Autonomous, Scalable, and Resilient Overlay Infrastructure

Khaldoon Shami, Damien Magoni, and Pascal Lorenz

Abstract: **Many distributed applications build overlays on top of the Internet. Several unsolved issues at the network layer can explain this trend to implement network services such as multicast, mobility, and security at the application layer. On one hand, overlays creating basic topologies are usually limited in flexibility and scalability. On the other hand, overlays creating complex topologies require some form of application level addressing, routing, and naming mechanisms. Our aim is to design an efficient and robust addressing, routing, and naming infrastructure for these complex overlays. Our only assumption is that they are deployed over the Internet topology. Applications that use our middleware will be relieved from managing their own overlay topologies. Our infrastructure is based on the separation of the naming and the addressing planes and provides a convergence plane for the current heterogeneous Internet environment. To implement this property, we have designed a scalable distributed k-resilient name to address binding system. This paper describes the design of our overlay infrastructure and presents performance results concerning its routing scalability, its path inflation efficiency and its resilience to network dynamics.**

Index Terms: **Addressing, autonomous, distributed, dynamic, naming, overlay, routing.**

I. INTRODUCTION

Designing an application level addressing, routing, and naming infrastructure for Internet overlays is challenging when no constraint is put on the topology of its members. However, it can be very useful to provide such a middleware for creating these overlays. For instance, setting up a tree topology is easy but provides very little robustness. Complex mechanisms must be used to avoid loops and to recreate the tree in case of branch failures. The advantages of allowing an overlay to have a free topology only restrained by the underlying network (i.e., the Internet for our purpose) are that it is very easy to add or remove nodes without breaking it and that redundant links provide load balancing and increased robustness. On the other hand, a free topology overlay requires a proper routing system that our infrastructure aims at providing. Furthermore, our infrastructure provides a separation between node addressing and naming. Thus, our overlay infrastructure enables applications:

- To run seamlessly over private and public addressing spaces (even over private networks having address overlapping),
- to use node mobility without network layer support,
- to mix node mobility with secured connections,
- to use application layer multicast,
- and to use name based instead of address based security.

The use of our infrastructure enables networked applications to work over the current heterogeneous Internet, for example, (i.e., NAT routers, IPSec tunnels, etc.) in a true end to end mode (necessary for P2P applications for instance) while using mobility, multicast, and security in a seamless way.

Our paper contains four sections. In Section II, we briefly present prior and related work on overlay networks. In Section III, we describe how our infrastructure is designed. In Section IV, we discuss the benefits brought by using this infrastructure. Finally, in Section V, we present the performance results of our infrastructure obtained by simulations and concerning its routing scalability, its path inflation efficiency and its resilience to network dynamics.

II. RELATED WORK

An important point that many people agree on is that naming and addressing should be separated [1]–[3]. Many problems could be elegantly solved if this feature was provided by the IP protocol. Solutions providing indirect addressing are proposed by many experimental protocols (e.g., INS [2], INPL [4], and i3 [5]) and especially by those designed for host mobility (e.g., TCP-Migrate [6] and Tribe [7]). All these solutions do create some forms of overlays, although not always at the application layer, in order to solve issues such as uniform addressing and mobility. Application layer overlays providing multicast support have also been designed and implemented (e.g., Narada [8], NICE [9], and ROMA [10]) in order to be used over networks that do not run multicast protocols. Overlays can also be designed to provide new services such as resilient networking (e.g., RON [11]) and peer-to-peer objet lookup (e.g., Chord [12], Pastry [13], etc.), thus, opening new opportunities for network applications.

On a more theoretical level related to local routing, the tradeoff between routing table sizes and path lengths has been actively studied by the distributed algorithm community. Eilam *et al.* have proved in [14] that it is possible to bound the average stretch (i.e., path length inflation) by 3 with routing tables of size $O(n^{3/2}\log^{3/2}n)$. Similarly, Cowen has proved in [15] that it is possible to bound the maximum stretch (i.e., path inflation) by 3 with routing tables of size $O(n^{2/3}\log^{4/3}n)$ and Krioukov *et al.* have shown that compact routing in the Internet yields an average stretch of 1.1 [16]. However, to achieve their goal, they use an appropriate labeling for every vertex and both do not describe how to implement it as a distributed algorithm. Concerning overlay topologies, Li *et al.* have shown in [17] that they do have an impact on the routing performances thus further motivating our work. In this paper, we present an infrastructure

Manuscript received April 14, 2006.

K. Shami and P. Lorenz are with the GRTC, Université de Haute Alsace, France, email: khaldoon.shami@uha.fr, lorenz@ieee.org.

D. Magoni is with the LSIIT-CNRS, Université Louis Pasteur, France, email: magoni@ieee.org.

1229-2370/06/$10.00 © 2006 KICS

where table sizes are not a function of the network size but a function of the node degrees. Although our infrastructure does not provide an upper bound on the average stretch, it is typically below 2.3 as shown in Section V.

Our work is related to network convergence as fully explained in Section IV because our overlay architecture restores, at the application layer, one of the fundamental Internet paradigms which is the existence of a unique public identifier for everyone. This was the key success of the network convergence leading to the creation of the Internet. Today, new networks rise again such as IPv6, private NATs, GSM, UMTS, etc. They all have different addressing schemes. As our architecture can cope with any underlying protocol, it can bind these heterogeneous networks in order to restore the use of a globally unique and reachable identifier. It concerns broadband in the sense that it is fully scalable and our experiments detailed in Section V show that our architecture works with more than 10,000 nodes. Thus, it is not destined to small private networks but to huge networking communities located everywhere in the currently interconnected networks (e.g., Internet, cellular networks, etc.). By introducing additional layers in the members of our overlay network we not only restore the end-to-end properties of the original Internet but we also provide support for advanced networking such as transparent mobility, group communications and security. All these services are currently hard to deploy at the network layer and are usually not factorized in a single architecture and software. Thus the advantages of our architecture when building an overlay are

1. the creation of a global dynamic addressing space at the application layer that ensures network convergence,
2. the creation of a persistent naming space that provides not only machine names but also service and group names and ensures user application level security,
3. the use of a scalable and dynamic binding system between these naming and addressing spaces in order to provide support for mobility, security, and group communications simultaneously.

The disadvantages of our architecture are

1. the introduction of a middleware that must run on the nodes participating in the overlay (however, nodes can withdraw and rejoin at will),
2. the slight loss of performance introduced by the protocols of our middleware layer (e.g., routing may be suboptimal compared to network layer routing),
3. the need of processing and storage power in the overlay nodes in order to cope with the autonomous distributed naming and addressing system that runs in our middleware.

III. INFRASTRUCTURE DESCRIPTION

As we have designed an infrastructure that puts no constraint on the topologies of the overlays, we have to define a distributed addressing mechanism in order to properly route data packets inside the overlay. Our overlay infrastructure is currently nicknamed DHARMA which stands for dynamic hierarchical addressing, routing, and naming architecture.

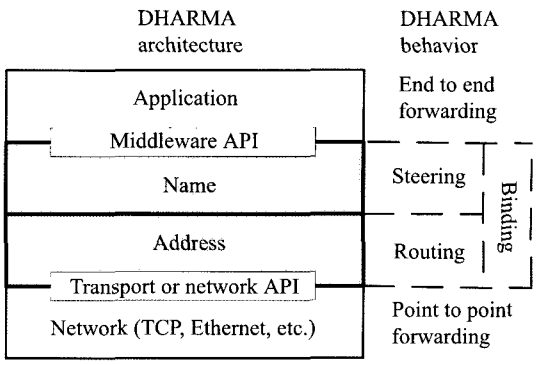

Fig. 1. Middleware layout.

A. Layout

Fig. 1 shows the infrastructure and middleware layout. We can see the addressing plane located over the network layer and the naming plane located over the addressing one. The process of forwarding packets in the overlay is divided into two parts.

The first part, called routing, is similar to the current routing in the Internet. Overlay routing only uses the overlay addresses. The network or transport connections between overlay nodes are considered point-to-point connections. If a reliable transport layer protocol is used (such as TCP) then no transmission reliability mechanism is needed in our middleware. If a data link layer protocol (such as Ethernet) or an unreliable transport layer protocol (such as UDP) is used then a transmission reliability mechanism is needed in our infrastructure. We have not studied this issue yet and we assume at the moment that the application using our middleware will take care of the transmitted data reliability if the connections are not using a reliable transport layer protocol. In the rest of this paper, we assume that the overlay nodes connect themselves via TCP.

