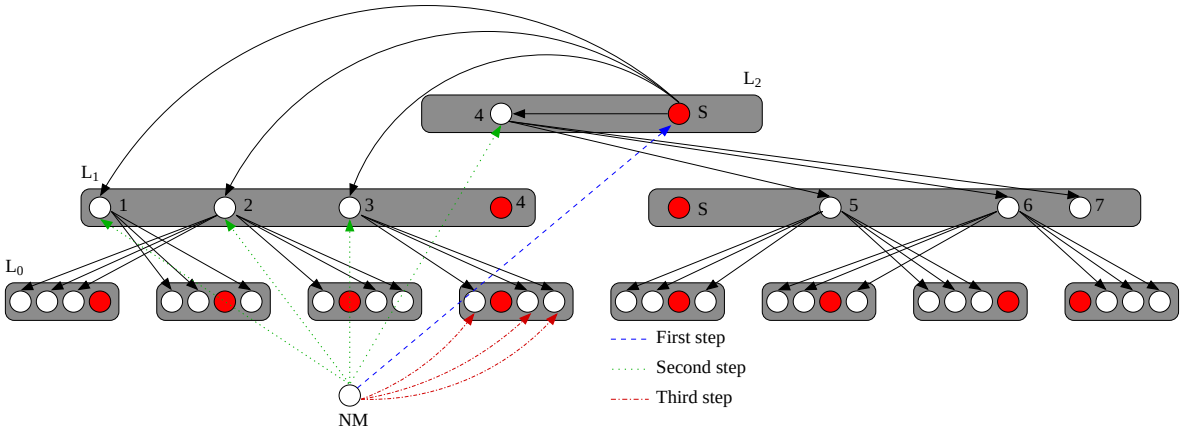

Figure 3.22: ZIGZAG. Joining procedure.

- a backbone-tree
- a delivery-tree

The backbone tree links nodes close to each other in order to reduce the communication costs. On top of this tree, links are added to build the mesh network. The protocol focused on a single-source multi-receiver topology, as it is the case with ZIGZAG (section 3.7). Both trees (backbone and delivery) are rooted at the source, S. An example of the topology is given in figure 3.23.

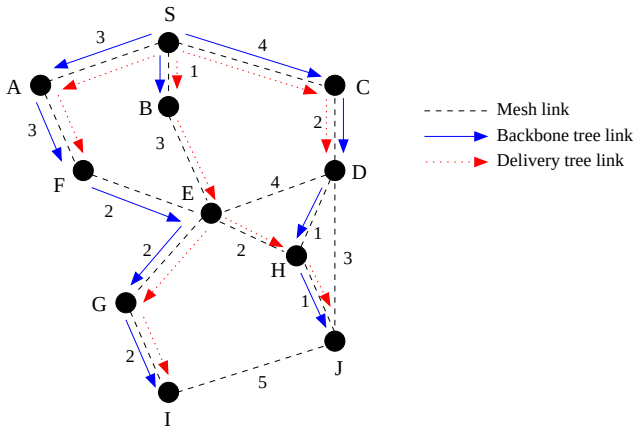

Figure 3.23: MeshTree. Hierarchical topology and multicast example.

In order to suits nodes' bandwidth limitation, each one of them is degree bounded. Therefore there is a maximum number of possible connections.

For the joining process, as in the other algorithms, the protocol deploys a Rendezvous

Point, RP. A new member, NM, will contact the RP in order to obtain a small list of members as well as the root address. Among this members, the NM selects a fixed number of members limited by its fan-out and send them join messages. When receiving the first positive answer, NM will be added to the backbone and delivery tree by its new father. Other members accepting NM will create links for the mesh network. This is illustrated in figures 3.24 and 3.25 where one can see that NM will finally be connected to A, D and J (assuming that it will not receive a positive answer from G).

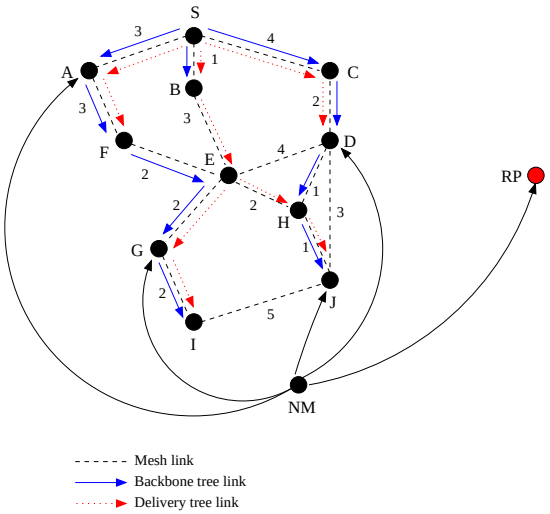

Figure 3.24: MeshTree. Joining procedure.

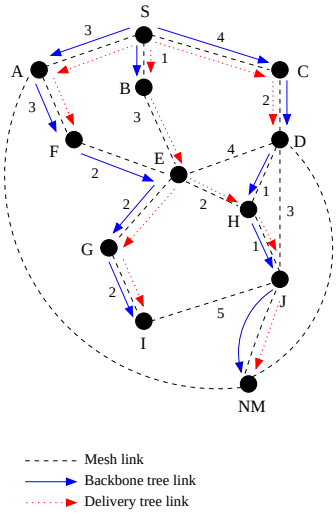

Figure 3.25: HOMP. New hierarchical topology.

Once the new member has joined the mesh, there is an improvement process explained in [18].

To define the delivery tree, each node maintains a routing table with the node to contact to reach the source. By exchanging messages with its neighbors, a node can determine the better (i.e. shortest) path to reach the source. Once done, and by reversing the procedure, the delivery-tree is obtained.

3.9 DS-P2P: Dominating Set based Peer-to-Peer Protocol

DS-P2P [1] has been developed by the University of Ottawa. It suggests a mesh network based on participants' geographical position and respecting each node bandwidth.

When a host wants to join the diffusion group, it registers to a Rendezvous Point, RP. The RP makes a list of possible edges and creates the backbone of the mesh based on the

minimum distance edges [2] (but including all nodes). Once the backbone is constructed, the *RP* greedily constructs the mesh based on the minimum distance edges that were not included in the first phase and respecting the free degree of each node. Given the distance matrix in figure 3.26, the corresponding mesh network is given in figure 3.27 (assuming a degree of 3 for each node).

	B	**C**	**D**	**E**	**F**	**G**
A	1	2	3	3	4	5
B	x	2	4	3	4	3
C	x	x	2	2	3	5
D	x	x	x	1	2	4
E	x	x	x	x	1	3
F	x	x	x	x	x	2

Figure 3.26: DS-P2P. Distance matrix.

Once the mesh network is created, the protocol will decide whether a node is promoted as a gateway or not. Gateway nodes will forward the messages while non gateway node will be the end of the data path. Initially, a node is marked as gateway if it has two unconnected neighbors. This is illustrated in figure 3.28. In order to limit the number of gateways, DS-P2P establishes two rules. But before explaining them we need to define the open neighbor set (ONS) and close neighbor set (CNS). For a node v, the ONS is defined by $N(v) = \{u | (v,u) \in E\}$ and the CNS by $N[v] = N(v) \cup \{v\}$. We can now enumerate the rules to reduce the cardinality of nodes:

Rule 1: If v is a gateway, for two nodes u and v, if $N[v] \subseteq N[u]$ and if key(v)< key(u) then change v as non-gateway.

Rule 2: If v is a gateway and u and w are two gateways neighbors of v, then if $N(v) \subseteq N(u) \cup N(w)$ and key(v)< key(u), key(w) change v as non-gateway.

These two rules are illustrated in figure 3.29 and 3.30. As we can see, there is no change after rule 2. This is due to the fact that C is no more a gateway when rule 2 is applied to the mesh.

The last step of the protocol consists in establishing the data path. Each source will send a message to reach every members of the diffusion group. Since a mesh is used, the protocol has to ensure that no loops are created. To do so when receiving duplicate messages, the node will create an ignore set (I). If a non-gateway node receives a message, it will not forward it. If a gateway v receives a messages from its neighbor u, then it will forward it to a set F

[2] The set of edges is defined as E.

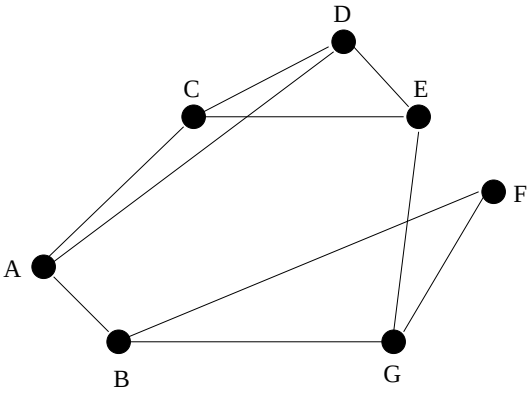

Figure 3.27: DS-P2P. Mesh topology.

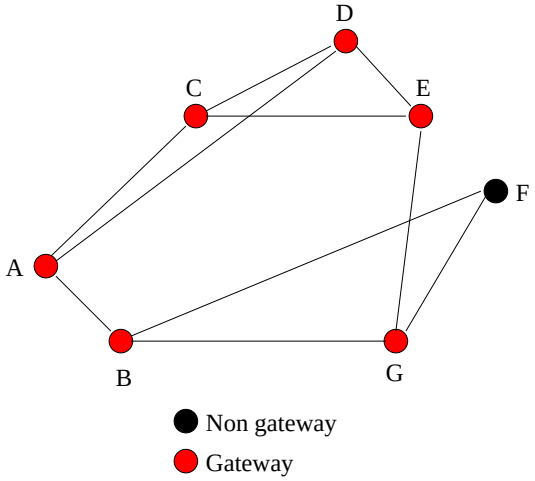

Figure 3.28: DS-P2P. A node becomes gateway if it has two unconnected neighbors.

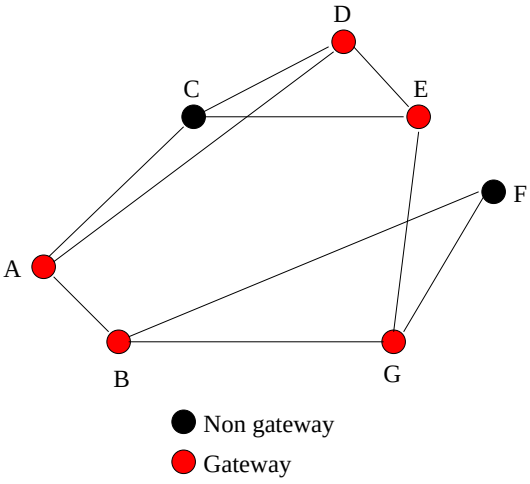

Figure 3.29: DS-P2P. Rule 1.

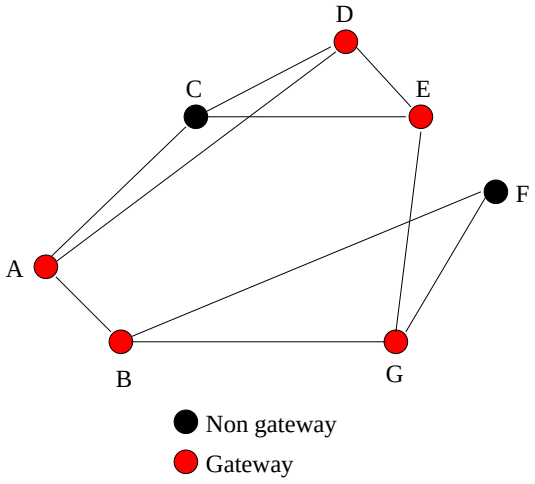

Figure 3.30: DS-P2P. Rule 2.

where $F = N(v) - N(u) - I(v)$. This is illustrated in figures 3.31 and 3.32 where A and E are the respective sources.

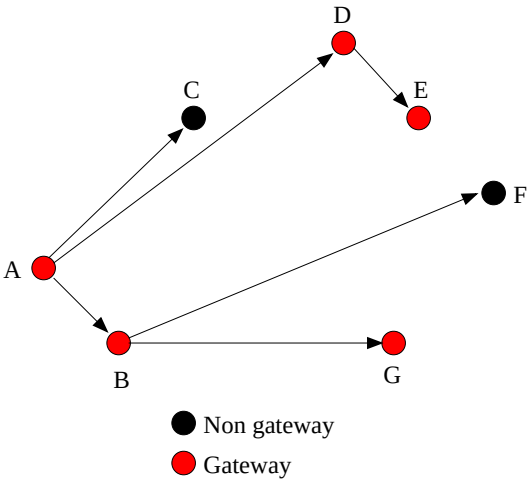

Figure 3.31: DS-P2P. Multicast example 1.

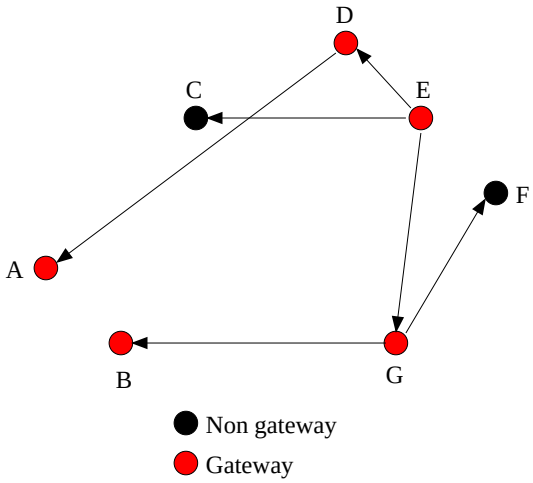

Figure 3.32: DS-P2P. Multicast example 2.

3.10 Conclusion

This chapter illustrated some technologies to build efficient overlay networks. The finality of all of them is to build a multicast tree usually focused on the best end-to-end delay between a source and the receivers. The construction of this tree will greatly vary depending on the main purpose of the protocol as we have seen in the previous sections. Some begin to construct a mesh network and then build a tree above it, other begin directly with a tree. The mesh network usually offers a better reliability since if a node fails others can still forward messages. We did not consider the messages' costs when we explained the different protocols. However, it is obvious that exchanging messages will consume bandwidth and that is the reason why [16], for instance, focus on decreasing these messages' exchanges. Some protocols are respectful of each host's bandwidth by limiting the number of connections they can do. This degree constraint permits to eliminate the risk of bottleneck.

