RAN Naming and Discovery

Balasubramaneyam Maniymaran

Department of Electrical and Computer Engineering,
McGill University,
Montreal, QC, Canada

bmaniy@cs.mcgill.ca

1. Introduction

Galaxy is the a *public computing utility* (PCU) project at the ANRL, McGill University. It is a community-based resource provisioning infrastructure which hires public resources and leases them to parties who are in need of resources. This on-demand based resource provision enables many different types of applications hosted on Galaxy. Implementing a PCU like Galaxy has to face many challenges such as resource naming and discovery, status monitoring and aggregation, resource allocation and job scheduling, quality of service (QoS) management, *service level agreement* (SLA) management, and security and trust management. This draft describes the design and architecture of the naming and discovery substrate of the Galaxy, the *resource addressable network* (RAN).

2. Architecture Overview

RAN is the (resource) naming and discovery substrate of the Galaxy. Hence, as illustrated in the organization of the Galaxy middleware (Figure 1), RAN is placed at the bottom layer of the Galaxy architecture. It provides necessary functionalities to the *grid resource management system* (GRMS) in the upper layer which handles the high level resource management such as resource allocation.

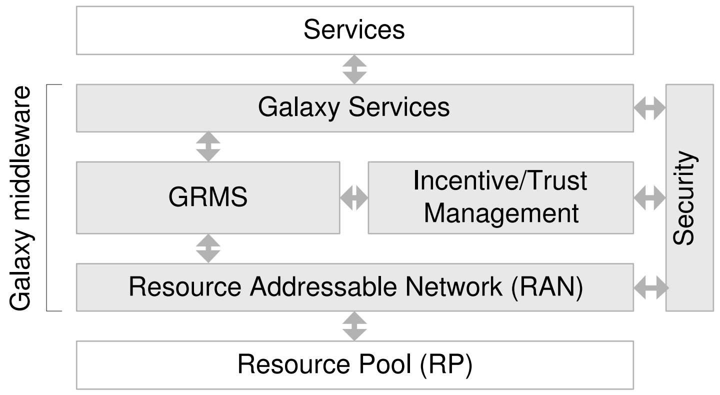

Figure 1: Galaxy middleware structure.

In the RAN level, the Galaxy resources form a self-organizing and self-healing P2P network, maintaining application level routing tables that enables a scalable decentralized resource discovery. RAN naming is composed of three parts: (a) based on *profiling* the resources into specific *types* (*profile-based naming*); (b) based on the locations of the resources in the network (*location-based naming*), and (c) based on the organizational *domains* the resources are attached to (*domain-based naming*). These components seamlessly combine together to enable a scalable and an efficient discovery mechanism. Especially, the *location-based naming* makes the discovery mechanism QoS aware. This is one of the novelties of the RAN design.

In a RAN *cloud* (a sector in the Galaxy), the resources are collected in multiple *type rings* each comprising the resources of the same type/profile [1] (Figure 2). Route entries called *connection pointers* connects the rings together. The decision of between which nodes a connection pointer is to be created is based on the profile-based names and resource join order. The type rings preserves the locality properties of the resource locations and arrange the resources according to their location-based names. A resource in a ring has at least two route pointers called *ring pointers* to form the ring that point to the left and right neighbors in the ring. Also it can have a variable number of pointers (with a maximum boundary) called *jump pointers* to non-neighbor resources. Jump pointers and connections pointers help (a) to route to a destination with a $O(\log n)$ number of routing hops and (b) the self-healing process. The type rings are actually *space filling curves* (SFCs) covering the Cartesian coordinate space that represents network delay.

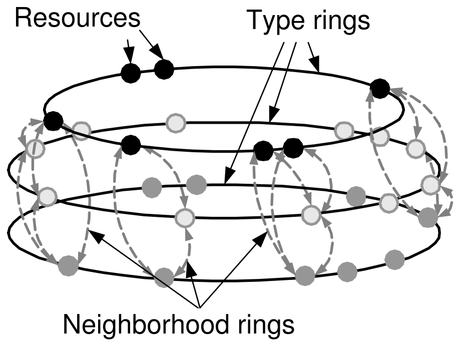

Figure 2: RAN type rings.

A resource can join RAN by contacting any already existing RAN node. At the time of join, the resource is profiled into one of the RAN-recognized types and it is joined into the specific type ring. Only the types with high popularity are recognized by RAN to cut-down the overhead in naming. The network location of the new resource is found relative to the RAN coordinate space using the *landmark aided positioning* (LAP) scheme. The insertion point of the resource in the type ring in which it belongs to is selected according to this location; A location-based name is assigned to the resource in the process.

Resource discovery is carried out with the same mechanisms used for a resource join: a resource request is profiled into one or more of the recognized types and the search is constrained within those specific type ring(s). A request can be profiled into multiple types in the case of wildcards present in the request. The required QoS descriptions in the request is used to constrain the search within an area in the RAN coordinate space and the location-base names of the resources are effectively used to find a suitable resource.

