Discrete Mathematics and Theoretical Computer Science **4**, 2001, 255–272

Oriented Multicast Routing Algorithm Applied to Network-level Agent Search

Damien Magoni and Jean-Jacques Pansiot

LSIIT (UPRES-A N° 7005), Université Louis Pasteur, Boulevard Sébastien Brant, Illkirch, France

received Dec 26, 2000, *accepted Aug 3, 2001*.

Many protocols need a discovery mechanism to enable a given node to locate one or several nodes involved in the same communication. However, there is no protocol ready to fulfill this service at the network-layer. Every protocol usually implements its own solution. In particular, multicast protocols often use a searching technique based on an algorithm called expanding ring search. This algorithm searches for nodes in all directions and thus uses much bandwidth. However a typical search can usually restrict its scan in a specific direction. To enable this broadcast restriction, we propose an oriented multicast routing algorithm. The algorithm's principle is to direct the multicast of packets towards a special node, involved in the communication, in order to search only in a limited area. The area must be as small as possible to reduce network flooding but still has to contain many nodes satisfying the search criteria. This new algorithm is the core part of a network-level node search framework also defined herein. A search protocol based on this framework could provide a network-level agent discovery service to current protocols. We have simulated an agent search with our algorithm on one side and with the expanding ring algorithm on the other side and we give comparative results.

Keywords: multicast routing, agent discovery, search protocol, expanding ring

1 Introduction

Many network protocols need information from a service discovery system before setting up a communication. A device such as an end-user computer or a router which hosts a service has to be advertised or found in order to be used. We call such a device an agent. Some services provide shared resources and the location of the agents that host them does not evolve over time. In this case, a service discovery based on a directory system works well. But many other services provide resources for a short period of time. In particular, services provided by multicast protocols such as multicast tree creation and maintenance, are very volatile. In this case the service discovery should be dynamic. This means that the search must be done at each request as the results are supposed to change rapidly. At the network level (i.e. the IP layer) carrying out an agent search means trying to locate an agent by broadcasting messages containing information about the service needed. The broadcast can be done by using IP broadcasting [Mog84], multicasting by Reverse Path Forwarding (RPF) [DM78] or anycasting [PMM93]. Multicast protocols usually implement the second solution in a mechanism called Expanding Ring Search (ERS). This is because IP broadcasting in not efficient considering its network resource consumption and anycasting is not yet ready for use. Although the ERS algorithm uses RPF and is incremental, it is not always efficient. This is because it scans in all directions and therefore sends packets in uninteresting areas. In many cases where only a specific area needs to be scanned for agents, a mechanism based on our new algorithm can give better results than the use of the ERS algorithm.

In this paper we give an overview of a search framework and a detailed description of its key mechanism: the oriented multicast routing algorithm. Section 3 contains the algorithm's description in pseudo-code as well as the data structures needed for the algorithm. Section 4 describes the framework of a network-level agent search. Its architecture and basic features are explained. Finally, section 5 gives simulation results of an agent search using the oriented multicast routing algorithm compared to an agent search using the ERS algorithm.

2 Related work

The classical multicast technique currently used by some protocols to carry out an agent search is the expanding ring search [CC97]. For instance, in a reliable communication, some nodes (called agents) are responsible for retransmitting data that has not been correctly received by one or more receivers. In reliable multicast protocols such as LGMP [Hof97], RMTP [PSLB97] and TMTP [YGS95], receivers search for these retransmitting agents by multicasting requests according to the ERS scheme. Receivers look for the nearest agent, but the agent should also be as close to the source as possible. We could orient the multicast by taking the multicast source as the target node. In multicast tree creation and maintenance protocols such as PIM-SM [EFH$^+$98], YAM [CC97] and QoSMIC [BFP98], when a receiver has succeeded in contacting the root of the multicast tree of the group, it must be inserted into the existing multicast tree. In YAM and QoSMIC, the receiver then uses an ERS to be able to graft itself to a node already belonging to the desired multicast tree. The receiver would like to be grafted to an on-tree node that minimizes the total distance receiver - in -tree node - root node, in order to minimize delays. However these protocols include more efficient search methods that are more expensive too. YAM has defined the directed spanning join which can be seen as a particular instance of our algorithm. QoSMIC has defined a dual method : search is accomplished simultaneously by a limited ERS from the initiator, and a search from the multicast tree. In our approach, we could take the root of the multicast tree as our target node. In protocols needing the use of gateways such as MeGa [AMK98], agents are servents grouped in clusters, that provide video streams adapted to the link capacity. At this time, clients use out-of-band means to contact these agents (e.g. via DHCP or a configuration file). However, the needs are the same than in the previous examples. The client search a servent which will adapt the video stream throughput sent by the source. We can see here that we need a third-party search mechanism. The aim here too is to minimize the total distance client-servent-source. Note that the video source would be the target of the search.