Chapter 4

Simulation environment

4.1 Introduction

Now that we have seen theoretical aspects of Application Layer Multicast and some of its applications in different protocols, we will describe briefly the simulation environment we worked with: *OPNET*.

Working with this software was a prerequisite of this master thesis. Unfortunately, with the education license comes really little help. It is as well quite hard to find good documentation about it since it is used mainly by industry. However some sources of other projects are available online and thanks to them [13, 17, 9], we were able to find appropriate functions for our project.

In this chapter we will explain what the main purposes of OPNET are. We will then mention some of the key features which simplify implementation of new model and give the steps to follow to create a new project with this software. We will finally say a word about the main development environments we used. Indeed, OPNET comes with many graphic tools to make development easier.

4.2 OPNET

OPNET modeler is a network simulation tool. It has two main purposes [12]:

1. "To study system behavior and performance"
2. "To deliver a modeling environment to end-user" or in other words to create a customized model available for a particular end-user.

Our use of OPNET, as one will see later, will combine these two aspects. Indeed, we will have to develop a model first and then to study its performance.

4.2.1 Key features

OPNET offers many useful tools to define networks. The list behind is not exhaustive but describes some of the main features OPNET provides. It would be too long to develop everything but details can be found in [12].

- Systems consist of objects, each having its own attributes. Objects belong to classes which will determine their behavior.
- Hierarchical model in compliance with the real Internet model (and the network hierarchy).
- Graphical model wherever it is possible to make development easier.
- Flexible models that can be customize to suit one's need. This functionnality allowed us to define a special application layer behavior.
- C-like programming allows to model every level layer of the TCP/IP model or to deploy easily new models.
- Integrated analysis tools in order to calculate statistics for some interesting parameters.
- Traffic can be added easily. Some specific protocols are already modeled to generate traffic. This is the case for FTP for instance.
- Packet format can be defined and used depending on the application.

4.2.2 Workflow

Developing a model with OPNET follows five major steps. It is an iterative process as it is shown in figure 4.1. The project we realized had to follow these phases. But we have to mention that it can still be improved by refining the model we already implemented. We will see later why and how.

4.2.3 Specification

The creation of the model (4.1) has to be done at many levels. Specifications are made through the use of specialized editors. Each editor can be seen as a level of specification in the design hierarchy. Indeed, from the most abstract level, which would be the network, to the most explicit one, which would be the process, OPNET offers specific editors. Editor's hierarchy can be seen in figure 4.2 taken from [12].

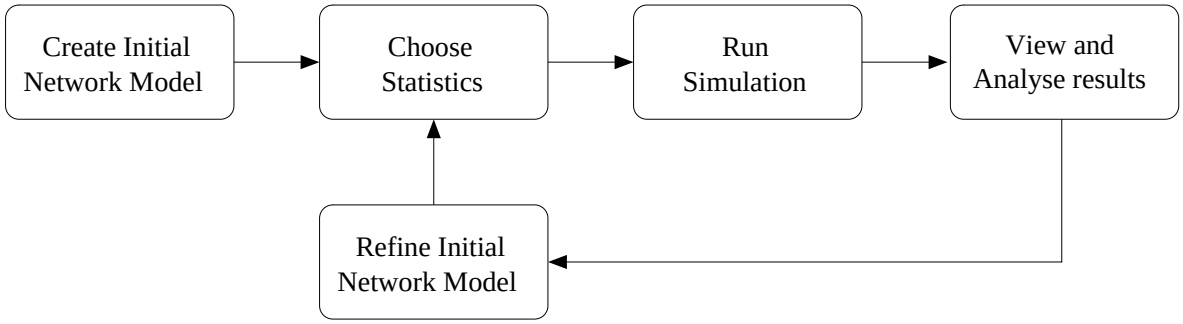

Figure 4.1: OPNET. Simulation project cycle.

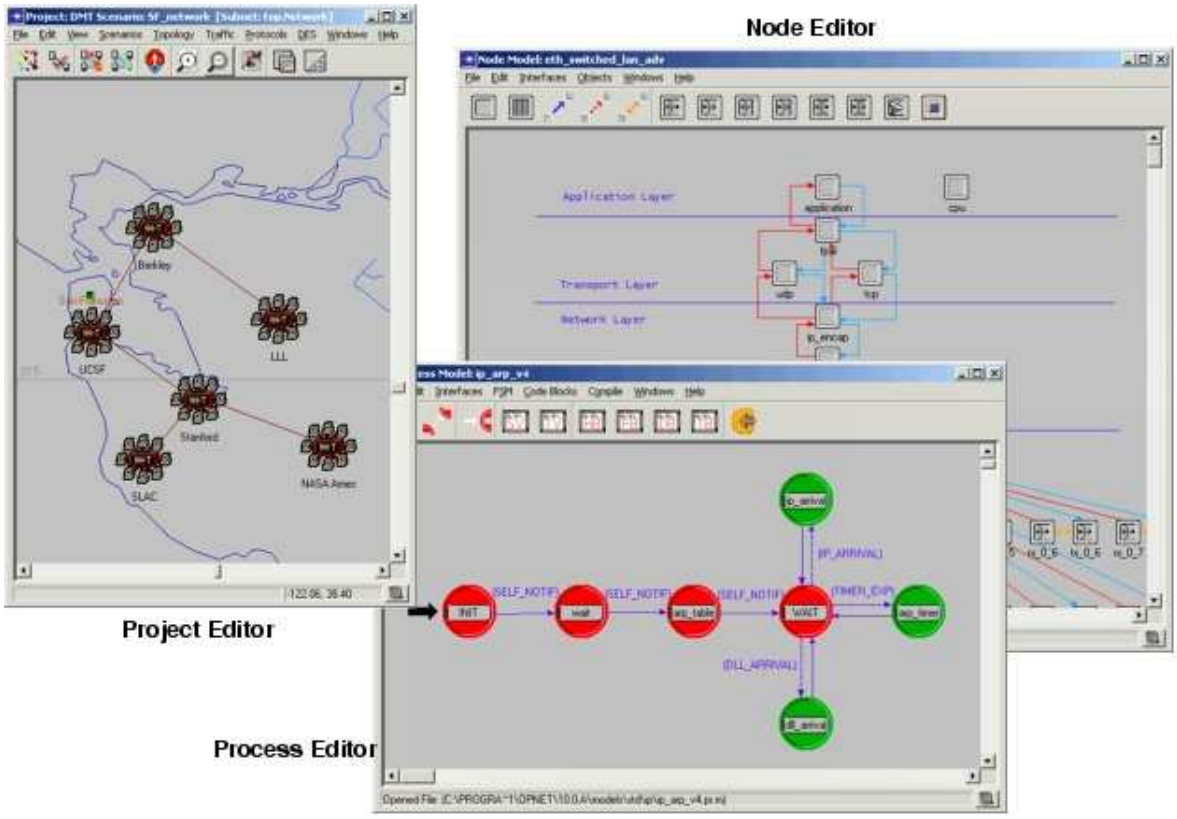

Figure 4.2: OPNET. Graphical Editors for Network, Node and Process models

The network editor is a graphic editor allowing designing the network. It offers a lot of pre-implemented models, such as computers, routers, IP clouds, ethernet links. Maps are also available so that it is easy to locate elements on it. For our project, we deploy between fifty and five hundred hosts on the U.S.A. map. This is illustrated for three hundred hosts in figure 4.3. Figure 4.4 shows what one can see when we zoom on a subnetwork. In our project, each subnetwork contains two other subnetworks in which are present an xDSL router and a computer.

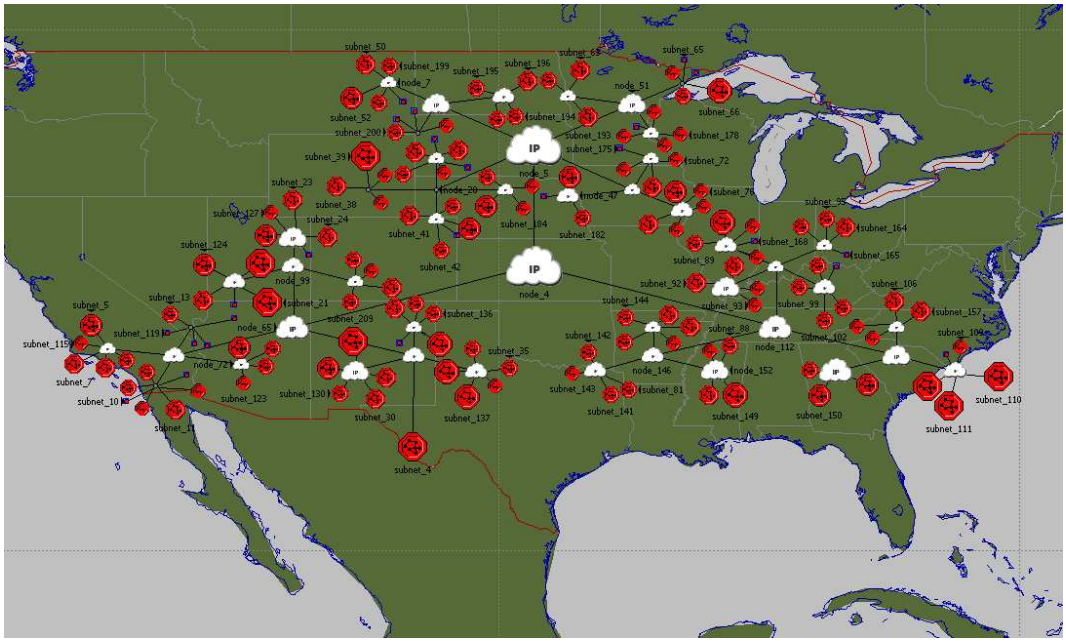

Figure 4.3: OPNET. Topology example with 150 subnetworks and 300 hosts.

The node editor reflects the structure behind each element of the network model. A node models contains modules which usually implement a layer of the TCP/IP model but not only. Indeed, in order to simplify some procedures, OPNET implements "between-layers" protocols. Figure 4.5 shows the node model we used in our project. As we said, between the transport and application layers one can see a "between-layer" module, *tpal*, which is an abstract module allowing to deal with TCP and UDP with the same procedures.

The process editor allows creating finish state machine which are behind each module created in the node editor. Each state is then implemented in a C-like language. Some example will be given later. In our project, we had to design a Rendezvous Point and different hosts. All of them are at least separated in two processes – a receiving and a sending process. In fact, to have better performances, we have many sending processes but we will see that later. Figures 4.6 and 4.7 illustrate the Rendezvous Point receiving and sending processes of one of the protocols, NICE, we modeled.

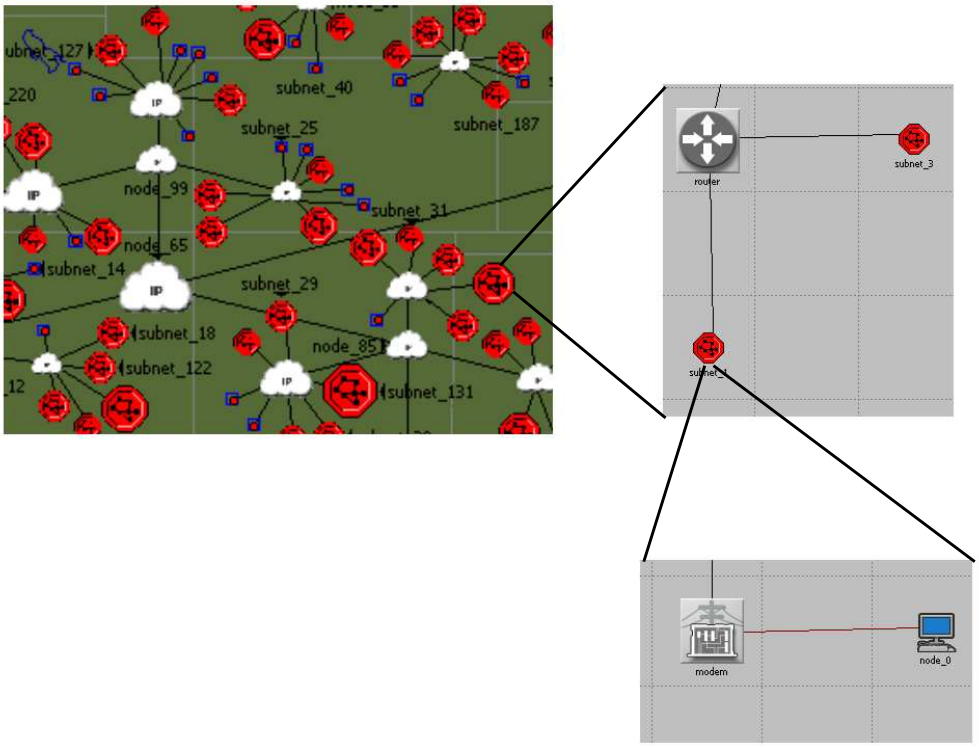

Figure 4.4: OPNET. Zoom on subnetworks.

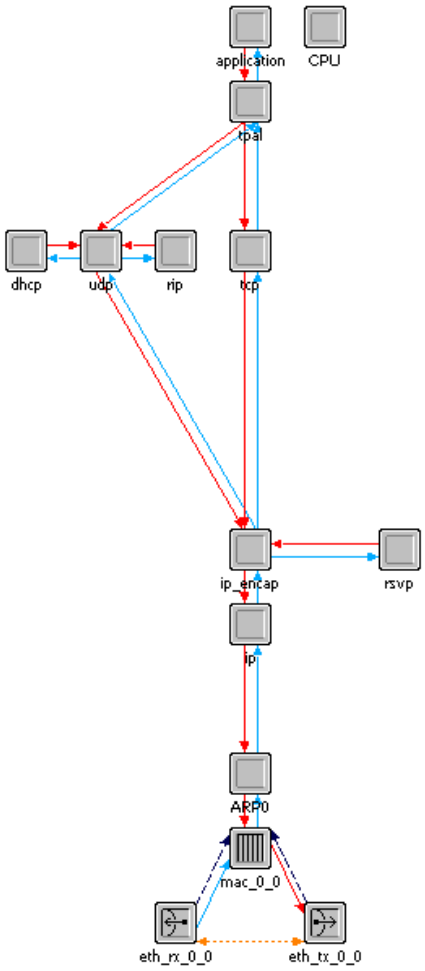

Figure 4.5: OPNET. Node editor.