The second part called steering is specific to our infrastructure and introduces more flexibility in the overlay communications especially for mobile, multicast, and secured communications as explained in Section IV. Overlay steering consists in binding the names of the overlay nodes to their current addresses before or during a data transmission between overlay nodes. It can be seen as a second stage routing system. It also binds various logical names (user, service, or group names) to overlay node names before or during a data transmission between overlay nodes. In order to make the steering work, the names must be properly mapped to the correct addresses. This binding is managed by a subset of the overlay nodes that agree to play the role of name servers.

B. Addressing

Each overlay node has an address. An address is composed of one or several fields containing numbers and separated by dots, one field for each level of the address hierarchy. Each field of an address is also called a label. The level of the address is equal to the number of fields in the address. The prefix of an address is equal to the address without the last field. The last field is called the local field or local label. The number of levels in the hierarchy is not fixed but totally dynamic. Its value depends on how the addresses are distributed at any given time. It is worth

noticing that the length of the label should be defined once at the creation of the overlay. Each node in the overlay network has at least one address and typically more in order to cope with the network dynamics as explained later.

The addressing plane contains zones that correspond to the address fields. The label size, thus, defines the maximum zone size. All zones have the same fixed size n (called the zone size). There is one level 1 (i.e., top level) zone containing n nodes (defined in the first address field). Then, there are at most n level 2 zones each containing at most n nodes (defined by the first two address fields). Then, there are at most n^2 level 3 zones each containing at most n nodes and so on. This means that all the address space can be theoretically distributed and if we have k levels and l bits per level, we can address $2^{k\times l}$ nodes. The aim of this hierarchical addressing is to enforce the construction of zones of limited size in order to make routing scalable.

The addressing of the overlay nodes is fully distributed: It relies only on local knowledge in a node. The only local knowledge we rely on is the degree of the node and the addresses and degrees of its neighbors. Any node is supposed to know this information. Let us assume that the zone size is n. Each node that has address $w.x.y$ is responsible for allocating the following addresses to its neighbors:

- The address $w.x.(y+1)$ (called a "next" address) where $(y+1) \leq n$,
- the address $w.x.y.1$ (called a "down" address),
- the addresses $w.x.y.z$ (called a "leaf" address) where $z > n$.

The first node of the overlay takes the address 1. The nodes connect to each other by the use of transport protocols (e.g., TCP, etc.). As transport protocols are used for setting point to point connections between pairs of overlay nodes, different protocols can be used simultaneously, layer 2 protocols can be used and unique network layer addresses are not required among all overlay nodes. In order to connect to an overlay, new or moving nodes must know the network address (usually IP addresses) of at least one overlay node. This address should be provided by out-of-band methods (e.g., email, WWW, SDP, etc.) via the application that is using this overlay. The overlay nodes directly connected by data link or transport protocols to a given node are defined as its *neighbors*. Nodes join the overlay successively by connecting themselves to already connected ones. When a node wants to join the overlay, it asks for address proposals to all its neighbors. Each neighbor proposes an address to the joining node, from the possibilities of the above list, that it has not already given to one of its other neighbors. The joining node then chooses one or more addresses with the following priority:

- The shortest address,
- in case of draw, the shortest "next" address,
- if no "next" address, the shortest "down" address,
- if no "down" address, the shortest "leaf" address.

Fig. 2 illustrates a joining node requesting addresses from its neighbors. As said above, a leaf address is an address whose local label is above the zone size value (e.g., if the zone size is 32, the first leaf label is 33). Nodes that have a leaf address can only route data to their father even if they are connected to other nodes, they are considered as leaf nodes for the overlay routing system. That is why they are chosen by newcomers with the lowest priority.

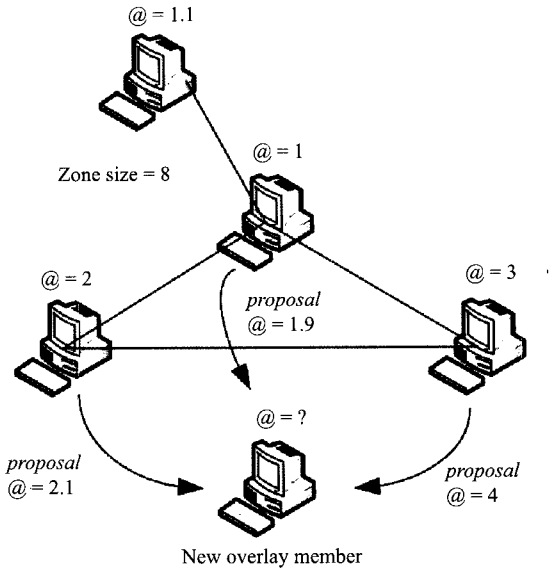

Fig. 2. Joining overlay node asking for addresses.

C. Routing

Routing in the overlay consists in using the addresses of the overlay nodes in order to transmit overlay-level data packets to their proper destination. As we saw in the previous section, the address(es) obtained by an overlay node depend(s) on its neighbors and thus on its localization. Thus, our routing mechanism is address-driven (i.e., some path information is stored in the addresses). The core principle of our infrastructure is that every node only needs to store the addresses of its neighbors in order to properly route the packets towards their destination. Thus, its routing table only contains the (few) addresses of its neighbors. However, as the path towards the destination is partly contained in the destination address itself, the path length depends on the efficiency of the address allocation.

In hierarchical routing, when a packet is routed in a node, if the destination address is down this node hierarchy, the packet is driven to the node of the current zone that leads further towards the destination zone (we call it the ingress node). If the destination address is not down the current node hierarchy, the packet is driven to the first node of the zone (i.e., the one with a local label of 1) in order to be sent to the upper level zone (we call it the egress node). When a packet is routed inside a zone because the destination is in the zone or to go to the ingress or egress node, it is routed by a technique that we call the closest label. This technique only needs to know the addresses of the neighbors. The closest label routing technique works as follows. If the destination local label is lower than the current node local label, then the packet is forwarded to the neighbor node which has the lowest local label higher or equal to the destination local label. If the destination local label is higher than the current node local label, then the packet is forwarded to the neighbor node which has the highest local label lower or equal to the destination local label. As the neighbors have a continuous label assignment, this technique ensures that the packet will reach its destination although not necessarily by a shortest path.

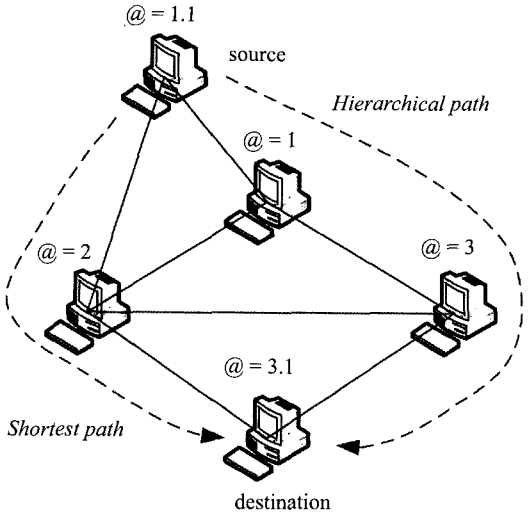

Fig. 3. Path inflation caused by hierarchical routing.

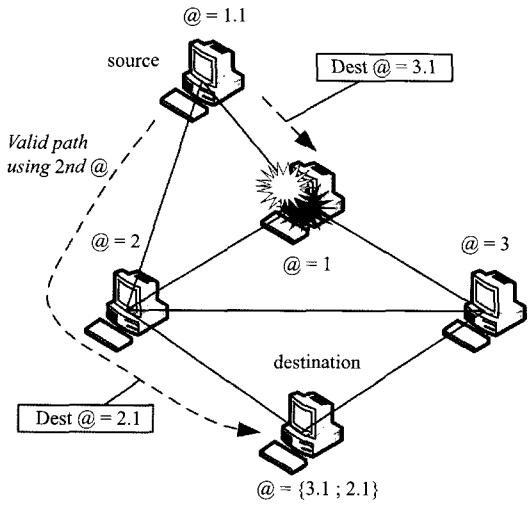

Fig. 4. Alternate routing for solving network dynamics.