As we saw above, ERS is a multicast technique widely used in current protocols. It is based on the use of a field of the IP header called TTL (Time To Live). This field is decremented each time the packet goes through a router. Its initial value, at most 255, is chosen by the source. When the TTL value reaches 0, the router destroys the packet. The original aim of this field is to avoid infinitely looping packets due to erroneous routing information. In the case of the ERS, the TTL field is also used for the following mechanism: packets are sent by the source with an arbitrary fixed TTL value. This value defines a *range* for the packet. The packet is then forwarded by a multicast technique called Reverse Path Forwarding (RPF) [DM78]. When a packet is received by a router, the latter checks if the packet comes from a shortest path to the source (RPF Check). If it is true, the packet is duplicated and sent on every router interface except the receiving one, if it is false the packet is discarded. When an agent (noted A) receives the search packet, it sends an answer packet to the source. Initially an ERS packet is sent with a small TTL (say 1 or 2). If no agent is found, a new packet is sent with an increased TTL and so on until an agent is found or a threshold value for the TTL is reached. Note that with a TTL greater than 10, one can probably reach millions of hosts in the Internet. The evolution of the TTL value can follow an arithmetic or a geometric progression. The nodes reached by this multicasting can be viewed as a disk centered at the initiator and with a radius equal to the highest TTL value used, hence the name expanding ring. In real life, use of an ERS with an initial TTL greater than 7 or 8 is hardly conceivable. A request packet propagated by an ERS by a source S does not take into account the position of the other actors of the communication and floods surrounding nodes in all directions. However we saw in each of the preceding examples that we could find a special target node serving as a beacon for the search area. Our aim is to use this information to orient the search and produce a more efficient agent search algorithm.

3 Oriented multicast routing algorithm

We propose an algorithm to define a new kind of multicast (i.e. oriented multicast). The idea is to perform a limited multicast channeled around the unicast path joining the sender to a specific destination, hence the use of the term *oriented*. This algorithm is close to RPF multicast algorithms but the flooding is much more controlled. The generic principle of the algorithm is to reach only the nodes located on or near the (or a) shortest path between the multicast initiator S (called source) and the target node D (called destination). Every packet contains a special field, besides the TTL, called *range*. As long as the packet travels along a shortest path between S and D, it is multicasted on every link of the node except the arrival link and the range is not decreased. When it goes out of the shortest path SD, the policy depends on the parameter settings of the algorithm. It can be propagated by Reverse Path Forwarding (RPF) or by lateral multicasting. Lateral multicasting means that the packet is only sent on links not leading by a shortest path to the source or the destination. This rule gives to the packets a lateral propagation effect intended to avoid packets redundancy.

In most cases the packet range is decreased for each hop and thus the maximum traveling distance (out of SD) for the packet is limited to at most the initial range value. In some case the initial range value is not fixed but depends on the position where the packet leaves SD. The initial range is then dynamically calculated at the point where the packet leaves the shortest path from S to D. The idea is that the distance that can be covered shall increase when the packet leaves the path farther from S up to the middle of SD and decrease when the packet leaves the path beyond the middle of SD up to D. In fact, when the packet has covered a distance lower or equal to half the distance SD, the initial range is set proportionally to the value of the distance covered. When the packet has covered a distance greater than half the distance SD, the initial range is set proportionally to the distance SD minus the distance already covered by the packet.

Although we talk about shortest paths, and SD shortest path in particular, through this paper, it is indeed the path given by the underlying unicast routing system. The fact that it is not always a true shortest path does not matter and our algorithm still works.

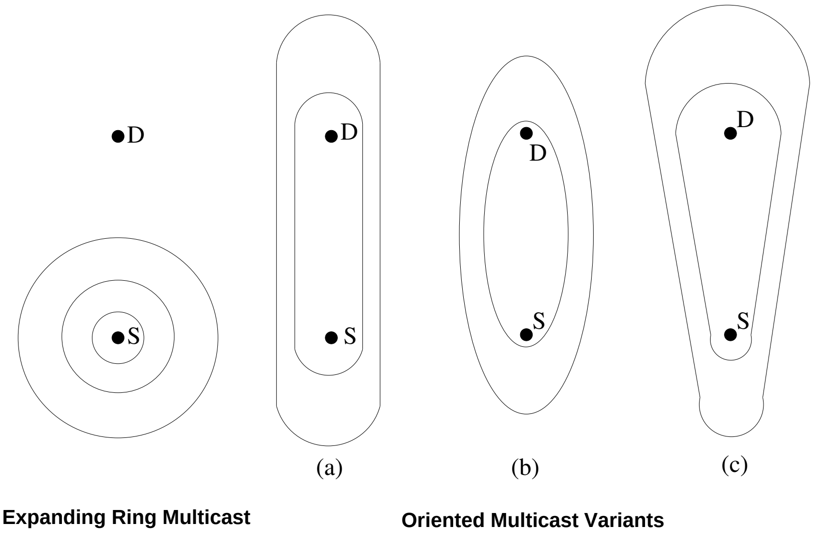

Fig. 1: Multicast areas of various models

3.1 Oriented multicast variants

Our algorithm is parameterized. By changing the values of the parameters we can obtain several different versions of our algorithm that we call *variants*. We study six of them in this paper. Figure 1 shows the areas covered by an expanding ring and some variants of our algorithm. Each thin black line defines the area covered in one attempt with a given value for the initial range. S represents the node initiating the multicast while D represents the target node of the multicast (used by our algorithm only). The notion of attempt is valid in a search context. If the search fails in the currently covered area, a new attempt will be made with an increased initial range to cover an extended area. In the expanding ring example of Figure 1 we represented three attempts having each an increased initial TTL value. Concerning the variants, three examples are shown in Figure 1. Each variant example is represented with two attempts. We can see that the area of a search attempt of a variant is always surrounding the SD shortest path.