All models, and whatever the editor they are created in, can be easily reused. It is really easy to integrate models from OPNET libraries. For example in our project, we define the network topology using computers, DSL modems, routers, IP clouds, T3 and fast ethernet links. Behavior of all elements is defined by OPNET, except for the computer in which we had to modify the application layer. But the modification we had to do is only visible in the process editor for the application layer node.

These three editors are the most interesting for our project. However, they are not the only one available and as we said earlier, a lot of parameters can be customized. This is the reason why OPNET also comes with:

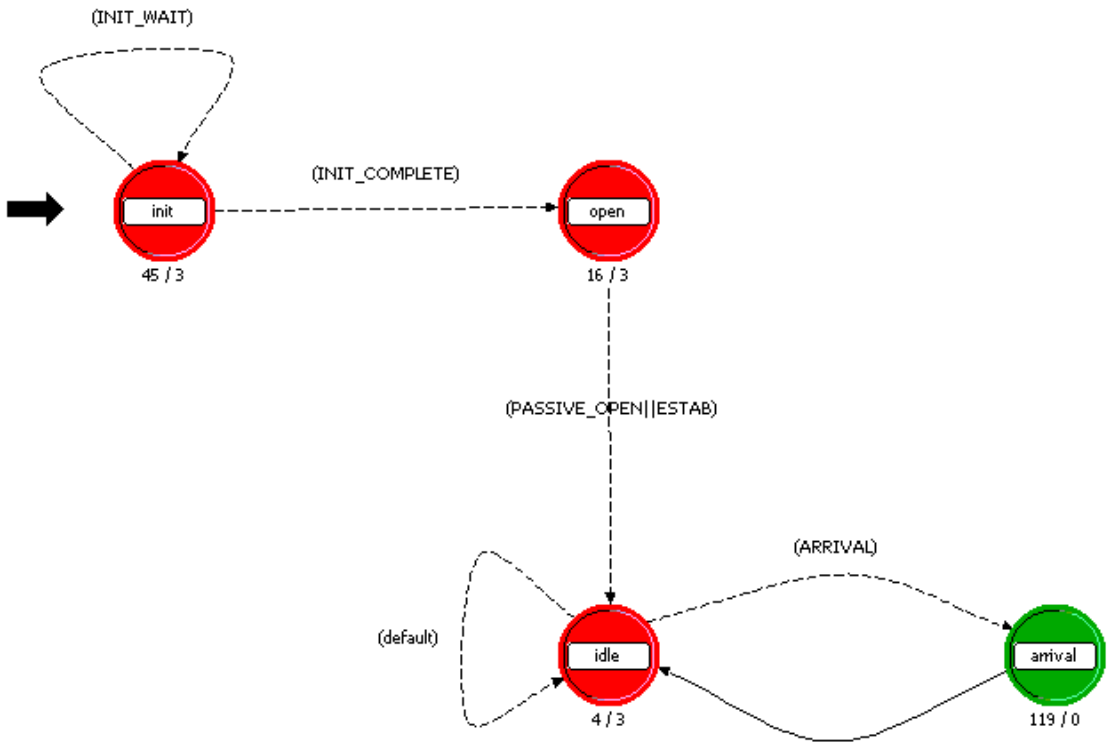

Figure 4.6: OPNET. Rendezvous Point receiving process.

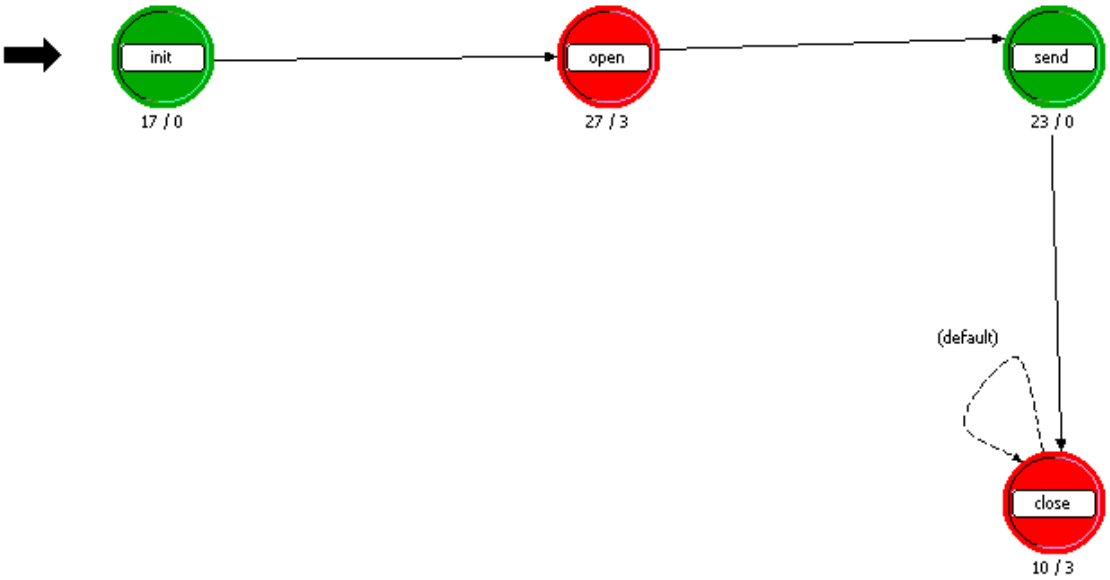

Figure 4.7: OPNET. Rendezvous Point sending process.

- External System Editor to integrate external system definition.
- Link Model Editor to define specific links.
- Packet Format Editor to define protocol specific packets.
- ICI Editor to "create, edit and view interface control information (ICI) format [which] are used to communicate control information between processes" [12].
- PDF (Probability Density Function) Editor to create specific probability distribution.

For more information about those specification editors, the reader should go to [12].

4.3 Conclusion

OPNET is a network simulation software providing many tools to study network behavior. Many models are already available from OPNET libraries and it is easy, for example, to study scalability of a system.

But in our case, even though we used existing models, we had to add new capacity at the application layer. Indeed, since we implemented some of the protocols studied in chapter 3, it was necessary to be able to re-define application layer behavior, but only this one. Therefore we used the classic TCP/IP stack, and we didn't have to focus on any layer underneath the one we were interested in.

Through the use of the network editor, we were able to deploy up to 500 clients all across the U.S.A really easily. We barely used the node editor since we let OPNET deal with the default behavior of TCP/IP but we used extensively the process editor to define new finite state machines to model the protocols we simulated. Next chapter will focus on our simulation work as we will explain in detail what we did for this project.

Part II

Personal work

Chapter 5

Simulations

5.1 Introduction

The following chapters will develop our personal work, in this case the implementation with OPNET of two ALM algorithms: NICE and DS-ALM (section 3.2 and 3.9). We will explain the models we constructed and detail the working process.

The simulations are available for a variable number of peers. The kind of computer and memory limitations are discussed in first section.

The network we created use some specific parameters, like the kind of cable to link two elements or the network elements (routers, modem). This will be discussed in the second section.

Third section will explain the difference between NICE and DS-ALM in terms of architectureand parameters. We will explain why DS-ALM free-degree parameter is a good benefit compare to NICE. We will as well make links with OPNET implementation.

Finally, we will go deeper in the implementation of both algorithms. We will develop the process models of Rendezvous Point and peers. Some similarities will appear but we will see that the implementation behind is completely different. Sending and receiving process are detailed but some processes will just be mentioned as they just have same structure as others and their roles are pretty clear.

5.2 Simulation setup

OPNET modeler is a powerful tool to model and simulate networks. It is one of the most used simulation tool coming with a GUI allowing deploying network easily. It provides accurate results which allow researchers to develop and test new models for different layers of the OSI model. In our simulations, we rewrote the application layer in order to model NICE and

DS-ALM algorithms.

We ran the software on a brand new computer (Intel Xeon 3.00 GHz, 2.00GB of RAM) which allows us to run simulation with up to 500 hosts. Unfortunately, the memory requirements when we are going over 500 hosts are too important. Indeed, depending on the model and the value of some parameters, each host requires between 3 and 6 MB of memory. We simulate the behavior of both algorithms for 50 to 500 workstations.

5.3 Technical considerations

As we mentioned earlier, OPNET allows deploying random topology quite easily. Using this tool, our simulation is based on a tree structure of IP clouds with an average node degree between 3 and 4. Leaves of the tree are subnetworks in which are present xDSL users. We show an illustration in chapter 4, figures 4.3 and 4.4.

The tree is based on the real Internet structure in which different tiers (National, Regional, and Local) are present. Considering this, all leaves are connected to local ISP (tier 3). However, we made an exception for the Rendezvous Point (*RP*) and connected it to a tier 2 cloud in order to make it accessible faster. Figure 5.1 explicits Internet hierarchy and shows in which subnetwork is situated the *RP*.

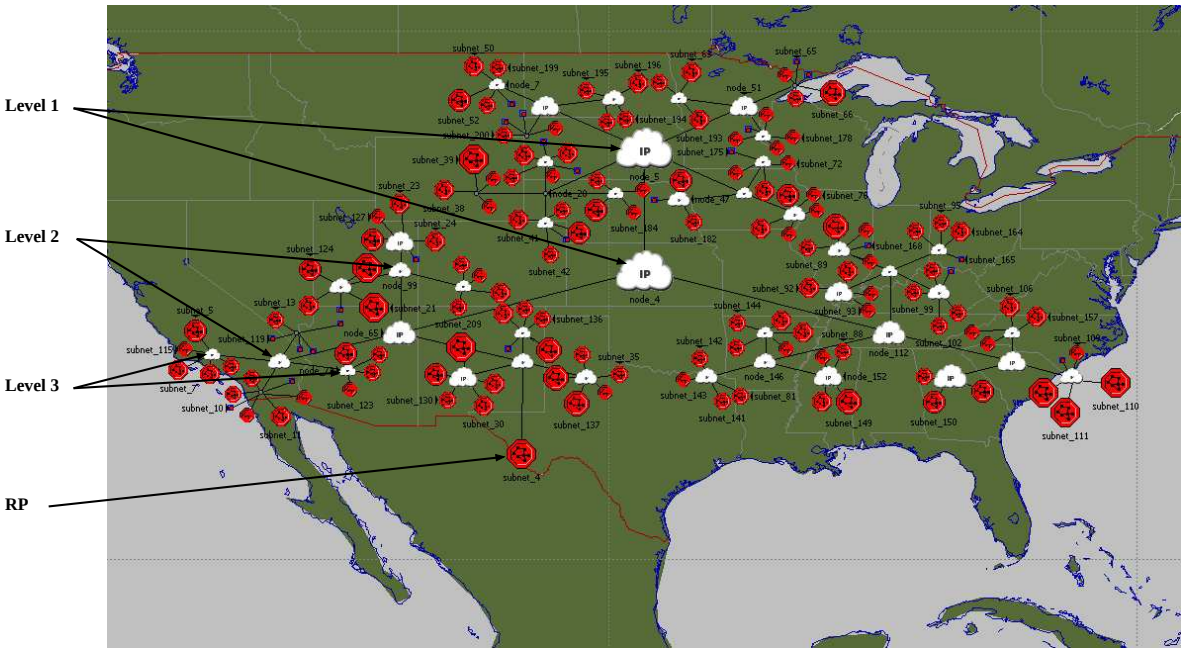

Figure 5.1: NICE and DS-ALM overall architecture.

The different IP clouds are connected using PPP DS3 links (45Mbps). The same kind of links is used between the routers and the clouds and between the routers and the xDSL

modems. On the other hand, end-hosts (clients and server) are connected to their modem/router through a 100Mbps Ethernet cable.

All the traffic generated by the clients is for application purpose only. We will not consider any overload on the traffic because we want to compare raw results of both algorithms. Both applications only use UDP as the transport layer which is quite logical since the purpose is to minimize the end-to-end delay while transmitting packets. Moreover, links in the simulation reflect real links since OPNET implements the correct behavior of UDP/IP layers. However, the amount of traffic generated was far less under real Internet traffic and therefore we did not observe any loss on the links.

5.4 Comparison of NICE and DS-ALM

We decided to compare DS-ALM to NICE [2] which is one of the best algorithms for application layer multicasting. NICE is a "tree-first" ALM approach in which each layer is divided in clusters. Each cluster contains between k and $3k-1$ members and a leader is elected in each cluster. Each leader is allowed to access the higher layer and the root of the tree is finally the leader of the cluster in the highest layer. NICE has only one parameter, k, defined above, which value has been set to $k = 3$ by NICE creators. On the contrary, DS-AML has a parameter we can play with, the degree. In order to be fair, we will compare NICE with a 3-7 degree version of DS-ALM. Indeed, most of the clients in NICE will only communicate with k to $3k-1$ (3 to 8) hosts. But when NICE elected a leader, this leader will communicate with nk to $3nk-n$ members (where n is the number of layers the leader belongs to). That means that for a tree containing 4 layers, and considering that the highest layer will only have one host, the root of the tree will have to send between 9 to 24 messages. Considering that most of the time the number of members within a cluster will be more around $1.5k$ (since the clusters split into two when they reach $3k$ members) the root will have to send more than 13 messages. Therefore, electing a bad leader can result in slowing the all forward message process. On the other hand, since DS-ALM allows its hosts to define how many members he is able to forward the message to, there will not be any risk of slowing down the forwarding process. For more information on NICE and DS-ALM, read chapter 3, sections 3.2 and 3.9.

Both algorithms are using different processes to send and receive messages. This allows a certain degree of multitasking and therefore reduces the delays to forward packets. We only implemented a basic version of each algorithm. Indeed, we did not take into account early departures and all statistics (diameter (time and hops), average (time and hops) and stretch) were recorded once the overlay topology was completely defined.

For NICE, we didn't take into account that the leader of a cluster could change during the

creation of the tree (of course this does not prevent new leader to be created when splitting one cluster into two when the number of hosts exceed $3k-1$). However, this should not have a big impact on the results since the leader only send messages to k to $3k-1$ hosts and all members of a cluster are the ones that minimized the end-to-end delay between them.