Fig. 3 illustrates the effects of hierarchical routing and shortest path routing between the nodes 1.1 and 3.1 on path lengths. The hierarchical routing forces the message to be routed via 1 and 3 thus giving a path length of 3 hops while a shortest path routing requires only 2 hops via 2 to reach the destination. We define the path length ratio (or path inflation) as the value of the hierarchical routing path length in hops divided by the shortest path routing path length in hops.

If a node moves or fails, thus invalidating its address, all packets routed to itself or to a destination address that contains its invalid address will not be able to be routed anymore. To solve this issue, two techniques are simultaneously used

- addresses are always stored temporarily (i.e., soft state) and nodes having lost connections will get back their address or get new addresses from the operational (working) nodes to which they reconnect to.
- Each node can simultaneously use several addresses provided by different neighbors, thus allowing itself to be reached by alternate paths if the current used path is blocked by a moving or failed node.

All addresses are given a timeout value v_1 and have to be periodically refreshed by hello messages sent by the address giving neighbor(s). Invalid addresses and derived addresses thereof will not be refreshed. If the owning node does not reappear again (i.e., in the case of a permanent move or failure) at the end of a timeout value v_2 ($v_2 > K_{\text{depth}} \times v_1$), the node responsible for this address (i.e., the one that gave it) will be able to allocate it to another node. In this case, all the addresses derived from the invalid address will also be flushed. If all destination addresses fail during the data routing, the sending node must wait until the connectivity to the destination is restored (i.e., connections are re-established and addresses are re-allocated).

Each node can be located by several addresses (i.e., more than one). The additional addresses can be chosen at the time of insertion in the overlay or later on when the overlay connectivity changes and more addresses become available to the node as a result. All the addresses owned by a given node must come from different neighbors (checks are made on the node's name). All the addresses owned by a given node must be different, that is they must not have a common prefix. Otherwise, if the disappearing node address is included in the common prefix, all addresses will not work. This multiple address allocation increases the amount of routing information to be stored by a factor equal to the number of addresses per node but the advantage is that the network dynamics are handled transparently by the addressing protocol. If a packet can not be routed because a node has disappeared, it can use one of the alternate addresses to get through to the destination as shown in Fig. 4. Two solutions are possible: Either the packet carries all the destination addresses and thus it can be rerouted on the fly by using its alternate addresses (but this uses more bandwidth) or a warning message is sent back to the source which then will change the destination address by an alternate one in all future packets.

D. Naming

We have seen above that every node in the overlay has one (or several addresses to cope with network dynamics). These addresses depend on the location of the node in the overlay and they are subject to network dynamics (i.e., addition, removal, or movement of nodes). In order to be able to communicate seamlessly, every node in the overlay has a unique name that remains the same over the lifetime of the node. Applications using our infrastructure use the names of the nodes to establish communication links between them. Thus, address changes in nodes are transparent to the applications. The binding between the names and the addresses are done via name servers. Name servers are regular overlay nodes that accept to carry on the name solving tasks because they have more capacities and they move much less than the other nodes (the node having address 1 is always a name server). In order to be scalable, the name servers are organized in a tree hierarchy. The tree depth plus one is equal to the hierarchy maximum level. Each name server has a hash table storing the addresses of its next level name servers and a name table storing the name-to-address mappings as shown in Fig. 5. A name is composed of several parts. Each part corresponds to one level of the name server hierarchy. It is not a problem if

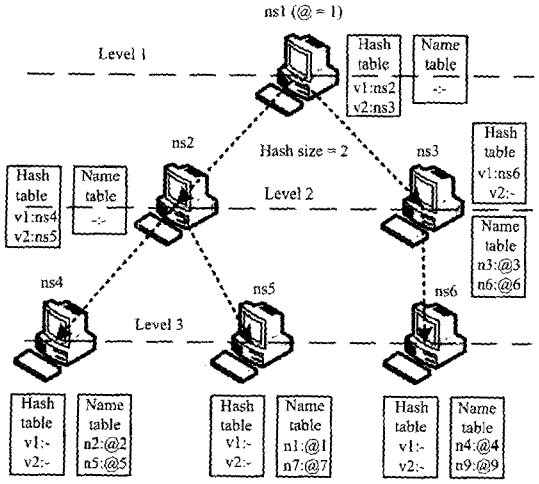

Fig. 5. Hierarchical organization of name servers.

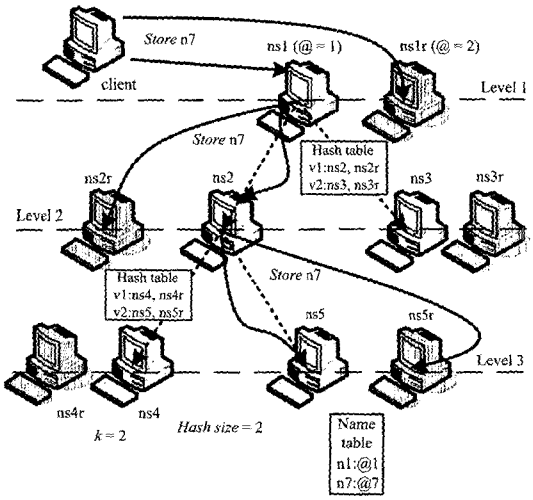

Fig. 6. Name to address storing procedure.

the number of levels is different from the number of parts. The name servers' hierarchy is totally independent of the addressing hierarchy. The scalability of the namespace storage is ensured by a procedure loosely based on the principle of distributed hash tables.

In the following, we assume that there are no name server dynamics (i.e., failures). When a new node has joined the overlay, it has selected one or several addresses. It then chooses a name and sends a message containing its name and addresses to the node 1 for storing this information in a name server. On reception, the node 1 performs a hash on the first part of the new node's name and sends the message to the corresponding second level name server. On reception, the second level name server hashes the second part of the name and sends the message as before, down in the name servers' hierarchy. If the name has no more part to hash or if there is no entry for the hash result, or if the hash table is empty in a name server, then this last server has to store the name and addresses in its name table if this name does not already exist (otherwise, another name must be chosen by the new node). When a node wants to obtain the addresses of a destination node given its name (the name is supposed to be known by an external mechanism), it sends a request message to the node 1 containing the name to solve and its own address. The request is forwarded to the proper name server by hashing the name exactly as during a store operation. The name server holding the name will send back a message containing the addresses of the destination name to the request node by using the sender address in the request.

In order to provide load balancing and cope with name server dynamics, we use a replication approach. We want our naming system to be resilient for up to k failures. Thus, we assume that a redundancy factor k is chosen at the start of the overlay construction. The total number of servers equals s. Each name server has $k-1$ exact copies called replicas. The k first level nodes having addresses 1 to k will be serving as first level name servers. They will have a copy of the hash and name tables of the node having address 1 and they will perform the same functions thus acting as redundant servers. Clients (i.e., nodes requesting addresses) can, therefore, send store and request messages to nodes 1 to k. Also each hash entry in any server in the hierarchy will store k name servers instead of one. The result of a hash will provide up to k suitable servers if all are operational and one will be picked randomly for receiving the message. In the lower levels, as for the first level, the k name servers corresponding to one hash entry will have to maintain the same hash and name tables as they act as redundant servers. Thus, there is a tradeoff between providing load balancing and fault tolerance and managing replication for the s cliques of k identical name servers. We also envision to use name caches in all nodes of the overlay in order to increase performances.

Fig. 6 shows how the names are stored in a naming system with $k = 2$ (i.e., one replica per name server). We suppose that no servers are down here. A client overlay node named n7 (and having address @7) sends a store message to the first level name servers, namely the overlay nodes having the addresses 1 and 2 (as $k = 2$). Then, the first server (with address 1 here) forwards the store messages to the level 2 name servers found in its hash table (here hashing part 1 of n7 gives v1 for instance which points to ns2 and ns2r). Then, the level 2 server (ns2 here) forwards the store messages to the level 3 name servers found in its hash table (here hashing part 2 of n7 gives v2 for instance pointing to ns5 and ns5r). Then, both ns5 and ns5r store the address @7 of the name n7 in their name tables as they are at the bottom of the hierarchy. Thanks to this procedure, all replicas of a given name server have exactly the same name table. Notice that a name server and each of its replicas maintain a list of themselves called a pool list. Thus, they can communicate between themselves and ensure that they have the same hash table. As Fig. 6 shows, a name server and its replicas have the exact same name table and hash table. If a server is down during a store operation that concerns it, it can, upon restart, wait until the next store message of the client (as the names are soft-state stored) or ask one of the other replicas by using its pool list.

