MASTER THESIS

TITLE: Real-time multicast algorithms for P2P networks

MASTER DEGREE: Master of Science in Telecommunication Engineering & Management

AUTHORS: Eva Aymamí Gili
Albert Rodríguez Viaplana

DIRECTOR: Javier Ozón Górriz

DATE: October 13th, 2011

Title: Real-time multicast transmission algorithms for P2P networks

Authors: Eva Aymamí Gili
Albert Rodríguez Viaplana

Director: Javier Ozón Górriz

Date: October 13th, 2011

Overview

Nowadays, many applications use multicast transmissions, such as online games, videoconference programs, or sharing applications in a P2P network. However, multicast transmission is a problem that has still not been satisfactorily solved.

In this work we show a family of algorithms capable to solve this problem, concretely focused on the real-time transmissions, in which a node called root or source sends information to a specific group of nodes. These algorithms take advantage of the transmission delay of a message between one node and another in order to send it towards another node.

In order to study the behaviour of these new real-time transmission algorithms we have worked with two virtual networks that models the IP network, to which we have added a number of users, from 10 to 1000. These users form the multicast group. Later, we have obtained the overlay network. This network is defined in the application layer, and the user nodes form it.

Finally, the multicast algorithms have been applied on those networks and results have been analysed to extract the conclusions for our original purposes.

Títol: Algorismes per la transmissió multicast en temps real en xarxes P2P

Autors: Eva Aymamí Gili
Albert Rodríguez Viaplana

Director: Javier Ozón Górriz

Data: 13 d'Octubre del 2011

Resum

Actualment, moltes aplicacions utilitzen les transmissions *multicast*, tals com jocs en línia, aplicacions de videoconferència o d'intercanvi d'arxius en una xarxa P2P. No obstant, la transmissió multicast és un problema que encara no ha estat resolt de forma satisfactòria.

En aquest projecte, es presenta i estudia una família d'algorismes capaços de resoldre el problema de l'encaminament multicast, focalitzats en escenaris de transmissió en temps real, en els qual un node denominat arrel o font ha d'enviar informació a un grup determinat de nodes. Aquests algorismes aprofiten el retard de transmissió d'un missatge entre un node i un altre per tal d'enviar el missatge cap a un tercer node.

Amb la finalitat d'estudiar el comportament d'aquests algorismes hem treballat amb dos models de xarxa virtual per tal d'emular la xarxa IP, als quals hem afegit un determinat número d'usuaris que han oscil·lat des de 10 fins a 1000. Aquests usuaris conformen finalment el grup multicast. A continuació s'ha obtingut la xarxa *overlay*, definida a la capa d'aplicació i formada pel conjunt de nodes d'usuari.

Finalment, s'han aplicat els algorismes sobre les diferents xarxes *overlay* i els resultats han estat analitzats amb l'objectiu d'extreure'n conclusions i conèixer si els algorismes han assolit les premisses definides inicialment.

INDEX

INDEX FIGURES

INDEX TABLE

INTRODUCTION

In recent years the number of computers connected to the Internet has grown up considerably, as well as the set of applications that can be executed over them. Very often, these applications consist in data transmission between one computer and another, or between one computer and a group of them. From the origins of computer networks, the unicast (from one computer to a single one) and the broadcast (from one computer to an undefined group of them) transmissions are available for communications among devices on a network. Moreover, the transmission from one single computer to a well-defined group of receivers is still an unsolved problem. This sort of transmissions is named multicast and lots of applications use it: videoconference calls, multiplayer games, or file sharing in a P2P network.

The multicast IP is an available solution to multicast routing problem, but it also has some (and relevant) drawbacks, as it was added to the original IP specification. First, the number of multicast addresses is small, which implies a limited number of groups and then, some multicast routing algorithms and protocols are complex. In any case, the most problematic point is the fact that all the network equipment, basically routers and switches, must understand these protocols and some of them are still not ready to support multicast services. In other words, most of the backbone equipment should be changed in order to provide a multicast IP service.

This master thesis is the continuation of previous works [13,14,15,16,17], where a different strategy for multicast transmissions, defined over the application layer, was proposed. With this purpose, they presented a family of routing algorithms which, due to its general characteristics, can be used at any layer of the protocol stack. These algorithms present as main features: simple implementation, low transmission delay and high scalability.

At the beginning [13,14,15], the algorithms were defined to send a single packet in a very homogeneous network. After that, more complex networks and scenarios were proposed, adding also some behaviour conditions with the aim of optimizing the total transmission delay [16,17]. But, in any case, no one of the preceding studies took into account the performance of the algorithms for a real-time transmission. Most of the multicast transmissions such as a retransmission of a live match or a concert, require a real-time communication, so our study is focused on the adaption of the former algorithms to properly work with real-time transmissions.

To study and check the performance of the modified algorithms, we have tested them on different simulated networks. In fact, we have worked with a virtual representation of an Internet backbone, which has transit networks (with high speed and delay) and access networks (slower, with lower delays, and connected to the transit nodes). So, two different classes of backbone networks are tested in this work, one formed by a single Autonomous System, and another which includes several AS.

After that, we have defined the overlay networks by adding to the previous backbone networks a set of user nodes (or peers) that form the multicast group. Once we have the overlay network, we have applied the algorithms over it. This execution will report the routing tree, the node rate, the time at which any node receives the packet, the number of nodes reached by the algorithm and the total transmission delay. We have analysed and compared the results for real-time algorithms with those ones resulting from previous works.

The memory of this project is structured in six chapters: first, we explain the multicast transmission problem and we present the Internet network model used for the simulations. The second section is a brief introduction to graph theory and the Dijkstra's algorithm, which is used to obtain the minimum path between two nodes. The third section defines the algorithms used to get the multicast routing tables. Fourth section describes the applications used to create the P2P overlay networks and to apply the algorithms on it. In the fifth chapter we present and analyse the results, and compare them with former performances of the algorithms. Finally, in the last section, we enumerate the conclusions of the study and analyze the environmental impact of this work.

CHAPTER 1. TRANSMISSION AND MODELING OF IP NETWORK

1.1. Multicast transmissions

Any communication network, including computer networks, responds to some transmission topology. The most usual transmission types are unicast or point-to-point transmission (where one single source sends information to one single receiver) and broadcast, where the data is transmitted from a single source to all the possible receivers in the network. An example of unicast communication is a call over the Public Switched Telephone Network (PSTN) while an example of broadcast transmission is the diffusion done by a radio station. Otherwise, if we want to transmit data from a node to a well-defined set of receivers, the transmission is more complex than in the previous cases. This is known as multicast transmission and means a set of clients receiving the same stream from a single source.

Multicast applications include video conferencing, multiplayer networking games, corporate communications, distance learning, software distribution, stock quotes or news. In the context of IP networks multicast was initially proposed to be implemented at the network layer [1], but it has not been widely deployed [2]. Multicast IP defines a multicast group where the clients receive the stream originated by a single source, which only sends one packet to the multicast group. It is forwarded to the multicast routers and replicated at points where the paths of the different clients diverge. By sending only one copy of the information to the network and allowing the network intelligence replicate the packet only when necessary, bandwidth and network resources may be efficiently exploited. But there are significant drawbacks for multicast IP, since the original IP design did not consider multicast transmission and it is an “add-on” to the IPv4 protocol. For example, the range of IP addresses to create multicast groups is very limited and most of them are already reserved. Additionally, complex algorithms capable to know all the devices involved during the transmission (end devices, routers…) are needed. An example of a protocol family that carries out this function is the Protocol Independent Multicast (PIM).

These reasons make multicast transmission complex and difficult before using IPv6. Also, the multicast group in IPv4 is created at network level of the protocol stack, so an application that needs to define and manage a multicast group (for example, a videoconference call application) may have difficulty to administrate the whole multicast group. New application layer based approaches have been deployed due to multiple existing drawbacks in IP multicast transmissions. These alternatives are built using peer-to-peer architectures [3].

In an application-layer multicast approach, also called overlay multicast, the participating peers organize themselves into an overlay topology for data delivery. In this topology each edge corresponds to a unicast path between two end-systems or peers in the underlying IP network. All multicast-related

functionality is implemented by peers instead of routers, with the goal of depicting an efficient overlay network for multicast data transmission. Obviously, the application-layer multicast is not as efficient as network-layer multicast because it adds delay and bandwidth consumption and also provides less stability in the multicast tree. This fact penalizes those applications that require a real-time communication, as a live broadcast football game or a concert, because both are delay-sensitive transmissions. They are also sensible to jitter delay (that is, the difference between transmission delays of two consecutive packets). In video on demand applications, receiving the information without delay is not as important as avoiding the delay jitter, since in this case we want to preserve the rate at which the packets arrive at the destination (rather than the delay at which they arrive) to be able to rebuild the video without interruptions.

This project presents a set of algorithms for obtaining multicast trees in order to minimize the total delay of the transmission –in other words, the time since the first packet is issued into the network until the last node receives the last packet of the transmission– and, also, in order to maintain the rate transmission above the source bit rate, to avoid network congestion.

There are several studies and proposals for application-layer multicast transmissions. These studies are mainly focused on protocols describing efficient overlay trees construction and maintenance. There are two basic approaches to this problem: *fixed nodes* and *dynamic nodes*. The first proposal, as proposed in [4] and [5], places strategically a set of some special nodes around the whole network when a multicast service is required by any application. Although the multicast tree defined is quite stable and easy to maintain, this solution has similar problems than IP Multicast [2]. In dynamic nodes-based approach, according to [6], [7] and [8], the group members are self-organized into an overlay multicast tree and they take care of all multicast functions. Since in large multicast groups there is a frequent joining and leaving movement of nodes, the adaptation of the network to possible changes is one of the main issues that should be considered together with the scalable formation of an efficient multicast tree.

1.2. IP network modeling

1.2.1. Computer network modeling

The use of real networks to study and analyse the multicast algorithms is not possible due to the difficulty to manage them. Moreover, they are usually not available for this sort of tests. These are some of the reasons why simulations will be used to test our designs.

So far, the models used to modelling computer networks usually are:

- Regular topologies, like rings, trees and stars.
- “Well-known” topologies, like ARPAnet or the backbone of NSFnet.
- Randomly generated topologies.

Obviously, these three options have some limitations. The regular or “well-known” topologies only represent a part of current networks and the random ones, usually, do not represent a real network. These limitations must be taken into account, since the performance of an algorithm can widely vary from one topology to another.

Other option to model a network is using a hierarchical model. Two choices are available: N-Level and Transit-Stub. The first one starts with a random connected graph and then, recursively, the nodes are substituted by another new connected graph. The result of this operation is a hierarchical level structure where domains and properties, like the cost of the links, can be defined by the user. The Transit-Stub model will be used for developing our simulations, so it will be described in a more detailed way in the following section.

1.2.2. Transit-Stub model

The graphs proposed by Ellen W. Zegura, Kenneth L. Calvert and Samrat Bhattacharjee [9] have been chosen by us to model the IP network. Their purpose is emulating, in the most realistic way, the *paths* (that is, the nodes sequence) through which information travels in a transmission between any pair of nodes in the IP network. Nodes represent networking devices as switches or routers, and edges represent the paths between interconnecting elements. The model only stands for the logical network structure and does not include individual hosts, that is, terminal equipment.

Figure 1.1 shows a network built following the Transit-Stub model. First, a random connected graph is defined, where each node represents a complete transit domain. Two different domains are shown in the figure; they are highlighted in grey. Then, each node is substituted by a new connected graph, which represents the backbone network for each transit domain (five and three nodes in the figure, respectively). After that, we generate a set of connected graphs and we randomly join one of the nodes of each connected graph to a transit node. Thus, each new connected graph (also called stub domain) represents an access domain for that transit node. Finally, some edges between transit domain and stub domain nodes (or even between stub domain and stub domain nodes) are added. Due to all the generated graphs are connected graphs, the resulting graph will be a connected graph as well. Finally, we must add that any of those random generated graphs can be created using any of the random plain graphs model. The parameters needed to create a network using the Transit-Stub model are:

- T: number of transit domains.
- N_t: average number of nodes per transit domain.
- K: average number of stub domains per transit domain.
- N_s: average number of nodes per stub domain.
- P_s: probability of connecting a stub node to another stub node.
- P_t: probability of connecting a stub node to a transit node.

With these parameters, we can use the tools (described in section 4.1) to create the network layer of each device. However, the Transit-Stub model only sets the backbone (that is, the network layer devices interconnection), which is not enough for our purposes, since we need individual hosts (or peers) to act as multicast group members. Therefore, we should add to the backbone network formed by the transit nodes *T* and the stub nodes *S*, the terminal or user nodes. These nodes will form the multicast group which will share information. In our study, the user nodes have been added to the network by a Java application (see section 4.2) that randomly connects each user node to a stub node of the backbone. This operation emulates the connection between an end user with an Internet access node (i.e. an ISP, represented in our model by an *S* stub node) through an access link. After this, the end user becomes a member of the overlay network.

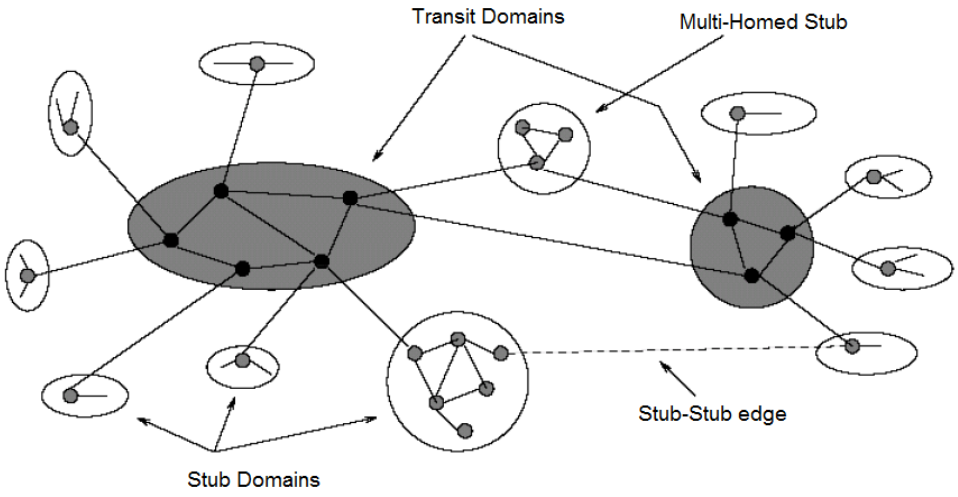


Fig. 1.1 *Transit-Stub model* structure

1.2.3. Random plain graphs

Another model that can be used for the multicast algorithms test is the random plain graphs model. Although this sort of graphs do not represent a real IP network, its simplicity makes them a good option in some network studies to measure the correct behaviour of the overlay algorithms. The basic model corresponds to the pure random model, which distributes the nodes randomly throughout the plane surface. Once distributed, edges are added between a pair of nodes *u* and *v* with an α probability.

Other random models also distribute the nodes randomly over the plain, but they modify the probability function to add an edge with the aim of getting a better approach of the network structure. The most used model is the Waxman one, where the probability of adding an edge between two nodes *u* and *v* is given by the expression:

$$P(u, v) = \alpha e^{-d/(\beta L)} \quad \textbf{(1.1)}$$

In this case, $\alpha>0$, $\beta\leq1$, *d* is the Euclidian distance between *u* and *v*, and *L* is the maximum distance between two nodes of the network. There are other possibilities such as replace the value of *d* by a random number between 0 and *L* or set $\alpha>1$.