5.5 NICE & DS-ALM implementation

As we said in chapter 4, we had to implement the processes of NICE and DS-ALM with the process editor. Since algorithms are different, so will be the processes. Even if some similarities can appear in the state diagrams, the implementation behind will be different. Implementation is written in a C-like language: *Proto-C*.

5.5.1 NICE

As we explained, we divided sending and receiving process, whether it is for the Rendezvous Point (RP) or for the peers. In addition of this division, we added other processes as we will see later.

Rendezvous Point

Sending and receiving processes for the Rendezvous Point are shown in figures 4.6 and 4.7. Red states are called unforced states and green states are called forced states. Each state is divided in three parts:

- Enter executive: code executed when the state is entered.
- Exit executive: code executed when the state is exited.
- Transition executive: code executed when a special event occurred.

The difference between forced and unforced states is that unforced states will wait once enter executive code is executed while forced state will execute directly the exit executive code and the transition to the next state. Another important thing to mention is that some state transitions are only done if a condition is true. This condition is expressed above transitional links which appear in dot instead of being plained.

As one can see, each process has four states. Each one begins with an *init* state but it is forced for the sending process and unforced for the receiving process. The following *open* state is different for the two. Indeed, in the receiver, the state is waiting for a peer to connect, while in the sender it connects to a peer. We will now separate the sender and the receiver to explain the last two states.

1. Receiver (figure 4.6): The idle state is a waiting state waiting for packet arrival from client. In this case it is waiting for the first connection of client to redirect them to the root of the tree. In the arrival state, for each client who is connecting, the *RP* will create a new sending process. This multitasking allows a fast reply to each client in the case where many clients would connect almost simultaneously.

2. Sender (figure 4.7): The sending state is reached once the connection with the client is established. When the information is sent, the sender goes directly in the closing state to close the connection with the client and to destroy the sending process in itself (to free memory).

Peer

The main process of the peer consists of twelve states (figure 5.2). We will give a explanation for each one of them and we will make links to some of the children processes (figure 5.3 to 5.5).

1. *init*. Many variables have to be initialized and we have to ensure that modules underneath the application layer are initialized correctly.

2. *begin*. Since there will be between fifty and five hundred clients we can not waste time by starting them manually. Therefore, each client will begin randomly.

3. *stop*. If an error occurs in the *begin* state, the client will go to this state.

4. *open_RP*. Once the client has begun, it will have to contact the Rendezvous Point (*RP*).

5. *join_RP*. Once the connection is established, the client will send information to the *RP*.

6. *setup*. In order to be able to receive the answer from the *RP*, the client has to set up a listening port to be contacted.

7. *idle*. It is a waiting state.

8. *receive*. Peers will exchange many messages, whether it is for "administrative" or application purpose. For application purpose, different messages exist: heartbeat (or keep-alive) messages, splitting messages, update messages, join message. Therefore, depending on the kind of message a peer will receive it will take appropriate action. The first message each peer will receive will come from the *RP* and will contains the address of the root of the diffusion tree.

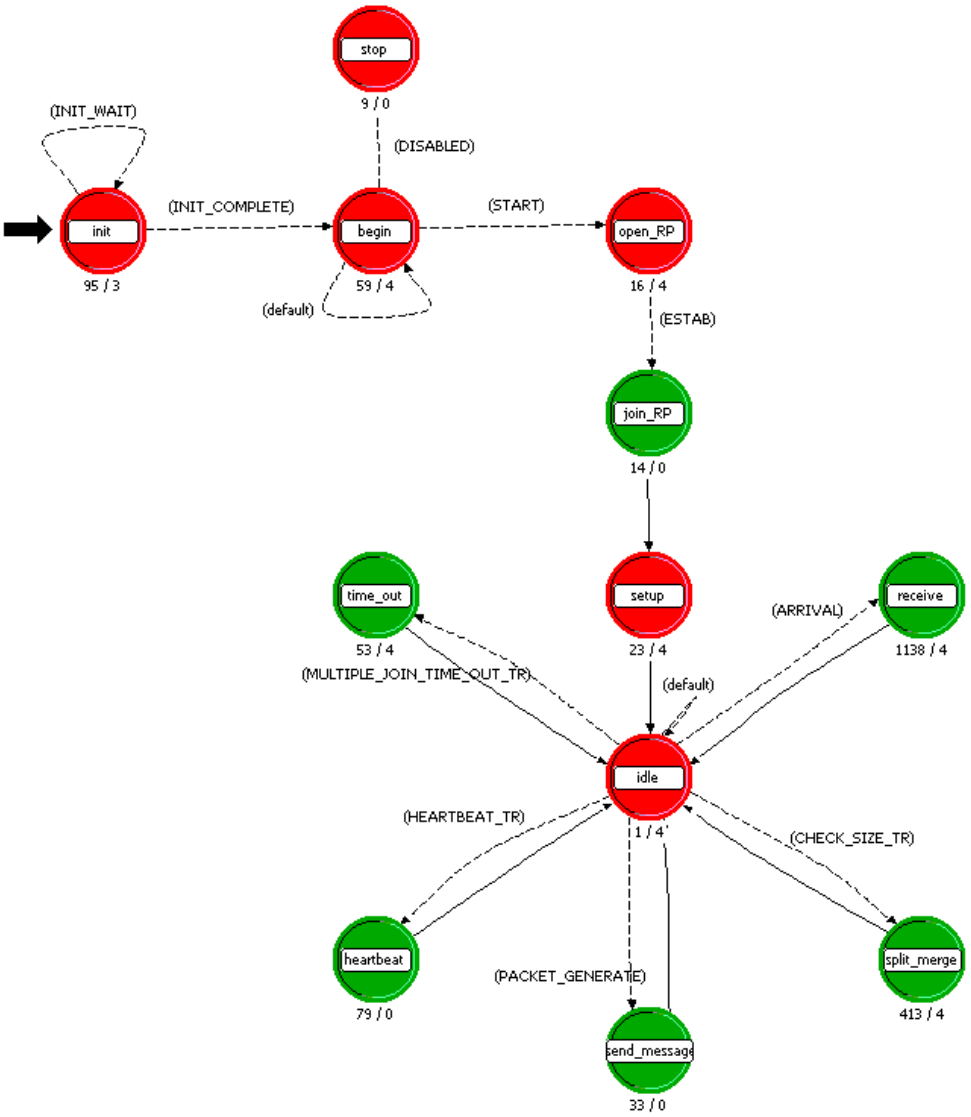

Figure 5.2: NICE. Client's main process.

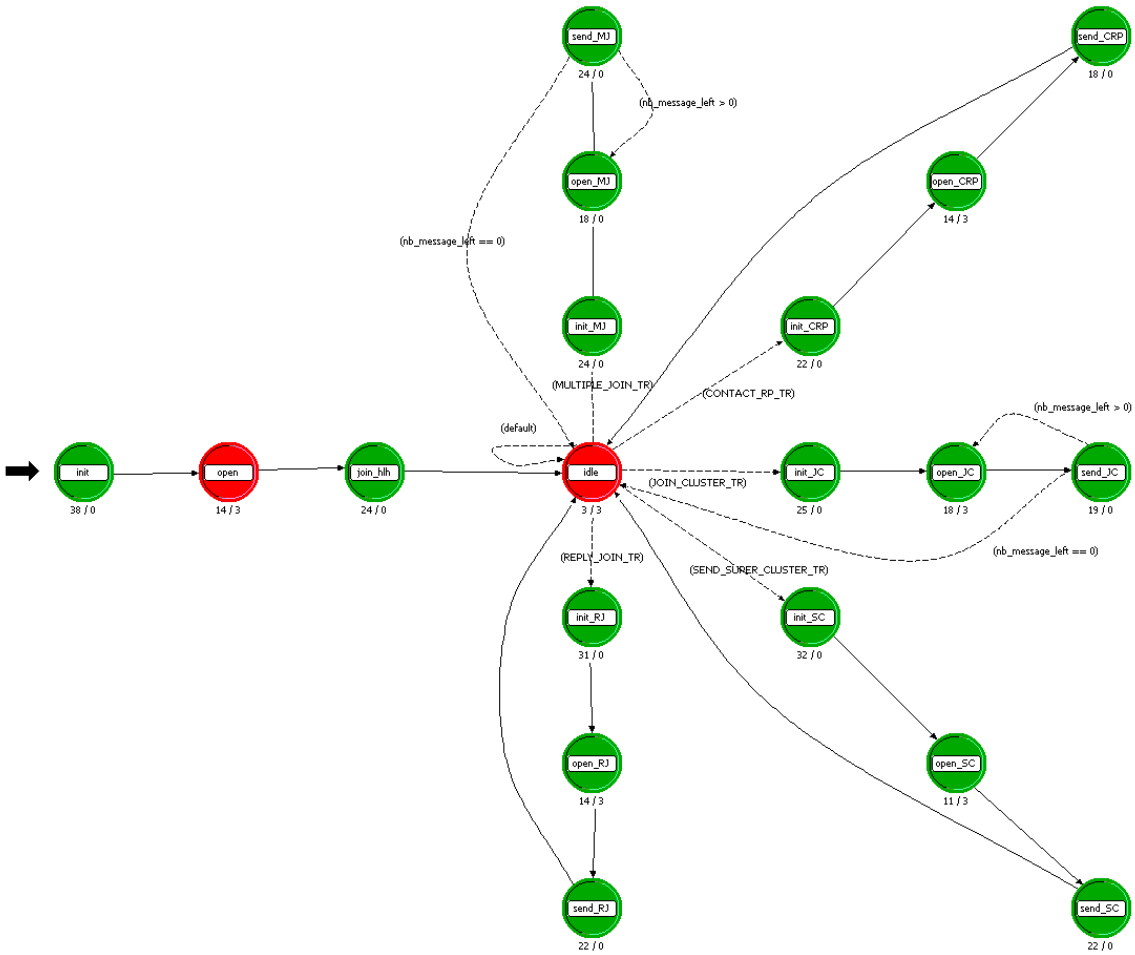

Figure 5.3: NICE. Client's first child process.

Once received, each peer will create a child process which will mainly focus on the joining process. The child process is given in figure 5.3. The FSM has the shape of a star in which each branch has a special purpose and where the center is an idle state, which will act as a dispatching state. Left branch is the first step and is used for the client to contact the root of the tree. Central bottom branch is used (by cluster leaders) to answer a join request from a new host. Central upper branch is an iterative process to go down the tree and contact leaders to find the closest one. Finally, right branch is the final step of the joining process when the new host will send a message to its new cluster members. Upper diagonal branch is only for the root of the tree and send update message to the RP. Lower diagonal branch is for cluster leaders and send the super cluster parameter to new hosts once they have joined a cluster.

9. *split_merge*. Each peer can be leader of a cluster and must have the capability of splitting or merging a group when the size of the cluster is respectively greater than $3k - 1$ or lower than k.

10. *send_message*. This is the state in which a peer goes when he wants to send a message to the group. In our simulation, peers send messages randomly once all members have joined the group.

11. *heartbeat*. Peers communicate with their cluster's neighbors and have to send messages every t seconds. To do that, it will create a heartbeat process given in figure 5.5. This process will be responsible of sending messages and once done will destroy itself. Once more, creating this child process allows the peer to receive at the same time messages from other peers and to process other tasks (multitasking).

12. *timeout*. Peers must be able to detect if other peers are not anymore present in the group. In our implementation, we did not implement this feature yet.

Some other processes have also been develop but are not represented here in order to concentrate on the most important ones. Those processes are made to:

- recontact the RP if there was a problem with the first connection,
- calculate the strech,
- update members of a cluster on special will of the cluster's leader,
- update RP if the root of the tree has new information to send to the RP.

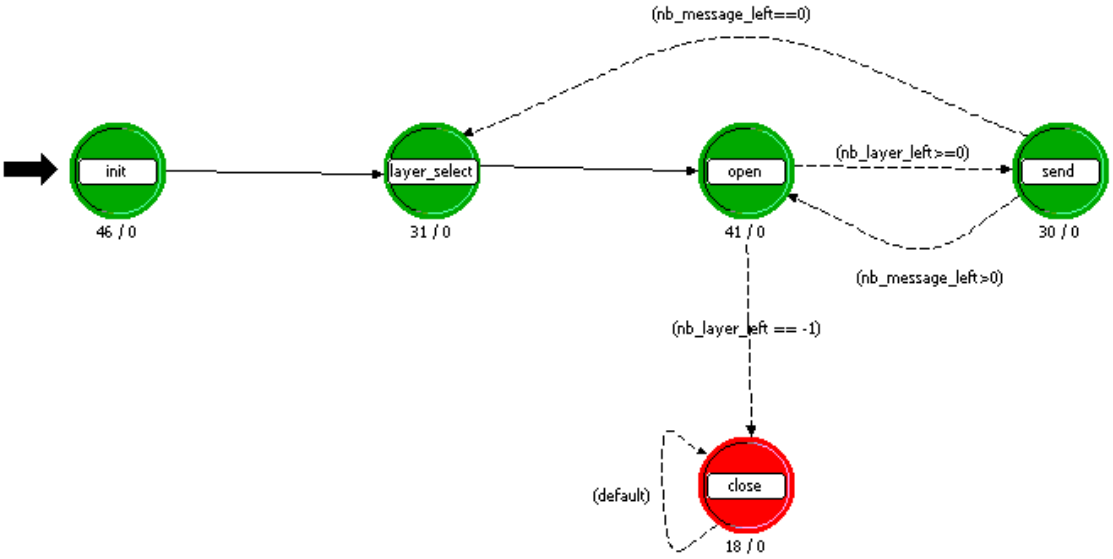

Figure 5.4: NICE. Client's forwarding process.