In the context of agent search described in section 4, it is clear that the agents that can be found by an ERS could be a the opposite of the SD direction and thus not optimizing the S-A-D distance. Although the variants will cover a greater area than the expanding ring when SD increases, they have a much greater chance to find a agent located near the SD shortest path and thus minimizing the S-A-D distance. The variants labeled (a) and (b) have been tested in our simulation. The variant (c) has not been tested yet but its use could be of interest for specific applications.

3.2 Definition of the algorithm and the data structures

The *variant* structure contains the parameters of the algorithm. Table 1 gives a detailed explanation for each field of this structure. The (a) kind variants of Figure 1 are obtained with the *shape* parameter set to ***fixed*** and the (b) kind variants with the *shape* parameter set to ***variable***.

Name	Possible Values	Description
$multicast\_method$	***broadcast***, ***lateral***	Defines the multicast mode when the packets have left the SD shortest path
$frontal\_range\_dec$	**true**, **false**	Boolean indicating if the packet range has to be decreased when a packet is on a shortest path to S or D but not on a SD shortest path
$shape$	***fixed***, ***variable***	Defines the way of calculating the range value when the packets leave the SD shortest path
$initial\_range$	$n \in \mathcal{N}^*$	Defines the initial range value of a packet

Tab. 1: Algorithm parameters

Table 2 lists the fields of a *packet* structure: it contains a number of fields concerning the IP protocol as well as our multicast algorithm and our search mechanism. From the packet current router position, we define E as being a neighbor router not belonging to a SD shortest path. The packet contains fields belonging to three categories. The first category contains a number of fields typical of the IP protocol (i.e. IP *source* address, IP *destination* address and *TTL*). The second category contains fields necessary to the oriented multicast routing algorithm (i.e. *SD*, *ES*, *ED*, *range* and *distance_covered)*. Finally the third category contains fields for the agent search framework (i.e. *type* and *function_searched*). It will contain more fields in the future when we have finalized our framework in a search algorithm (however we already designate it as the *search algorithm* in some parts of the paper).

Name	Possible Values	Description
$source$	$@ \in \mathcal{A}$	IP address of the search initiator
$destination$	$@ \in \mathcal{A}$	IP address of the special node (target)
TTL	0 - 255	Time To Live: maximum number of hops left
$distance\_covered$	0 - 255	Number of hops already done
SD	**true** , **false**	Boolean indicating if the packet comes from a link on the shortest path from S to D
ES	**true** , **false**	Boolean indicating if the packet comes from a link on a shortest path (from E) to S (E is not on a shortest path from S to D)
ED	**true** , **false**	Boolean indicating if the packet comes from a link on a shortest path (from E) to D (E is not on a shortest path from S to D)
range	$n \in \mathcal{N}$	Integer used to define the degree of multicasting (the bigger it is, the farther the multicasting will be from the S, D axis)
$type$	$request, answer$	Defines the packet's type
$function\_searched$	$f \in \mathcal{F}$	Identifies the function searched by the source router

Tab. 2: Packet fields

Table 3 gives an explanation of the only field of a *router* structure. In our simulation each node of a graph represents a router (the algorithm's extensions to hosts will be done at a later time). The only information currently used by the search framework is the *function_id*. It indicates what service a router (or a host attached to the router's local network) is able to provide. Notice that our oriented multicast routing algorithm maintains no state information in the routers.

Name	Possible Values	Description
$function\_id$	$f \in \mathcal{F}$	Identifies the function performed by the router's agent

Tab. 3: Router attribute

Table 4 gives an explanation of the only field of an *interface* structure. The information contained in this field is needed for any routing process (whether unicast or multicast) and it is not specific to our oriented multicast routing.

Name	Possible Values	Description
$receiving\_router$	$@ \in \mathcal{A}$	Defines the address of the next hop router of the interface

Tab. 4: Interface attribute

The algorithm's functions are given below in pseudo-code. These functions will mainly be executed in routers as a core part of an eventual protocol implementation. The algorithm's parameters are accessible through the *variant* structure.

The ROUTE function provides an access to the underlying unicast routing system. It can be called in any router. It takes a router *address* and a network interface number. If the given network *interface* leads to the router having the given *address* by a shortest path then the function returns **true** else it returns **false**.

ROUTE$(address, interface)$
if $interface$ leads to $router.address$ by a shortest path
  then return true
  else return false

When an agent search is triggered at the application layer, the first *packet* is created and fields that are specific to the algorithm are initialized by the INITIALIZE function of the source node S. Notice that the last two fields belong to the search algorithm and should not be set at the oriented multicast routing layer but we have set their values here for completeness.

```
INITIALIZE(packet)
packet.TTL ← 255
packet.distance_covered ← 0
packet.SD ← true
packet.ES ← false
packet.ED ← false
packet.type ← request
packet.searched_function ← function_id
```

Notice the initialization of the fields *XX*. Indeed at startup the *packet* is on a shortest path from S to D (it is in S !), but is not on a shortest path from E to S or D because, according to our previous definition, E represent a router which is not on a shortest path from S to D. After the initialization of the first *packet*, PROCESS is called. This function is crucial because it is at the start of a series of recursive function calls that which model the travel of the *packet* in the network. A part of the network will be flooded by the *packets* in a depth-first order.