Fig. 7 shows how the names are retrieved in a naming system with $k = 2$. We assume that node named n7 tries to solve its own name in order to be consistent with the previous figure

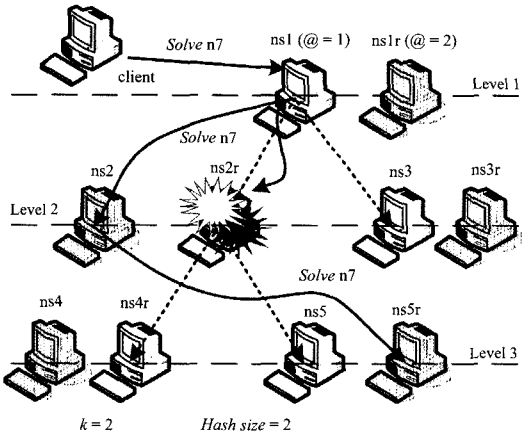

Fig. 7. Name to address solving procedure.

This client thus sends a solve message to one of the first level name servers. Server ns1 is up and tries to forward the request to ns2. As ns2 is down, ns1 tries to forward the request to the next replica of ns2 found in its hash table, namely ns2r. This behavior ensures the resilience of the naming system. As ns2r is up, it receives the request and sends it to ns5r instead of ns5 thus providing load balancing.

There are at least two major differences between our naming distribution strategy and the famous DNS [18] that maps domain names to IP addresses. First, domain names are still mainly aliases of IP addresses and if an IP device goes in a different IP network, it will obtain a different IP prefix and thus it will usually not be able to keep its domain name (some dynamic DNS services keep up to date with changing IP, mobile IP can help, too). Second, our solution does not make use of iterative calls such as in the DNS. Client messages are sent to top level name servers that forward them to servers down the hierarchy and finally, the server holding the desired addresses replies directly to the client. Finally, our infrastructure must be autonomic for each and any overlay in order to be deployed by hosts without constraints thus we can not use or ask for modifications in the current DNS service.

E. Steering

Steering consists in mapping a name to an address at any moment during the routing process inside the overlay. Thus, any overlay packet contains the addresses as well as the names of the overlay source and destination nodes (in unicast). The primary aim of steering is to enable the applications to establish connections by using names rather than addresses thus hiding any topology changes from applications. Steering can be done at the beginning of a connection or on the fly by any overlay node during a connection if one of the source or destination nodes is changing its address or if a group name is used as destination name (as in multicast). The connection of two overlay nodes by using their names is called a virtual connection. As a name is an invariant over the lifetime of the entity and is unique, their use can restore the original assumptions initially put on Internet addresses: Uniqueness and durability. These properties of original IP addresses have been exploited in many ways, in particular by incorporating them in transport identifiers. Thus, they have been built into transport check-sums, cryptographic signatures, web documents, etc. The name layer of our middleware restores the property of Internet end-to-end transparency currently lost because of dynamic address allocation, firewalls, NATs, etc. The basic sequence for establishing an unicast communication in the overlay between node named A and node named B is

1. a joins the overlay network by connecting to an overlay node C located close to A and discovered via an out-of-band source (web, email, etc.),
2. a retrieves the name of B from a directory distributed in the overlay (this is currently left to be managed by the application using the overlay) or from an out-of-band source (web, email, etc.),
3. a retrieves the address of B from an overlay name server by using B's name and sending a request to one of the overlay nodes that have addresses between 1 and k (level 1 name servers),
4. a sends packets inside the overlay to B and includes its own address in the first few packets so B can reply without querying a name server.

IV. INFRASTRUCTURE BENEFITS

The benefits provided by the separation of naming and addressing in our infrastructure are multifold.

A. Networking Convergence

In a world where IPv6 would be used everywhere as well as mobile IPv6 and router level multicasting, there would most probably be no need for our overlay infrastructure. However, we think that this idealistic scenario will require several years before becoming a reality. Distributed applications might want to use advanced network services now in the real world where IPv4 and IPv6 co-exist, IP mobility and router level multicast are very scarcely deployed and public IP addressing is coping with private IP addressing enabled by NAT. Our overlay infrastructure is designed to provide a uniform addressing space, mobility management, and multicast support to networked applications deployed over the currently highly heterogeneous Internet. As our overlay infrastructure is completely autonomic in terms of addressing, routing, and naming, only the computers willing to use an application using our infrastructure will have to run an instance of our middleware.

Fig. 8 shows a small scenario where four overlay nodes are connected by using our middleware. Each node is an overlay neighbor of its leftside node and rightside node. On this figure, the real network connectivity between the nodes is shown (instead of the overlay connectivity as on the other figures). The laptop is connected to the leftmost desktop computer at the OSI data link layer by using IEEE 802.2. The leftmost computer is connected to the center computer at the transport layer by using a TCP connection. The center computer is connected to the rightmost computer also at the transport layer by using UDP. As explained in Section III-A, these data link or transport layer connections are considered point-to-point overlay connections.

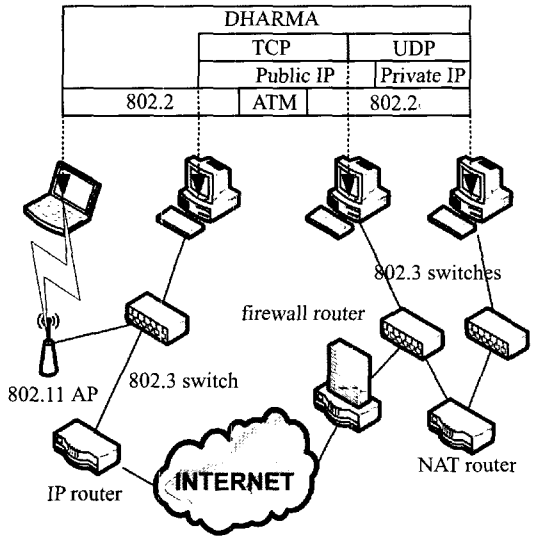

Fig. 8. Network convergence with our middleware.

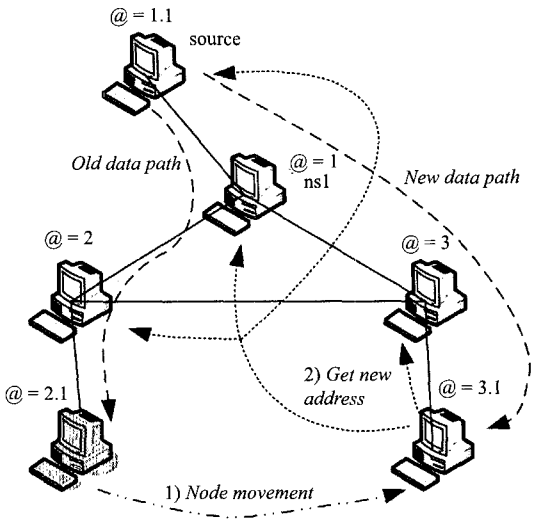

Fig. 9. Mobility scenario.

Reliability must currently be provided by the application if no reliable transport protocol is used. However, we are currently implementing reliability in our middleware to free the application from this task. In this scenario, if the laptop changes its IP address (because of DHCP or moving in another IP subnet) the overlay communications will not be broken as they rely only on the overlay name. Even if the laptop moves to another WiFi network, it will only need to connect (bootstrap) to another overlay node and gain a new overlay address but communications will be maintained. The firewall can be crossed by encaspulating DHARMA data in TCP connections using open ports. The NAT can be made transparent to the application once a transport connection is established between one private IP addressed node and a public IP addressed node (the connection is initiated by the private node). Thus, a distributed or P2P application can fully work on top of our overlay infrastructure even if the underlying nodes are located in a mix of public and private IP subnets.