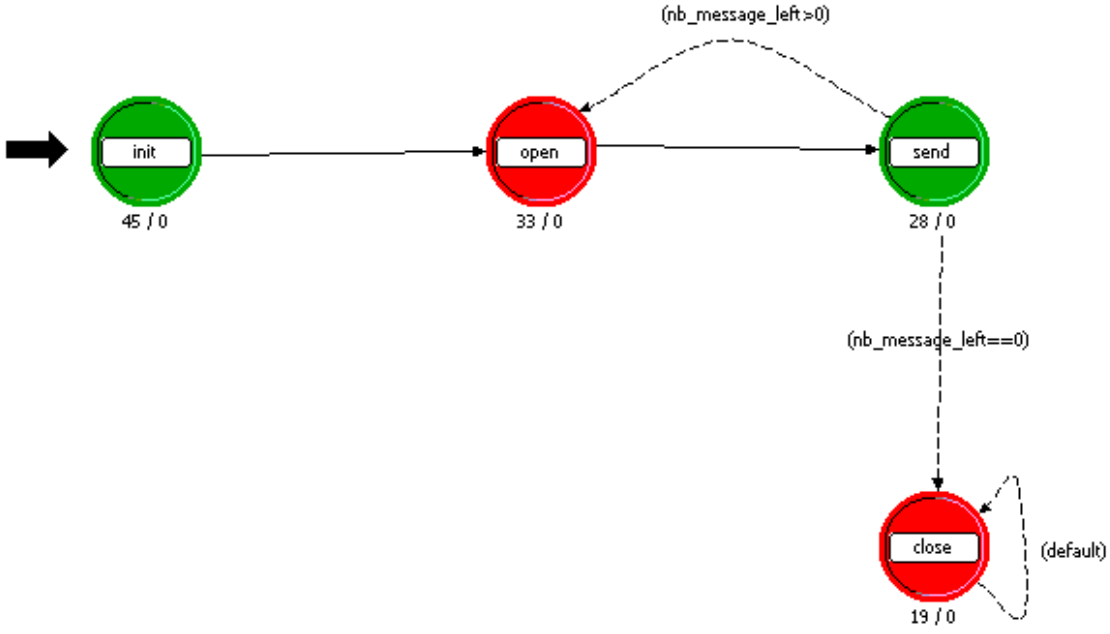

Figure 5.5: NICE. Client's heartbeat process.

5.5.2 DS-ALM

DS-ALM simulation was a little bit easier to realize than NICE. Indeed, during the first step of the protocol, the *RP* is calculating an initial mesh network and will reply to hosts once the result is computed. Therefore, in our simplified implementation we did not have to care about the heartbeat message for example. On the other hand, the *RP* was more "intelligent" than just redirecting new hosts to the root of the tree and therefore more work has to be done on the *RP* implementation.

Rendezvous Point

Some similarities appear with NICE. First of all, the finite state machines are the same (figures 4.6 and 4.7). But of course, the implementation beneath is different. Codes are not given here for an obvious lake of place.

The two first states have exactly the same functionnalities as before, i.e. initializing variables and opening connections. The two other states have the following purpose:

1. Receiver. The idle state is waiting for peer connection. Once packets arrive, the *RP* registers each host information. When x members are connected, the *RP* is able to compute the mesh network and will then create the sending process.

2. Sender. The sending process will send a message to all members present in the mesh network computed by the receiving process.

Peer

The main process (figure 5.6) looks a lot like NICE main process, although some states are not present in DS-ALM FSM since they are not defined. As we explained, the main difference concerns the joining process. This is the reason why there is not an equivalent to the main child of NICE. Many of the states have the same functionalities than NICE. Important differences occur in two states:

1. *receive*. As we said, in this state, new hosts will not have to create the (complex) main child responsible of finding its place in the tree. Instead they will create the four-states activation process given in figure 5.7.

   The activation process goal is that neighbors exchange messages so that all members of the mesh know if their neighbors are connected and if they are gateway nodes.

2. *route_act*. Once the activation process is over, hosts send a message to reach all others group members. It will create the process illustrated in figure 5.8. Gateway nodes will

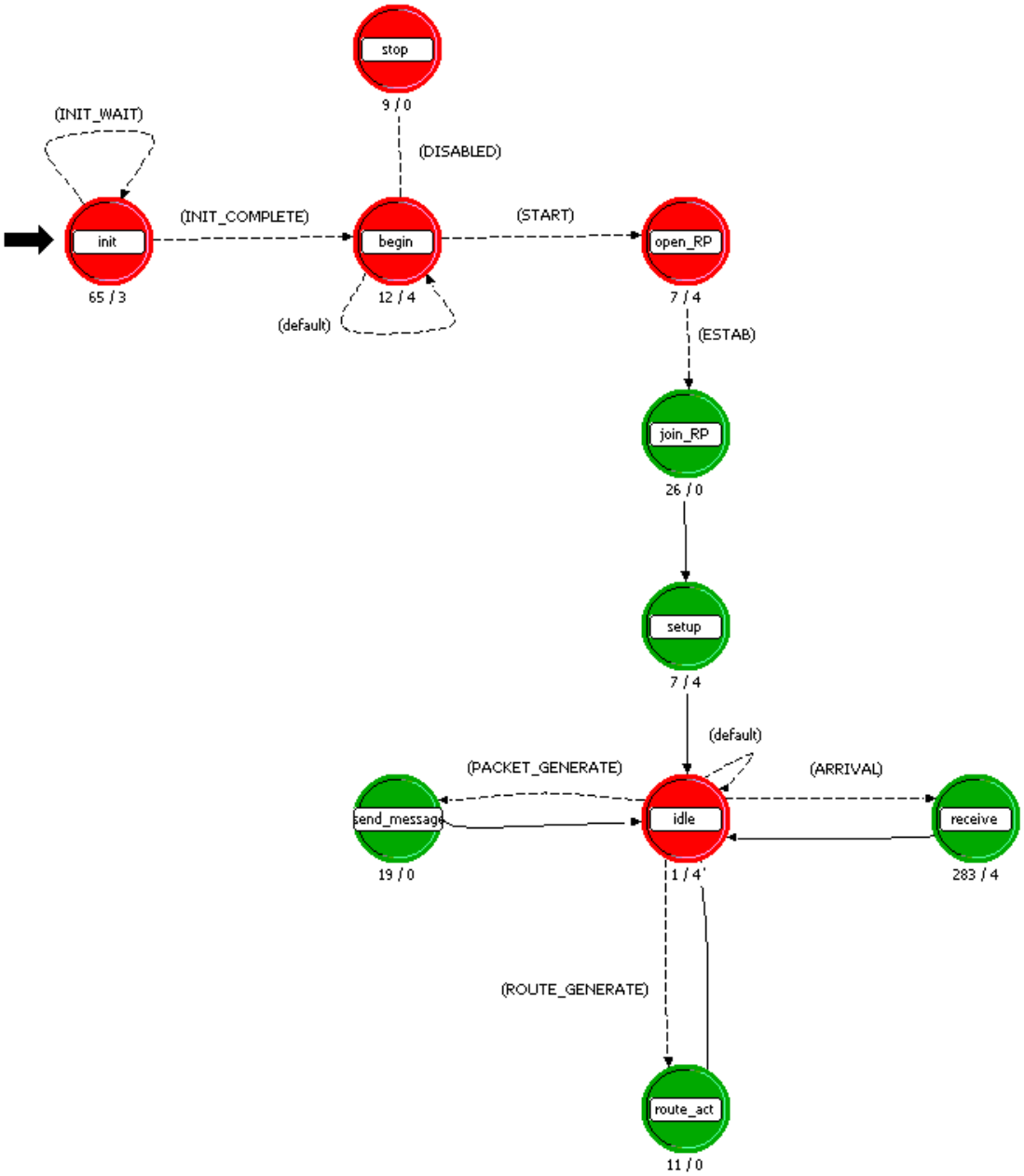

Figure 5.6: DS-ALM. Client's main process.

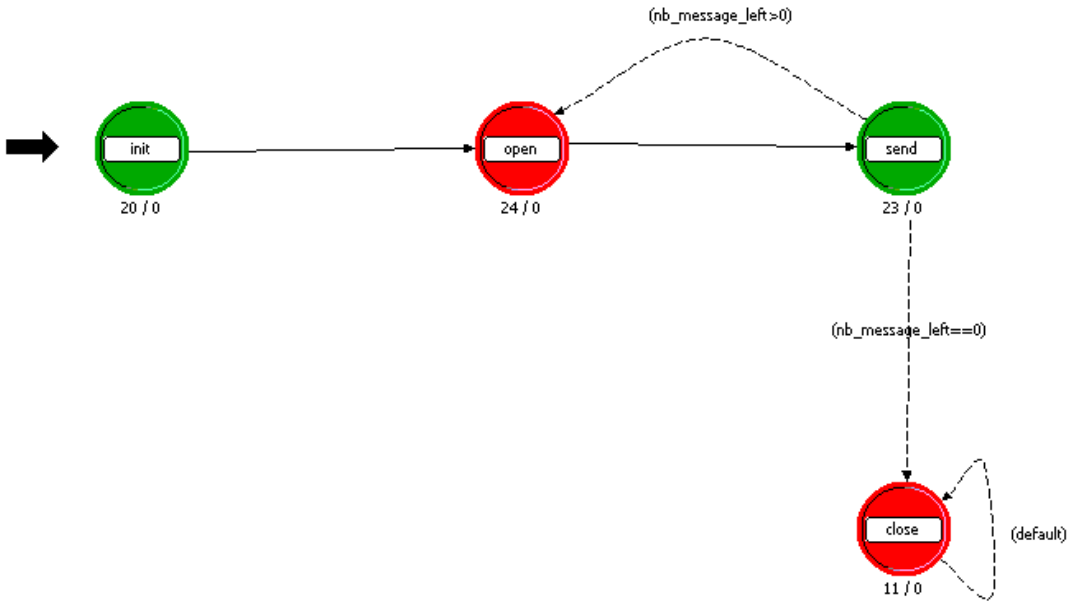

Figure 5.7: DS-ALM. Client's activation process.

have to forward this message to all neighbors except the one that it comes from and will have to discard doubles. This step will build a special diffusion tree for each source.

As it was the case with NICE, another state was created in order to collect the stretch results from the simulation. We did not reproduce the forwarding process (figure 5.4) as the idea is the same as in NICE.

5.6 Conclusion

We used OPNET in our project in order to simulate NICE and DS-ALM algorithms. We develop models with between fifty to five hundred workstations and one Rendezvous Point. Limitations in the number of workstations are linked to memory requirements.

In order to be as realistic as possible, we develop the Internet model based on the real Internet topology. Internet clouds simulate the behavior of multiple routers although we do not know exactly how it is done by OPNET. All elements used in simulation are OPNET elements. We did not modify clouds, nor routers or links. All default behavior have been tested and compared to reality by OPNET creators and/or users and we assumed a correct behavior for this project. The only part that we had to modify was the application layer in

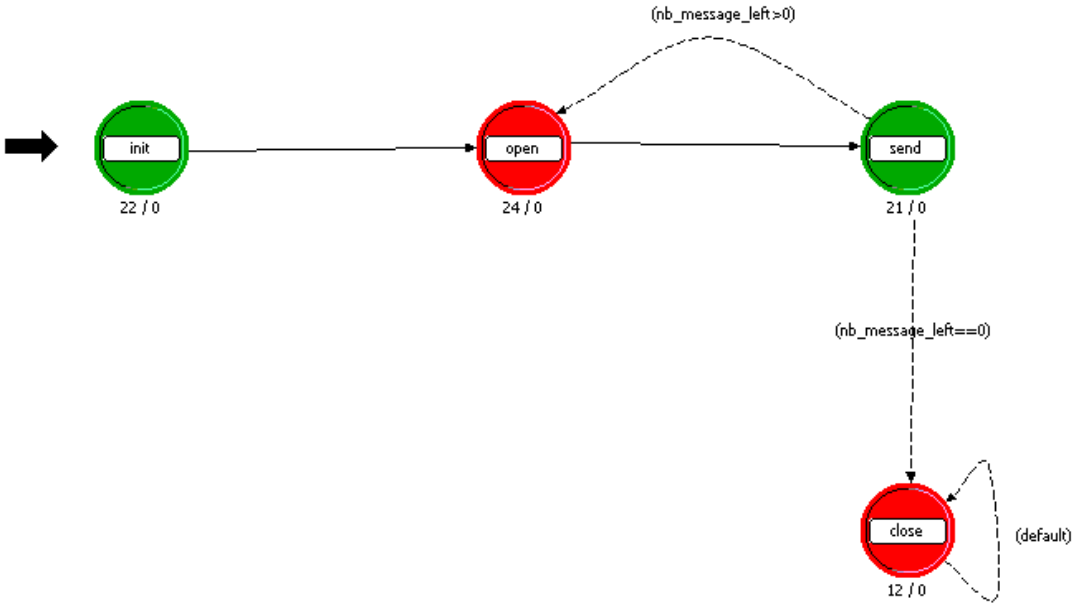

Figure 5.8: DS-ALM. Client's route activation process.

which we create different processes to simulate NICE and DS-ALM algorithms. We usually create a receiving process and at least a sending process. There were often more since different tasks in the algorithms required to send messages.

We focused only on the application and this is the reason why all we compared were raw behavior. We did not introduce any other latency or traffic that the one that was generating by the algorithms. UDP was used since main goal of both algorithms is to have a minimum end-to-end delay.

Next chapter will present the results of our simulation as well as comments on these results.

Chapter 6

Results

6.1 Introduction

After implementing the two algorithms, we had to compare their behavior. We have seen some parameters in section 2.4 and we will use some of them to see the difference between the two algorithms. Parameters that we will study are:

- Diameter (in term of time)
- Diameter (in term of hops)
- Average number of hops
- Average end-to-end delay
- Stretch

First three sections compare DS-ALM and NICE based on a free-degree of DS-ALM between 3 and 7. As we saw earlier, this is a fair way to compare both algorithms since most of the node in NICE will have between k and $3k - 1$ members (i.e. 3 to 8 members).

But since we can play on something with DS-ALM, we will see the evolution of studied parameters when we increase the free degree to go until 25. Unfortunately, increasing the degree requires more memory and at some points we were not able to simulate the behavior for more than two hundred hosts.

Finally, last section will compare NICE and DS-ALM for different value of the degree.

6.2 Diameter

Maximum diameter diagrams (figures 6.1 and 6.2) show that despite a maximum number of hops obtain with DS-ALM the end-to-end delay remains lower.

