Application Layer Addressing, Routing and Naming Framework for Overlays

Damien Magoni, Pascal Lorenz

Application Layer Addressing, Routing and Naming Framework for Overlays

Damien Magoni
Université Louis Pasteur – LSIIT
Boulevard Sébastien Brant, 67400 Illkirch, France
magoni@dpt-info.u-strasbg.fr

Pascal Lorenz
Université de Haute Alsace – GRTC
34 rue du Grillenbreit, 68008 Colmar, France
lorenz@ieee.org

***Abstract*— A growing number of applications create 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. However overlays require some form of internal addressing, routing and naming. Therefore their topologies are usually kept simple but this limits their flexibility and scalability. Our aim is to design an efficient and robust addressing, routing and naming framework for complex overlays. Our only assumption is that they are constrained by the Internet topology. Applications using our framework will be relieved from managing their own overlay topologies. This paper presents our framework in detail as well as some performance results concerning its routing efficiency, its reliability to network dynamics and its naming scalability.**

I. Introduction

Designing an application level addressing, routing and naming framework 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 service for creating these overlays especially the bigger ones. For instance, setting up a tree topology is difficult and 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 to it and redundant links provide increased robustness. On the other hand, the overlay requires a proper routing system that we aim at providing. Furthermore our framework provides a separation between node addressing and naming. Thus our overlay enable applications to run seamlessly over private and public addressing spaces, to mix node mobility with secured connections, etc. Our paper contains three sections. In section II we briefly present prior and related work on overlay networks. In section III we describe how our addressing, routing and naming framework is designed. Finally in section IV we present some performance results of our framework obtained by simulations and concerning routing efficiency, reliability to network dynamics and naming scalability.

II. Related work

An important point that many people agree on is that naming and addressing should be separated [1], [2], [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-to-peer objet lookup (e.g., Chord [12], Pastry [13], etc) thus opening new opportunities for network applications. On a more theoretical level, Cowen has proved in [14] 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 [15]. However both do not describe how to implement it as a distributed algorithm. Concerning overlay topologies, Li *et al.* have shown in [16] that they do have an impact on the routing performances thus further motivating our work. In this paper we present a framework where table sizes are not a function of the network size but a function of the node degrees. Although our architecture does not provide an upper bound on the average stretch, it is typically below 2.3 as shown in section IV. This paper contains many issues that were not studied in our first work [17]. These new topics concern distributed addressing, resistance to network dynamics and naming management.

III. Framework description

As we have designed an architecture that puts no constraint on the topologies of the overlays, we have to define a routing mechanism in order to route data packets inside the overlay. Our routing mechanism is partly address-driven (i.e., some path information is stored in an address), that is why we define here the addressing first.

A. 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 hierarchy. Each field of

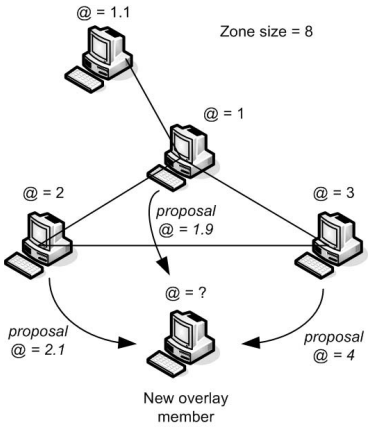

Fig. 1. Joining overlay node asking for addresses

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 a given time. It is worth noticing that the size 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 plan 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 primarily 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 others by the use of transport protocols (e.g., TCP, SPX, 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 transport addresses are not required among all overlay nodes. In order to connect to an overlay, new or moving nodes must know the transport address (usually IP addresses) of at least one overlay node. This address should be provided by out-of-band methods (e.g., E-mail, WWW, SDP, etc) and provided to the application that is using this overlay. The overlay nodes directly connected by 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 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", the shortest "down" address,
- if no "down", the shortest "leaf" address.

Figure 1 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.

B. Routing

The core principle of our architecture 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, its length depends on the efficiency of the address allocation.

Hierarchical routing works in the following way. 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

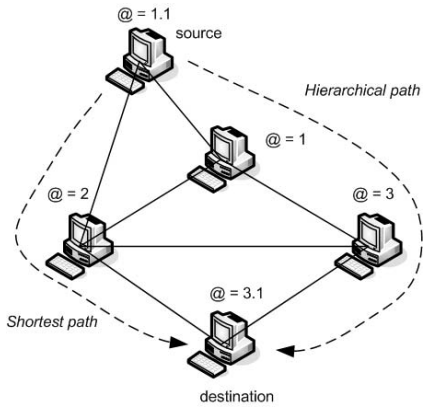

Fig. 2. Path inflation caused by hierarchical routing

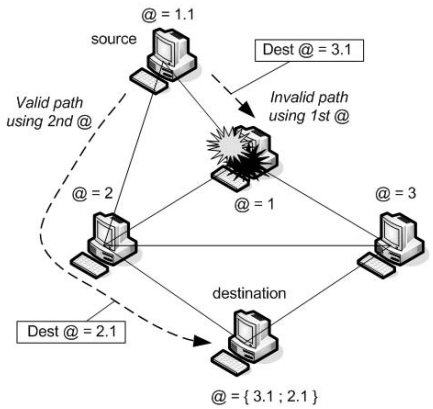

Fig. 3. Alternate routing for solving network dynamics

a continuous label assignment, this technique ensures that the packet will reach its destination although not necessarily by a shortest path.