Another similar topology to the Waxman's model is the exponential model. In this case, the probability of adding an edge between any pair of nodes *u* and *v* is:

$$P(u,v)=\alpha e^{-d/(L-d)} \quad \textbf{(1.2)}$$

In both cases the probability of adding an edge between two nodes decreases as the distance between them increases.

Finally, on the localization model, the nodes are divided into categories and a link probability is assigned to each of them. For example, for two categories, the probability of an existing edge between *u* and *v* is:

$$P(u,v)=\begin{cases}\alpha \text{ if } d < r\\ \beta \text{ if } d \geq r\end{cases} \quad \textbf{(1.3)}$$

In this case, *r* is a parameter used to establish the boundaries of the category. The Table 1.1 summarizes the random models:

Table 1.1. Random graph models

Model	Edge probability
Pure random	$P(u,v)=\alpha$
Waxman	$P(u,v)=\alpha e^{-d/(\beta L)}$
Exponential	$P(u,v)=\alpha e^{-d/(L-d)}$
Localization	$P(u,v)=\begin{cases}\alpha & \text{if } d<r\\ \beta & \text{if } d\geq r\end{cases}$

In our case, as described in future sections, we have chosen a set of nodes and they have been joined among them, creating a complete graph according to the Transit-Stub model. The edges have been described by two random parameters: transmission time and propagation delay. These values have been chosen randomly and uniformly within a range of allowed values. We have

defined for simplicity's sake a symmetric topology, and so the values of the edge that joins node *u* to node *v* are the same as those of the opposite edge that goes from node *v* to node *u*.

CHAPTER 2. GRAPH THEORY

This chapter presents graph theory, some definitions and the Dijkstra's shortest path algorithm. With these tools, we will model Internet and create the overlay graph to represent our multicast group and test the routing algorithms.

2.1. Definitions

A pair *(V(G),E(G))* is a simple graph *G* where *V(G)* is a finite set of elements called vertices or nodes of *G* and *E(G)* is a finite set of non-sorted pairs of nodes, called edges or links. The order *n* of a graph *G=(V,E)* is the cardinal of *V(G)*, that is, the number of nodes that form it. The graph size *E* is defined as the number of edges of the graph, i.e. the cardinal of *E(G)*. Usually, a graph is sketched through points, which represent nodes, and lines representing the edges joining the nodes, as it can be seen in Figure 2.1. By definition, a simple graph *G* has no repeated edges (pair of nodes linked by more than one edge). From now on, unless specified otherwise, whenever we say graph *G* we will refer to a simple graph.

Both nodes and edges of a graph may include one or more labels identified from applications Φ:*V(G)*$\rightarrow$*R* and Φ':*E(G)*$\rightarrow$*R*. Then, each node has associated a label that typically is an integer or a real number. These labels (also known as weights) can be used to identify the elements of the graph, or set some property of these elements, like link bandwidth. In this project we have assigned two labels –represented by real numbers– to each edge, one for transmission time and another for the propagation delay between each pair of nodes.

Two nodes *u* and *v* are adjacent (or neighbours) if they are linked by an edge *uv*. In this case, nodes *u* and *v* are adjacent to edge *uv*, and edge *uv* is also said to be adjacent to nodes *u* and *v*. Two edges are adjacent when they have one common node. The degree $\delta(v)$ of a node *v* is the number of adjacent edges to *v*.

Given a simple graph of order *n*, the maximum number of edges that it can have is equal to *n(n-1)/2*, that is, all the possible combinations of *n* nodes taken in groups of two. The density $\rho(G)$ of a graph *G* of order *n* is the ratio between the number of edges of the graph *G* and the maximum number of edges that can contain a graph of order *n*. Thus:

$$\rho(G) = \frac{E}{n(n-1)/2} = \frac{2E}{n(n-1)} \qquad \textbf{(2.1)}$$

We define an edge sequence as a succession of consecutive edges $v_0v_1, v_1v_2, v_2v_3, \ldots, v_{m-1}v_m$. This sequence draws a continuous path over the graph. A sequence with no repeated edges is called *path*, and if there are not repeated

nodes, it is called a *simple path*. A *cycle* is a path such that the first node and the end node are the same. Note, however, that any node of a cycle can be chosen as the start node, so the start is often not specified.

Two nodes *u* and *v* are connected if *G* contains a path from *u* to *v*. If every pair of nodes in *V(G)* is connected, then the graph *G* is *connected*. The distance *d(u,v)* between two nodes *u* and *v* is the length of the shortest path between them, that is, the minimum number of edges that we need to go from one node to the other. If a graph is not connected and two nodes belong to two different connected components, we say that their distance is infinite. The eccentricity *ε(v)* of a node *u* in a graph *G* is the maximum distance from *u* to any other node of the graph. The *diameter D(G)* of a graph *G* is the maximum eccentricity of all the vertices in the graph –that is, the maximum distance between two nodes–, and the *radius R(G)*, the minimum.

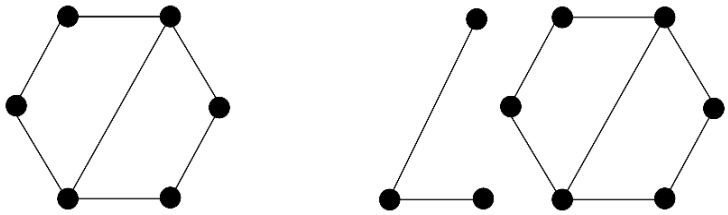


Fig. 2.1 Example of a connected graph and a disconnected graph

A tree is a connected graph with no cycles, or alternatively, a graph in which any two nodes are connected by exactly one path.

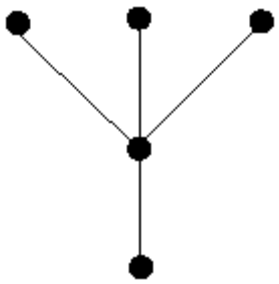


Fig. 2.2 Example of a tree graph

It is easy to prove [10,11] the equivalence of next propositions, related to tree definitions and their properties:

i. *T* is a tree composed by *n* nodes.

ii. *T* has no cycles and has *n-1* edges.

iii. *T* is a connected graph and has *n-1* edges.

iv. *T* is connected and each of its edges is an isthmus, that is, the removal of any of its edges splits off the graph into two connected parts. Moreover, these parts do not have cycles.

v. Each pair of nodes of *T* is connected through just one path.

vi. *T* has no cycles, but the addition of any new edge will create exactly one cycle.

A complete graph K_n is a simple graph in which all the pairs of nodes are linked by an edge, and thus its size is $n(n-1)/2$. In addition, if all nodes in a graph have the same degree, it is called a *regular* graph; and, in particular, if all nodes have degree *r*, it is called a regular graph of degree *r* or *r-regular* graph.

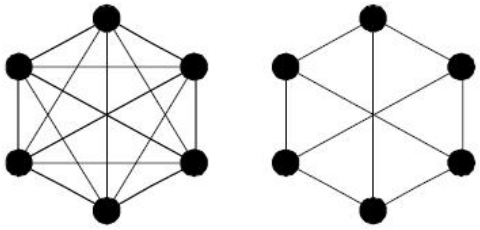


Fig. 2.3 Example of a complete graph and a regular graph

Two graphs $G=(V,E)$ and $G'=(V',E')$ are isomorphic if exists a bijection $\Phi : V \rightarrow V'$ such that for any pair of nodes $u,v \in V(G)$ the edge uv belongs to $E(G)$ if, and only if, the edge $\Phi(u)\Phi(v)$ belongs to $E'(G')$. In this case we say that Φ is an isomorphism of *G* to *G'*. Two isomorphic graphs *G* and *G'* can be represented graphically in the same way. An automorphism of a graph *G* is an isomorphism from *G* to *G*. A graph $G=(V,E)$ is vertex-transitive or vertex-symmetric if given any arbitrary pair of nodes $u,v \in V(G)$ there exists an automorphism Φ of *G* such that $\Phi(u)=v$. Given a vertex-transitive graph, all nodes are interchangeable and have the same properties.

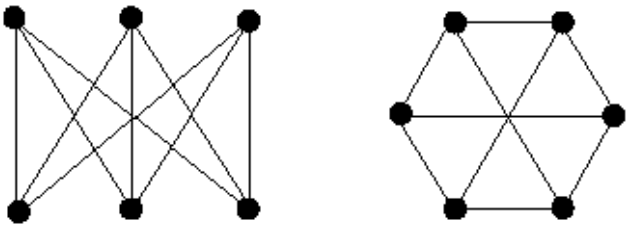


Fig. 2.4 Example of an isomorphic vertex-transitive graph

A directed graph or digraph *G* is an ordered pair $(V(G),A(G))$, where $V(G)$ is a finite, non-empty set of elements called vertices or nodes, and $A(G)$ is a finite set of ordered pairs of $V(G)$, called directed edges or arrows. In this case, the edges have directions (plotted by an arrow) and “goes” from a node *v* to another

one *w*. This way, the edge *vw* and the edge *wv* are different and we can say that *vw* is the inverted edge of *wv.* If *G* has not any edge from one node to itself (that is, an arrow *vv* called loop), and all the edges in *G* are different, then *G* is called a simple *digraph*. If *G* is a digraph, the graph obtained by deleting the arrows (or the directions) of the arcs is called base graph of *G*.

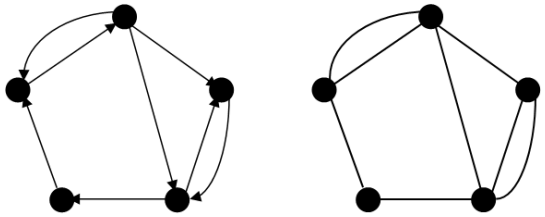


Fig. 2.5 Example of a digraph and its base graph associated

All the definitions given for a simple graph can be extended to a digraph. Thus, we can define labelling applications $\Phi:V(G)\rightarrow R$ and $\Phi':E(G)\rightarrow R$ on the set of nodes and on the set of edges, respectively, and also describe finite sequences of edges $v_0v_1, v_1v_2, v_2v_3, \ldots, v_{m-1}v_m$ where neither edges nor nodes are repeated.

A digraph *G* is called connected or weakly connected if the base graph of *G* is a connected graph. Otherwise, *G* is called strongly connected if it contains a directed path for every pair of nodes $v,w \in V(G)$. While all strongly connected digraphs are connected, not all connected digraphs are strongly connected.

The difference between weakly connected digraphs and strongly connected digraphs is easier to understand if we consider a map of a city whose streets are all one-way. Saying that the map is connected is equivalent to say that we can move from one point of the city to somewhere else, ignoring the required traffic directions. By other hand, if we say that the map is strongly connected we can keep driving from any point of the city to any other else, but always taking care about the allowable direction of the streets.

2.2. Shortest path search. Dijkstra's algorithm

2.2.1. The shortest path problem

The main objective of this project is to study the behaviour of a set of algorithms in order to optimize the transmission of information over a multicast network. To do this, it will be necessary to calculate the shortest path between any pair of nodes of the network and the cost (or delay) to send data between them. This is a classical problem of graph theory, solved optimally by the Dijkstra's algorithm. In particular, Dijkstra's algorithm searches the minimum distance from a node *u* to the rest of the nodes of the graph. To find the shortest path between any pair

of nodes, the algorithm must be run consecutively, taking in each case, as the source node, each node in the graph.

2.2.2. Dijkstra's algorithm principles

In this algorithm, each node *v* of *G=(V,E)* has a label *L(v)* associated. This label shows the shortest known distance needed to move from one given node *u* to that node *v*. Initially, the value of *L(v)* is the weight *w(u,v)* of the edge that connects the nodes *u* and *v*. If this edge does not exist, this value is set to infinity. Also, *L(u)=0*, so the cost to stay in the node itself is 0.

The algorithm defines a set $T \subset V$ which contains the nodes whose shortest path has already been found (from *u* to each of them). Initially, *T* contains only the node *u*. When the algorithm application ends, the *L(v)* value is equal to the cost of the shortest path from *u* to node *v* of *V(G)*.

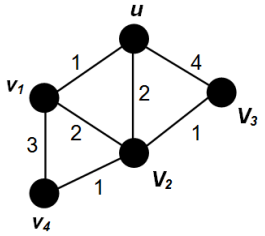
At each iteration, the algorithm adds a new node to the list *T*. This is done by choosing the node *v'* which does not belong to the list *T* and which has the minimum label *L(v')*. In other words, the selected node is the nearest node *v'* to *u* among those that have not been selected. Once this is done, the adjacent nodes to *v'* must update their label, so the distances between *u* and these nodes are recalculated. This process is repeated until all the nodes of the graph have been added to the list. In Figure 2.6 we can see the code which shows the steps performed on the algorithm execution.

```
for all v≠u   L(v)=w(u,v)
L(u)=0
T={u}
while T≠V
begin
find v'∉T such that for all v∉T  L(v')≤L(v)
T=T∪{v'}
for all v∉T such that v' is adjacent to v
        if L(v)>L(v')+w(v',v)
        then L(v)=L(v')+w(v',v) end if
end for all
end while
```

Fig. 2.6 Dijkstra's algorithm

Dijkstra's algorithm is optimal. To prove this, let's see that each time a node *v'* is added to *T*, the label *L(v')* is the minimum distance from *u* to *v'*. Using a proof by contradiction, let's suppose that *L(v')* is not the shortest path between *u* and *v'*. Then, let's say that w_2 is the first node through which passes this new shortest way between *u* and *v'*, shorter than *L(v')*. This w_2 node must belong to *T* by construction, as the distance from *u* to w_2, which the algorithm knows since the first iteration, must be shorter than *L(v')*.

The same argument can be repeated for the next node in the path, w_3, which must have shorter distance from u than $L(v')$, calculated after adding w_2 to T, and therefore this node w_3 must also belong to T. Hence, when the algorithm reaches v', if it would exist a shortest path from u to v' than the indicated by $L(v')$, the algorithm would have added to T these nodes w_2,w_3,w_4,… which form this shortest path, and it would have found this path. Finally, as to complete the algorithm execution all nodes must be in T, the Dijkstra's algorithm finds the shortest path from one node u to any other node in the graph.



Iteration	V'	L(u)	$L(v_1)$	$L(v_2)$	$L(v_3)$	$L(v_4)$	T
0	-	0	1	2	4	∞	$\{u\}$
1	v_1	0	1	2	4	4	$\{u, v_1\}$
2	v_2	0	1	2	3	3	$\{u, v_1, v_2\}$
3	v_3	0	1	2	3	3	$\{u, v_1, v_2, v_3\}$
4	v_4	0	1	2	3	3	$\{u, v_1, v_2, v_{3,}\ v_4\}$

Fig. 2.7 Dijsktra's algorithm execution example for a node u

It is also easy to prove that Dijkstra's algorithm has a complexity of $O(n^2)$, something that, in practice, means that shortest paths can be found in a low computing time. To get the minimum $L(v')$ (fifth line of the algorithm pseudo code) we make $O(n)$ comparisons, and the seventh line does not need more than n allocations. These two lines are in the while loop from line 4, executed $(n-1)$ times. Then, this algorithm can be completed in $O(n^2)$ computation time.

Finally, it must be noted that this algorithm not only calculates the minimum cost (or distance) between any two nodes, but also draws the path which connects them. This can be done by adding a new label in each node, in such a way that when $L(v')$ value is updated, this new label keeps the node v from which the new value for $L(v')$ has been calculated.

