Hierarchical Addressing and Routing Mechanisms for Distributed Applications over Heterogeneous Networks

Damien Magoni

Université Louis Pasteur – LSIIT
magoni@dpt-info.u-strasbg.fr

Abstract. Although distributed applications such as grids and peer-to-peer technologies are quickly evolving, their deployment across heterogeneous networks such as non-globally addressable networks has not received much attention. The lack of IP addressing space for instance has created an explosion of network address translators and thus a large amount of non-globally addressable hosts. Whereas these boxes do not disturb much the behavior of a client-server communication, this is no longer true in a distributed application built on the peer-to-peer model where any entity can play both roles and thus needs a globally known address. Application level gateways are proposed to solve this problem but there are usually specific to a given distributed application. In this paper we propose an application level addressing scheme and a routing mechanism in order to overcome the non-global addressing limitation of heterogeneous networks such as those brought by network address translation technologies. Our architecture is designed to bring increased scalability and flexibility for communications between members of a distributed application. We give simulation results concerning the efficiency of the addressing and routing mechanisms.

1 Introduction

In this paper we present the basic mechanisms of an architecture designed to overcome the limitations caused by non-globally addressable members of a distributed application such as a grid or a P2P application. The idea is to create an application level overlay to manage addresses specific to our architecture. Network level connections are managed between members and thus they can belong to different addressing spaces. Our architecture must provide some routing but in a much simpler way than what can be found in routers for instance. We are currently investigating the efficiency of our architecture by simulation and thus we have not yet started its implementation. The rest of this paper is organized as follows. In section 2 we give some information on our architecture. Section 3 contains the algorithms for providing the hierarchical addressing and routing architecture to be used in the overlay. Finally, in section 4 we give simulation results obtained by simulation on the efficiency of our hierarchical addressing and routing scheme.

2 Architecture

Our architecture consists in creating a network overlay that has his own addressing scheme. Each host willing to use our architecture, will run our network middleware (to be implemented soon). Any distributed application on the host can use our middleware and there will be (at least) one overlay per distributed application instance. A new member, for a given distributed application, will connect to one or more members by using an end-to-end network connection such as a TCP connection and then it will be addressable by any host in the same overlay. Thus an overlay member being a NAT box [1] will be able to connect to members located in the public Internet while also connecting members in a private non-globally addressable network and it will forward messages by using our application level routing. Furthermore every member whatever its type of network address (e.g. public IP, private IP, IPX, etc.) will be globally addressable in the overlay. This can be very useful for many distributed applications. Notice that NAT boxes as-is do not provide this important property (i.e. a NAT box just lures both hosts by making them believe that itself is the end-point of both connections). Thus high-level protocols such as TCP/IP only serve for connecting neighbor members of an overlay (thus behaving like data-link protocols). We have chosen a hierarchical addressing scheme in to order to make the architecture scalable to a large number of members (i.e. overlay size). We also have chosen to allow the overlay to have a graph structure instead of a tree structure (typically found in overlays) in order to introduce some robustness through the use of alternate paths and better path lengths. This, in turn, increases the complexity of the overlay routing mechanism. Please note that the mechanisms needed to pipe connections in members acting like routers in the overlay is not described here due to space limitations.

3 Hierarchical Addressing and Routing

As hosts can use different protocols to communicate, the address attribution process to the hosts must be protocol independent and uniform. In order to make the addressing as well as the routing scalable, we define a hierarchical addressing. Figure 1 illustrates a network of hosts with their corresponding addresses. An address is composed of one or several fields separated by dots. 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 latest field.

3.1 Hierarchical Addressing

We have designed a centralized algorithm (for the moment) to address all the hosts of a distributed application overlay in order to build a hierarchical structure. This algorithm is given in the **Create-Hierarchy** procedure below. This algorithm uses one parameter called **zone-size**. This parameter controls the number of hosts in the top level zone (i.e. hosts having a level 1 address) as well as the

Fig. 1. Hierarchical addressing and routing example.