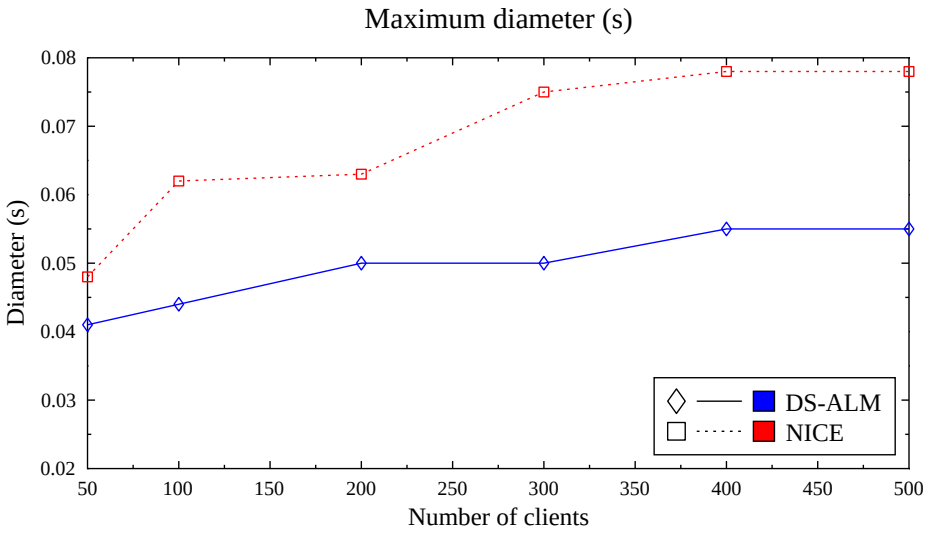

Figure 6.1: Maximum diameter time of NICE and DS-ALM

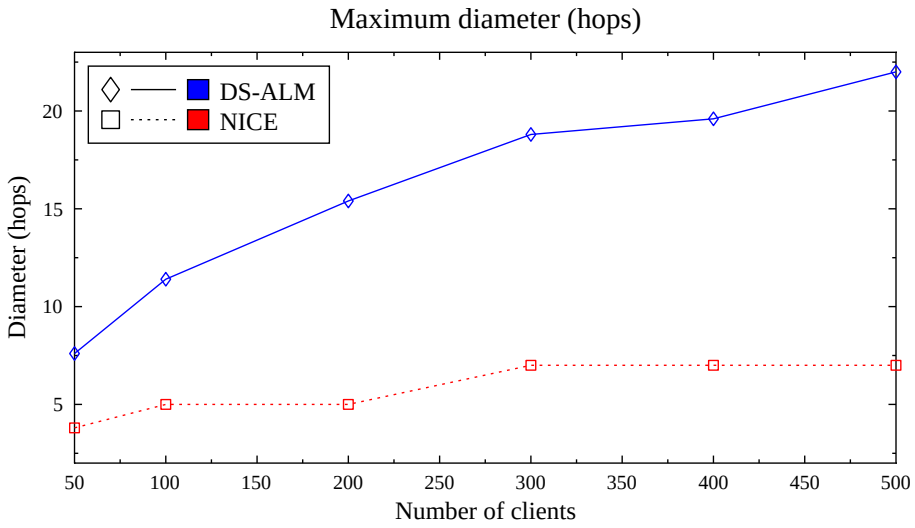

Figure 6.2: Maximum diameter in term of hops of NICE and DS-ALM

This is possible because of the nature of each host. In DS-ALM a lot more nodes are considered as gateway node while only the leader of a cluster is a gateway in NICE. In addition, since DS-ALM create an ignore set for each source, this ensure that depending on the source, the gateway used to forward the message will be the one who minimize the end-to-end delay. This concept is illustrated in figure 6.4 and 6.5 where one can see that some nodes will receive the message from different gateways depending on the source.

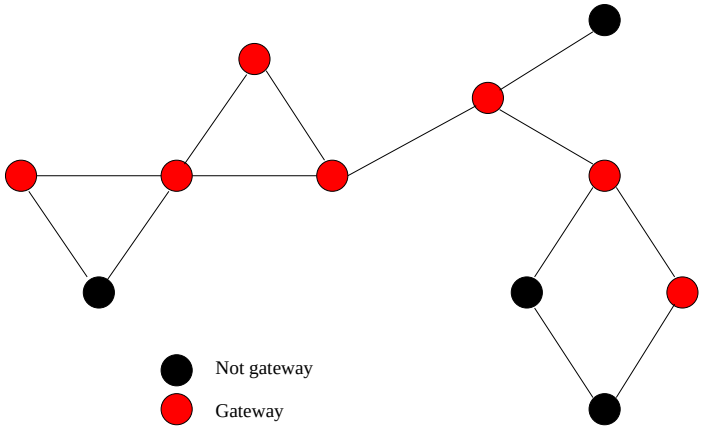

Figure 6.3: Example of a mesh network

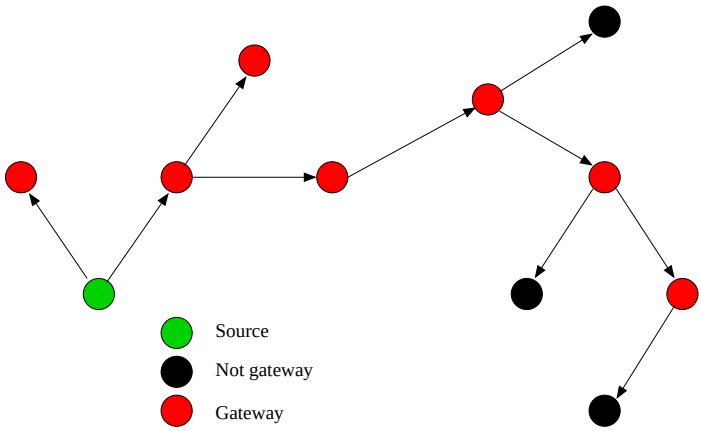

Figure 6.4: Multicast diffusion. Scenario 1.

6.3 Average

Concerning the average end-to-end delay (figure 6.6), we can observe that it is smaller with DS-ALM even if the average number of hops (figure 6.7) is greater than NICE. However, the

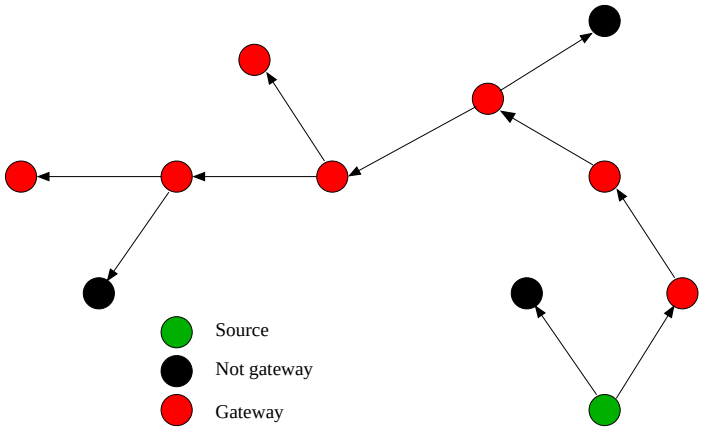

Figure 6.5: Multicast diffusion. Scenario 2.

difference between the two algorithms is lower than with the diameter and especially for the average number of hops. This difference concerning the number of hops can be explained by the fact that the diameter in term of hops in the case of DS-ALM is an isolated value.

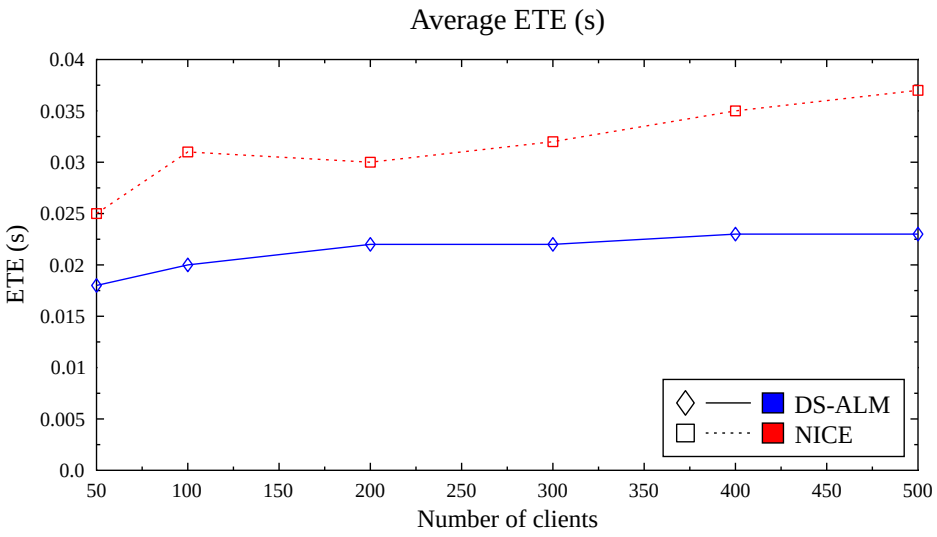

Figure 6.6: Average end-to-end delay of NICE and DS-ALM

6.4 Stretch

The stretch is calculated by the formula $s = i/d$ (where i is the time taken to send a packet through a path established by the ALM algorithm and d is the time taken to send a packet

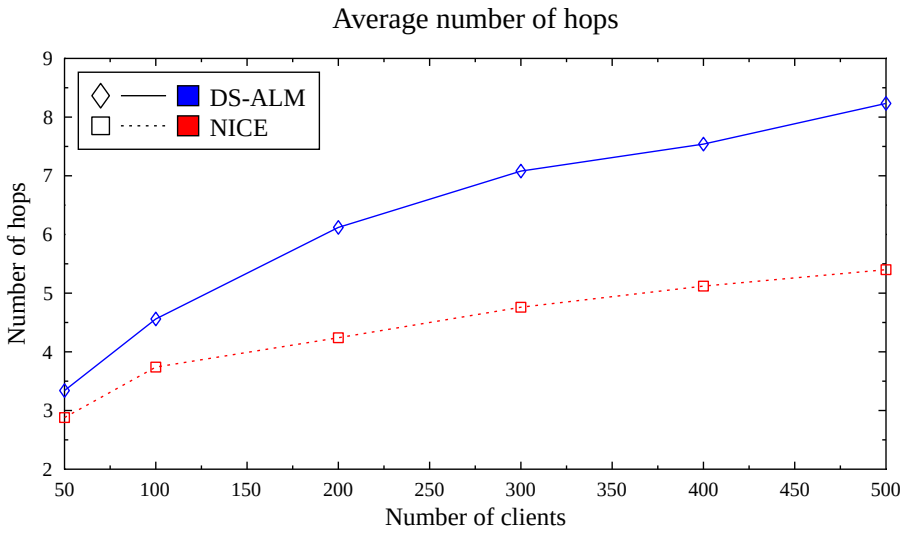

Figure 6.7: Average number of hops of NICE and DS-ALM

with direct unicast). The results (figure 6.8) show that the average stretch is better with DS-ALM than NICE. We can see that the two curves are parallel above 200 clients. We can suppose that for 50 to 100 clients the real curves should be parallel as well but that the simplifications we made in the implementation result in larger stretch for NICE. However, the fact that the stretch is smaller in DS-ALM seems logical. Indeed, the average end-to-end delay is lower in DS-ALM which will results in lower i values and therefore for the same d values, the stretch will be lower.

6.5 DS-ALM evolution with free-degree

Figures 6.9 to 6.13 illustrate the impact of different degrees on the results commented above. As a general remark, we can observe that when the degree increases, all parameters studied decrease. In addition, we can observe that the maximum difference between the different curves is usually obtained between 3-7 degree and 8-12 degree.

The maximum diameter in term of delay is clearly better when going over a degree 7 (figure 6.9). After that, the diameter decreases slowly and it may not be interesting to have a too high degree considering the small improvements. The diameter evolution seems to forward a logarithmic scale when the first small modification improves a lot the characteristic and when the next significant improvement will require a lot more resources.

The diameter in term of hops (figure 6.10) shows small improvement for low number of

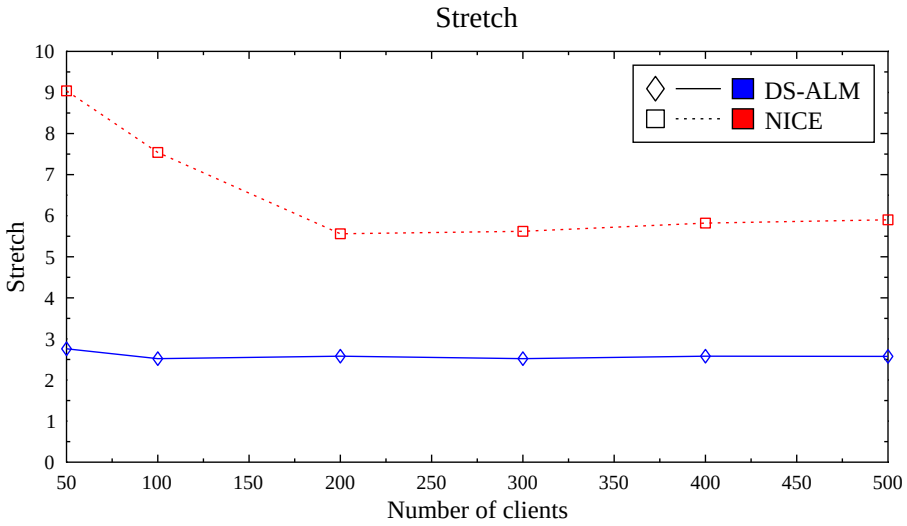

Figure 6.8: Stretch of NICE and DS-ALM

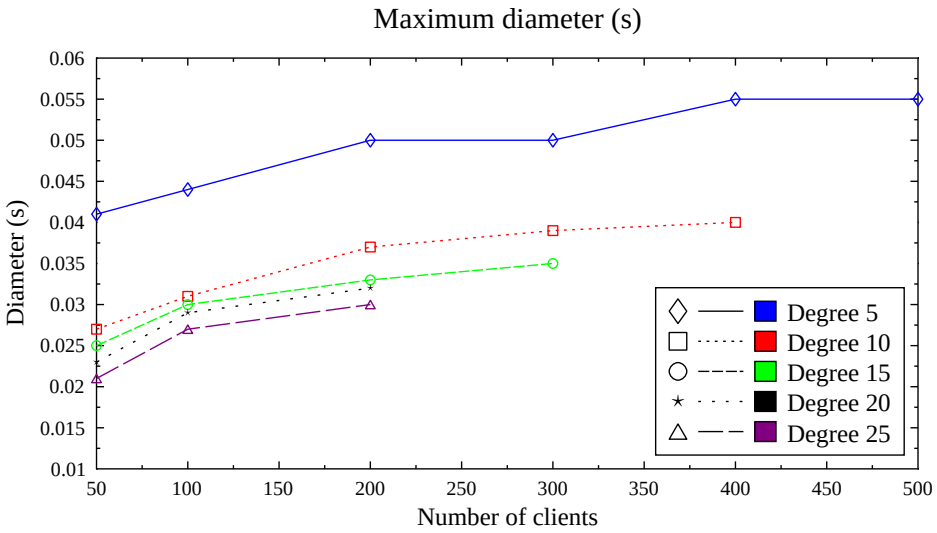

Figure 6.9: Maximum diameter (s) of DS-ALM (various degree)

clients and significant ones for higher number. However, since we observed earlier that the diameter hops number was an isolated value it will be more interesting to compare the average value of this parameter.