B. Mobility

Let us assume that a *destination* mobile device establishes a connection with a *source* correspondent having address 1.1. Fig. 9 shows the location of the mobile node, the source and three other overlay nodes. A binding is created between the mobile original address 2.1 and the source. The packets are routed in the overlay from the source through nodes with addresses 1 and 2 to the mobile with address 2.1 (thick dashed arrow line). If the mobile moves to a new location, it will get a new address from its new neighbor and then it will send update messages (thin dotted arrows) to the name server ns1 and to the source and optionally to its previous neighbor having address 2 in order to give them its new address (i.e., to update the mapping between its name and its address in the caches of the source, the previous neighbor and in the naming system). Some packets will be lost but eventually, the source will send them to the mobile through nodes 1 and 3 without breaking the application layer binding. The optional update message to overlay node with address 2 will enable it to reroute late packets to the new mobile address (i.e., to steer packets). This mechanism will also work for network mobility. However, when a network connects itself to a new location, the new prefix will have to be propagated everywhere inside it and sub-level prefixes will have to be rebuilt.

C. Security

The use of IPsec can be hindered when coping with mobility in the current Internet architecture. In IPsec, a security association (SA) is uniquely identified by a triple consisting of a security parameter index (SPI), an IP destination address, and a security protocol (AH or ESP) identifier [19]. However, if the destination is a mobile, the SA will be invalidated when the IP address of the mobile changes. It is possible to use middleboxes such as firewalls to securely tunnel the traffic through the public Internet. But, what if the spy is inside the network of the end-user? Securing strictly end-to-end will always be safer than tunneling and it will relieve the firewalls from becoming hot spots for attacks. By using the name defined in our infrastructure instead of the address, a SA can remain valid even if the underlying topology changes (e.g., the mobile moves and its address changes).

Furthermore, a name in the overlay can define a user or a group rather than simply a node, which is much richer semantically and thus enables a finer control of the communication policies. It also makes management and auditing easier. Instead of having to authorize all possible node names from where a given person could connect to a given service, only the name of the person will need to be stored in the policy database.

Firewalls will also make a great benefit of using the name rather than the address. By using a name referring to a service instead of a node (for a server), firewalls will be able to filter the traffic depending on its meaning whatever the networking conditions (e.g., it will be possible to filter encrypted data from a mobile source). This is not possible today especially when the ESP protocol [20] is used. By encrypting the transport header, ESP prevents firewalls from filtering on the application type. Finally, firewall configuration will be easier and more meaningful

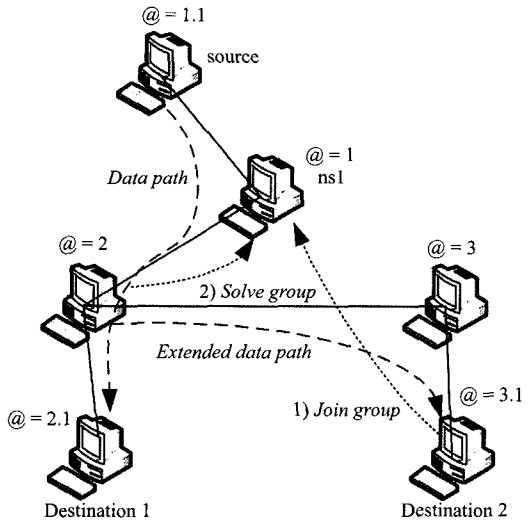

Fig. 10. Multicast scenario.

by using names. All these issues lead us to conclude that deploying a secured infrastructure over our middleware is much more flexible and efficient than deploying it over a regular IP network.

D. Multicast

In multicasting, two name levels are used. One virtual name for the group and the names of the overlay nodes participating in the group. The name servers store and solve all types of names (group or node). Let us assume that a multicast source with address 1.1, having registered the group name to the overlay naming system, starts emitting a video stream to this group name. This group consists in node Destination 1 with address 2.1 and will be joined by the overlay node Destination 2 with address 3.1 as shown in Fig. 10. Destination 2 sends a join message (thin dotted arrow) to the name server ns1 in order to be bound to the group name. Overlay nodes on the data path (i.e., nodes with address 1 and 2) will solve the group name stored in the packets into the overlay node names and will duplicate the packets if necessary (i.e., if the group members' nodes are not reachable by the same output connection). Thus node with address 2 will duplicate the packets as shown in the figure by the thick dashed lines.

V. EXPERIMENTS

A. Settings

In order to evaluate our overlay infrastructure, we have used one 12 k-node IPv4 map made in July 2004 and one 4 k-node IPv6 map made in June 2003, both collected by using our IP topology mapping *nec* software [21]. These maps are more accurate with regard to their amount and placement of links than the maps produced by previous efforts as we show in [22]. We assume on first approximation that overlays deployed over the Internet can be represented as subgraphs of these maps.

For the overlay construction, the first node is a randomly chosen node having an above average network level degree (>10). Overlay nodes and links are then gradually added to the overlay from the nodes and links of one given map.

For network dynamics, we have analyzed periodical percentage of random node removal ranging from 0 to 50% of the map size and allocating 1 to 4 (at most) addresses to each node. Network dynamics are a macroscopic way to simulate the addition, removal, movement and failure of the overlay nodes. At the beginning of the simulation all nodes belong to the overlay. Before the simulation starts, a given $x\%$ of nodes are randomly selected and removed from the overlay. After every 10 runs, all the removed nodes are re-inserted in the overlay and again the same % of nodes are randomly selected and removed from the overlay. Although x remains the same, the actual nodes that are removed each time will be different most of the time especially when x is low. This simulates the addition, removal, movement and failure of the overlay nodes while keeping the size of the overlay equal to $(100 - x)\%$.

For name servers, we have selected random nodes having an above average network degree (>5). For name server dynamics, we have studied periodical percentage of random node removal ranging from 0 to 50% of the total number of name servers. Name server dynamics are handled as regular node dynamics. At the beginning of the simulation all name server are up and running. Before the simulation starts, a given $x\%$ of all the name servers are randomly selected and labeled as failed servers. After every 10 runs, all the failed servers are re-inserted in the overlay and again the same % of servers are randomly selected and labelled as failed. Although x remains the same, the actual failed servers will be different most of the time especially when x is low. This simulates the random failures of the overlay name servers while keeping the size of the running servers equal to $(100 - x)\%$.

As the process of generating addressing planes, selecting name servers and selecting source and destination nodes involve random selection (and thus random rolls), we have used a sequential scenario of simulation [23] to produce the results shown in the next section. As the random rolls are the only source of randomness in our simulation, we can reasonably assume that the simulation output data obey the central limit theorem. We have performed a terminating simulation where each run (one run is the time horizon) consists in picking an overlay source node and an overlay destination node and determining:

- The success of the name resolution of the destination node,
- the distance in hops of the name resolution including the reply,
- the total amount of packets sent for the name resolution,
- the flat and hierarchical distances between the source and destination nodes,
- and the success of the hierarchical routing in presence of network dynamics.

In order to reduce calculation costs, the creation of the addressing and naming planes as well as the placement of the name servers are done every 100 runs, the network nodes and name servers dynamics described above happen every 10 runs and the sequential checkpoints are carried out every 5 runs. This explains the small fluctuations found in some plots. All the simulation results have been obtained assuming a confidence level of 0.95 with a relative statistical error threshold of 5% for all mea-

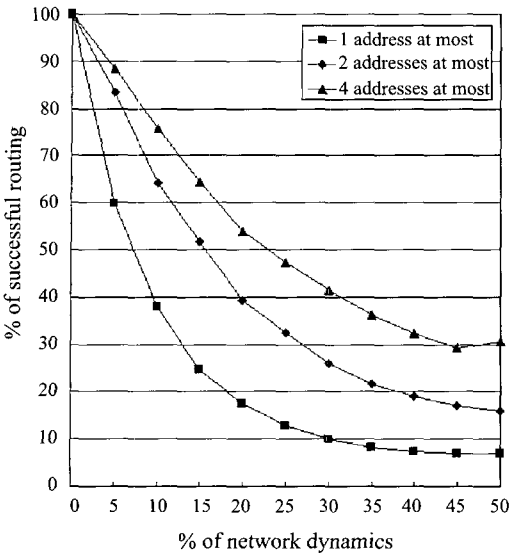

Fig. 11. Percentage of success vs. network dynamics.