number of hosts in each sub level zone. First the biggest host with respect to the degree (i.e. number of communication routes) is addressed as host 1. All his neighbors are added to a **degree-pick-list**. Then, the highest degree host of the list is picked and given the same level next address (e.g. 2 for the second one). The neighbors of the next chosen host are added to the list (if they are not in it already). The process is repeated until a top level zone having **zone-size** hosts is obtained. Top level hosts form a connected graph. Then each top level host addresses its **zone-size** highest degree unaddressed neighbors with level 2 addresses where the prefix is its own address. This process is done recursively until no more host can be addressed. So when all level 1 hosts have addressed their **zone-size** highest degree unaddressed neighbors, level 2 hosts do the same, etc. Of course in an implementation, this algorithm will have to be distributed. The top level address attribution protocol may be a bit tricky, but the protocol for sub levels address attribution should be fairly easy to implement. After this phase, some hosts may still not be addressed especially if the **zone-size** parameter is small. We call them leftovers. Each unaddressed remaining host picks its first found addressed neighbor and take the next level next address (e.g. if the neighbor has address 4.5 and has addressed 7 neighbors, host will take address 4.5.8). This means that the number of hosts per zone prefix is not strictly limited to **zone-size** hosts. The **addr-host-list** is a list of the currently addressed hosts.

```
Create-Hierarchy(zone-size)
1. Empty addr-host-list
2. Empty degree-pick-list
3. Give the level 1 address 1 to the host with the highest degree
4. Add its neighbors to the degree-pick-list
   // select top level hosts
5. while zone-no <= zone-size
6.   Remove of the degree-pick-list the host with the highest degree
7.   Give the level 1 next address to this host
8.   Insert this host in addr-host-list
9.   Add its neighbors to the degree-pick-list if not already in it
   // select sub level hosts
10. level-no = 1
11. while addr-host-list < number of hosts
12.   if previous loop addressed no host
13.     break
```

```

14.   foreach host
15.       if host address level = level-no
16.           Address-Neighbors(host)
17.   level-no++
18.   foreach host
19.       if host address level = level-no
20.           Insert host in addr-host-list
21. // add non addressed hosts
22. foreach host
23.     if host has no address
24.         foreach neighbor of host
25.             if neighbor has an address
26.                 Give next level next address to host
27.                 Insert host in addr-host-list
28.                 break

```

The `Address-Neighbors(host)` function is called by our addressing algorithm. The host given in parameter will address its **zone-size** highest degree unaddressed neighbors with the next level next address. For instance, if host has address 4.5, the first highest degree unaddressed host will be given address 4.5.1, the second 4.5.2, and so on until the last 4.5.**zone-size**.

```

Address-Neighbors(host)
1.   Empty degree-pick-list
2.   foreach neighbor of host
3.       if neighbor does not have an address
4.           if degree-pick-list size < zone-size
5.               Add neighbor to the degree-pick-list
6.   foreach neighbor of host
7.       Remove the highest degree neighbor from the degree-pick-list
8.       Give to neighbor the next level next address

```

3.2 Hierarchical Routing

We propose an algorithm for routing inside our hierarchical addressing structure. The algorithm is distributed (i.e. each host will run this algorithm in its routing module). The `Hierarchical-Routing` function returns true if the next-hop host given in argument does lead to the destination host by a shortest path or false otherwise. The algorithm uses the address of the host running it (**host-addr**), the address of the destination host (**dest-addr**, found in the messages to route) and the address of the next-hop host (**next-hop-host**). Level 1 routing is different because hosts do not have a tree-like relation. It is explained later in this section. In the code, address A is included (strictly) in address B means that A is at most a prefix of B. For instance addresses 1, 1.3 and 1.3.5.2 are all included in address 1.3.5.2.1. Figure 1 illustrates the effects of hierarchical routing and flat routing between the hosts 3.2 and 2.2.1.1 on path length. The hierarchical routing forces the message to be routed via 2 thus giving a path length of 5 hops while a flat shortest path routing requires only 4 hops to reach the destination.

```

Hierarchical-Routing(next-hop-host, dest-addr) returns bool
1.   if host-addr level = dest-addr level
2.       if host-addr prefix = dest-addr prefix
3.           if host-addr label = dest-addr label
4.               // we are at dest!
5.           else
6.               Up-Routing(next-hop-addr, dest-addr)

```

```

7.      else
8.          return (next-hop-addr is included in host-addr)
9.  else
10.     if host-addr is included in dest-addr
11.         if next-hop-addr = dest-addr
12.             return true
13.         else if next-hop-addr is included in dest-addr
14.             and host-addr is included in next-hop-addr
15.                 return true
16.         else
17.             return false
18.     else
19.         Up-Routing(next-hop-addr, dest-addr)

```

The `Level-Routing(next-hop-addr, dest-addr)` function is called by our routing algorithm when the message to be routed is in a level 1 host. The host needs to know the distance between its neighbors and all the level 1 hosts. In our simulation we run a centralized Floyd-Warshall algorithm on the top level zone to obtain this topology knowledge. In an implementation we will propose a path-vector distributed routing protocol (i.e. a very lightweight BGP-like protocol). The number of hosts involved in level 1 routing will be, by design, limited to **zone-size** thus making the job of the protocol easier. Experiments presented in the next section tend to show that this architecture can scale. In the code, **level-1-addr** of the destination address is the value of the first field of the destination address. The `Up-Routing(next-hop-addr, dest-addr)` function is called by our routing algorithm when the message to be routed must go towards lower level hosts (i.e. hosts located higher in the hierarchy).