First a very important remark concerning the PROCESS function. It belongs to the agent search algorithm and *not* to the oriented multicast routing algorithm. However we have included it here for a better understanding of the function calls and because it is currently the only function of the search algorithm (i.e. putting it alone in a subsection would not have made the presentation clearer). The PROCESS function checks if it can reply positively to the search request. Then it performs a check on the *packet* hop counts: if the *TTL* or the *range* equals zero, it destroys the *packet*. Otherwise it forwards the *packet* to the SWITCH function.

```
PROCESS(packet)
switch packet.type
  case request :
        if function_id = packet.searched_function
          then // send answer to source by unicast
        if packet.range ≤ 0 or packet.TTL ≤ 0
          then DISCARD(packet)
          else SWITCH(packet)
        break
  case answer :
        // forward answer to application layer
        break
```

The aim of the SWITCH function is to determine the position of the *packet* (e.g. is it on a shortest path from S to D?), in order to select the multicast technique. Notice that the *multicast_method* parameter affects this choice. Its value depends on the choice of the *variant*.

```
SWITCH(packet)
if packet.SD = true
  then MULTICAST(packet, broadcast)
  else MULTICAST(packet, variant.multicast_method)
```

The MULTICAST function will spread the *packet* by various methods thus obtaining the different variants. The support for RPF-like forwarding is given by the ***broadcast*** option. The ***lateral*** option only spread packets on links which are orthogonal to the SD direction (i.e. links not leading to S or D by a shortest path). The router's interfaces are noted output_interface for output interfaces and input_interface for input interfaces.

```
MULTICAST(packet, method)
switch method
   case broadcast :
         for each output_interface
            do if output_interface ≠ packet.input_interface
                  then SEND(packet, output_interface)
         break
   case lateral :
         for each output_interface
            do if output_interface ≠ packet.input_interface
                  and ROUTE(packet.source, output_interface) = false
                  and ROUTE(packet.destination, output_interface) = false
                  then SEND(packet, output_interface)
         break
```

The SEND function sets the *initial_range* value if the *packet* is leaving the SD shortest path. It also sets the *ES* and *ED* booleans if the *packet* is already out of an SD shortest path. It calls the RECEIVE function of the router at the other end of the sending *interface*. This function call models the travel of the *packet* between two routers.

```
SEND(packet, interface)
if packet.SD = true
   then if ROUTE(packet.destination, interface) = false
            then packet.SD ← false
                  if variant.shape = fixed
                     then packet.range ← variant.initial_range
                     else cd ← packet.covered_distance
                           if cd ≤ packet.SD_dist ÷ 2
                              then packet.range ← cd ÷ (4 − variant.initial_range)
                              else packet.range ← (packet.SD_dist − cd) ÷ (4 − variant.initial_range)
   else if ROUTE(packet.destination, interface) = true
            then packet.ED ← true
            else packet.ED ← false
         if ROUTE(packet.source, interface) = true
            then packet.ES ← true
            else packet.ES ← false
interface.receiving_router.RECEIVE(packet)
```

The last function of the algorithm is the RECEIVE function. It performs a RPF check on the *packet* and discards it if necessary. It also decreases the values of the *TTL* and the *range*, if needed, and transfers the *packet* to the PROCESS function.

```
RECEIVE(packet)
if ROUTE(packet.source, input_interface) = false
   then DISCARD(packet)
packet.input_interface ← input_interface
packet.TTL ← packet.TTL − 1
if packet.SD = false
   then if variant.frontal_range_dec = true or ( packet.ES = false and packet.ED = false )
           then packet.range ← packet.range − 1
packet.covered_distance ← packet.covered_distance + 1
PROCESS(packet)
```

4 Network-level agent search framework

In many setup phases of network protocols, one or several network nodes with a specific functionality need to be selected. Depending on the specific problem, these nodes, that we call agents, can be network devices like routers, specialized servers or even end-user computers. Some examples of these agents include:

- A print server.
- A multicast tree node to graft a new member.
- A multicast retransmission node for reliable multicast streams.
- An application gateway that will convert data between incompatible clients.

Note that in the first example the server can be looked for in any direction, whereas in the three other cases, the search could be directed toward a specific node called the target. For the multicast tree, the target could be the root of the tree. For the reliable multicast, the target could be the source of the multicast stream. For the application gateway, the target could be the distant client. Moreover in these cases, it is desirable for the agent to be as close as possible to the shortest path from the initiator to the target. Such an agent can be found by several methods:

- Lookup in a directory, such as a name server. In this case, it is hard to choose the location of the agent. Potential agents must be declared in advance in the directory.
- Anycasting [BEH+98]: the routing system is in charge to forward requests to the nearest agent. Potential agents must declare themselves to the routing system. Directed anycasting is not currently considered.
- Network search: search packets are multicasted through the network until they reach agents, such as in the expanding ring search. Potential agents need not to be declared in advance.

We saw in section 2 that the first two methods are not suitable for finding agents belonging to the last three agent categories. In particular, multicast protocol agents are usually discovered by a network-level search.

We propose, in such cases, to replace the commonly used ERS by a search protocol using our oriented multicast routing algorithm. It is oriented towards the target (defined in each of the above examples) and it could be more efficient than the ERS method. Moreover our algorithm is designed to find agents as close as possible to a shortest path from the initiator to the target which is of a great interest to the above protocols. Our aim is to build an mechanism that uses the position of the other devices implied in the search. The search framework has to find a relay agent named A located somewhere between a node called S and a node called D, the target, in such a way that the search of this agent is done in a channel area between S and D. Our oriented multicast routing algorithm provides such a mechanism. Its use ensures that flooding is avoided and that the SAD distance is bounded. Compared to the expanding ring search, our algorithm needs an additional information which is the address of the target router D. We are free to increase the area of the search if no agent node has been found (i.e. like the ERS does).