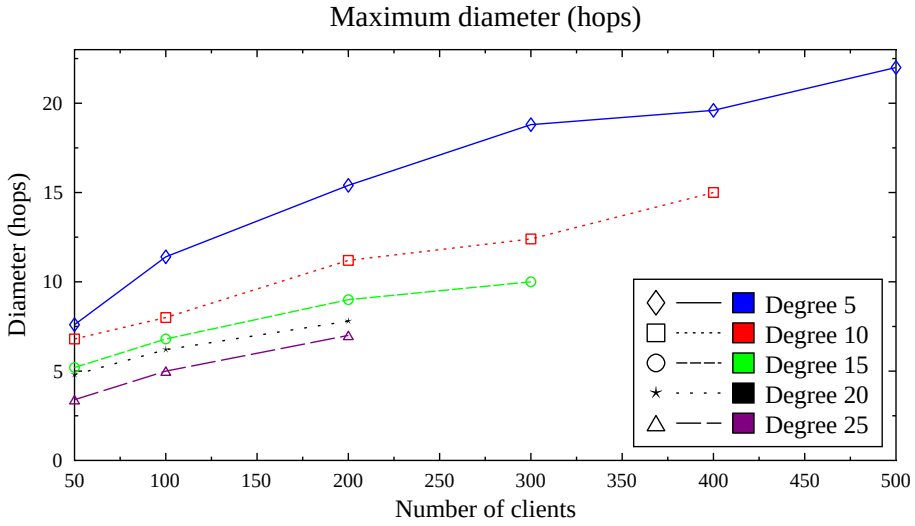

Figure 6.10: Maximum diameter (hops) of DS-ALM (various degree)

The average ETE curves (figure 6.11) follow the diameter but we can observe that the difference between values of high degree is even smaller. As we were saying earlier, it may be not interesting to go over a specific degree considering the small improvements it will procure.

As with the diameter the average number of hosts (figure 6.12) does not show big improvements when the degree increases except for high number of clients.

The stress evolution graph (figure 6.13) is clearly different from the previous one. Indeed, degree 3-7 curve has a different profile than all the other ones. The different curve styles can be explained by the fact that we obtain more high isolated values for low number of clients when the degree is lower. Indeed, the higher the degree, the higher the chance that a better path is found in the overlay network. However, the same comment concerning the logarithmic improvement can be seen since when the degree goes over 10, the stress decrease is barely visible.

6.6 NICE and DS-ALM degree comparison

Figures 6.16 to 6.20 illustrate the comparison of NICE and DS-ALM for different degree values. Comparisons are made for 200 clients.

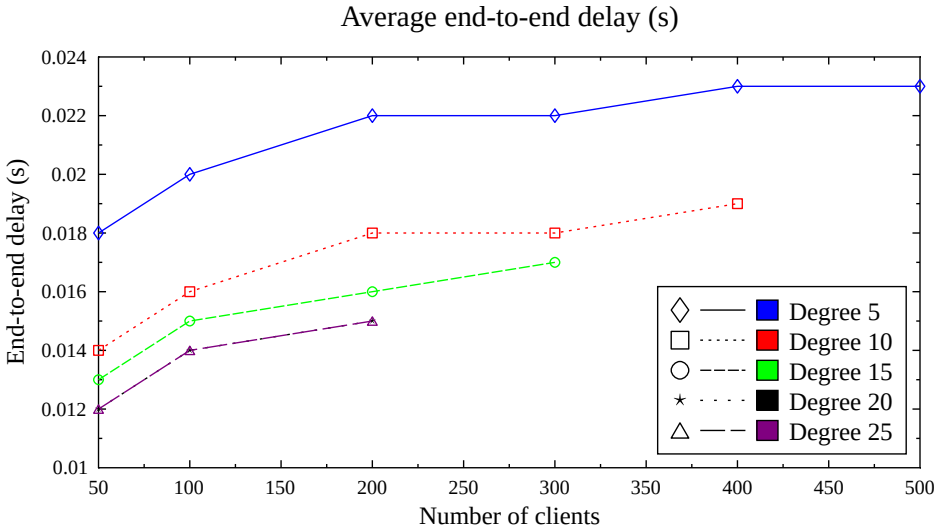

Figure 6.11: Average end-to-end delay (s) of DS-ALM (various degree)

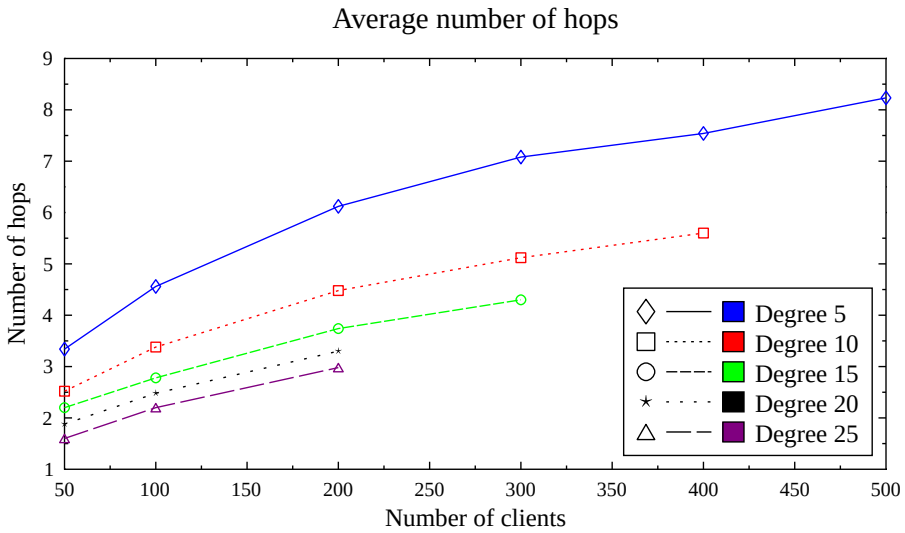

Figure 6.12: Average number of hops of DS-ALM (various degree)

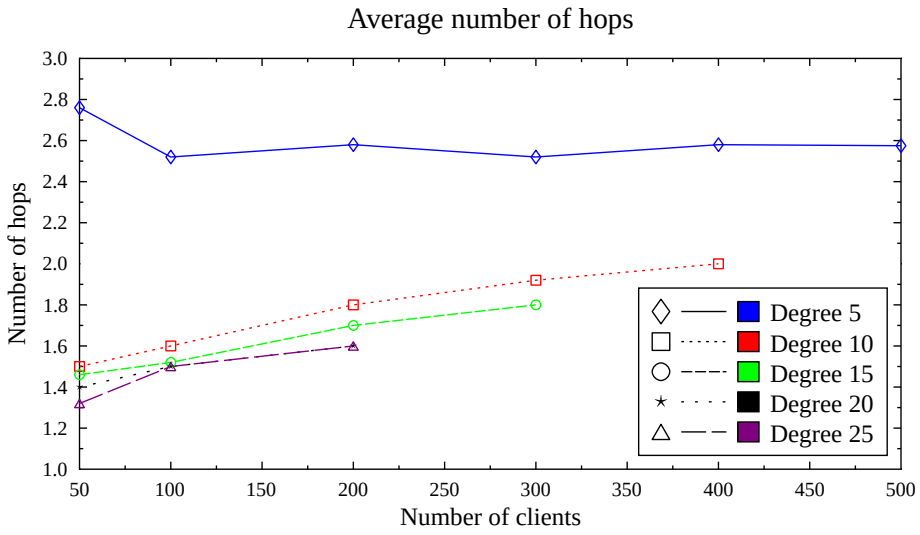

Figure 6.13: Stretch of DS-ALM (various degree)

Figure 6.16 illustrates the decrease of diameter for the same number of clients when the degree increases. We also observe that the diameter is lower for DS-ALM and can be as low as two times better as NICE. The decrease of the diameter with the degree is easily understandable as it is shown in figures 6.5 and 6.15. When the degree increases, the number of connection a node can do increases. Therefore some nodes will receive messages directly rather than indirectly and the end-to-end delay will decrease. It will also affect the number of hops as we can see in figure 6.15.

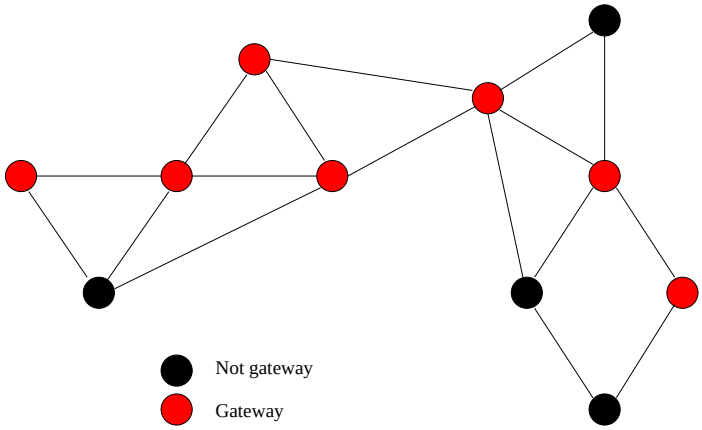

Figure 6.14: Mesh network for a higher degree

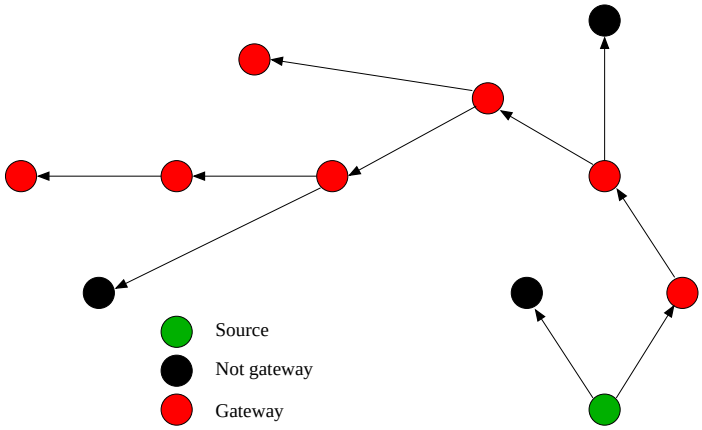

Figure 6.15: Multicast path with a higher degree

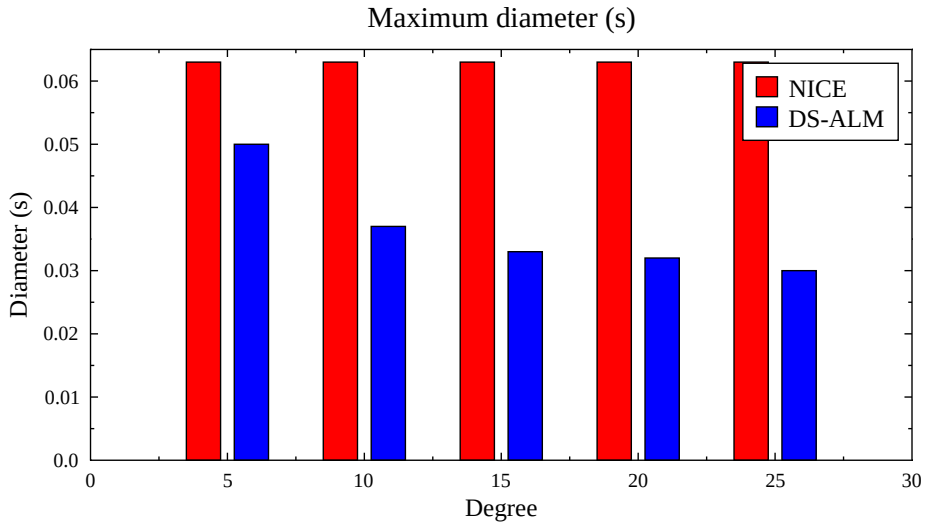

Figure 6.16: Diameter (s) for 200 hosts

As we explained, the maximum number of hops decreases with the degree. But as we can see in figure 6.17, even with a degree of 25, NICE still have a better number of hops. This is due to the tree-based algorithm while DS-ALM is meshed-based. Therefore, the tree constructed by DS-ALM will depend on the source and can results in a higher number of hops. However, as we mentioned earlier, a minimum number of hops does not offer a minimum end-to-end delay diameter as we saw it in the previous graphics.