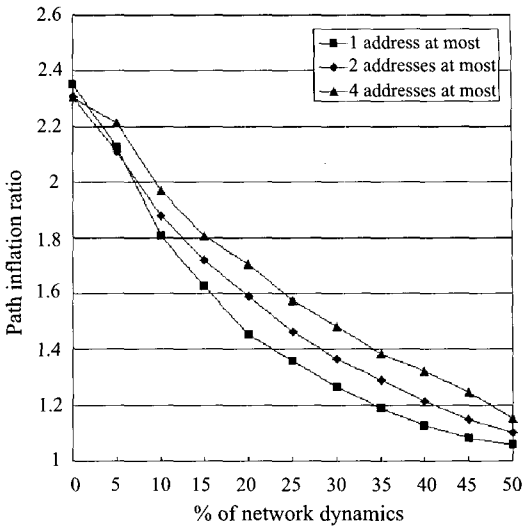

Fig. 12. Path inflation vs. network dynamics.

sured metrics. Simulations have been carried out in our static simulator *nem* software [24].

B. Results

In all our simulations, the results obtained with the IPv6 map were very close to the ones obtained with the IPv4 map when expressed as percentages (e.g., of the number of nodes in the overlay, of the average path length, etc.). Thus, we only show here the results for the IPv4 map unless specified otherwise.

Fig. 11 shows the percentage of successful routing attempts as a function of the network dynamics percentage. As explained above, a given percentage of nodes are absent, thus the overlay may not be connected but composed of multiple connected components. The percentage is calculated as the number of successful hierarchical routing attempts divided by the number of successful flat routing attempts. As the hierarchical path is longer than the flat (i.e., shortest) path, it may go out of the source-destination component and thus it will make the routing fail. We can see that with only one address (i.e., no route alternative), 20% of dynamics makes the success rate fall under 20%. However, the addition of addresses to the nodes heavily increases the routing success. With up to 4 addresses per node and 20% of dynamics, the success reaches 55%. Increasing the maximum number of addresses per node does not linearly improve the success because the maximum number of addresses per node is still bounded by its neighborhood size (and this is small for most of the nodes because of the underlying Internet topology).

Fig. 12 shows the path inflation as a function of the network dynamics percentage. First we can see that the path inflation is around 2.3 when all overlay nodes are operational. This path inflation result not taking dynamics into account is coherent with those of our early work [25]. This is a good ratio for an application layer routing protocol as discussed in Section II. The path inflation is decreasing when the dynamics % is increasing because as the network becomes more fragmented the connected components become smaller and so do their inner paths.

We saw in Section III that addresses are stored in soft state

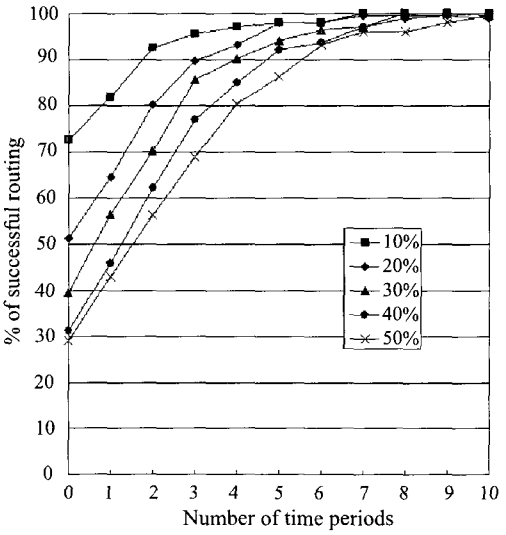

Fig. 13. Percentage of success vs. addressing convergence time.

and have to be refreshed at a regular given time interval. We call this duration a time period. We assume that all overlay nodes are using the same time period value but it could also be depending on the mobility of the node (we leave this evaluation for future work). Fig. 13 shows the percentage of successful routing attempts as defined before as a function of the number of time periods. Each plot corresponds to a given percentage of network dynamics ranging from 10% to 50%. For these plots, each node could have up to 4 addresses at most (the plots with lower maximum addresses per node are worse but have the same characteristics). These plots show the percentage of chance that an overlay source node has to reach an overlay destination node. We can see on these plots that given a few time periods (typically more than 5), there is a convergence of the addressing algorithm that almost guarantees the routing to succeed (above 95%).

We evaluate now the efficiency and scalability performances of our naming resolution system. Fig. 14 shows the average

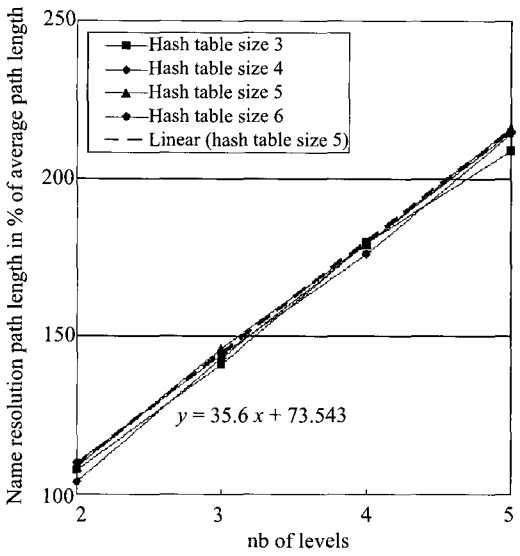

Fig. 14. Name resolving path length ratio vs. number of levels.

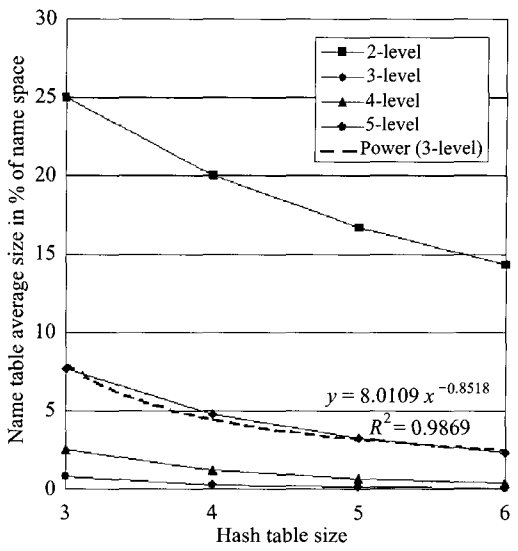

Fig. 15. Name table size vs. hash table size.

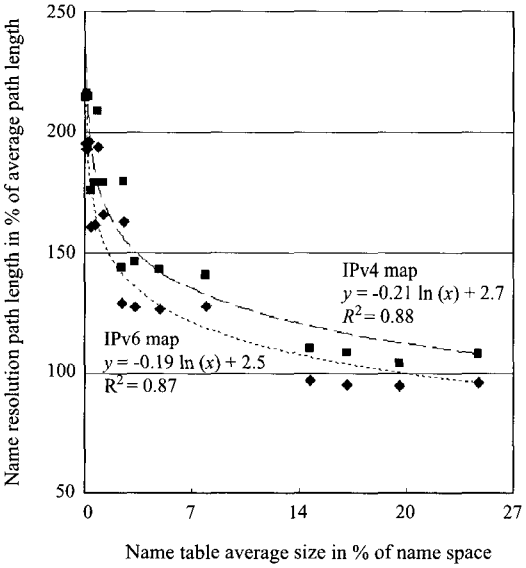

Fig. 16. Name resolving path length ratio vs. name table size.

path length or distance d (expressed as a % of the average round trip in hops) of a name resolution including the answer with respect to the number of levels in the hierarchy l and the hash table size h. Recall that h is the maximum number of next level name servers under one name server (fanout). Although this is expected that the distance is increasing as the levels increase, the plots surprisingly show a linear fit (as shown in the figure for a hash table size of 5). However, the values remain always below 2.5 times the average round trip between any pair of nodes. These results show that the name resolution has a reasonable distance (and thus delay) cost. Indeed, 5 hierarchy levels can handle a very large amount of names. We can also see on the plots that the distance does not vary with the hash table size as all the plots are very close. This is expected as the hash table size will only have an effect on the distribution of the name load at each level. Thus we can write $d \propto l$ (1) when l is small.

Fig. 15 shows the name table size n (expressed as a % of the name space) with respect to the hash table size h and the number of levels in the hierarchy l. We can see that n is a function of h to the power of a constant for l fixed (as shown in the figure for a 3-level hierarchy) and that it decreases when h increases. This is expected by the theoretical equations. If we call s the number of name servers in our system, we have $s = \frac{h^l - 1}{h-1}$ (2). Furthermore, if the names of the overlay members m are well distributed in the s servers, then the names are stored in the leaves of a balanced h-ary tree and we can approximate $n \approx \frac{m}{h^{l-1}}$ (3). The explanation is that increasing the hash table size increases the number of servers and thus reduces the burden on each server. The plots show that the number of levels also have an important impact on the name table sizes for the same reason.