CHAPTER 3. MULTICAST ALGORITHM

In this chapter we describe an algorithm, named *Message Stream Multicast MSM*, for multicast data transmission. Initially we describe the *postal model* where a source node sends information to a group of nodes. Afterwards, a set of modifications, which have been carried out over the initial algorithm, have been proposed, developed and compared among them.

3.1. The postal model

To improve data transmission between nodes, or peers in the P2P network context, A. Bar-Noy and S. Kipnis introduced in [12] the MPS(n) Postal Model, which characterizes message-passing system, where packet switching techniques are employed. Also a latency parameter $\lambda \geq 1$ is defined. It corresponds to the time elapsed since a peer starts sending a message until it is fully received by another peer. Therefore, the latency can be calculated as the sum of transmission and link propagation times.

The basic idea of the model proposed in [12] is to use the transmission delay to send the message to other peers. This means that the source forwards the message to more than one peer in order to flood the network as soon as possible, instead of remaining idle after it has send the message once. Similarly, the peers who have already received a message, forward it to other peers before receiving the next one.

The postal model looks for optimum routing trees based on Fibonacci, in spite of the traditional binominal trees. Although the presented model in [12] is designed for broadcast transmissions, it can easily be extrapolated to multicast communications, organizing the peers in groups and using broadcast transmissions to share the information among the nodes belonging to the same group.

Figure 3.1 shows the transmission from peer p_0 to seven different peers, using a binomial tree and a Fibonacci tree. In both cases, each peer can transmit a packet every time unit. The latency value is $\lambda=2$ because two time units are required to go from one peer to other one. Thus, propagation time is equal to one time unit. The left tree is based on a binomial model and requires six time units to send the data message to all the network peers. The right one uses the postal model based on Fibonacci tree, and only needs five time units to reach all the network peers.

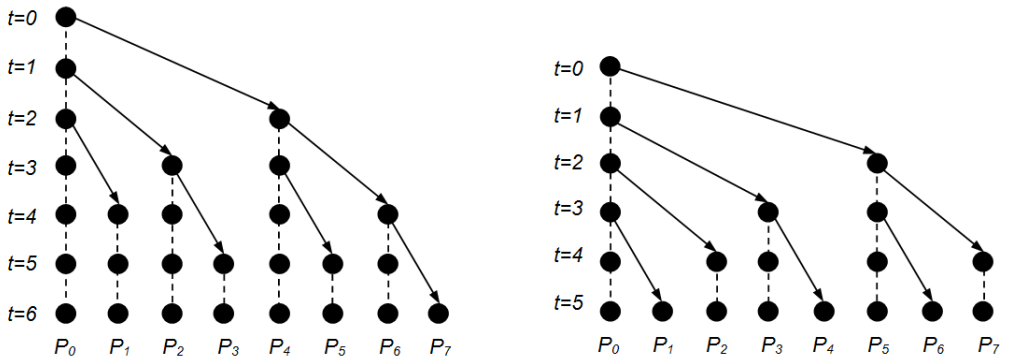


Fig. 3.1 Transmission trees (Binomial on the left, Fibonacci on the right)

3.2. The extended postal model

In this document, continuation of previous projects [13,14,15,16,17] we work with an extended postal model, denominated *Extended MPS* or by the acronym EMPS. For this, we have defined a message-passing system EMPS (n, λ, μ) of *n* peers, with full-duplex connectivity among peers $\{p_0, p_1, \ldots p_{n-1}\}$, where each peer *p* can simultaneously send a message to a peer *q* and receive from a second peer *r* another message according to the parameters listed below:

- For each peer *p* in a message-passing system, we define the transmission time μ_p as the time that *p* requires to transmit a message *M* of length *L*. We denote μ as the vector of all μ_p. In a more detailed context we can define a transmission time μ_{pq} for each pair of peers *p* and *q*. In this case μ, is a square matrix with all-time transmissions μ_{pq}.

- For each pair of peers *p* and *q*, in a message-passing system, we define the communication latency λ_{pq} between two peers *p* and *q*. If at time *t* peer *p* starts sending a message *M* to the peer *q*, then the peer *p* sends the message *M* during the time interval $[t, t+\mu_{pq}]$, and the peer *q* will receive the message *M* during the interval $[t-\mu_{pq}+\lambda_{pq}, t+\lambda_{pq}]$, as shown in Figure 3.2. As defined, the latency is the sum of transmission time of the peer *p* and the propagation delay between *p* and *q*. We denote λ as the square matrix of all λ_{pq}.

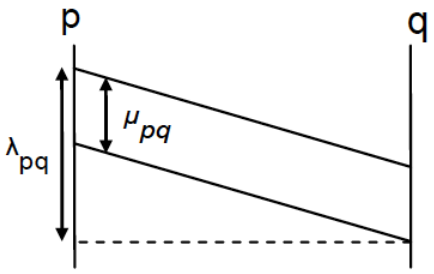


Fig. 3.2 Time transmission and latency

Although an overlay network normally is a fully connected network (each peer in the overlay network is able to send an end-to-end message to any other peer), the EMPS (n, λ, μ) does not require full connectivity. Hence, the definitions and results depicted in this project will fit both fully connected and not fully connected networks. We also assume that the processing delay of the peers is negligible. However, the model could easily be modified when a peer *p* has a processing delay different than zero just adding the processing delay to the latency each time that peer *p* forwards the message for the first time.

EMPS (n, λ, μ) is a generalization of MPS (n) in [18]. In MPS (n) model all peers are identical, so μ_p is equal to the unit value for all peers, and λ_{pq} has also the same value for any pair of peers *p* and *q*. Furthermore, the overlay network is a full connected graph. On the other hand, the EMPS (n, λ, μ) model considers heterogeneous nodes, so we can model different transmission times for different peers and also different communication latencies between any pair of peers, since the underlay network consists of a set of links with different characteristics and network devices.

For simplicity, we assume that for same length messages, the communication latency is constant as a function of time. This means that we do not consider the possible variation of the communication latency due to the load and broken links of the underlying network. Application-layer networks use the services provided by the underlying network, such as a TCP/IP network, to establish unicast full-duplex connections between any pair of peers. However, in this project, we will ignore the characteristics of the layers below the application layer. The term message refers to any atomic piece of data sent by one peer to another using the protocols of the underlying layers.

Thus, we denote by EMPS (n, λ, μ) the message-passing system with *n* peers, a communication latency matrix λ and a transmission time vector μ.

3.3. Single message multicast algorithm SMM

The problem of multicasting one message in a message-passing system is defined as follows: let p_0 be a peer in EMPS (n, λ, μ) model which has a message *M* to multicast to the set of receiving peers $R=\{p_0, p_1, \ldots p_{n-1}\}$, at time $t=0$. On this scenario we have to find an algorithm that minimizes the multicast time t_M, that is, the time at which the last peer of *R* receives the message *M*. Although the result of EMPS (n, λ, μ) is a multicast spanning tree (that is, a tree connecting all the peers of the network), Figure 3.3 shows that this problem is different from the well known Minimum Spanning Tree problem, in which, for a given a network, we have to find the spanning tree with minimum weight. Furthermore, in our problem the time delay between two peers *p* and *q* is not always the weight λ_{pq} of the edge that joins them, since if the peer *p* has forwarded the message to other peer before, we must add the transmission time μ_p to the delay.

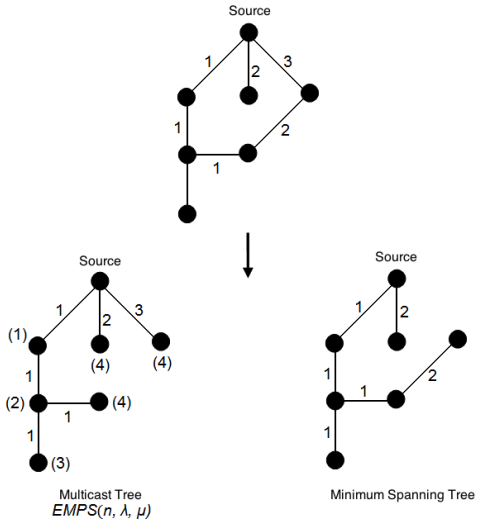


Fig. 3.3 Comparison between EMPS (n, λ, μ) and Minimum Spanning Tree. The transmission time μ is one for all peers and the latency is the weight of the edge. Brackets show the time that each peer receives the message

In [18] the authors define the algorithm BCAST which provides time-optimal multicast trees for the full connectivity case, and for $\mu_p=1$ and $\lambda_{pq}=1$ for any pair of peers p and q. Such time-optimal multicast trees are based on generalized Fibonacci numbers, and they refer to these trees as generalized Fibonacci trees. The authors also state in [18], that in any optimal strategy each peer once have received message M, has to forward it to a new peer each time unit (so, transmission time is considered to be one). This idea also applies to the extended postal model EMPS (n, λ, μ), with the difference that now message retransmissions of peer p have to occur each transmission time μ_p. The algorithm that we propose, called SMM *Single Message Multicast*, is outlined in Figure 3.4.

```
Data: EMPS(n,λ,μ)
Result: routing[i].send[j]
send ¬ 1;
routing[i].send[j] ← 0 ∀i, j ;
routing[i].tnext ← ∞ ∀i ;
routing[root].tnext ← lowest latency of root;
while send <n do
    i ← imin();
    next ← routing[i].index;
    routing[i].send[next] ← 1;
    update_i(routing[i].index);
    update_t(routing[i].tnext);
    update_i(routing[next].index);
    update_tn(routing[next].tnext);
    send ¬ send+1;
end
```

Fig. 3.4 SMM *Single Message Multicast* Algorithm

The variables, arrays, and functions that the algorithm uses are the following:

- *routing[i].send[j]*: the routing table. Initially all its values are 0. When SMM has finished, *routing[i].send[j]* equal to 1 means that peer *i* has to forward the message to peer *j*. If it equals to 0, then peer *i* will not send the message to peer *j*. After the algorithm's execution, since each peer has an ordered list of its neighbours according to their distance, once peer *i* has received the message, it will forward it to the first peer *j* such that *routing[i].send[j]=1*. After that, it will forward the message to the next peer *k* with *routing[i].send[k]=1* and so forth.
- *i*: the peer that sends the message at each step.
- *next*: the peer that receives the message at each step.
- *routing[i].index*: points to the closest peer to *i* which has not yet received the message. We understand as closest peer the one who has a smaller latency (transmission time add on propagation time) in front of *i*.
- *routing[i].tnext*: time at which if peer *i* sends a message, it will arrive at its closest peer, chosen among the unvisited peers. This receiving peer is *routing[i].index*.
- *imin()*: chooses the peer with lowest *routing[i].tnext*.
- *update_i()*: searches the nearest peer to *i* from the set of peers which have not yet received the message.
- *update_t()*: once *i* has forwarded the message, *update_t()* computes the next value for *routing[i].tnext*. That is, it subtracts from its previous value the last latency, and adds up to it the next latency plus its transmission time.
- *update_tn()*: the same as *update_t()* but it applies to a peer which has just received the message. To the time at which the peer receives the message, we add up the closet peer latency, chosen among the peers that have not yet received the message.

The algorithm operation is simple. At each step, SMM chooses the peer that has not yet received the message and has the lowest cost. That is, the unvisited peer that can be reached with the minimum time from any peer, among those that has already received the message. Once the message has been received by the new peer, the algorithm recalculates the arrival times of the remaining peers (considering that the new peer can forward the message immediately), chooses the peer with the lowest arrival time and forwards the message to it. The arrival time calculations are made under the assumption that when a peer finishes the message transmission to another peer, it begins immediately with another destination peer.

The SMM algorithm is very similar to Dijkstra's shortest path algorithm [19] with the difference that in EMPS (n, λ, μ) the time delay between two peers *p* and *q* is not constant. Actually, in EMPS (n, λ, μ) this delay is equal to λ_{pq} plus μ_p multiplied by the number of previous retransmissions of peer *p* (or, when the transmission time *p* varies according to the recipient, is equal to the sum of λ_{pq} and the total time of all previous retransmissions of *p*).

Consider the network depicted in Figure 3.5 where the edges weights correspond to the communication latency λ_{pq} (transmission time plus

propagation time) between nodes that the edge joins. For simplicity, we have considered that the transmission time is equal to one for all peers. We also assume that p_0 is the source peer. At $t=0$, p_0 sends the message to peer p_1 which receives the message at time $t=10$. At $t=1$, p_0 has its output link free and can send the message M to the next closest peer p_2, which will receive the message at time $t=11$. Likewise, for p_3 and p_4, the arrival times from the p_0 peer is calculated as $t=22$ and $t=23$ respectively, whereas from p_1 those arrival times are $t=20$ and $t=21$. So, in this case, the algorithm will forward the message to peers p_3 and p_4 from peer p_1.

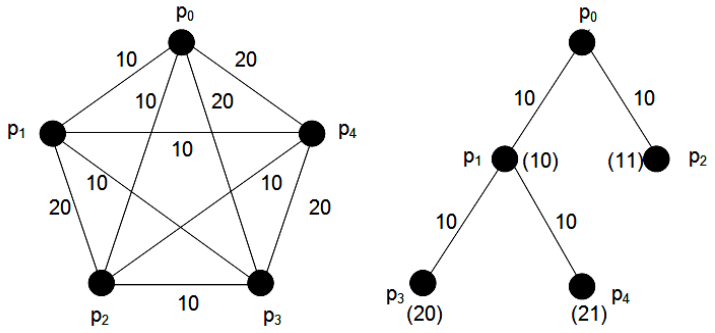


Fig. 3.5 SMM algorithm example in a full-connected network

Observe that if peer p_2 sends the message to peer p_4, the message also would arrive at $t=21$ instant. The selection of either p_1 or p_2 to send the message to p_4, depends on a strict comparison or not when the algorithm checks if the next peer has to be selected (i.e. the use of "<" versus "≤"). This consideration has an effect on the peers degree in the multicast tree, and in turn, on the peers load in terms of network computing. Although the effects on computing load are not the focus of our study, it seems clearly useful to preserve the minimum peers degree. This has a certain importance since in overlay networks the peers correspond to end-users devices.

We can prove that the multicast time achieved by the algorithm SMM is minimum, when for all peers, $\mu_p=0$. The proof is simple: in this case, since $\mu_p=0$ for all peers, the time delay between two peers p and q is always the weight λ_{pq} of the edge that joins them. Thus, the SMM algorithm corresponds to the optimal Dijkstras' algorithm of complexity $O(n^2)$. In a general case, however, the SMM algorithm is not always optimal. In Figure 3.6 we show a network where SMM is not optimal. On the left, we apply SMM with a result of a 7 multicast time. On the right, we apply another multicast transmission order and we show that multicast delay can be reduced to 5. Hence, if the source begins with peer p_3, follows with p_1 and finishes with p_2, the multicast delay will be 5. In the picture, the time that the packet reaches each peer is showed in brackets. Transmission time is also 1 for all peers.