Figure 2 illustrates the effects of hierarchical routing and shortest path routing between the nodes 1.1 and 3.1 on path lengthes. 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 a destination address that contains the invalid address will not be able to be routed anymore. To solve this issue, 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 figure 3. 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.

All addresses are given a time-out value v_1 and have to be refreshed by hello messages sent between neighbors. 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 time-out value v_2 ($v_2 > K_{depth} \times v_1$), 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 addresses fail during the 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).

C. Naming

We have seen above that every node in the overlay has one or several addresses. 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 framework 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 figure 4. A name is composed of several parts. Each part corresponds to one level of the name server hierarchy. It is not a problem if the number of levels is different from the number of parts.

In the following, we assume that there are no network 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

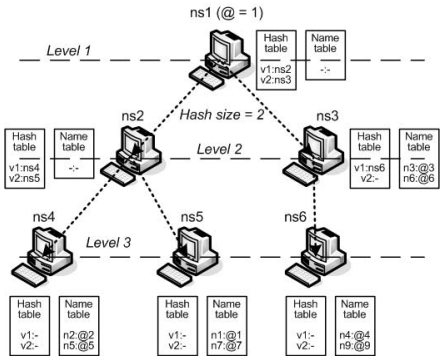

Fig. 4. Hierarchical organization of name servers

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(es). 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 network dynamics and attacks, we use a replication approach. We assume that a redundancy factor k is chosen at the start of the overlay construction and that the total number of servers equal s. 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.

There are at least two major differences between our naming distribution strategy and the famous DNS 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 framework 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.

IV. Experiments

A. Settings

In order to evaluate our addressing, routing and naming design, we have used one 12k-node IPv4 map made in July 2004 and one 4k-node IPv6 map made in June 2003, both collected by using our IP topology mapping *nec* software [18]. 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 [19]. We assume on first approximation that overlays deployed over the Internet can be represented by subgraphs of these maps.

In our experiments, nodes and links are gradually added to the overlay from one given map with some nodes and links of the map being discarded because of the addressing plan construction and the network dynamics. For the overlay construction, the first node is a random node having an above average potential degree ($>$10). For network dynamics, we have analyzed periodical percentage of random node removal ranging from 0 to 50% of the overlay size and allocating 1 to 4 (at most) addresses to each node. For name servers, we have selected random nodes having an above average potential degree ($>$5). 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 $100 - x\%$.

As the process of generating addressing plans, selecting name servers and selecting source-destination nodes involves random selection (and thus random rolls), we have used a sequential scenario of simulation [20] 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

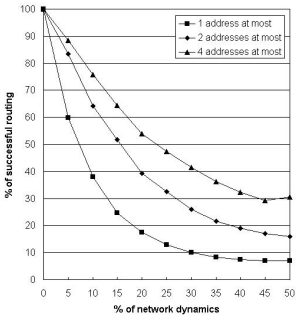

Fig. 5. Percentage of success *vs* network dynamics

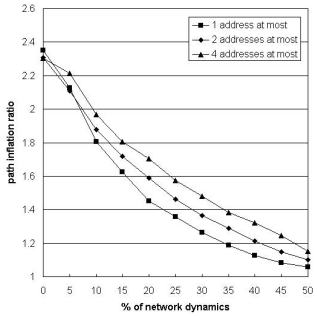

Fig. 6. Path inflation *vs* network dynamics

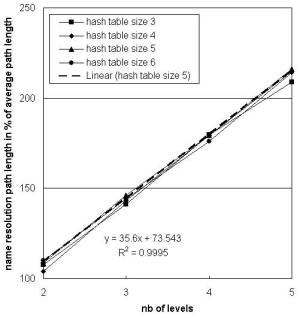

Fig. 7. Name resolving path length ratio *vs* nb of levels

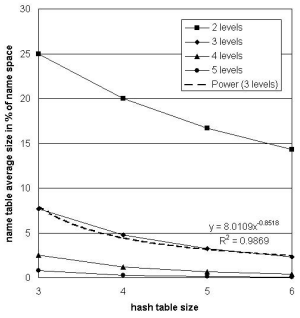

Fig. 8. Name table size *vs* hash table size

each run (one run is the time horizon) consists in picking two nodes and determining:

- the flat and hierarchical distances between them
- the success of the hierarchical routing in presence of network dynamics
- the distance of the destination name resolution including request and answer

In order to reduce calculation costs, the choice of the first node and subsequent addressing plan creation and the placement of the name servers are done every 100 runs, the network dynamics described above happen every 10 runs and the sequential checkpoints are carried out every 5 runs. All the simulation results have been obtained assuming a confidence level of 0.95 with a relative statistical error threshold of 5% for all measured metrics. Simulations have been carried out in our static simulator *nem* software [21].