Fig. 16 shows the average distance d of a name resolution with respect to the name table size n. Equations (1) and (3) give us $d \propto \frac{\log \frac{n}{m}}{\log h}$. However, we can see that the plots do not properly fit the data (0.88 correlation coefficient only). Our multiple approximations may be the cause of this phenomenon. Nevertheless, this plot shows that we can obtain a good tradeoff between the cost of name resolutions and the cost of storage in each name server by choosing values in the bottom-left area.

Fig. 17 shows the name table size n (expressed as a % of the number of names) with respect to the number of name servers (expressed as a % of the number of overlay members). We can see that n is inversely proportional to m. This is expected by the theoretical equations. Equations (2) and (3) give $s \propto \frac{\frac{mh}{n} - 1}{h-1}$ which gives $s \propto \frac{mh}{n(h-1)}$ if $\frac{mh}{n} \gg 1$, which for m and h fixed gives $n \propto s^{-1}$. The plot shows that we can achieve a good tradeoff between the % of name servers required and the size of the name tables to be stored in each of them by choosing values in the bottom-left area.

We evaluate now the resilience performances of our naming resolution system. We first fix l and h in order to reduce our parameter space. The results previously shown in Figs. 14 and 15

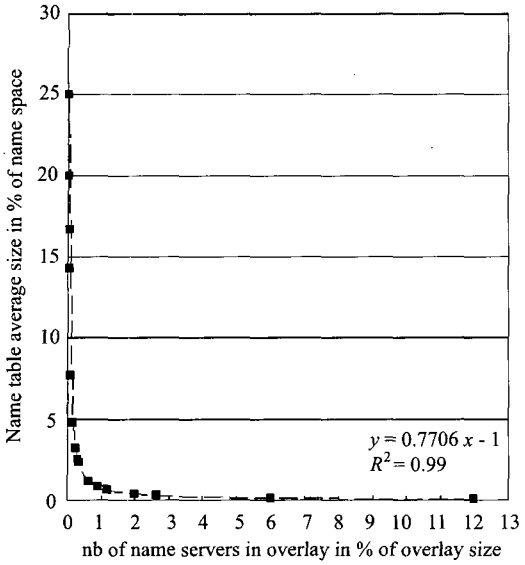

Fig. 17. Name table size vs. number of name servers.

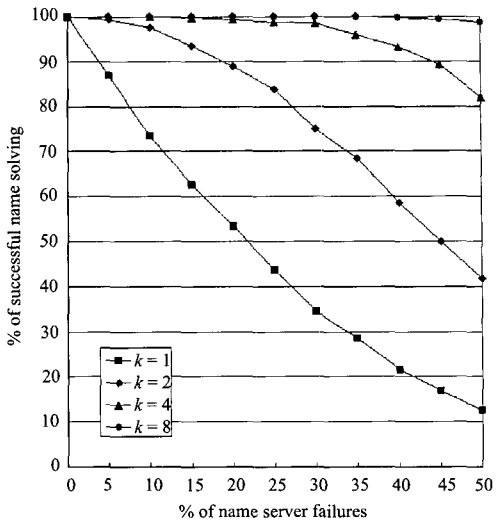

Fig. 18. Percentage of successful resolutions vs. name server failures.

encourage us to choose $l = 3$ hierarchy levels in order to keep the name resolution distance reasonable and a hash table size of $h = 6$ in order to limit the name table size. These parameters yield a number of name servers of 43, each having a name table containing around 300 names. Remember that the total overlay size is equal to 12977 nodes. Thus, the name servers represent 0.33% of the overlay nodes and they hold each 2.3% of the naming space. If we look at the plot of Fig. 17, we can see that these percentage values define a point located in the lower left area of the figure where the trade-off between the name table size and the amount of name servers is optimal. We have evaluated redundancy values k ranging from 1 (no replica) to 8 (7 replicas per name server). Thus, the number of servers ranges from 43 to 344. This remains appropriate given the total number of overlay nodes. Although the replicas increase the overall number of servers, the name table size of the servers remains the same (i.e., around 300).

Fig. 18 shows the percentage of successful naming resolution as a function of the name servers' dynamics percentage. As explained above, a given percentage of name servers are down. We can see that with no replica, 20% of dynamics makes the name resolution success rate fall under 50%. However, the addition of 3 replicas to each of the name servers heavily increases the naming resolution success. With up to 3 replicas per server and 50% of dynamics, the success rate still remains above 80%. Increasing the number of replicas above 3 per server does not linearly improve the success as it gets closer of 100%.

Fig. 19 shows the average number of times that a request between a client and a server or between two servers has failed because of a fallen name server. The maximum value is $l(k-1)$ for $k > 1$ and 1 for $k = 1$. As we have 3 levels in the hierarchy, the maximum value is 3 for $k = 2$, 9 for $k = 3$, and 21 for $k = 8$. For any value of k, the plot shows that this value increases when the % of failures increases. For no replicas, the plot tends quickly towards 1 but for $k > 1$, we can see that the system is far from using all the possible attempts when the % of failures is equal or less than 50%. For $k = 8$ and 50% of

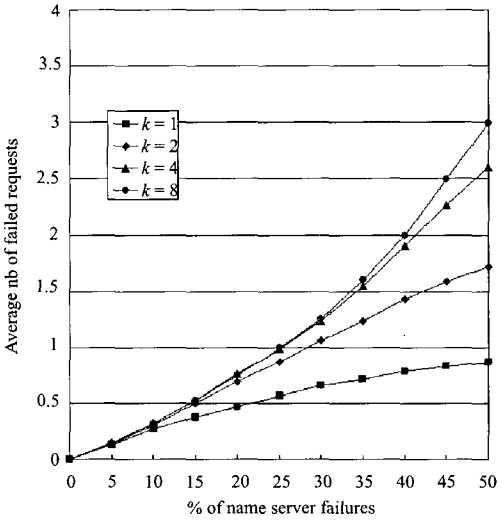

Fig. 19. Average number of failed requests vs. name server failures.

failures, this value is just above 3 while it could reach 21.

Fig. 20 shows the average distance covered by a name resolution in hops including the reply and expressed as a ratio of the average round trip distance measured in the overlay. We have made this measurement as a function of the levels in Fig. 14 (with no replicas) and have seen that for 3 levels in the hierarchy this value is roughly equal to 150%. For 0% of failures, we observe the same value here (taking into account the relative statistical error of 5%). For $k < 3$, this value is decreasing nearly linearly as the amount of failed requests (and thus the name resolutions) increases. For $k > 2$, this value is increasing a little bit when the % of failures increases. This shows that most of the requests are successful (thus making a round trip) even though the increasing number of resubmitted request (to avoid fallen servers) makes the overall paths a bit longer.

Fig. 21 shows the average total number of packets emitted per

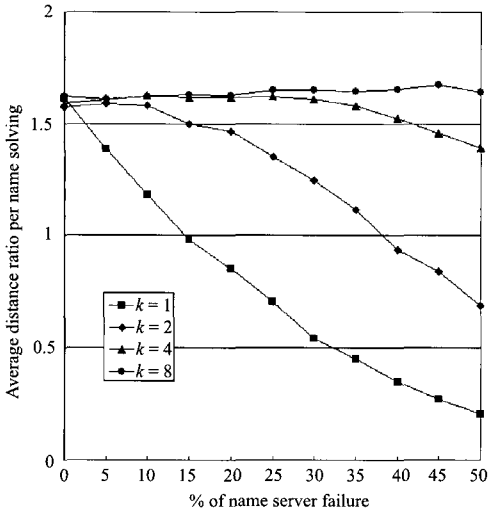

Fig. 20. Average name resolution distance vs. name server failures.