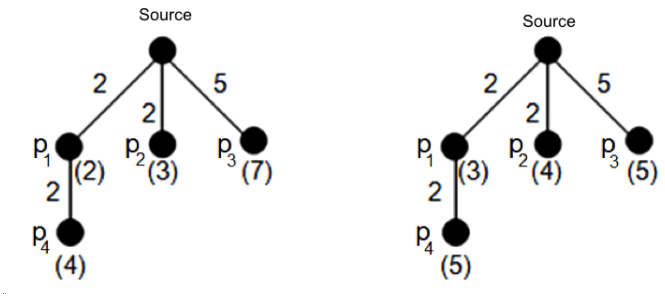


Fig. 3.6 Network example where SMM is not optimal

Nevertheless, for an overlay network we can assume that $\mu_p \ll \lambda_{pq}\ \forall p,q$ and thus consider $\mu_p \approx 0$. This means that in an overlay network the SMM algorithm may be described as near optimal (since for $\mu_p = 0$ the algorithm is optimum). Moreover, in the general case where $\mu_p \neq 0$ an optimal solution could be found by means of redefining the order of transmissions for each peer, using an exhaustive exploration (i.e. by analysing all the possible multicast trees). First of all we choose the peer where the root node sends the message for the first time, which gives *n-1* possible combinations. After that, we choose the second receiver peer among the *n-2* remaining peers, as well as the peer who sends the message, from one of the two nodes who have already the message (the root node or the first node who has received the message). Thus give a total of $2\cdot(n-2)$ possible combinations. For the next peer we can choose between *n-3* receivers and 3 transmitters, so we will have $3\cdot(n-3)$ possible combinations, and so forth until the last peer, where we can choose between *n-1* transmitters and just one receiver. Summarizing we have $1\cdot(n-1)\cdot 2\cdot(n-2)\cdot 3\cdot(n-3)\ \ldots\ (n-2)\cdot 2\cdot(n-1)\cdot 1=(n-1)!^2$ possible combinations, and thus $(n-1)!^2$ possible multicast trees. Once each tree has been completed, we calculate the total delay transmission of one message and choose the best possible tree. Anyway, the value $(n-1)!^2$ is not suitable in practise for high values of *n*. Even a *n=10* network will give $13{,}17\cdot 10^{10}$ possible trees, which leads to an unacceptable computational time.

At this point, we can prove that SMM algorithm for EMPS (n, λ, μ) has complexity $O(n^2)$. At each step, SMM searches the peer that has not yet received the message and has the lowest cost, that is, the peer that can be reached at minimum time. As the maximum number of unvisited peers is *n*, this operation requires at most *n-1* comparisons. Moreover, each iteration is executed once for every peer that receives the message, thus, the total number of steps equals the number of peers. So we have *n* steps and at each step we perform at maximum *n-1* comparisons plus some basic operations, resulting a complexity of $O(n^2)$.

3.4. Message Stream Multicast Algorithm MSM

The SMM algorithm was designed for a single-message multicast environment. This is a drawback since usually we desire to send more than a single message. So, SMM is not very practical and would only apply where the time difference between two consecutive messages is larger than the multicast delay. If, for example, we consider a real application-layer of multicast video streaming, we see that what we call message in EMPS (n, λ, μ), may be a video frame. In practise, two consecutive video frames are provided with a time difference equal to the inverse of the frames per second (fps), which usually has a value between 16 and 32 fps, depending on the video quality. Thus, the source must multicast two consecutive messages with an interval value between 31,25 and 62,5 ms.

The first approach to message streaming is to repeat indefinitely the routing table obtained with SMM algorithm, multicasting each message as if it would be completely independent of the others. That means that when one message finally arrives at all the multicast group members, the message source would proceed to multicast the next message, and so forth.

The total multicast time delay of the stream would be, in this case, the total number of messages *M* multiplied by the multicast SMM delay t_0 for one single message. The main obstacle of this solution is that the source cannot send the next message until the previous one has been received by all the group members, and this, may increase the message rate. As defined by the SMM algorithm, the source usually retransmits the message until the communications end, in order to flood before the multicast group and to minimize the total delay of a single message.

Next, we consider a new possibility for multicast transmission when we transmit more than one single message. Before the first message has arrived to all peers, the source could stop sending it and begin with the second message. With this restriction, the multicast time t_0 of the first message, individually considered, will be increased, but we will begin to send before the second message, and so the third, and the fourth, and so on. This saving of time between the sending of two consecutive messages will be progressively accumulated and, if the number of messages is large enough, it will compensate the increase of the first message multicast time t_0.

The modified algorithm, called MSM-s Message Stream Multicast Algorithm, will allow each peer to transmit *s* times the same message. At this point, the peer will start to send the next message, and so on, with the feature that it will be able to send the second message before the first has been received by all the network peers. The MSM-s definition is the same as SMM adding the restriction that each peer can send *s* times the same message, and this particularity is applied to successive messages as well.

Therefore the scheme in Figure 3.4 is for the SMM algorithm can be extended to MSM-s with the only difference that function *imin()*, will choose the next peer among the peers which have not yet forwarded the message *s* times or, as

shown in section 3.5, within the peers that have forwarded the message during a time lower than a certain value.

The number of retransmissions restriction could isolate some peers if we do not have full connectivity, as shown in Figure 3.7 when $s \leq 3$. In this case, the MSM-s algorithm should choose a minimum restriction number s to guarantee that all peers receive the messages. Moreover, when we restrict the number of transmissions for each peer, MSM-s has to take into account the packet or messages generation rate of the source. That is, if the source sends at most s times the first message and then, after $s \cdot \mu_r$ time units, stops the transmission of the first message to begin with the second one, we must be able to assume that the source has the second message ready to forward. This means that we must assume that the rate of the source is high enough to provide a new message each $s \cdot \mu_r$ time units. Otherwise, the source would stop sending the first message before having the second one and would remain unnecessarily idle, with the consequent loss of efficiency. Therefore, MSM-s has to choose a minimum restriction number s not only to avoid the isolation of the peers but also in order to avoid the source idleness. Hence, $s \cdot \mu_r$ cannot be much lower than the time spent between the generation of two consecutive messages.

To calculate the total transmission delay, we denote M as the number of messages transmitted, μ_r as the source transmission time and t_0 as the total single message delay. If the rate of all peers is greater than the rate of the source, will not be necessary store packets in queues (buffers) and t_0 delay will be the same for all packets. In this case, the total delay transmission, assuming that the source transmit the message $s_r(s) \leq s$ times, is:

$$T = (M-1) \cdot s_r(s) \cdot \mu_r + t_0 \qquad \textbf{(3.1)}$$

As mentioned above, limiting s will increase the t_0 delay for a single message; on the other hand, the first term will decrease. This reduction will compensate the t_0 increment if the number M of messages is large enough.

For a given network, if we apply MSM-s instead of MSM-(s+1), that is, if the maximum number of messages sent by any peer is s instead of $s+1$, then the first message will arrive later to all the destination peers (that is, t_0 will increase), but we will start to send the second message before, and so the third, and the fourth, and so on. We have already pointed that for every new message we will save a little time. In this case, if the number of messages is large enough, the increase of the multicast time for one single message will be compensated and thus MSM-s will be faster than MSM-(s+1). In [16] has been proved that under certain conditions it is possible to calculate a minimum number $M\sigma$ in such a way that if the number of messages is equal or larger than $M\sigma$ then MSM-σ is better than MSM-(σ+1).

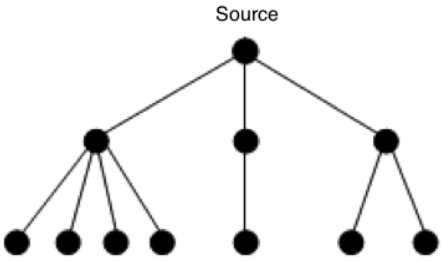


Fig. 3.7 A network in which MSM-s isolates some peers for $s \leq 3$

Finally we can demonstrate that MSM-s algorithm for EMPS (n, λ, μ) has complexity $O(n^2)$. The proof is the same as for SMM, since MSM-s performs exactly the same operations with the difference that at each step MSM-s has to limit the number of message retransmission, for each peer, to s.

3.5. MSM algorithm with time restriction

Figure 3.8 shows a peer q that has the same transmission time μ_p than a peer p that forwards to q the message. Let $s_p(s) \leq s$ be the actual number of times that p forwards the message for MSM-s. In this case the second message will be received at q with a time delay of $s_p(s) \cdot \mu_p$ time units respect to the first message, since the second message follows the same path but with a time delay, from p, equal to $s_p(s) \cdot \mu_p$. That is, p will send the first message $s_p(s)$ times and then, $s_p(s) \cdot \mu_p$ time units later, it will begin with the second and so forth. In this case, if we also limit the time transmission of q to $s_q(s) \cdot \mu_q$, and moreover $s_q(s) = s_p(s)$, then peer q will receive the second message at the same time that it finishes the sending of the first message for the last time. Although this is not important since we have usually full-duplex connections, it could be avoided if $s_q(s) = s_p(s) - 1$.

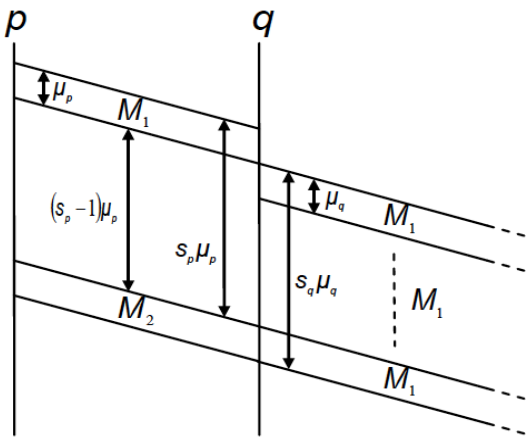


Fig. 3.8 Transmission of two peers p and q

In a more general case where peers has different transmission times, when the forwarding period $s_q(s)\cdot\mu_q$ of peer q is higher than the forwarding period $s_p(s)\cdot\mu_p$ of peer p, which is in a higher level of the multicast tree, the second message could arrive at peer q before it has finished forwarding the first message and then, the second message would have to be buffered, with the consequent time delay. This buffering delay would be accumulated by the third message and by the forth message and so on. So if we do not do anything, it can result in a message loss. This situation can be avoided by limiting the time period $s_q(s)\cdot\mu_q$ while each peer forwards the message. Thus, the forwarding time $s_q(s)\cdot\mu_q$ of each peer q must be less or equal than the retransmission time $s_p(s)\cdot\mu_p$ of any peer p that is in the path between the source and the peer q (including the source). That is equivalent to say that $s_q(s)\cdot\mu_q<s_p(s)\cdot\mu_p$. Furthermore, we do not want $s_p(s)\cdot\mu_p \gg s_q(s)\cdot\mu_q$ since in this case q would stop forwarding the first message long time before receiving the second one and the algorithm would lose efficiency or, in other words, it would waste bandwidth.

3.6. MSM algorithm with rate restriction

Since the MSM-s algorithm depends on some parameters (the number of retransmissions of the root node, or the transmission time limit for any of the other peers, as seen in previous section), we can apply some modifications over the original algorithm to improve its performance. Thus, all the algorithms defined in the present work are based on the definition of Figure 3.4, with some changes in the *imin()* function that chooses the next forwarding peer. In fact, these changes affect mainly two different points: the condition that considers one peer better than other to be the next to forward the data; and the strategy that we follow when, under certain conditions, the algorithm cannot find any node to send the data and as a consequence total connectivity is not reached.

The first algorithms that were defined in previous projects [13,14,15,16,17] were the following. We first call *Cadence* the MSM algorithm with time restriction, as it has been specified in previous section. In this case, the limit time b_0 is defined as the value of s multiplied by the mean of all the root transmission times (note that the transmission time between the root and any other peer is not always the same, since it depends on the bandwidth of the links of the path between the root and the peer). The root can send the packet at most s times, while the other peers will be able to send the packet while their rate time t_{cad} (this means, the time during a peer forwards the same message) is lower than b_0. Note that, with this algorithm, depending on the value of s and the link bandwidth, some peers can be isolated (for example, if one peer has a very slow access, and the time to transmit the packet to it is higher than b_0), resulting an incomplete tree. Also, we can find some cases where a node fulfils the condition for its rate time, but this time is higher than the rate time of the root. For example: set s to 5, and the value of b_0 to 50 ms (because the average μ of the root node is 10); then, if the root sends the packet to the nearest peers, whose μ is 9 for all of them, we can have a total rate time for the root of 45 ms. As the time limit is b_0, some other peer can have a rate time of 48 ms, which fulfils the restriction of being lower than b_0. Anyway, this rate time is higher than root rate, which could result in a congestion issue.

Another algorithm designed in previous works is the *Forced leaves*. In this case, the source can send the message *s* times, and also the b_0 limit is imposed to the rate time t_{cad} of each peer. Similarly, peers can send the packet as long as its rate time is less or equal to b_0, but this time, if some peer does not fulfil the retransmission condition, and there are still isolated peers, the algorithm forces one or more peers (always chosen to minimize the arrival time of the messages) to send the message once, among those who have not yet retransmitted the message. This algorithm allows the generation of complete trees, but now some peers can exceed the limit b_0 and also the rate time of the source, causing congestion problems (unless we limit the rate of the source to the rate of slowest rate peer in the network, a discussion that we will state later). With the name *Forced all peers* is denominated an algorithm quiet near form the previous one, with the difference that if there are still peers for receive the message, any peer can be forced to retransmit the information (and not only those who have not yet retransmitted the message).

The algorithms that follow the previous ones are listed next. They were developed during a previous project [17] and our work consists on improving them for real-time multicast. A common point among all of them is that they are defined according to two conditions: first, a restriction criterion and, after that, a selection criterion. All the next algorithms have the same restriction criterion. In this case, we allow the source to send the packet up to *s* times, while the rest of peers can forward the message as long as its rate time t_{cad} does not exceed the source rate time at that concrete time (this will be modified in our work, as it will be explained later in section 3.7). The selection criterion varies according to the algorithm.

Optimum t0 (optt0): In this algorithm, as in previous ones, the source can send any message at most *s* times (in practice, we can increase the value of *s* until we get full connectivity). At each step, the algorithm chooses as a transmitter the peer that has already received the message and can reach its closest neighbour at minimum time. In this case, we can apply an additional restriction in such a way that a peer will be allowed to retransmit the message only if its total rate time t_{cad} (which, as mentioned, is the time that a peer has transmitted the same message) is less or equal than the rate time of the source. If we assume this restriction we will avoid congestion problem, since the source will have the highest rate time of the network, and thus all the peers will have finished sending a message before receiving the next.

Optimum rate (optcad): This algorithm is quite similar to the optt0 algorithm but now we change the selection criterion that chooses the next forwarding peer at each step. Now, selection criterion tries to minimize the amount of time that a peer transmits the same message, instead of minimizing the arrival time on the next peer. Therefore, we choose as a transmitter peer the one who has accumulated less rate time t_{cad}. Thus, limiting the rate time of the different nodes, the next messages could be sent earlier and if the number of messages is large enough, we will get a smaller total delay. In this case, as for optt0, we can optionally allow a peer to retransmit the packet only when its total rate time t_{cad} accumulated is less or equal than the rate time of the source.

Optimum rate version 2.0 (optcad2): This algorithm is a redefinition of the above algorithm, optcad. In this case, the selection criterion does not choose as a transmitter peer the one who has accumulated less transmission rate t_{cad}, as in optcad. Now we choose the peer who will accumulate a lowest rate time if transmits the message. That is, optcad2 takes into account the future transmission while the original optcad does not. As for both optt0 and optcad, now we can also apply the restriction criterion for avoiding congestion.

3.7. MSM algorithm for real-time transmission

The algorithms presented so far have been focused on minimizing the time rate t_0 or t_{cad}, depending on the algorithm applied (optt0 and optcad as well as their evolutions, respectively), in order to reduce the total delay when M is large enough. However, this sometimes does not consider real-time transmission. For example, if we get a time rate of $t_{cad}=12\ ms$, but, the source generates a packet each 23,4375 ms (corresponding to a 512 Kbps rate if every packet has a length of 1500 bytes), we will lose bandwidth since the source will transmit the first packet in 12 ms and then it will remain idle for almost 12 ms before having the second packet ready to transfer.