`Level-Routing(next-hop-addr, dest-addr)` returns bool

```

1.  if next-hop-addr level = 1
2.      d1 = hops from next-hop-addr host to
3.          level-1-addr host of dest-addr host
4.      foreach neighbor of host
5.          if neighbor-addr level = 1
6.              d2 = hops from neighbor-addr host
7.                  to level-1-addr host of dest-addr host
8.              if d2 < d1
9.                  return false
10. return true
11. else
12. return false

```

`Up-Routing(next-hop-addr, dest-addr)` returns bool

```

1.  if host-addr level = 1
2.      return Level-Routing(next-hop-addr, dest-addr)
3.  else
4.      return (next-hop-addr is included in host-addr)

```

4 Experiments

In this section we present preliminary results obtained by simulation for evaluating the feasibility and efficiency of our hierarchical addressing and routing architecture. We show that the overhead incurred on path length is quite low compared to the gain in routing scalability.

4.1 Simulation Settings

We have used power law graphs for modelling distributed application overlay inter-host connections in accordance with our reasonable assumption that hosts will connect freely and thus will most probably create this kind of topology. We have used a huge 280k-node router-level Internet map [2] collected by Govindan *et al.* and we have sampled it to produce peer-to-peer overlay maps of sizes ranging from 1000 to 5000 nodes which can be considered as large scale distributed application overlays. We have used the *nem* software [3,4] to produce these overlays. We have constructed various addressing plans on them by using zone sizes ranging from 8 to 512. Table 1 shows the parameter space of our simulations.

Table 1. Simulation Parameters

Parameter	Values
Number of members / Overlay size	1000, 2k, 3k, 4k, 5k
Zone size	8, 16, 32, 64, 128, 256, 512

Then we have randomly picked up source and destination host pairs in the distributed application overlays and have computed the shortest path length and the hierarchical path length between the source and destination. They can differ as shown for instance in figure 1. In our simulations, we have measured all the metrics shown in table 2. The number of levels and leftovers are measured after each addressing plan creation and the path length and ratio are measured after each source and destination host pair selection.

Table 2. Measured Metrics

Metric
Flat path length (in hops)
Hierarchical path length (in hops)
Ratio hierarchical/flat path length
Number of levels in hierarchy
Number of leftovers

As the process of generating distributed application overlays and selecting hosts involves random selection (and thus random rolls), we have used a sequential scenario of simulation [5] to produce the results shown in the next section. We have used the Mersenne Twister code [6] for producing the random numbers needed in our simulation. 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 is made of picking a source and a destination and determining the number of hops between them using flat (shortest path) and hierarchical routing (i.e. one run is the time horizon). Ideally we should have created a distributed application overlay, planned the addressing, selected two hosts in this overlay, measured the metrics and carried out a sequential checkpoint. However this would have been too costly in terms of computing power. Thus we have performed a sequential checkpoint each time after having created a distributed application overlay, planned the addressing and selected and measured routing distances for 100 pairs of hosts in this overlay. All the simulation results have been obtained assuming a confidence level of 0.95 with a relative statistical error threshold of 1% for the path length distances and ratio and 5% for the number of levels and leftovers.

4.2 Results

Figure 2 shows the ratio between the path length provided by the hierarchical routing and the shortest path length. We call this ratio the *routing overhead*. The routing overhead is shown as a function of the zone size for several overlay sizes. Very surprisingly, the overhead is quite low for this kind of overlay routing, especially for zone sizes above 64. A 4000-node overlay yields an overhead of 32% when the zone size is equal to 128. Furthermore the routing overhead becomes really low (around 10%) when the zone size is equal to 512. We believe that the routing power necessary to manage a 512-node top level zone in a host should not be overwhelming especially when compared to the 120000+ network prefix entries that a BGP core router has to handle! The results do however slightly depend on the distributed application overlay size which can tend to question the scalability. We discuss this issue in the next paragraph. Finally the wavy aspect of some parts of the plots are probably an artefact of our simulation. Our technique consisting in running a number (i.e. 100) of runs per overlay and then performing a sequential checkpoint may slightly affect the statistical results. If we had chosen other values for the number of runs we may have obtained smoother plots. Globally, the overhead cost of using the hierarchical routing is always below 50% and it can be kept around 10% for large zone sizes. As a comparison, in [7] Govindan *et al.* have shown that the routing overhead incurred by routing policies in the Internet is 25% for half of the IP level paths. They also have found that 20% of the paths are inflated by more than 50%! We have also shown in [8] that the routing overhead at the Autonomous System level of the Internet was 8.7% on the average in year 2000 and that 25% of the AS paths were inflated (i.e. longer than their corresponding shortest path). As any BGP router can be considered as knowing the whole AS topology, these results are not good and clearly demonstrate the impact of routing policy in the Internet. Concerning other kinds of routing, Waldvogel *et al.* have shown by simulation in [9] that their topology aware routing architecture for overlays yields a routing overhead of more than 200% for 40% of the paths when using 4 dimensions (however they use the delay as a metric instead of the hop count).