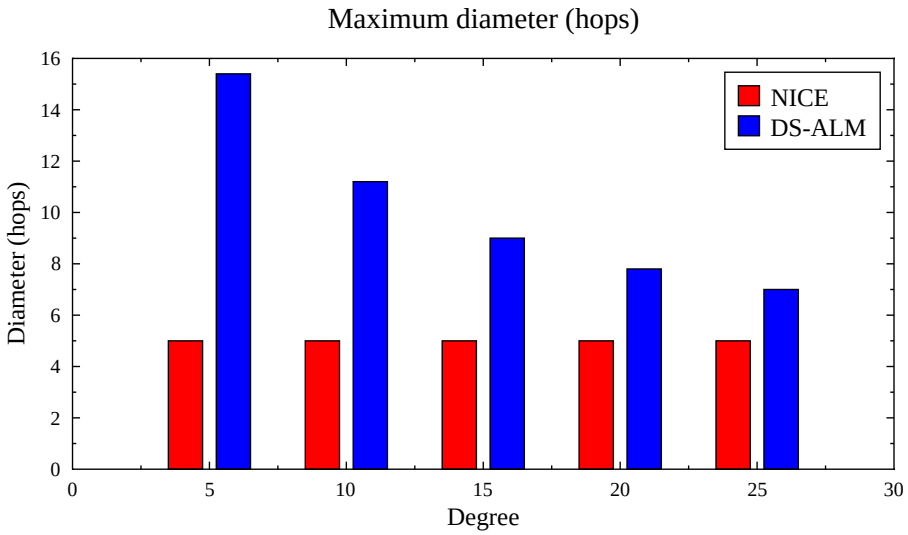

Figure 6.17: Diameter (hops) for 200 hosts

Diameter in term of time and average end-to-end delay graphics (figures 6.16 and 6.18 are quite similar. The evolution of the average end-to-end delay with the degree is following the same curve and the higher the degree, the higher the difference with NICE to go until an end-to-end delay which is half of NICE one.

In terms of average number of hops (figure 6.19), the evolution is interesting. Indeed, when the degree is higher than 10, DS-ALM's average number of hops goes under NICE. And for a degree of 10, the difference is quite small. This is related to what we explained earlier concerning the highest isolated values obtained for the diameter. Besides this high diameter's values, DS-ALM find quite short paths and especially when the degree increase since the number of possible paths increase exponentially.

As other parameters (diameter, number of hops, end-to-end delay) the stretch is getting better when the degree increase (figure 6.20). But in term of ratio when compared to NICE (stretch_DS/stretch_NICE), the difference is not really important since there is a big difference between DS-ALM and NICE.

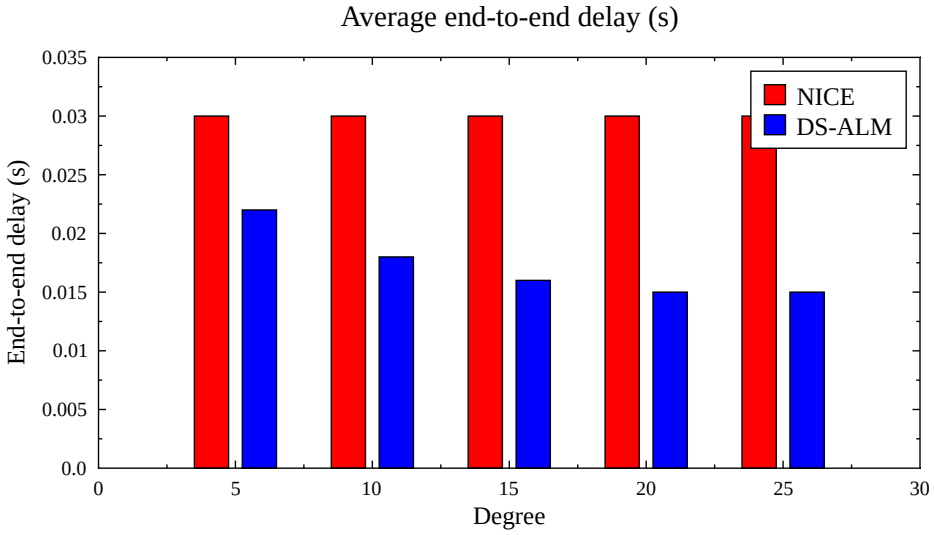

Figure 6.18: Average end-to-end delay (s) for 200 hosts

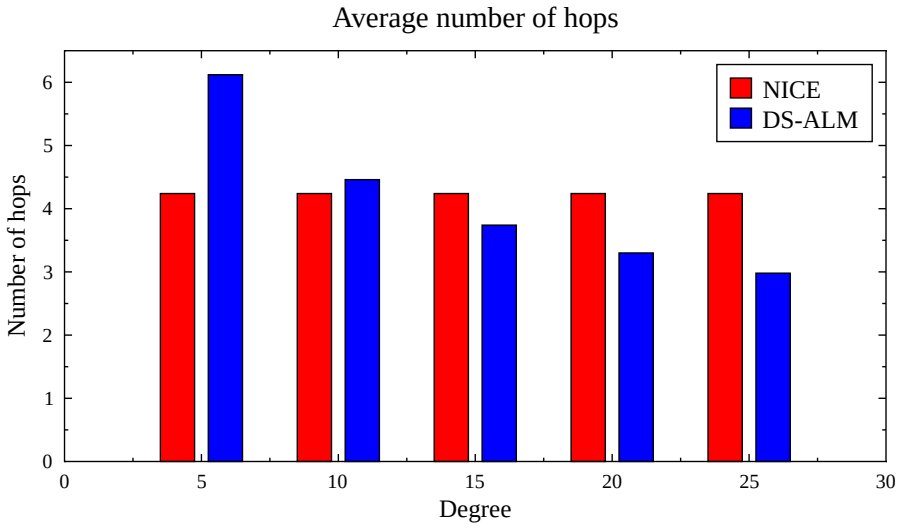

Figure 6.19: Average number of hops for 200 hosts

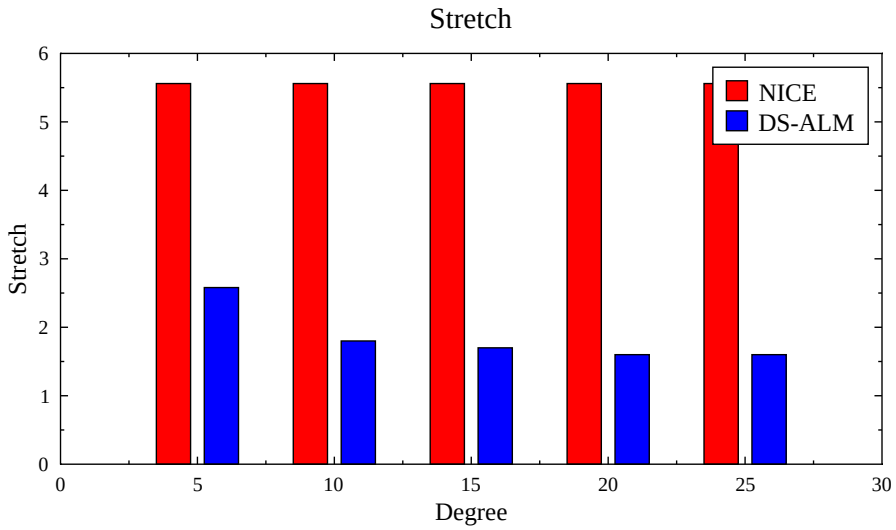

Figure 6.20: Stretch for 200 hosts

6.7 Conclusion

Comparison of DS-ALM with the tree-based ALM algorithm NICE, shows that it is suitable for real time application. Indeed, DS-ALM provides a lower diameter time as well as a lower average end-to-end delay even if messages are going through more nodes. This, a priori non sequitur has finally a simple explanation: the number of possible gateways. Even though the mesh is first established using the real geographic distances between nodes, the ALM path are then based on the minimum end-to-end delay between nodes depending on the source. DS-ALM also has a better stretch, which is finally a consequence of the lower end-to-end delay observed previously.

All this previous results are getting even better when considering a higher degree, the parameter of DS-ALM algorithm. The degree let each host determine the number of clients it can exchange messages with. This allows clients to have different bandwidth and ensure that only the ones who really can are able to forward messages. NICE does not have such a parameter and it implies that leaders can finally be slow hosts.

But increasing drastically the degree will not necessarily provide drastically better results. Indeed, the evolution follows a logarithmic scale which means that finally one of the best alternative would be to increase the degree by only a small value (8-12).

Chapter 7

Conclusion

Even though IPv4 protocol have an ideal solution – *IP multicast* – for group communication, this method suffers from a lake of consistency. Indeed, it is not mandatory for routers or hosts to implement this solution. That is the reason why researchers come with a solution: application-layer multicast.

Although application-layer multicast is a quite new concept, many solutions have been developed and offer many benefits. There is no need to change the Internet architecture since technology is deployed at end-hosts. Intelligence of replicating and forwarding packets is also at end-host which can handle easily this kind of overhead. On the other hand, this will add a load on the network since some links will see the same packet more than once and some end-hosts may receive the same packet twice.

In our thesis we concentrate on comparing NICE, a well-known proven ALM algorithm, to DS-ALM developed at the University of Ottawa. In order to obtain characteristics corresponding to a real situation we had to simulate both algorithms with a network simulation tool. OPNET was the choice for its use in industry and the correct results it provides. What is more, it allowed using existing libraries so that we were able to focus only on the application layer.

Our simulations implement simplified versions of both algorithms. Indeed, we focused on the behavior once the group was created and members were registered to it. Once groups were established we calculate statistics without early departure, nor late comers. In addition, maintenance operations were most of the time skipped for the same reason. We concentrate on end-to-end delay, number of hops and packets duplication at end-host. This allowed us to have a good idea of algorithms' performance.

Our results show that on two studied points DS-ALM has better characteristics. Indeed it provides a faster forwarding process since end-to-end delay is lower than NICE. In fact, the most interesting variable is certainly the diameter time since it gives an idea of the worst

transmission delay from one sender to a receiver. In addition, the more members join the group, the better the performance of DS-ALM compare to NICE. On the other hand, DS-ALM has a worse number of hops linked to the hierarchy schema used to create the overlay network. Of course having too many hops can be annoying but everything depends on the final purpose of the application it is used in. Finally, stretch is better as well in DS-ALM. Since it reflects the comparison of an algorithm with unicast method, it is equivalent to compare algorithms to IP-multicast.

In addition to this basic comparison, DS-ALM allows to play with a parameter: the free-degree of each host. And the results show that the higher the free-degree, i.e. the number of connections a host is allowed to do, the better the results. End-to-end delay, number of hops and stretch decrease with the free-degree.

Of course, many improvements can be done with our simulations. The first one is probably to model a more dynamic model able to take into account members when they leave the group or join it later. Other improvements can certainly be done with memory use. Indeed, our basic knowledge of OPNET was not enough to deal with too many hosts because of memory errors. But the main concern of our thesis was to compare raw results of both algorithm, and on this point, we achieve our goal since we were able to publish an article [2].

Bibliography

[1] D.T. Ahmed, S. Shirmohammadi,and A. El Saddik. A Dominating Set Based Peer-to-Peer Protocol for Real-Time Multi-Source Collaboration. In *Proc. IEEE Workshop on Collaborative P2P Information Systems*. Jun. 2007.

[2] D.T. Ahmed, S. Shirmohammadi, J.C. Oliveira, and J. Bonney. Supporting Large-Scale Networked Virtual Environments. In *Proc. IEEE Conference on Virtual Environments, Human-Computer Interfaces, and Measurement Systems*. Jun. 2007

[3] B. Baccala. Connected: An Internet Encyclopedia. Available online at: <http://www.freesoft.org/CIE/index.htm>. Apr. 1997.

[4] S. Banerjee, and B. Bhattacharjee. Analysis of the NICE Application Layer Multicast Protocol. In *Technical Reports from UMIACS*. Jun. 02.

[5] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable Application Layer Multicast. In *Proceedings of ACM SIGCOMM*, Aug. 2002.

[6] BitTorrent, Inc. Protocol. <http://www.bittorrent.com/protocol.html>. 2005.

[7] G. Coulouris, J. Dollimore, and T. Kindberg. Distributed Systems, Concepts and design. Fourth edition. Pearson eduction limited. 2005.

[8] Y. Kulbak and D. Bickson. The emule protocol specification. In *Technical report TR-2005-03*. 2005.

[9] Secciò de l'EPSEGV. OPNET: Manual de usuario. Available online at: <http://www.opnet.com/university_program/teaching_with_opnet/-textbooks_and_materials/materials/OPNET_Modeler_Manual.pdf>. 2004.

[10] J.-M. de Goyeneche. Multicast over TCP/IP HOWTO. Available online at: <http://www.linuxjunkies.org/html/Multicast-HOWTO.html>. Mar. 1998.

[11] T.-M. Kwan, and K. Yeung. On Overlay Multicast Tree Construction and Maintenance. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing*. Dec. 2005.

[12] OPNET Technologies, Inc. Modeler Documentation Set. Version 12.0. 2006.

[13] X. G. Pañeda, D. Melendi, M. García, V. García, R. García, and E. Riesgo. Analysis Tool for a Video-on-Demand Service Based in Streaming Technology. In *LNCS 2720*. 2003.

[14] B. Rong, I. Khalil, and Z. Tari. Making Application Layer Multicast Reliable is Feasible. In *31st Annual IEEE Conference on Local Computer Networks (LCN)*. Nov. 2006.

[15] S. Shirmohammadi. Internetworking and the Internet Protocol (IP). In *Higher Layer Network Protocols* lecture. Fall 2006.

[16] A. Sobeih , W. Yurcik and J. C. Hou. VRing: A Case for Building Application-Layer Multicast Rings (Rather Than Trees). In *Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*. 2004.

[17] T. Svensson. A. Popescu. Development of laboratory exercises based on OPNET Modeler. Master Thesis available online at: <http://www.opnet.com/-university_program/teaching_with_opnet/textbooks_and_materials/materials/-Lab_Exercices_Modeler.pdf>. Jun. 03.

[18] S.-W. Tan, G. Waters, J. Crawford. MeshTree: A Delay-optimised Overlay Multicast Tree Building Protocol. In *Proc. of 11th International Conference on Parallel and Distributed Systems*. Jul. 2005.

[19] TANENBAUM Andrew. 2005. *Réseaux*. 4e édition. Paris : Pearson Education France. 908 p.

[20] D. A. Tran, K. A. Hua, T. Do. ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming. In *Proc. of IEEE INFOCOM 2003*. Apr. 2003.

[21] C. Zhang, H. Jin, D. Deng, S. Yang, Q. Yuan and Z. Yin. Anysee: Multicast-based Peer-to-Peer Media Streaming Service System. In *Asia-Pacific Conference on Communications*. Oct. 2005.

[22] Q. Zaho, Y. He, and J. Zhang. A Hybrid Approach for Overlay Multicast. In *Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences*. 2006.