Since the aim of the present project is to carry out real-time transmissions in an optimal way and the algorithms presented so far do not consider real-time transmissions, now we propose a new algorithm. First, we define the packet transmission period T_p (for simplicity, we will refer to it as the *packet period*) as the time that the information source needs to produce two consecutives packets. Thereby, considering that we have a real-time communication scenario, the source starts to send each packet when it is ready, and so the second packet will be sent T_p seconds later than the first one and it will reach the destination nodes with the same delay. We consider that the network works with a good performance and it does not have congestion problems.

The algorithm proposed follows the same model as the former algorithms: it has a restriction and a selection criterion in order to allow a peer to transmit and choose the next peer that will transmit each packet. Since now it has no sense to minimize t_{cad} as explained before (it has no sense to achieve a t_{cad} of 12 ms if we have to wait 23,4375 ms for the next packet, due to the information rate), we will try to minimize t_0 and then we apply the same selection criterion than optt0. Moreover, we use a new restriction criterion: choose the next transmitter node only among those peers who will return a rate time equal or less than the period packet T_p, in other words, between those nodes whose time rate is finally lower than the time elapsed between the creation of two consecutives packets.

CHAPTER 4. ALGORITHMS APPLICATION

In the previous chapter we have described the fundamental aspects of the routing algorithms, aim of our study. Next, we present the procedures and tools used to perform and analyse the behaviour of the algorithms over a simulated network. With this purpose, we use the Transit-Stub model, described in the first chapter, in order to simulate Internet. Internet is not the unique scenario for real-time multicast transmission. However, it is the most common environment where we can find this sort of communications. So, from now on, we will talk either about Internet or backbone network. Furthermore, on the network generated using a Transit-Stub model, we have added a set of user nodes with different link speeds.

4.1. Backbone network

As we described previously, to simulate an overlay network first of all we have to define a backbone network. The backbone network that we want to simulate is Internet, a network without general topological constrictions. Thus, nobody can give us today a fully detailed map of the Internet. Fortunately, some aspects of the Internet topology are easier to capture than others. Hence Internet is a collection of Autonomous Systems (AS) [20], where each AS may be understood as a collection of connected IP routing prefixes, under the control of one or more network operators that presents a common, clearly defined routing policy to the Internet.

Summarizing, on the Internet context, an AS is the unit of router policy, either a single network or a group of networks that is controlled by a common network administrator (or administrators) on behalf of a single administrative entity (such as a university, a business enterprise, a business division or an ISP). Networks within an AS communicate routing information to each other using an Interior Gateway Protocol (IGP). An AS shares routing information with other AS using the Boarder Gateway Protocol (BGP). Previously, the Exterior Gateway Protocol (EGP) was used. In the future, is expected to replace BGP with the OSI Inter-Domain Routing Protocol (IDRP).

There are several backbone network simulators. Some offers simulations closer to the Internet topology than others. The one considered in the present project is the *Georgia Tech Internet Topology Models* GT-ITM [21], a group of applications which uses both a random and a Transit-Stub model. In our case, we have used the Transit-Stub model. The GT-ITM group of applications package works with a data file, which contains all the backbone network parameters. In our case, we have also employed other tool in order to convert the output file of the GT-ITM application (“.gb”) to an “*.alt” file, with intelligible information. The resulting file contains the graph information (nodes, edges and weights). The topology of this graph is similar to the real Internet topology, with a set of nodes acting as routers on the backbone network and high-speed links

that interconnect the routers to the stubs. These stubs represent the ISP or corporative LAN networks connected to the backbone network. Finally, we have added to the ISP network the user nodes, which form the multicast group of the P2P network.

On the present project we have work with two sorts of backbone network topologies. The first one simulates a network formed by a single Autonomous System and hence it simulates just the network of a single ISP. The second backbone network simulation are composed by multiple Autonomous Systems and therefore simulates in a closest way the Internet topology, which is built by a collection of AS.

4.1.1 Single Autonomous System simulation

As it has been explained, the Internet topology is formed by a group of AS, where each AS represents a single ISP. So, a simulation with a single AS is a simulation quite far from the real Internet network topology. However, we have work with this sort of backbone network in order to compare the results of the new routing algorithms presented on section 3.7, with the ones presented on previous works [13,14,15,16,17].

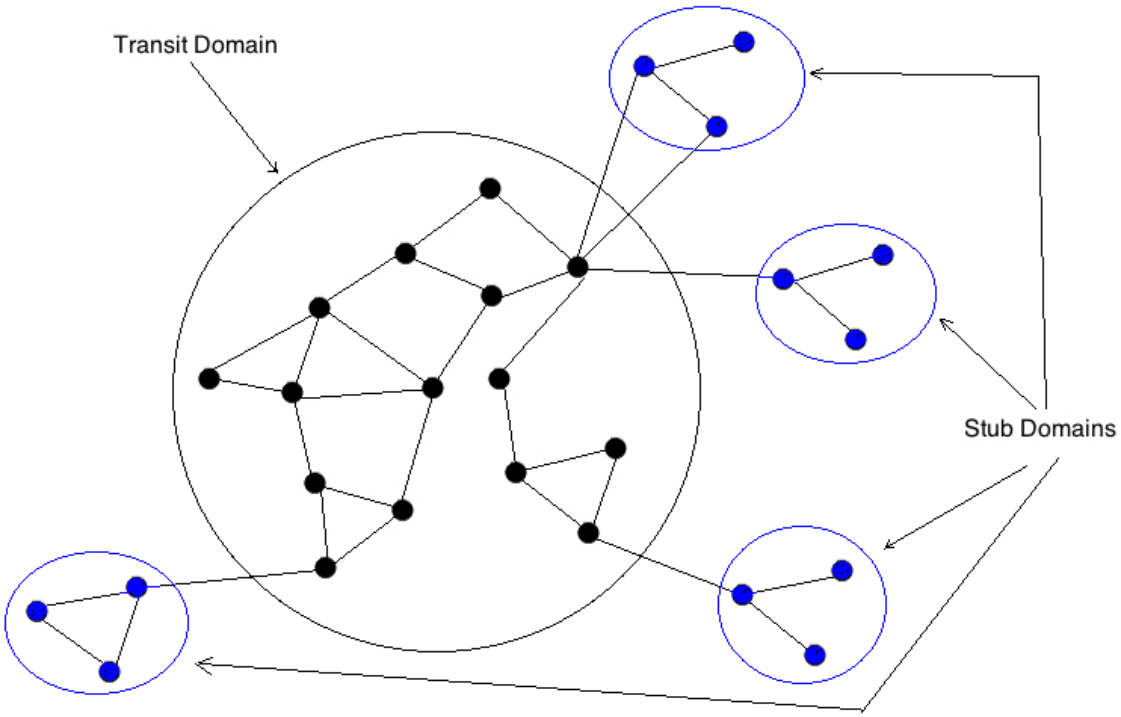


Fig. 4.1 Single Autonomous System simulation graph

4.1.2 Multiple Autonomous Systems simulation

For the present project we introduce the simulation of a backbone network as a set of multiple Autonomous Systems. This is a scenario much closer to Internet topology than the one presented in the previous section.

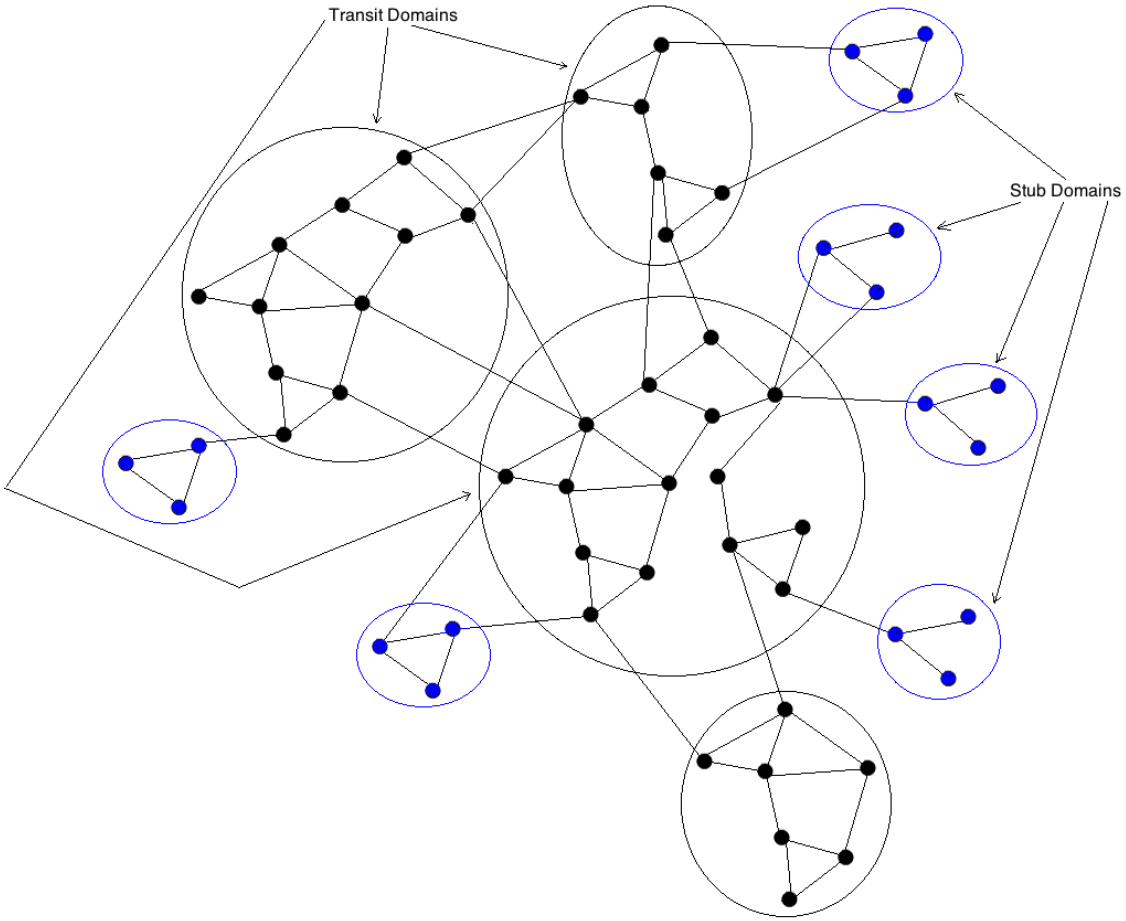


Fig. 4.2 Multiple Autonomous System simulation graph

4.2. MSM algorithm simulation

Once the backbone network has been depicted, we use an application created in previous projects [13,14,15,16,17] to design the overlay network and apply the routing algorithms over it. The application, named MSM or MSM Algorithms Simulator, which is developed in Java [22] and RCP Rich Client Platform [23], has been used to obtain the backbone graphs as well as the user nodes that, finally, will form the overlay network.

After that, we have used two more applications. The first of them is the Dijkstra's algorithm, also developed in Java. The result of this application is an overlay graph which contains the user nodes that forms the multicast group and the corresponding delays to send a packet from one to another node, that is,

transmission time, propagation time, and also the sum of these two variables (called *latency*).

The second application reads the overlay graphs created with the previous application and, then, executes the routing algorithms. This application is programmed in C. On every execution, the program returns three files. The first is a ".res" file extension which contains the execution results. The second file just contains the delays t_{cad} and t_0 for each value of s in ".csv" format. Finally, the program returns a ".res" file with the routing tree obtained as well as the arrival and departure times of each message between two consecutives nodes of the tree.

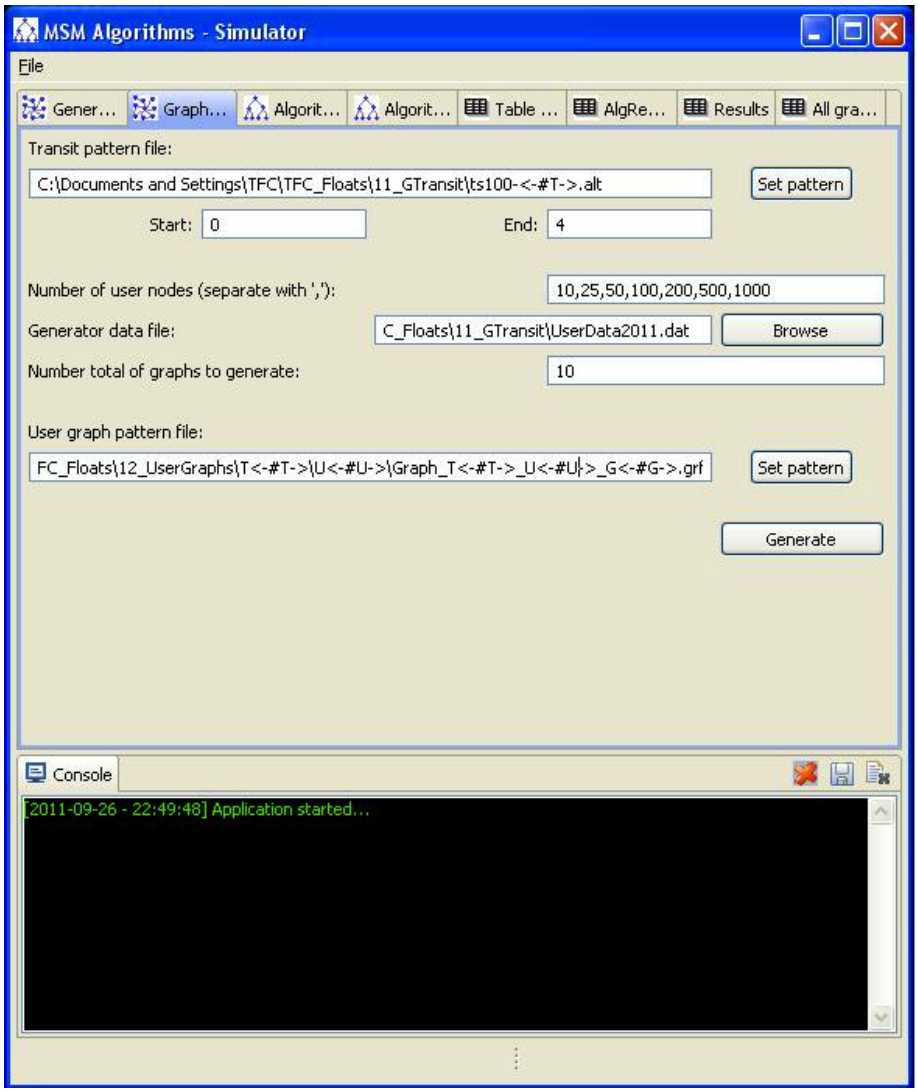


Fig. 4.3 MSM Algorithm Simulator user interface

The node access rates to the network have been updated respect previous works, as shown in next chapter. Moreover, we have modified both codes, the code for Dijkstra's application that generates the user graphs and the C

application which executes the routing algorithms. Initially, these codes worked with milliseconds units represented by integer values, which in the old stages were enough. However, in our case, by including faster links, the access rates obtained goes up to 20 Mbps, that involves transmission times of 0,6 ms (considering a packet size of 1500 bytes). In that case, it does not make sense to round the value to an integer value.