Fig. 2. Average path length inflation as a function of the zone size.

In figure 3, the overhead is shown as a function of the overlay size for several zone sizes. We can see that the routing overhead does increase with the overlay size, topping at 50% for a 5000-node overlay with a zone size of 8. However, the overhead is diminishing when it gets towards the right part of the plots which could show that the overhead increase is not linear. This is a positive indication of scalability. Furthermore for overlays with a zone size of 512, the overhead increases very slowly although in this case we do not know if it diminishes at some point. We are confident that large enough zone sizes can handle large scale overlays for a reasonable routing overhead cost. We plan to do more simulations on larger graphs to confirm this point.

Fig. 3. Average path length inflation as a function of the overlay size.

Figure 4 shows the number of levels used to hierarchically address all the hosts of a distributed application overlay. This number seems to be exponentially inversely proportional to the zone size. This property can be related to the property that the depth of a tree is proportional to the logarithm of its number of nodes. The number of levels seems to be loosely dependent (maybe by a log

function) on the overlay size. We can see that for large overlays, the number of levels needed may be quite high thus producing long addresses. A solution to reduce this is to build the sub-levels like the top level and not only by using the neighbors. This will reduce the number of levels and the routing overhead, however it will increase the complexity of the architecture. We plan to develop and analyze this scheme in a future work.

Fig. 4. Average number of levels as a function of the zone size.

The number of leftovers produced during the addressing phase is shown in figure 5 as a function of the zone size. It can be very high especially when the zone size is small. The very important thing to notice is that leftovers do require more addresses and thus can inflate the size of a given zone (i.e. defined as the number of hosts having the same prefix and address level) but they do not imply any penalty on the routing architecture as they can not be part of the top level hosts. Thus the routing information to collect and manage by the top level hosts is always bounded by a number of entries equal to zone size.

Fig. 5. Average number of leftovers as a function of the zone size.

5 Conclusions

Designing an application layer addressing and routing overlay network for distributed applications in order to overcome the addressing limitations found in heterogeneous networks is not an easy task. And making it scalable and flexible is even more challenging. In this paper we have proposed such an architecture that will be implemented as a communication middleware for distributed applications. We have shown by simulation that our architecture yields promising results concerning the efficient handling of the addressing across non-globally addressable networks. However there is still a lot of work left to build a prototype of our middleware. We still have to design the distributed version of our addressing mechanism, to create the level routing protocol needed inside the top zone and define algorithms for the dynamic addition and removal of hosts within our architecture. We hope that our proposal will bring to distributed applications more flexibility and more independence towards the underlying network addressing.

References

1. Srisuresh, P., Egevang, K.B.: Traditional ip network address translator. Request For Comments 3022, Internet Engineering Task Force (2001)
2. Govindan, R., Tangmunarunkit, H.: Heuristics for internet map discovery. In: Proceedings of IEEE INFOCOM'00, Tel Aviv, Israël (2000)
3. Magoni, D.: nem: a software for network topology analysis and modeling. In: Proceedings of the 10th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Fort Worth, Texas, USA (2002)
4. Magoni, D., Pansiot, J.J.: Internet topology modeler based on map sampling. In: Proceedings of the 7th IEEE Symposium on Computers and Communications, Giardini Naxos, Sicily, Italy (2002) 1021–1027
5. Law, A., Kelton, W.: Simulation Modelling and Analysis. 3rd edn. McGraw-Hill (2000)
6. Matsumoto, M., Nishimura, T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation* **8** (1998) 3–30
7. Tangmunarunkit, H., Govindan, R., Shenker, S., Estrin, D.: The impact of routing policy on internet paths. In: Proceedings of IEEE INFOCOM'01, Anchorage, Alaska, USA (2001)
8. Magoni, D., Pansiot, J.J.: Analysis of the autonomous system network topology. *Computer Communication Review* **31** (2001) 26–37
9. Waldvogel, M., Rinaldi, R.: Efficient topology-aware overlay network. In: Proceedings of ACM SIGCOMM HotNets'02. (2002)