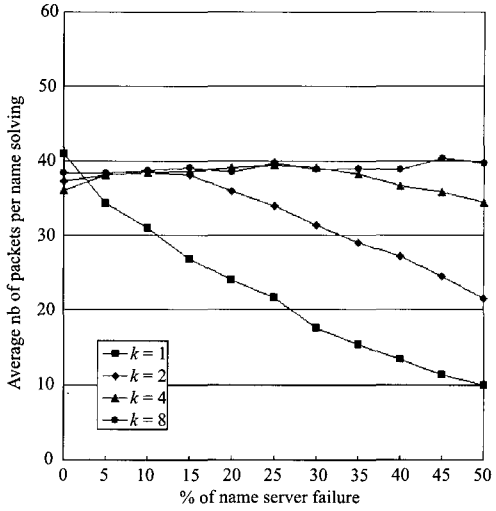

Fig. 21. Average amount of packets sent vs. name server failures.

name resolution including the reply. Any crossed link counts for one packet. The trend of these plots closely and naturally maps the trend of the plots of Fig. 20 albeit with different values. The total number of packets decreases when k is low and requests fail to complete while it increases when k is high and a large number of alternate request are possible via the replicas.

VI. CONCLUSION

In this paper, we have proposed a scalable, resilient, and autonomous overlay infrastructure based on a distributed hierarchical addressing, routing and naming framework. The corresponding middleware is designed for applications creating and using overlays set up on top of the Internet. We have defined a local knowledge routing scheme based on an efficient localization driven addressing and the simulation results obtained upon two realistic Internet maps have shown that our solution yields a reasonable path inflation ranging between 10% and 130% depending on the network dynamics. We have described how to cope with network dynamics and simulations have shown that our multiple address allocation scheme multiplies by 2 the routing success percentage when the network dynamics are equal or above 10%. We have also shown that routing is nearly guaranteed given enough convergence time (5 or more time periods). Finally, we have designed an autonomous scalable distributed k-resilient name to address binding system for efficiently separating the naming and addressing planes thus allowing applications to seamlessly use advanced network services such as mobility and multicast. Simulation results have shown that the tradeoff between resolution costs and name table sizes can be optimized. We also have shown that by using a reasonable amount of name server replicas (3 or more), we can cope with the failure of up to 50% of the name servers while maintaining a name solving success ratio above 80%. Our simulations were done on overlays having up to 12,000 nodes which makes us believe that our infrastructure is scalable to very large overlays.

We are currently implementing our infrastructure as a host level network middleware. A basic prototype written in C by graduate students is available at [26]. This prototype uses the *sockets* API for portability and currently runs over LINUX. Applications using our middleware will be able to set up self-organizing efficient and scalable overlays. They will provide an autonomic support for addressing and naming management thus freeing the applications of many network level limitations. Our future work is targeted at improving our address allocation scheme, evaluating the overlay data forwarding performances (e.g., TCP pipelining), implementing data forwarding reliability when needed (i.e., over 802.2 and UDP) and testing our middleware in real situation.

REFERENCES

[1] F. Teraoka, Y. Yokote, and M. Tokoro, "A network architecture providing host migration transparency," in *Proc. ACM SIGCOMM'91*, 1991.

[2] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "The design and implementation of an intentional naming system," in *Proc. 17th ACM SOSP'99*, 1999.

[3] D. Magoni, "A scalable and unifying architecture for deploying advanced protocols in the internet," in *Proc. 10th Int. Conf. Telecommun. 2003*, Papeete, Tahiti, French Polynesia, Feb. 2003, pp. 1001–1007.

[4] P. Francis and R. Gummadi, "Ipnl: A nat-extended Internet architecture," in *Proc. ACM SIGCOMM 2001*, 2001.

[5] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," in *Proc. ACM SIGCOMM 2002*, 2002.

[6] A. Snoeren and H. Balakrishnan, "An end-to-end approach to host mobility," in *Proc. 6th ACM MobiCom 2000*, 2000.

[7] A. Viana, M. D. de Amorim, S. Fdida, and J. F. Rezende, "Indirect routing using distributed location information," in *Proc. IEEE PerCom 2003*, Mar. 2003, pp. 224–234.

[8] Y. H. Chu, S. Rao, S. Seshan, and H. Zhang, "Enabling conferencing applications on the internet using an overlay multicast architecture," in *Proc. ACM SIGCOMM 2001*, Aug. 2001.

[9] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller, "Construction of an efficient overlay multicast infrastructure for real-time applications," in *Proc. IEEE INFOCOM 2003*, 2003.

[10] G.-I. Kwon and J. Byers, "Roma: Reliable overlay multicast with loosely coupled TCP connections" in *Proc. IEEE INFOCOM 2004*, Mar. 2004.

[11] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proc. 18th ACM SOSP*, Oct. 2001.

[12] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," in *Proc. ACM SIGCOMM 2001*, Aug. 2001, pp. 149–160.

[13] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach, "Security for structured peer-to-peer overlay networks," in *Proc. 5th OSDI 2002*, Dec. 2002.
[14] T. Eilam, C. Gavoille, and D. Peleg, "Compact routing schemes with low stretch factor," in *Proc. 17th ACM Symp. Principles of Distributed Computing*, Aug. 1998, pp. 11–20.
[15] L. Cowen, "Compact routing with minimum stretch," in *Proc. 10th ACM-SIAM Symp. Discrete Algorithms*, Jan. 1999.
[16] D. Krioukov, K. Fall, and X. Yang, "Compact routing on internet-like graphs," in *Proc. IEEE INFOCOM 2004*, Mar. 2004.
[17] Z. Li and P. Mohapatra, "Impact of topology on overlay routing service," in *Proc. IEEE INFOCOM 2004*, Mar. 2004.
[18] P. Mockapetris, "Domain names—implementation and specification," *RFC*, IEFT, Nov. 1987.
[19] S. Kent and R. Atkinson, "Security architecture for the internet protocol," *RFC 2401*, IETF, Nov. 1998.
[20] S. Kent and R. Atkinson, "IP encapsulating security payload (esp)," *RFC 2406*, IETF, Nov. 1998.
[21] M. Hoerdt and D. Magoni, "*Network cartographer (nec)*," Université Louis Pasteur, available at https://dpt-info.u-strasbg.fr/~magoni/nec/.
[22] M. Hoerdt and D. Magoni, "Completeness of the Internet core topology collected by a fast mapping software," in *Proc. 11th Int. Conf. Software, Telecommun. and Computer Networks*, Split, Croatia, Oct. 2003, pp. 257–261.
[23] A. M. Law and W. D. Kelton, *Simulation Modelling and Analysis*, 3rd ed., McGraw-Hill, 2000.
[24] D. Magoni, "Network manipulator (nem)," Université Louis Pasteur, available at https://dpt-info.u-strasbg.fr/~magoni/nem/.
[25] D. Magoni, "Hierarchical addressing and routing mechanisms for distributed applications over heterogeneous networks." in *Proc. 3rd Int. Conf. Computational Science—Workshop on Innovative Solutions for Grid Computing*, Melbourne, Australia, June 2003, pp. 1093–1102.
[26] D. Magoni, "Dharma," Université Louis Pasteur, available at https://dpt-info.u-strasbg.fr/~magoni/dharma/.

Khaldoon Shami received a M.Sc. degree in computer science from the University of Reims, France in 2004. He is currently a Ph.D. student at the University of Haute Alsace under the supervision of Pascal Lorenz and co-supervision of Damien Magoni. His major interests are Internet protocols and overlay networks.

Damien Magoni received a Ph.D. degree in computer science from the University Louis Pasteur of Strasbourg, France in 2002. He is currently an assistant professor at the University Louis Pasteur and is a member of the LSIIT research laboratory. He is also an IEEE member and serves as a reviewer for numerous conferences and journals. He has co-authored more than 30 research papers. His research interests focus on Internet topology, architecture, and protocols.

Pascal Lorenz received a Ph.D. degree from the University of Nancy, France. Between 1990 and 1995, he was a research engineer at WorldFIP Europe and at Alcatel-Alsthom. He is a professor at the University of Haute-Alsace and responsible of the Network and Telecommunication Research Group. His research interests include QoS, wireless networks, and high-speed networks. He was the program and organizing chair of the IEEE ICATM'98, ICATM'99, ECUMN 2000, ICN 2001, ECUMN 2002, ICT 2003, and ICN 2004 conferences and co-program chair of ICC 2004. Since 2000, he is a technical editor of the IEEE Communications Magazine Editorial Board. He is the secretary of the IEEE ComSoc Communications Systems Integration and Modelling Technical Committee. He is senior member of the IEEE, member of many international program committees and he has served as a guest editor for a number of journals including Telecommunications Systems, IEEE Communications Magazine and LNCS. He has organized and chaired several technical sessions and gave tutorials at major international conferences. He is the author of 3 books and 135 international publications in journals and conferences.