4.2.1. User graphs

As it has been said, to represent the overlay network it is necessary, first of all, to define the backbone network graph and, after that, to add the user nodes. These second task is carried out using the MSM-AS application, which works with an ".alt" file that contains the backbone network. The application adds *n* nodes (acting as multicast members) to the backbone network and connects each one of them to just one stub node. According to Figure 4.3, the input data required by the application are: the backbone network graph (".alt" file), the user nodes and a data generator file (".dat" file). The ".dat" file contains the statistics of the user nodes that form the multicast group, in particular, the links bandwidth, and the probability of having each kind of link. On section 5.2 we specify these values with more detail.

This information establishes the parameters of the application, which finally adds each peer to the network. First, we choose a node among the stub nodes of the backbone network. This selection has been done considering an equal probability for all stub nodes. In this way, we ensure that all user nodes are connected to one access network, since, as we remarked in the first chapter, the stub domain represent the access networks. Once we have the stub node, we add the new user link, with a bandwidth value chosen randomly according to the information file ".dat". As a result, the application gives a ".grf" file, which adds the user nodes to the initial backbone network (conformed by 100 nodes for single AS simulation and 1000 nodes for multiple AS simulation). When we have the ".grf" files, we can use the applications described below in order to obtain the overlay graph of the multicast group.

4.2.2. Overlay graphs

The next step consists of depicting the overlay network. With this objective, we have used an application that obtains the overlay network from the initial network (".grf" file), and returns a graph formed just by the user nodes and the distances between them (measured in transmission and propagation times). The resulting files ".ovr" allows the execution and the analysis of the routing algorithms.

This application reads the ".grf" file previously generated and applies, over it, the Dijsktra's algorithm in order to obtain the shortest path between any pair of users, from the delay (propagation plus transmission) point of view. In our case, the total propagation delay is calculated as the sum of all links that we pass-through when we go from one node to other one. The transmission time

between two users is calculated as the transmission time of the slowest link. The result is a complete graph, formed by the user nodes and represented with a square matrix of ordered nodes. For each node, we define a list of neighbour nodes, ordered by transmission time or latency (the sum of the transmission and propagations times), depending on the routing algorithm that we apply later: the optcad algorithm and its redefinition use the transmission time ordination, whereas the optt0 algorithm and its redefinition use the latency ordination. Figure 4.4 shows the matrix structure for a node belonging to a 10-user graph, differentiating both ordinations cases.

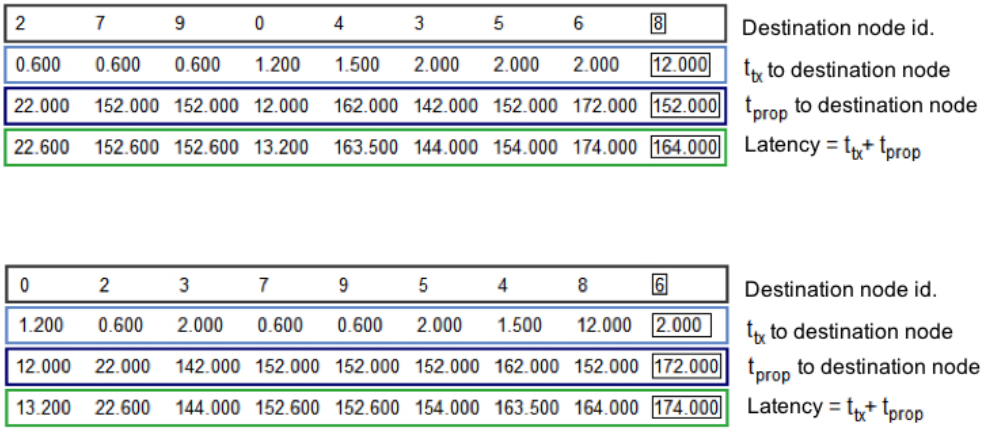


Fig. 4.4 P2P overlay network node example

4.2.3. Algorithms simulation

Once the overlay network has been described, we can execute the routing algorithms. With this purpose, we have used an application that executes, over the overlay graphs, the algorithms described on the previous chapter (the two redefinitions of the optt0 and optcad2 algorithms). This application gives as a result three information files, which contains the delays, the nodes affected by congestion (no one in our case do to restriction included by us for real-time transmissions) and the multicast trees.

First of all, this application determines the root node, which is a random selected node among those capable of transmitting (that is, with a bandwidth higher than the source rate). We have to indicate to the application the range of *s* values that limits the number of retransmissions of the source (which depends on the number of users) as well as the routing algorithm that will be executed (mainly optt0 redefinition and optcad2 redefinition). In our case, we have developed the algorithms in a ".c" extension file, which has been compiled and executed on a Linux operating system.

CHAPTER 5. ALGORITHMS EVALUATION

In this chapter we present the parameters that we have considered to define our simulated networks as well as the results, once we have applied on them our multicast algorithms. We evaluate them, mostly, in terms of delay and percentage of nodes which receive the information and we compare them with former results [17].

5.1. Transit network parameterization

As it has been specified on section 4.1, the first step to simulate an overlay P2P network consists of depicting the transit or backbone network. We have used the GT-ITM tools to build five network cores based on the Transit-Stub model, according to the two sets of parameters listed on Table 5.1.

The first group of parameters simulates a single Autonomous System. Though this is not the proper scenario to simulate Internet –since Internet's topology is made of multiple AS– we have consider these parameters in order to compare our results with previous works.

Additionally, one of the objectives of our study is to simulate a backbone or transit network closer to the actual Internet network, where multicast transmissions are usually carried out. Thus, we have considered a new group of parameters with the aim of simulating multiple ISP networks, where each AS represents the network of an ISP.

As it can be observed on Table 5.1, both nodes and transit domain parameters have been increased, resulting on a new network with $5 \cdot 8 \cdot (1+3 \cdot 8)=1000$ nodes instead of the $1 \cdot 4 \cdot (1+3 \cdot 8)=100$ nodes of the first network. Moreover, network bandwidth has been updated to actual link capacities [24] and propagation delays have also been modified in order to represent the actual Internet scenario in a more realistic way. To obtain the propagation delays let's see an example. Consider two routers (i.e. transit nodes) between Europe and United States. We can assume that the distance to go from one to another point is 5000 Km approximately. This is probably one of the longest existing links between two AS. So, in the worst case, the propagation delay is:

$$\text{Propagation delay} = \frac{d}{v} = \frac{5000\ [\text{Km}]}{0{,}7 \cdot 3 \cdot 10^8\ [\text{m/s}]} = 23{,}8\ \text{ms} \approx 25\ \text{ms} \qquad (5.1)$$

where d is the distance between the nodes and v is the wave propagation speed. We consider $v = 0{,}7 \cdot c$ where c is the speed of light. This resultant value determines the propagation delay between two transit nodes.

Table 5.1. GT-ITM parameterization

	Single AS	Multiple AS
Transit Domains	1	5
Transit Nodes (T) per Transit Domain (average)	4	8
Stub Domains per Transit Node (average)	3	3
Stub Nodes (S) per Stub Domain (average)	8	8
Link Probability (between nodes in the same Transit domain)	0.6	
Link Probability (between nodes in the same Stub domain)	0.42	
Transit-Stub extra links	0	
Stub-Stub extra links	0	
Transit-Transit bandwidth	1 Gbit/s	10 Gbit/s
Transit-Transit propagation delay	100 ms	25 ms
Transit-Stub bandwidth	100 Mbit/s	2,5 Gbit/s
Transit-Stub propagation delay	10 ms	10 ms
Stub-Stub bandwidth	100 Mbit/s	2,5 Gbit/s
Stub-Stub propagation delay	10 ms	5 ms

5.2. Graph creation

In order to create the overlay network –as an enlargement of the Transit-Stub model– where the algorithms will be tested, we have applied the generator described in section 4.2. In this case, we have considered the access rates published in the thirteenth AIMC quiz [25] over Internet ADSL users, released in 2011. The ADSL users connected to an ISP network represent our peers. The peer access rates and its percentage are detailed in table 5.2. As it can be seen, all the nodes have an access rate higher than 512 Kbps, which is considered good enough to transmit information in real time. Anyway, the quality of service (QoS) of the communications will rely on the codec used by the multicast user applications. This will be discussed in more detail in section 5.3.2.

We have considered a propagation delay for the links connecting the peers to a stub node equal to 1 ms because user and access nodes are usually close. In the table, the link rates are symmetric, unlike what happens in practise with ADSL. Anyway, the progressive increment of the upload speeds mixed with the use of the high-speed links, as for example the FTTH (Fiber To The Home), makes this restriction unimportant.

Table 5.2. Access rates (AIMC quiz, 2011)

Link base	Absolute users	%
It is not known	3229	8,2
512 Kbps	507	1,3
1 Mbps	2303	5,9
2 Mbps	1311	3,3
3 Mbps	5157	13,2
4 Mbps	1085	2,8
6 Mbps	10977	28
8 Mbps	621	1,6
10 Mbps	6192	15,8
12 Mbps	2871	7,3
20 Mbps	3687	9,4
More than 20 Mbps	1162	3
Do not know/No answer	47	0,1

5.3. Results and comparisons

Next, we present the results, comparisons and analysis of the algorithms applied over the single Autonomous System backbone network. As it has been said previously, this is not a very realistic scenario. However, we have used it in order to compare our results with previous works. After that, the results and analysis of the algorithms applied over multiple Autonomous Systems backbone network are introduced.

5.3.1. Algorithms execution over simulated IP networks (Transit – Stub model)

With the values presented on Table 5.2 we have generated, for each of the 5 backbone networks, 10 overlay graphs of 10 users each one, representing a total of 50 networks with 10 user nodes. After that we have repeated the same operation for 25, 50, 100, 200, 500 and 1000 users, respectively. In total we have obtained 50 networks of 10 users, other 50 of 25 users and so on. In order to calculate the transmission times we have assumed that the packets transmitted have a size of 1500 bytes.

After that, we have applied on each network the former optt0 and optcad2 algorithms (section 3.6) and their respective evolutions for a real-time transmission (section 3.7). Remember that the difference between the former and the actual algorithms is that now we have real-time transmissions, and then we restrict the rate time t_{cad} of each node to the packet period. This means that each node has to send each packet a period of time lower than the packet period, that is, than the time elapsed between the generation of two consecutive packets.

We present now the statistics of applying the arithmetical mean to the 50 overlay networks (10 overlay networks per each one of the 5 backbone networks) created for each number of users. The variables studied and analysed are listed and described below:

- t_0: is the transmission delay of a single packet, that is, the time elapsed since a packet is injected into the network by the source until it is received by the last node in the multicast group. In real-time transmission t_0 represents the delay of the communication, that is, the time elapsed between the event and its reception at home.

- t_{cad}: the rate time is the time that a node forwards the same packet to one or several nodes. For the network, we select the time of the node with a higher rate time, since this node will limit the rest of the nodes if we want to avoid congestion, as explained in former chapters.

- s: is the number of times that the root sends a packet.

- Covered nodes: the number of reached nodes by the algorithm. Note that since we restrict the rate time of each node, we cannot assure full connectivity and we can have isolated nodes at last, that is, nodes which do not receive the information.

- Worst case: if exists, the number of nodes covered in the worst scenario or overlay network.

- Total connected networks: if exist, the amount of scenarios where all nodes are reached.

Both t_0 and t_{cad} times are expressed in milliseconds (ms).

5.3.2. Single Autonomous System backbone network emulation

5.3.2.1. Former algorithms from previous projects

As it has been explained, the real-time algorithms proposed in this work are an evolution of algorithms whose intention were to minimize t_0 or t_{cad} depending on the algorithm applied, oppt0 and optcad2 respectively. This is the reason why a brief explanation an analysis of the results obtained by them is exposed next.

According to the optt0 definition given on section 3.6, optt0 tries to minimize t_0 by selecting as the next transmitter the node which obtains a lower partial t_0. In an analogous way, the optcad2 algorithm tries to minimize the rate time t_{cad} of the nodes by selecting as the next forwarding node that one which gives us back a better t_{cad}.

That was the original description of the algorithms. However, when applying them some nodes were affected by congestion. This is the reason why a

redefinition of the algorithms was implemented by introducing a common new restriction for both of them. This new restriction consists of limiting the possible transmitter nodes to those which has a t_{cad} lower than the t_{cad} accumulated by the source at that time. We refer to these algorithm redefinitions as optt0 and optcad2 “with restrictions” while for the original ones, we talk about optt0 and optcad2 with “no restrictions” (see Table 5.3).

5.3.2.2. Real-time transmission algorithms

The principal evolution of our new algorithms consists in restricting each peer rate time t_{cad} to the period packet time T_p of the source. This means that any peer will be able to transmit a packet only if its rate time t_{cad} is less or equal than T_p. So, finally, for optt0 and optcad2 evolutions we have:

- As a restriction criterion, we apply the new method just explained on the previous paragraph to carry out real-time transmissions. This new restriction substitutes any other restriction applied earlier. As explained before, this restriction avoids any possibility of congestion, since each node will finish the transmission of each packet before receiving the next one.
- As a selection criterion, we apply the same as described in section 3.6, that is, we will chose at each step the peer that returns a better partial t_0 and t_{cad} for optt0 and optcad2, respectively.

According to section 4.2.2, each node of the overlay network has a sorted vector of neighbours that is used to determine the next receiver node at each step. For optt0, this vector is sorted in function of the latency time (t_{tx}+t_{prop}) while in optcad2, an arrangement by t_{tx} is applied. During the optt0 algorithm application, some node evaluations may be truncated if we have a neighbour whose t_{tx} overcomes the maximum t_{cad} limited by the period packet T_p. But, due to latency sorting, the next node of the neighbours’ vector may have a lower t_{tx} time (if its latency time is dominated by a high propagation time that makes the final latency worse than its predecessor node, while its t_{tx} is lower). So, the optt0 algorithm has to take this possibility into account. That is, if it finds in the neighbours’ vector a possible receiver which does not accomplish the rate restriction, then it has to check the next node in the neighbours’ vector and so forth.

Since we are considering real-time transmission we select a codec of at least 256 Kbps, which is considered the minimum rate able to offer a real-time transmission with acceptable quality. This codec determines the transmission capacity of the source node, which at the same time determines the period packet time T_p defined in section 3.7 as the time that the information source needs to produce two consecutives packets.

Given a 512 Kbps codec and considering packets of 1500 bytes, the source will generate a packet each T_p=23,4375 ms. For a good real-time transmission, the destination node must receive the packets at the same rate fixed by codec. In our case, this means receiving a packet every 23,4375 ms in order to allow the

destination node to rebuild the streaming flow properly, without interruptions. Then, a minimum link bandwidth equal to the rate of the codec is required. Moreover, if we assume ideal backbone networks, congestion will not appear because any node will finish forwarding each packet before the reception of the next one, T_p seconds later, and so every peer will receive the information at the source rate.

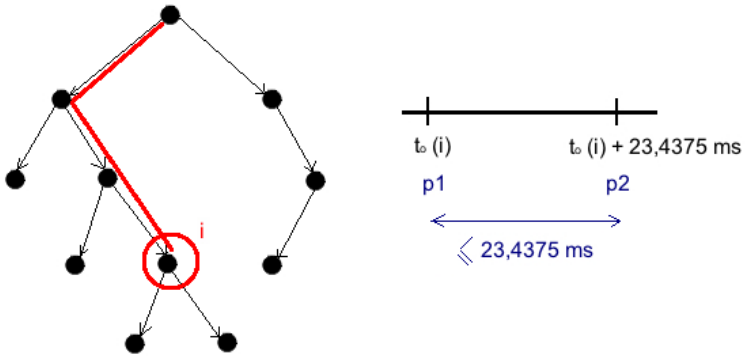


Fig. 5.1 Algorithm behaviour - Congestion avoidance

To understand better the congestion avoidance achieved by these algorithms, see Figure 5.1. Assuming $t_0(i)$ as the partial t_0 time of the node i, this node can forward the same message to its neighbours while $t_{cad}(i)$ does not reach 23,4375 ms or in, other words, before $t_0(i)$+23,4375 ms, instant at which the second packet will arrive to i. With this restriction, no queues are needed on nodes because when the second packet is received it may be processed. Hence, no congestion occurs on network.