B. Results

In all our simulations, the results obtained with the IPv6 map were very close to the ones 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 IPv4 map results unless specified otherwise. We evaluate now the performances of our addressing and routing framework especially in the presence of network dynamics. Figure 5 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). Figure 6 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 first work [17]. 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 evaluate now the performances of our naming resolution framework. Figure 7 shows the average 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. Figure 8 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

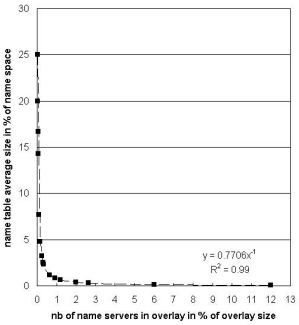

Fig. 9. Name table size *vs* nb of name servers

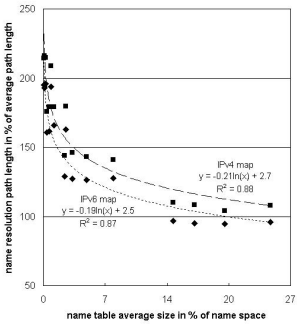

Fig. 10. Name resolving path length ratio *vs* name table size

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. Figure 9 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. Figure 10 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.

V. Conclusion

In this paper, we have proposed a distributed hierarchical addressing, routing and naming framework designed for creating and managing overlays set up on top of the Internet. We have defined a routing scheme based on efficient addressing and simulation results obtained upon two realistic Internet maps have shown that our solution yields a reasonable routing overhead ranging between 10% to 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 designed a scalable name to address resolution scheme and simulations have shown that the tradeoff between resolution costs and name table sizes can be optimized. We are finishing the implementation of our framework as a host level network middleware. It is written in C, 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 that will provide 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 our name server replication scheme, evaluating the transport level performance issues, evaluating how the performances of this new overlay scheme compare with other existing ones and testing our middleware in real situation.

References

[1] F.Teraoka, Y. Yokote, and M. Tokoro, "A network architecture providing host migration transparency," in *Proceedings of 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 *Proceedings of 17th ACM SOSP*, 1999.

[3] D. Magoni, "A scalable and unifying architecture for deploying advanced protocols in the internet," in *Proceedings of the 10th International Conference on Telecommunications*, Papeete, Tahiti, French Polynesia, February 2003, pp. 1001–1007.

[4] P. Francis and R. Gummadi, "Ipnl: A nat-extended internet architecture," in *Proceedings of ACM SIGCOMM'01*, 2001.

[5] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," in *Proceedings of ACM SIGCOMM'02*, 2002.

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

[7] A. Viana, M. D. de amorim, S. Fdida, and J. F. Rezende, "Indirect routing using distributed location information," in *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom)*, March 2003, pp. 224–234.

[8] Y. hua Chu, S. Rao, S. Seshan, and H. Zhang, "Enabling conferencing applications on the internet using an overlay multicast architecture," in *Proceedings of ACM SIGCOMM'01*, August 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 *Proceedings of IEEE INFOCOM'03*, 2003.

[10] G.-I. Kwon and J. Byers, "Roma: Reliable overlay multicast with loosely coupled tcp connections," in *Proceedings of IEEE INFOCOM'04*, March 2004.

[11] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proceedings of the 18th ACM SOSP*, October 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 *Proceedings of ACM SIGCOMM 2001*, August 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 *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, December 2002.

[14] L. Cowen, "Compact routing with minimum stretch," in *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*, January 1999.

[15] D. Krioukov, K. Fall, and X. Yang, "Compact routing on internet-like graphs," in *Proceedings of IEEE INFOCOM'04*, March 2004.

[16] Z. Li and P. Mohapatra, "Impact of topology on overlay routing service," in *Proceedings of IEEE INFOCOM'04*, March 2004.

[17] D. Magoni, "Hierarchical addressing and routing mechanisms for distributed applications over heterogeneous networks," in *Proceedings of the 3rd International Conference on Computational Science*, Melbourne, Australia, June 2003, pp. 1093–1102.

[18] M. Hoerdt and D. Magoni, *network cartographer (nec)*, Université Louis Pasteur, https://dpt-info.u-strasbg.fr/~magoni/nec/.

[19] ——, "Completeness of the internet core topology collected by a fast mapping software," in *Proceedings of the 11th International Conference on Software, Telecommunications and Computer Networks*, Split, Croatia, October 2003, pp. 257–261.

[20] A. Law and W. Kelton, *Simulation Modelling and Analysis*, 3rd ed. McGraw-Hill, 2000.

[21] D. Magoni, *network manipulator (nem)*, Université Louis Pasteur, https://dpt-info.u-strasbg.fr/~magoni/nem/.