4.1 Context of use

Our search framework is designed to operate in IP networks. The requirements on the routers are:

- All routers should run an instance of our search algorithm and an instance of our oriented multicast routing algorithm. In particular no tunneling should be done otherwise the oriented multicast routing behavior could become unpredictable.
- Routers may declare hosting a service on behalf of a host being on its local network (in a IGMP-like mechanism [Fen97]).
- No TCP-like stream facilities are needed by our algorithms.
- Each router should allow access to its unicast routing table, calculated by an independent algorithm.

In this context, we also ensure that the algorithms have the following properties:

- The search algorithm is fully dynamic (although a short-time cache system is not excluded) and thus does not rely on out-of-band methods (e.g. use of directories) to find agents.
- The oriented multicast routing algorithm will mainly run in routers and thus will not rely on any centralized mechanism. In particular, this algorithm only needs information from the underlying unicast routing system. No knowledge of the global network topology is required.
- Both the oriented algorithm and the search algorithm do not maintain any state information in routers for scalability reasons.

4.2 Architecture

An overview of the search framework is given in Figure 2. It shows how a request packet is handled by a router and forwarded to a next-hop router. Our oriented multicast routing algorithm will probably be implemented in a packet as an extension of a regular IP packet (e.g. like ICMP is). Of course in the routers, our oriented multicast routing algorithm will replace the regular IP packet forwarding mechanism

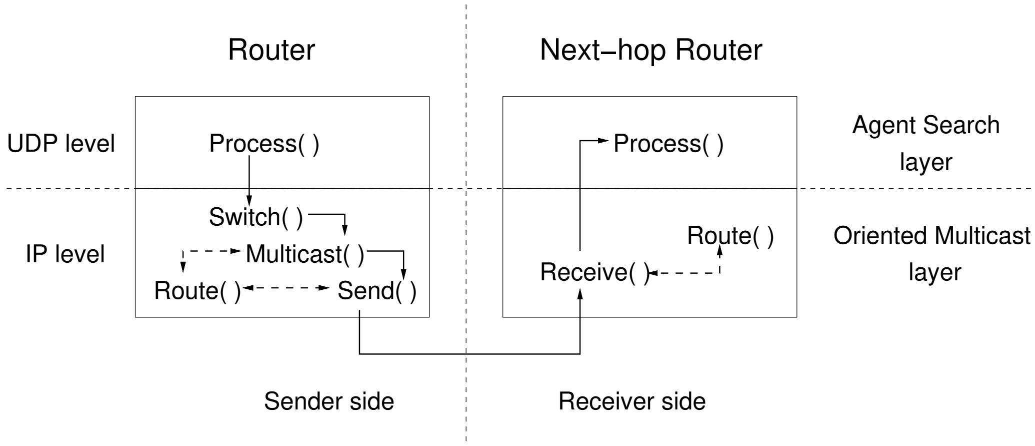

Fig. 2: Architecture

(and thus will place itself at the same level as the IP protocol). Concerning the search algorithm no choice has been made yet. It could be implemented directly on top of our oriented multicast routing algorithm or be implemented above UDP by using a specific port number. As we said before our search algorithm will not use stream facilities for speed and efficiency reasons. So UDP can provide the necessary mechanisms for implementing our search algorithm. Notice that for the moment only the process function belongs to the search layer whereas all the other functions belong to the oriented multicast routing layer. As we will develop our search algorithm, new functions will probably be added.

5 Agent search simulation

We have implemented our oriented multicast routing algorithm in an agent search framework to be able to compare it to the ERS algorithm. This implementation was done in the simulation module of our network manipulator (*nm*) software tool. With this module, we can run static simulations to evaluate informations on the behavior of the algorithms such as the number of packets multicasted or the number of agents hit. The *nm* simulation module can be used for algorithm simulation but not protocol simulation because:

- It does not take into account the temporal aspects of networks (propagation delays, packet sequencing on emission or reception, ...).
- It only performs evaluations on static objects (no possibility of broken links, failed routers, routing information changes, ...), in other words the topology is frozen.
- Its evaluations are done in an arbitrary order which does not correspond to reality (graph traveled in depth-first order). In particular, the moves of packets are not concurrent.

However the *nm* simulation module provides enough results for a first evaluation of our new algorithm before going deeper into the algorithm's description. This would require the definition of additional parameters such as timers that do not exist at this point. It is also for this reason that we did not use the LBNL simulator *ns* [VIN]. We will use *ns* when we will have taken into account every temporal parameter needed by the algorithm.

5.1 Simulation parameters and environment description

We have carried out our simulation runs over network topologies modeled by graphs. In a graph, each node represents a router and each edge represents a link between two routers. It could be a high throughput ATM link or a slower Frame Relay or ISDN link. We suppose that links are bi-directionals and we use the hop count as the only distance metric. The graphs have been created by using the Georgia Tech - Internet Topology Model package developed by Zegura et al. [ZCD97].

The parameters used for the creation of the graphs are given in Table 5. The transit-stub method has been chosen because it generates more realistic graphs than those obtained by using only the Waxman method [Wax88]. The parameters concerning the AS have been chosen to match as close as possible the values of the AS network topology studied by Magoni et al. in [MP01].