Also, a clarification of forwarding process for a node is exposed in Figure 5.2.

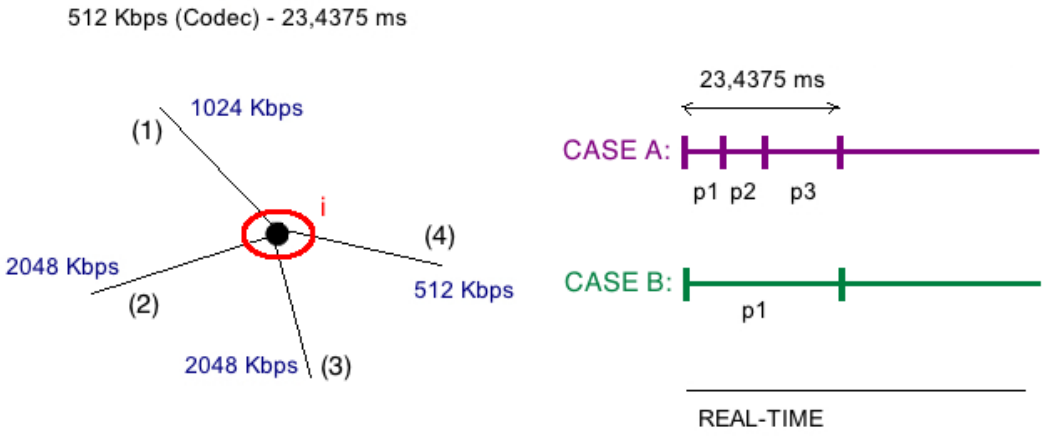


Fig. 5.2 Algorithm behaviour – Codec vs. bandwidth

Suppose that the node *i* can forward the same message to a determined number of nodes while its $t_{cad}(i)$ does not exceeds T_p. The number of nodes to where the message is transmitted during T_p is limited by the bandwidth of the links evaluated. In case "A" we observe that the packet is forwarded up to 3 times. This is possible due to links 2 and 3 use a quarter of the T_p available time because both links have a bandwidth of 2048 Kbps. This rate is 4 times higher than the 512 Kbps rate of the codec, so the transmission time needed to transmit a packet over these links is 5,8594 ms. Besides, the link bandwidth of 1024 Kbps offers a transmission time of 11,7187 ms, which means the half of the T_p time. In contrast, case "B" (link 4) allows node "i" to transmit only one packet because the link has a bandwidth of 512 Kbps, the same as the codec.

5.3.2.3. Comparison, analysis and results

Some multicast P2P applications require real-time communications. As it has been said, the former algorithms definition does not consider this requirement. So, in most cases, they do not fulfil the characteristics required by this sort of communications: sometimes by introducing congestion and other times by exceeding the codec rate.

Formerly, the results obtained were filtered according to the T_p time given by a codec of 512 Kbps and considering packets of 1500 bytes. This means that any node of the network could not have a t_{cad} time higher than the period packet T_p=23,4375 ms, determined by the codec. Networks with a t_{cad} exceeding the T_p were directly eliminated because they would not support a real-time traffic flow. In this case, packets would not arrive to some nodes at the rate at which they were generated causing a bad rebuilding of the streaming flow and, additionally, congestion would appear on network. So, the arithmetic means were calculated only with the compliant networks, that is, the trees with a rate time t_{cad} lower than T_p. We do not show the results for optcad2 since we consider now t_0 as a comparing parameter, and optcad2 were designed for minimizing t_{cad}.

The results of this filtering operation are presented in Table 5.3, where we can differentiate the results obtained by optt0 algorithm with and without restrictions. Note that as the number of users increases, the number of compliant networks decreases exponentially. This is because the original optt0 algorithm is designed for minimizing the t_0, which implies a deterioration of the rate time t_{cad}. That is, if we increase the number of users, we need more packet retransmissions and then the rate time t_{cad} also increases, resulting in more cases with a t_{cad} higher than T_p.

If we analyse the behaviour of both versions of the original algorithm, we realize that better results are obtained without restrictions. This was expected and has a simple explanation: algorithms with no limits have more freedom and have been able to find trees with lower delays. Anyway, these results are not significant due to the limited number of compliant networks.

Table 5.3. Real-time transmission algorithm results

		Number of users *n*					
Algorithm	**Parameters**	**10**	**25**	**50**	**100**	**200**	**500**
optt0 - no restriction	**Available networks**	40	34	2	1	0	0
	t_0	273,91	267,62	212,19	109,13	-	-
	t_{cad}	19,42	21,63	22,33	23,44	-	-
optt0 – with restriction	**Available networks**	43	34	16	9	12	9
	t_0	291,43	303,56	312,70	325,16	338,27	436,64
	t_{cad}	19,22	21,33	22,25	21,58	23,44	23,44
optt0 evolution	**Available networks**	47	50	49	49	48	47
	t_0	268,52	303,84	310,92	303,19	310,04	323,87
	t_{cad}	18,75	22,33	23,16	23,22	23,36	23,44
	Covered nodes	9,85	24,58	48,86	98,43	196,33	491,91
	Covered nodes (%)	98,51%	98,32%	97,71%	98,43%	98,17%	98,38%
	Worst case	9	22	46	96	193	486
	Total connected networks	40	32	13	6	1	0
	Total connected networks (%)	80%	64%	26%	12%	2%	0%

At this point starts our development by introducing the modification presented in section 5.3.2.2. Remember that this one consists of restricting each peer rate time t_{cad} to the period packet time T_p of the source. Thus, any peer will be able to transmit a packet only if it has forwarded this same packet less than T_p seconds.

As it can be observed in last section of Table 5.3, after applying the restriction for transmitting in real-time almost all the networks are available. Moreover, the ones that have not been evaluated are not considered because the source node has not enough bandwidth to transmit the information at the codec rate. An example for that situation is the scenario where all links from the source node have a maximum bandwidth of 256 Kbps, while the source codec rate is of 512 Kbps. In this case, when the second packet is ready (T_p reached), only the half of the first packet has been transmitted and therefore a real-time transmission is not possible. In these scenarios, any packet will overcome the t_{cad} restriction for real-time transmissions and therefore no packet will be transmitted.

From Table 5.5 we see that, in general, t_0 and t_{cad} increase with the number of nodes. These results were expected, but the transmission delay t_0 does not increase significantly due to more users implies more transmissions, but also more transmitting resources. In this context, the value of t_{cad} is irrelevant and never exceeds the value T_p= 23,4375≃23,44 ms because the rate of the data emission is fixed by the source node itself.

Moreover, the percentage of covered nodes is high. Approximately, 98% of the nodes are reached independently of the size of the network. This implies that the second modification presented in previous section to reach the maximum number of users works properly. Although the percentage of the covered nodes remains almost constant for all network sizes, the total connected networks

percentage decreases with the number of nodes. Anyway, this is not a big deal since we are interested in connecting the maximum number of users as possible and although not all users are connected, getting a 98% of connected users can be considered good enough.

5.3.2.4. Real-time transmission algorithms tuning

For simplicity, all the algorithms presented up till now select as a source node the first user of the overlay network, which always corresponds to the node number 100 (for a single AS, nodes from 0 up to 99 correspond to backbone nodes). Anyway, if this source node has not enough bandwidth to transmit the information at the codec rate, the network is not compliant, as explained before. So from now on we choose as a source a user connected to the network by at least one link with a bandwidth equal or higher than the codec rate.

Once this new condition has been considered, the results obtained by optt0 evolution algorithm are presented in Table 5.4. Note that the concept “available networks” has disappeared from it. This is because after applying the new restriction, all networks can support real-time transmissions. In the table, the *s* parameter represents the average number of retransmissions done by the source node.

In Table 5.4, results are evaluated for different codec rates from 256 to 4096 Kbps. The same overlay networks have been used in order to compare the results obtained by the different codec rates. This means that the source node must have at least one link bandwidth equal or higher than 4096 Kbps, which is the most restrictive codec. In this case, for 1500 byte packets it results T_p=2,9297≃2,93 ms.

Table shows that the total delay t_0 does not increase significatively with the number of users. This is explained by the fact that although more users have to be achieved, the network also has more available resources. That is, more nodes retransmit at the same time, which finally is translated only into a low increment of t_0. The differences on t_0 time values are minimal between bandwidths from 256 to 1024 Kbps. More appreciable differences are observed when codec rate is augmented to 2048 Kbps or, especially, when it is raised to 4096 Kbps. Those differences are not significant, but they exist because an increase in the codec rate implies a more limited algorithm. In these cases, the rate time t_{cad} for node retransmissions is lower due to source period packet T_p, fixed by codec, is also lower. Thus, transmissions over some links are not allowed because t_{cad} could overcome T_p. Despite of a transmission may not be allowed, each node has its own neighbours’ vector sorted by latency time, so a transmission to other nodes with a lower t_{tx} time (but worse latency) may be possible and this worsens the total t_0. Anyway, this increase of t_0 is small, as shown in Table 5.4.

Table 5.4. Real-time transmission optt0

		Number of users *n*						
Codec rate	**Parameters**	**10**	**25**	**50**	**100**	**200**	**500**	**1000**
256 Kbps	**s**	6,78	12,10	12,85	12,34	12,30	14,06	14,20
	t_0	247,79	269,22	291,68	321,37	311,39	306,28	319,12
	t_{cad}	21,36	31,43	35,52	37,09	39,37	41,21	42,67
	Covered nodes	10	25	50	100	200	500	1000
	Covered nodes (%)	100%	100%	100%	100%	100%	100%	100%
	Worst case	-	-	-	-	-	-	-
	Total connected networks	50	50	50	50	50	50	50
	Total connected networks (%)	100%	100%	100%	100%	100%	100%	100%
512 Kbps	**s**	6,22	9,55	9,15	8,86	8,06	9,40	9,74
	t_0	248,87	270,65	295,44	327,87	313,46	307,72	320,59
	t_{cad}	18,64	21,48	23,00	23,43	23,44	23,44	23,44
	Covered nodes	10	25	50	100	200	500	1000
	Covered nodes (%)	100%	100%	100%	100%	100%	100%	100%
	Worst case	-	-	-	-	-	-	-
	Total connected networks	50	50	50	50	50	50	50
	Total connected networks (%)	100%	100%	100%	100%	100%	100%	100%
1024 Kbps	**s**	4,92	5,64	5,16	4,66	5,36	5,92	5,96
	t_0	261,75	279,96	304,59	330,57	318,39	309,81	321,28
	t_{cad}	10,79	11,72	11,72	11,72	11,72	11,72	11,72
	Covered nodes	9,80	24,60	49,32	98,52	197,54	492,64	985,94
	Covered nodes (%)	98,00%	98,40%	98,64%	98,52%	98,77%	98,53%	98,59%
	Worst case	9	23	48	96	193	484	979
	Total connected networks	40	32	27	10	4	0	0
	Total connected networks (%)	80%	64%	54%	20%	8%	0%	0%
2048 Kbps	**s**	2,79	3,03	2,75	3,00	2,82	3,42	3,42
	t_0	282,25	308,44	329,40	352,44	337,60	322,41	331,34
	t_{cad}	5,65	5,86	5,86	5,86	5,86	5,86	5,86
	Covered nodes	9,31	23,12	45,96	91,58	185,60	460,72	923,28
	Covered nodes (%)	93,09%	92,48%	91,92%	91,58%	92,80%	92,14%	92,33%
	Worst case	7	18	42	87	179	449	904
	Total connected networks	23	6	2	0	0	0	0
	Total connected networks (%)	46%	12%	4%	0%	0%	0%	0%
4096 Kbps	**s**	1,60	1,49	1,59	1,50	1,48	1,64	1,58
	t_0	336,02	393,30	401,37	426,37	402,99	373,11	373,93
	t_{cad}	2,45	2,82	2,87	2,93	2,93	2,93	2,93
	Covered nodes	7,55	18,57	35,53	71,52	143,72	353,66	712,38
	Covered nodes (%)	75,53%	74,27%	71,06%	71,52%	71,86%	70,73%	71,24%
	Worst case	5	14	30	62	132	333	677
	Total connected networks	2	0	0	0	0	0	0
	Total connected networks (%)	4%	0%	0%	0%	0%	0%	0%

If we look at the evolution of t_0 when the number of users increases, we notice the decrement of this time when 100 and 200 user networks are compared. The same occurs between 200 and 500 users networks. This behaviour for t_0 occurs in all codec scenarios, which is logical because we are using the same overlay

networks and the final multicast transmission trees should not vary too much among all codec rates. This unexpected behaviour is due to the own distribution of user nodes through the network. In this scenario, the user nodes always belong to the same transit domain and they are distributed among different stub domains. This means that communications through these overlay networks have always (of nearly always) to cross exactly 3 transit-transit edges (we have four transit nodes), which contributes most to the delay. Thus, the statistical deviations due to user nodes distribution will cause this unexpected behaviour in the t_0 function. To check it, we expand the study over these networks by calculating the arithmetical means with 20 overlay networks per transit network instead of 10. In this case, the results consider 100 cases in total instead of 50. As we expected, an increasing behaviour of t_0 was recorded for all the values of n and the statistical deviations were corrected by expanding our statistics.

As explained in section 5.3.2.3, the value of t_{cad} is irrelevant in this scenario because the rate of the data emission is fixed by the codec. Moreover, the number of covered nodes and its percentage is also high. The only scenario where this value must be pointed out is the 4096 Kbps, where only 70% of the users are reached, independently on the size of the network. Actually, this value corresponds to the amount of users with an Internet access equal or higher than 4 Mbps (see Table 5.2) and although it is not as good as those presented by the other codecs, where more than 90% of the end users are reached in all cases, it is clear that if the access link is not good enough to support a certain codec rate, the transmission is not possible. In this case, the problem is not the algorithm but the network itself.

Finally, we point out the average number of times s that the source sends each packet. Obviously, this s is directly related with t_{cad} of the source, so the higher the rate of the codec is, the lower the value of s. On the other hand, this value increases with the number of nodes, since if we want to flood a higher number of users, the source will usually send more times each packet, always within the constriction $t_{cad}<T_p$.

Through this entire report we have said that the modification to get real-time transmissions would be applied in both optt0 and optcad2 algorithms. Anyway, until this point, only results for original and evolved optt0 algorithms have been presented. In Table 5.5, we show the results provided by optcad2 evolution, that is, after the real-time modification $t_{cad}<T_p$ has been applied over the original algorithm.

The original optcad2 tries to optimize the rate time t_{cad} of all nodes in the network. This is reached by penalizing the t_0 time, as it is explained in the report of the preceding study [17], which makes sense if the number of messages M of the flow is large enough. In this scenario all the nodes could forward each packet before (thanks to a better t_{cad}) and the total transmission time could be better than for optt0 (optcad2 saves time each time a node sends a new packet). Anyway, in Table 5.5 we show that optcad2 has no sense in real-time scenarios where t_{cad} is limited by the period packet time T_p of the source node.

Table 5.5. Real-time transmission optcad2

Codec rate	Parameters	Number of users *n* 10	25	50	100	200	500	1000
256 Kbps	s	1	1	1	1	1	1	1
	t_0	1022,3	2285,57	3848,22	5936,99	8472,43	13112,7	17728,1
	t_{cad}	10,04	15,66	17,58	20,86	21,8	23,44	23,44
	Covered nodes	10	25	50	100	200	500	1000
	Covered nodes (%)	100%	100%	100%	100%	100%	100%	100%
	Worst case	-	-	-	-	-	-	-
	Total connected networks	50	50	50	50	50	50	50
	Total connected networks (%)	100%	100%	100%	100%	100%	100%	100%
512 Kbps	s	1	1	1	1	1	1	1
	t_0	1022,3	2285,57	3848,22	5936,99	8472,43	13112,7	17728,1
	t_{cad}	10,04	15,66	17,58	20,86	21,8	23,44	23,44
	Covered nodes	10	25	50	100	200	500	1000
	Covered nodes (%)	100%	100%	100%	100%	100%	100%	100%
	Worst case	-	-	-	-	-	-	-
	Total connected networks	50	50	50	50	50	50	50
	Total connected networks (%)	100%	100%	100%	100%	100%	100%	100%
1024 Kbps	s	1	1	1	1	1	1	1
	t_0	1025,11	2206,37	3758,83	5720,98	8147,23	12360,5	16636,6
	t_{cad}	8,71	10,66	11,48	11,72	11,72	11,72	11,72
	Covered nodes	9,92	24,56	49,34	98,6	197,72	492,14	986,04
	Covered nodes (%)	99,20%	98,24%	98,68%	98,60%	98,86%	98,43%	98,60%
	Worst case	9	23	48	96	193	484	978
	Total connected networks	46	31	24	11	7	0	0
	Total connected networks (%)	92%	62%	48%	22%	14%	0%	0%

5.3.3. Multiple Autonomous System backbone network emulation

As seen in section 4.1, the routing algorithms presented have been tested over two sorts of backbone network topologies. The first one emulates the network of a single ISP and is the one considered until now. From now on, the backbone network simulates multiple Autonomous Systems, much closer to the actual Internet topology. So, next, the optt0 evolution with all the modifications is evaluated over this scenario.

As it were expected, the final results obtained on this set of tests are similar as the results presented in section 5.3.2.4. However, a lower delay t_0 is obtained since the propagation delays of the new backbone networks are lower.

According to Table 5.6, the t_0 delay does not increase significantly with the number of nodes. This fact occurs because although the number of nodes increases, the network resources increase as well, as explained before. Another relevant result is the dependence of the total delay t_0 on the codec rate. Although total delay t_0 increases with the codec rate, this growth is insignificant.

All these behaviours were also obtained in the previous section, were algorithms were tested over simple networks.

The same that is explained in section 5.3.2.3 (first optt0 evolution) for t_{cad} and covered nodes, applies in this new scenario. That is, the value of t_{cad} is irrelevant because the rate of the data emission is fixed by the source node itself. Furthermore, the number of covered nodes and its percentage is again high. For codec rates up to 512 Kbps all user nodes are reached independently on the size of the network while for the 1024 Kbps codec rate case some users were not reachable due to their access links were not high enough.

Table 5.6. Real-time transmission algorithm over multiple SA emulation

		Number of users *n*						
		10	**25**	**50**	**100**	**200**	**500**	**1000**
Codec rate	**Parameters**							
256 Kbps	**t_0**	188,77	220,28	236,73	249,04	263,82	270,91	276,81
	t_{cad}	28,71	41,11	43,51	45,91	46,53	46,80	46,86
	Covered nodes	10	25	50	100	200	500	1000
	Covered nodes (%)	100%	100%	100%	100%	100%	100%	100%
	Worst case	-	-	-	-	-	-	-
	Total connected networks	50	50	50	50	50	50	25
	Total connected networks (%)	100%	100%	100%	100%	100%	100%	100%
512 Kbps	**t_0**	202,62	230,86	247,14	257,61	273,96	278,24	282,80
	t_{cad}	20,57	22,41	23,37	23,44	23,44	23,44	23,44
	Covered nodes	10	25	50	100	200	500	1000
	Covered nodes (%)	100%	100%	100%	100%	100%	100%	100%
	Worst case	-	-	-	-	-	-	-
	Total connected networks	50	50	50	50	50	50	50
	Total connected networks (%)	100%	100%	100%	100%	100%	100%	100%
1024 Kbps	**t_0**	223,08	254,60	271,60	282,99	296,35	299,54	305,56
	t_{cad}	11,16	11,70	11,72	11,72	11,72	11,72	11,79
	Covered nodes	9,88	24,84	49,48	98,5	197,3	492,54	981,19
	Covered nodes (%)	98,80%	99,36%	98,96%	98,50%	98,65%	98,51%	98,12%
	Worst case	8	24	48	96	193	488	978
	Total connected networks	45	42	26	12	4	0	0
	Total connected networks (%)	90%	84%	52%	24%	8%	0%	0%

The upward tendency for t_0 as the number of users increases is graphically represented in Figure 5.3. The low variations respect to the number of users shows that if a new user is added, t_0 will almost not be affected. This is because the packet could be send to the new user through an idle user long before the packet has reached to all the users. Moreover, if the bandwidth of the source increases there are also moderate delay increases. This is because by increasing the source rate will be some links that cannot be used, and so slower links (worst latency time) will be used instead. However, as the delay growth from one link to another is always gradual, the final increase will be also gradual. Besides this, if one link is used instead of other due to rate time

restrictions, the new link will have a higher latency (t_{prop}+t_{tx}), but on the other side, the transmission time will be lower (if not, it will not be valid because it will result in a higher rate time). Hence, the propagation growth will be compensated by a lower transmission time. This is why, when we increase the bandwidth of the source, the delay is not increased notably.

All this conclusions mean that defined trees will give very good results, with a total delay lower than a second, a perfectly acceptable value for the transmission of live events. Furthermore, this transmission delay will not grow up notably if we increase both the transmission quality (bandwidth of the source) and the number of connected users. However, if transmission quality is augmented up to 1024 Kbps, some users (always in a small percentage) will not get connected.

Finally, on multiple AS scenarios the statistical deviations due to the distribution of the user nodes does not occur often (remember the deviations on the single AS networks in section 5.3.2.4). This fact is due to the backbone network topology. On the new backbone networks, where there are 40 transit nodes, in general, as the number of nodes increases more Transit-Transit edges must to be crossed. Thus, if we increase the number of users in general we will cross more Transit-Transit edges which contribute most to the total delay, and thus statistical deviation will be less probable. This is also why on the multiple AS scenario, t_0 increases as the number of users grows up more than on the single AS scenario, though this increment is irrelevant in practice.

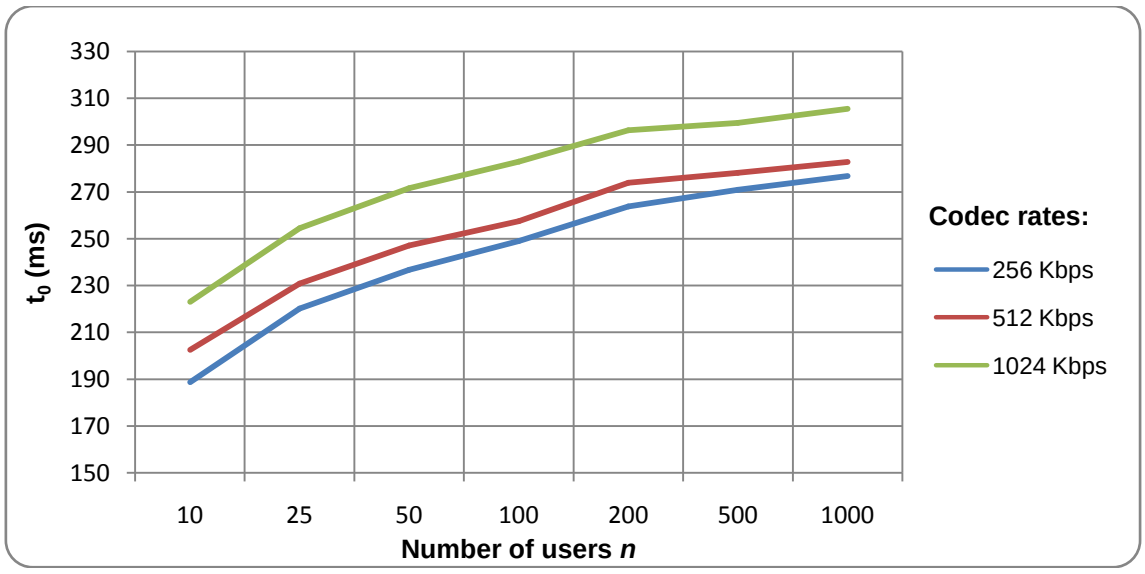


Fig. 5.3 Algorithm behaviour – t_0 vs. n

CHAPTER 6. CONCLUSIONS

In this project, continuation of previous works [13,14,15,16,17], we have presented a set of real-time multicast routing algorithms for P2P networks Those networks are also named overlay networks because they are defined on the application layer. Starting from an algorithm based on a single message transmission, we have described a group of algorithms that allows the transmission of multiple messages on a multicast group, considering always real-time communications. Our analysis has been focused on evolutions of optt0 and optcad2 algorithms, looking for the application of those algorithms on real-time scenarios. Both algorithms and their modifications have a complexity of $O(n^2)$.

We have worked with two different network backbone models. The first of them, which is the one used in previous works, is based on the representation of the backbone network as a single Autonomous System. The second one assumes that the Internet topology is based on multiple Autonomous Systems. Over those backbone networks simulations, we have tested the real-time multicast algorithms described in the present project.

Initially, we took the original algorithms optt0 and optcad2 and analysed their results. In this case, for real-time transmission we need to obtain a rate time, for each user, equal or lower than the source packet rate time (time elapsed between the creation of two consecutives packets). Hence, based on previous results, a filtration was done by selecting those networks with a rate time for the worst node lower than the packet period T_p. In this case, the number of available overlay networks was very small.

After that, the rate restriction described in former paragraph was introduced on the algorithms performance. Therefore, the rate time of the nodes has been limited to the period packet T_p of the source. These modifications have been tested over a backbone network topology represented by a single Autonomous System. The results obtained in this case, compared to the ones obtained previously, offer a higher amount of available networks where a real-time communication could take place. With this same purpose, we have chosen later as source a user connected to network by at least one link with a bandwidth equal or higher than the codec rate. After this modification all the networks were available. Finally, the optt0 evolution has been tested over a backbone network conformed by multiple Autonomous Systems in order to represent, in a more realistic way, the actual Internet topology, all with similar results.

Summarizing, the total delay of the first message t_0 (the one that we try to minimize) does not depend highly on the number of overlay users nor on the bandwidth of the source. This time t_0 has always been lower than one second, even in the case of 1000 users, which is completely proper for the transmission of life events. Furthermore, the results presented offer a transmission rate for each user equal or higher than the codec rate, an essential requirement to carry

out real-time communications. Hence, the initial objective of the present work is achieved satisfactorily.

The project environmental impact can be evaluated taking into account that with the algorithms introduced, the messages transmission time is reduced and the resources and bandwidth are optimized either in physical or logical terms. In smaller networks, formed by 10 users, such resources optimization is relative, but in large networks, with up to 1000 users, the resource savings can be significant.

BIBLIOGRAPHY REFERENCES

[1] S. E. Deering and D. R. Cheriton, “Multicast routing in datagram internetworks and extended LANs”, *ACM Trans. Comput. Syst.*, vol. 8, no. 2, pp. 85-110, 1990.

[2] M. H. Ammar, “Why Johnny can’t multicast: lessons about the evolution of the internet” in *NOSSDAV ’03: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, New York, NY, USA: ACM Press, pp. 1-1, 2003.

[3] J. H. Saltzer, D. P. Reed and D. D. Clarck, “End-to-end arguments in system design”, *ACM Trans. Comput. Syst.*, vol. 2, no. 4, pp. 277-288, 1984.

[4] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O’Toole, Jr., “Overcast: Reliable multicasting with an overlay network”, in *USENIX Symposium on Operating Systems Desing and Implementation*, pp. 197-212, 2000.

[5] Y. Chawathe, S. McCanne, and E. A. Brewer, “RMX: Reliable multicast for heterogeneous networks”, in *INFOCOM.* IEEE, pp. 795-804, 2000.

[6] A.-M. K. M. Castro, P. Druschel and A. Rowstron, “SCRIBE: A largescale and decentralised application-level multicast insfrastructure”, *IEEE Journal on Selected Areas in Communication (JSAC)*, vol. 20, no. 8, October 2002.

[7] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, “Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination”, in *NOSSDAV*, June 2001.

[8] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A scalable content addressable network”, in *ACM SIGCOMM*, 2001.

[9] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, “How to Model an Internetwork” in *IEEE Infocom*, pp. 594-602, IEEE, San Francisco, 1996.

[10] Gimbert, R. Moreno, J.M. Ribó, M. Valls, *Apropament a la teoria de grafs i als seus algorismes*, Edicions de la Universitat de Lleida, Lleida, 1998.

[11] R.J. Wilson, *Introducción a la Teoría de Grafos*, Alianza Universidad, Alianza Editorial, Madrid, 1972.

[12] A. Bar-Noy, and S.Kipnis, *Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems*, *ACM Symposium on Parallel Algorithms and Architectures*, pp. 12-22, 1992.

[13] Van Creij, V., *Algorismes Multicast en Xarxes Overlay*, Final project thesis, Escola Politècnica Superior de Castelldefles - UPC, 2009.

[14] S. Domínguez, *Análisis de aloritmos multicast en redes overlay*, Final project thesis, Escola Politècnica Superior de Castelldefels - UPC, 2006.

[15] J. Pratsevall, *Anàlisis d'algorismes de multicast en xarxes overlay*, Final project thesis, Escola Politècnica Superior de Castelldefels – UPC, 2005.

[16] J. Pratsevall, *Application-Layer Multicast Algorithms for Bounded Delay Transmissions*, Master Thesis, Escola Politècnica Superior de Castelldefels – UPC, 2008.

[17] A.J. Alcaide, *Algoritmos Multicast en Redes P2P*, Final project thesis, Escola Politècnica Superior de Castelldefels – UPC, 2010.

[18] A. Bar-Noy, and S. Kipnis, *Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems* in *ACM Symposium on Parallel Algorithms and Architectures,* pp. 13-22, 1992.

[19] E. W. Dijkstra, *A note on two problems in connextion with graphs*, *Numerische Mathematik,* vol. 1, pp. 269-271, 1959.

[20] Autonomous System, AS [On-line] Available: "http://en.wikipedia.org/wiki/Autonomous_system_(Internet)".

[21] GT-ITM, Modeling Topology of Internetworks [On-line] Available: "http://www.cc.gatech.edu/computing/Networking/projects/gt-itm/".

[22] Eclipse.org home [On-line] Available: "http://www.eclipse.org".

[23] Jeff McAffer, Jean-Michel Lemieux, *Eclipse Rich Client Platform: Designing, Coding and Packaging Java Applications*, Addison-Wesley Professional, 2005. [On-line] Available: http://eclipsercp.org/.

[24] Caida.org home [On line] Available: "http://www.caida.org/".

[25] Amic.es home [On line] Available: "http://www.aimc.es/".