Parameter name	Value
Generation method	Transit-Stub
Nb of transit AS in top domain	20
Nb of nodes per transit AS	4
Nb of stub AS per transit node	2
Nb of nodes per stub AS	12
Total nb of nodes	2000
Edge connection method	Waxman
α	0.5
β	0.5

Tab. 5: ITM network generator parameters

Table 6 shows the average property values of the 20 graphs used for simulation. More than 75% of the nodes are in the mesh which is not very realistic compared to real Internet map values. The average degree is nearly three which is an acceptable value. Yet the average degree distribution of the graphs is not following the power laws 1 and 2 defined by Faloutsos et al. [FFF99] in their AS and router level analysis. The average distance, eccentricity, radius and diameter have values that make realistic the use of the SD distances of 4, 8, 12 and 16 for simulation. Finally the trees have acceptable characteristics even though their tree size and depth distributions do not follow power laws 6, 7 and 8 defined by Magoni et al. [MP01].

Simulation parameters are shown in Table 7. The agent nodes have been randomly chosen and there are only representing 1% of all the nodes. This means that the agents are quite rare. It is worth noticing that if the agents are numerous (e.g. half or one-third of all the nodes) our search algorithm becomes uninteresting as it is trivial to find an agent in such conditions. Furthermore the random placement of the agents may not reflect reality. The agent placement will probably depend on the kind of application using the search algorithm. There have been one run for each variant (plus the ERS), for each source-destination

Property name	Average value
Nb of nodes	2000
Nb of nodes in the mesh	1603.6
Nb of leaf nodes (degree 1)	302.1
Nb of mesh nodes on a bridge	56.8
Nb of cut-point mesh nodes	306.5
Average degree	2.99
Average mesh degree	3.24
Average distance	9.8
Average eccentricity	16.7
Diameter	22.6
Radius	11.7
Nb of trees	255.9
Average tree size	2.54
Maximum tree size	13.3
Average tree depth	1.34
Maximum tree depth	7.3

Tab. 6: Average properties of ITM networks

Parameter name	Value(s)
Nb of ITM networks generated	20
Source-destination distances tested	4, 8, 12, 16
Nb of source-destination pairs tested per distance	500
Nb of variants tested per source-destination pair	6
Percentage of agent nodes	1
Routing	Multiple equal-distance route management

Tab. 7: Simulation parameters

pair, for each distance and for each graph which gives a total of 280000 runs. The results of these runs have been merged to give the average results shown in the next subsection.

Table 8 shows the parameter settings corresponding to the 6 variants that have been tested. All the variables in the first column are fields of the *variant* data structure used in the oriented multicast routing algorithm. The explanation of the possible values taken by these variables has been given previously in Table 1. In Table 8 the *initial_range* value is not given. This is because it depends on the attempt number. For the ERS, the TTL is increased by 1 while no optimal agent is found, starting at 1 up to 5 (i.e. 1, 2, 3, 4 and 5). For all the six variants, the *initial_range* is increased by 1 starting at 1 up to 3. As for the ERS, each oriented multicast with an increased *initial_range* is counted as one more attempt. We deduce that in the case where no optimal agent is found, an ERS will make at most 5 attempts while an oriented search will make at most 3 attempts. In a future implementation it is probable that only the most efficient variant will be retained and the values of the parameters will be definitely set according to this specific variant.

No of variant	1	2	3	4	5	6
multicast_method	***broadcast***	***broadcast***	***lateral***	***broadcast***	***broadcast***	***lateral***
frontal_range_dec	**true**	**false**	N/A	**true**	**false**	N/A
shape	***fixed***	***fixed***	***fixed***	***variable***	***variable***	***variable***

Tab. 8: Parameter settings for the studied variants

5.2 Simulation results

In all the following figures, the values shown are always the ratio of the variant's value versus the ERS's value (i.e. it is always a relative value). Furthermore the value is an average of the four SD distances tested. As 500 source-destination pairs are tested for each SD distance and for each of the 20 graphs, the ratio of a given measure (e.g. bandwidth usage) for a given variant is an average of 40000 values. We did not include the separate results for each SD distance because it would have taken too much space.

Figure 3 shows the ratio of the bandwidth usage of each variant versus the ERS. The bandwidth usage is equal to the average number of packets created in the network for an search. For example, variant 1 has a ratio of 3.2 which means that variant 1 creates 3.2 more packets than the ERS for an average search. This can be explained by the fact that the ERS is limited to a TTL of five while an oriented search covers an area that spans the SD distance (i.e. for 4 to 16 hops).

The number of optimal agents hit for each variant are shown in Figure 4. This is perhaps the most important figure as it shows the rate of success of each variant. For example, variant 1 has a ratio of 3.6. This means that variant 1 finds on average 3.6 more agents than an ERS. But this value can be interpreted differently. A search stops when one (or several) agent(s) are found. So the ratio also means that for one successful ERS search, 3.6 searches using variant 1 are successful. Indeed at an SD distance of 12, an ERS search is successful half of the time (i.e. finds 0.56 optimal agents on average) while a search using variant 1 is always successful (i.e. finds 1.58 optimal agents on average). We see that the variant having the highest ratio (i.e. rate of success) is variant 2 with a value of 4. The worst variant

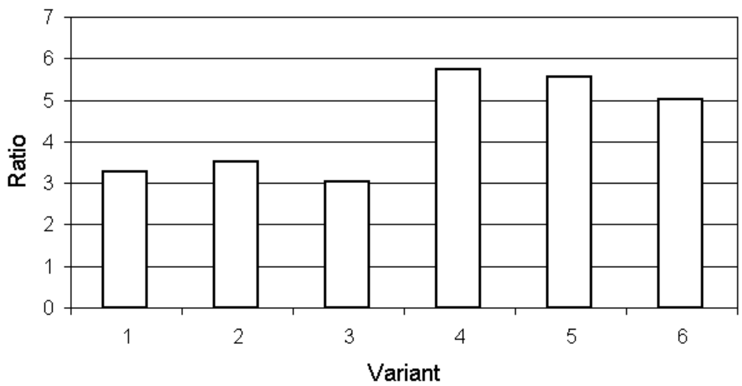

Fig. 3: Bandwidth usage

Fig. 4: Optimal agents hit

(i.e. no 6) still outperforms the ERS by a factor of 2.8. To conclude we can say that any variant of our oriented multicast-based search algorithm is on average three times more successful than an ERS for any SD distance between 4 to 16. In other words any variant of our oriented multicast-based search algorithm always finds at least one optimal agent whereas an ERS succeeds only once per two or three search attempts.

Figure 5 shows the average number of attempts needed to find at least one optimal agent. The ERS needs on average 4.5 attempts to succeed which corresponds to a maximum TTL of 4 to 5. The variants need on average around 2 attempts to succeed which corresponds to a maximum range of 2. As we can see in the figure the variants only need 44% to 47% of the number of attempts required by an ERS. This means that if we introduce time in the simulation, our variants will probably be faster than an ERS. Although

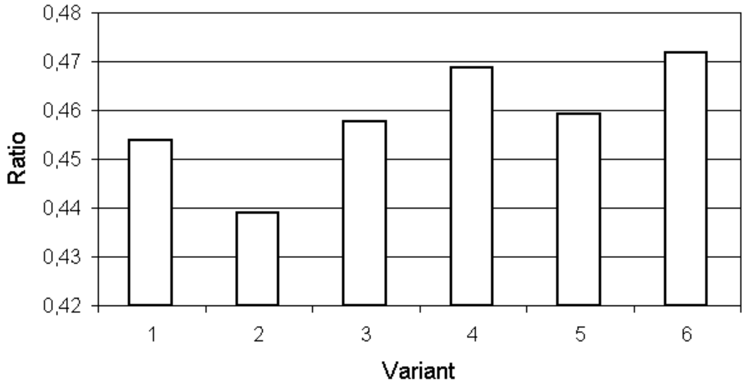

Fig. 5: Number of attempts

not twice as fast because the variants multicast the packets at a greater distance than the ERS thus each attempt will take a longer time. It's worth noticing that variant 2 has the best ratio (the lowest).

The number of routers that have received packets are shown on Figure 6. As usual it is the ratio of each variant score divided by the ERS score. Every variant hit from 3 to nearly 5 more routers than the ERS. This must be compared to the number of packets emitted in the network. We see that there is not an exact correlation with the bandwidth usage. This means that some variants have more redundant packets than others (i.e. a given router is hit by many packets which is useless). From these two figures we can stem the average number of packets received per router and thus evaluate the redundancy of each variant. In the figure we can clearly see two groups of variants. The first group of variants 1 to 3 have nearly the same ratio value of 3, their common point is that their initial range is fixed. The second group of Variants 4 to 6 also have the same ratio but its value is nearly 5, and their initial range is variable. It is clear that it is the initial range parameter value that make the difference between the variants' results. Although the second group hits on average many more routers than the first, they find less optimal agents as we can see in Figure 4. Therefore they are less efficient.

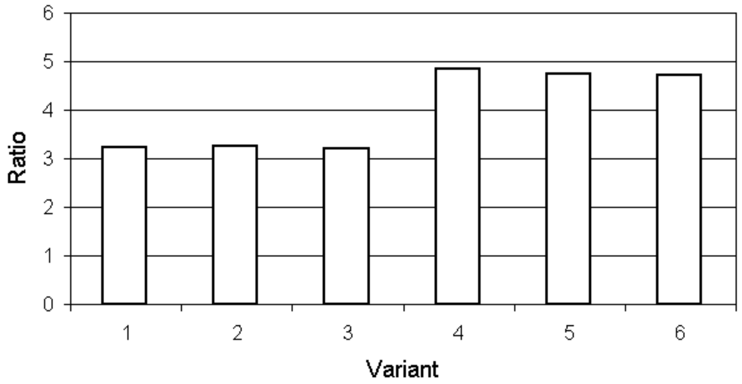

Fig. 6: Routers involved

To compare the variants in a concise way, we have defined a ratio called efficiency. It is equal to the number of optimal agents hit divided by the number of packets emitted in the network. The efficiency of a variant is then divided by the efficiency of the ERS for a quick comparison. The resulting ratio is shown in Figure 7 for each variant. The variants having a fixed initial range (i.e. corresponding to the (a) shape of Figure 1) have a 50% better efficiency than the variants having a variable initial range (i.e. the (b) shape). These variants numbered 1 to 3 have roughly the same efficiency and it is 20% above the ERS efficiency. Variant 3 is slightly ahead. Concerning variants 4 to 6 they also have roughly the same efficiency and it is 20% below the ERS efficiency. Although variant 3 has the best efficiency ratio, we remind the reader that variant 2 has the highest successful search rate, and the two are not balanced. This means that for an efficiency ratio loss of 0.04, the optimal agent hit ratio gains 0.55. So variant 2 is more interesting than variant 3.

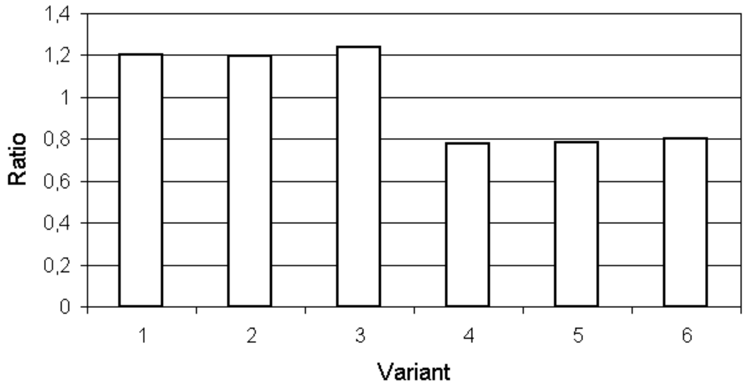

Fig. 7: Efficiency

6 Conclusion

Contribution: We have designed an original oriented multicast routing algorithm that avoids the packet exploding phenomenon correlated to multicast methods such as flooding or reverse path forwarding. We showed that, used in a search framework, our algorithm outperforms the expanding ring algorithm. An agent search protocol based on our framework could give to the client the possibility to find a topologically smart-placed intermediary in the network. Indeed the search optimizes the distance criterion between the concerned entities, which is a matter of interest for any protocol using our search service.

Future work: The second phase of the search algorithm has to be completely defined. We have to study the reply methods of the responding agents as well as the optimization of these reply messages to the source (e.g. by a filtering or aggregation technique). Once the answers are collected, we have to define criteria to select the best agent if more than one are found. The protocol implementation of the oriented multicast and agent search algorithms still have to be done. This would require to further define the algorithms' messages and to introduce timers and finite state machine diagrams. The resulting agent search protocol and its underlying oriented multicast routing protocol could serve many network protocols and particularly multicast protocols. We also have to simulate the dynamic behavior of our algorithms.

References

[AMK98] Elan Amir, Steven McCanne, and Randy Katz. An active service framework and its application to real-time multimedia transcoding. *ACM Computer Communication Review*, 28(4):178–189, August 1998.

[BEH+98] Erol Basturk, Robert Engel, Robert Haas, Vinod Peris, and Debanjan Saha. Using network layer anycast for load distribution in the internet. In *Proceedings of GLOBECOM/GIS'98*, December 1998.

[BFP98] Anindo Banerjea, Michalis Faloutsos, and Rajesh Pankaj. Designing qosmic : A quality of service sensitive multicast internet protocol. In *Proceedings of ACM SIGCOMM'98*, Vancouver, BC, Canada, September 1998.

[CC97] Ken Carlberg and Jon Crowcroft. Building shared trees using a one-to-many joining mechanism. *ACM Computer Communication Review*, 27(1):5–11, January 1997.

[DM78] Yogan Dalal and Robert Metcalfe. Reverse path forwarding of broadcast packets. *Communications of the ACM*, 21(12):1040–1048, December 1978.

[EFH+98] Deborah Estrin, Dino Farinacci, Ahmed Helmy, D Thaler, Stephen Deering, M Handley, Van Jacobson, Ching-Gung Liu, Puneet Sharma, and Liming Wei. Protocol independent multicast-sparse mode (pim-sm) : Protocol specification. Request For Comments 2362, Internet Engineering Task Force, June 1998.

[Fen97] William Fenner. Internet group management protocol version 2. Request For Comments 2236, Internet Engineering Task Force, November 1997.

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *Proceedings of ACM SIGCOMM'99*, Cambridge, Massachusetts, USA, September 1999.

[Hof97] Markus Hofmann. Enabling group communication in global networks. In *Proceedings of GLOBECOM'97*, Calgary, Alberta, Canada, June 1997.

[Mog84] Jeffrey Mogul. Broadcasting internet datagrams. Request For Comments 919, Internet Engineering Task Force, October 1984.

[MP01] Damien Magoni and Jean-Jacques Pansiot. Analysis of the autonomous system network topology. *Accepted for publication in ACM SIGCOMM Computer Communication Review*, 31(3), July 2001.

[PMM93] Craig Partridge, Trevor Mendez, and Walter Milliken. Host anycasting service. Request For Comments 1546, Internet Engineering Task Force, November 1993.

[PSLB97] Sanjoy Paul, Krishan Sabnani, John Lin, and Supratik Bhattacharyya. Reliable multicast transport protocol (rmtp). *IEEE Journal on Selected Areas in Communications*, 15(3), April 1997.

[VIN] The VINT project, http://www.isi.edu/nsnam/vint/. *network simulator (ns-2)*.

[Wax88] Bernard Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, December 1988.

[YGS95] Rajendra Yavatkar, James Griffioen, and Madhu Sudan. A reliable dissemination protocol for interactive collaborative applications. In *Proceedings of ACM Multimedia'95*, 1995.

[ZCD97] Ellen Zegura, Kenneth Calvert, and Michael Donahoo. A quantitative comparison of graph-based models for internetworks. *IEEE / ACM Transactions on Networking*, 5(6):770–783, December 1997.