Another important part of the RAN architecture is the *profile adaptation* engine. Profile-adaptation is a feed-back control system that monitors the incoming queries and their results to regulate the number of resource types that should be recognized by RAN. Lesser the number of types recognized, more the queries fail to match exact resource description; on the other hand, the higher the number of recognized types, the higher the overhead in handling the names. Therefore, in effect, profile adaptation trades off the performance for reduced overhead. All the RAN nodes participate in the profile-adaptation to distribute the load of this decision making process.

The security in RAN is implemented in the form of *self-certifying type names* and *profile certificates*. Self-certifying type names are used to establish a secure connection with a specific resource while the profie certificates are used to validate the profiles presented by the resources. The trust/incentive management in the GRMS level also indirectly participates in the security in RAN level: the incentive mechanism imple-

[1]the words *profile* and *type* are used in this draft interchangeably

ments a penalty system that penalize the resource who falsely advertise their attributes (that affect it profile). Other than the explicite security mechanisms in RAN, as the RAN algorithms are all completely decentralized, attacking or compromising a single RAN node have only a minor impact in the overall system performance.

Section 3 provides an concise description on *space filling curve* which is used in many phases of RAN design. The rest of the draft describe the RAN architecture in details.

3. Space Filling Curves

RAN structure uses the space filling curves in many phases of its design. This section gives a brief description of space filling curves and its characteristics.

A *Space filling curve* (SFC) [Sag94] is a curve that passes through all the points in a given bounded space. There are a many such curves defined such as *Hilbert curve*, *Peano curve*, and *Sierpinski curve*. A real such curve is practically impossible and useless to produce, and hence mostly approximated versions of these curves are used.

In RAN, SFCs are used in the profile-based and location-based namings and hence to create the rings. The SFC used in RAN is Hilbert curve. Different levels of approximation of the Hilbert curve is shown in Figure 3. (When the approximation level tends to infinity the approximated SFC becomes a real SFC). The RAN type rings are actually SFCs constructed in coordinate space representing network delay. The resources are placed at the nodes of the curve and the lines between nodes denote the ring pointers. Actually an SFC is open as shown in the figure, but the end points are connected in RAN to create the “rings”.

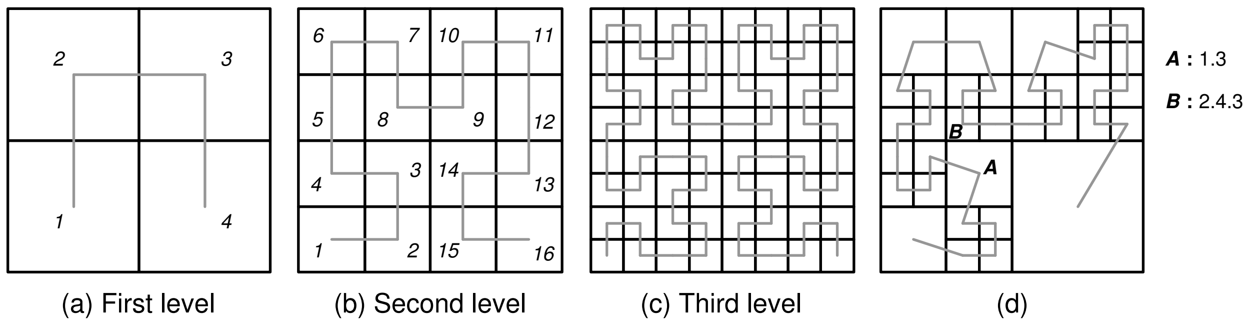

Figure 3: Hilbert space filling curves.

There are three characteristics of the SFCs that make them useful in RAN [ShC02, But71]:

1. Mapping of points in multi-dimensional space to a single dimensional curve: useful to place the resource in multi-dimensional (landmark) space (Section 5) onto a curve (the type rings) or compress the attribute-value pairs into a single type name (Section 4).
2. The mapping is such that the points that are closer to each other in the multi-dimensional space are closer on the single dimensional SFC too: useful to construct a a proximity based discovery mechanism.
3. SFCs can be produced recursively:
    - Can build up the curve incrementally and asymmetrically; Here asymmetry means that the curve in all the Cartesian regions need not be in the same approximation level (Figure 3(d)). This

conveniently matches the non-even distribution of resources in the RAN.

- Enables a hierarchical naming of the resources. For example, Node B in Figure 3(d) can be named as 2.4.3 in the hierarchical naming. Hierarchical naming can be efficiently used in routing (Section 7).

4. Profile Based Naming

One important common characteristic of the existing resource discovery systems is that they all implement some kind of *description-based naming*. Description-based naming does not provide any kind of a “name” for a resource, but describes it with its attribute-value tuples. While the description-based naming targets on providing a highly flexible resource discovery (resource of any characteristics can be queried), it has two important drawbacks: (a) the tremendous overhead in the form of size and number of the messages; and (b) the complex attribute-value matching algorithm.

Contrast to the description-based naming, RAN uses a *profile-based naming* where resources and queries are *profiled* into specific resource *types* based on the resource descriptions (Figure 4). Only popular resource descriptions are allowed as recognized types in RAN and by allowing attribute-value tuple involving inequalities, ranges of resource descriptions can be compressed into a single resource type. Once profiled the resources or queries can be simply identified by their *type IDs* instead of complex descriptions. It greatly cuts down the size of the RAN messages. Also this moves the load of the matching algorithm to the profiling algorithm which is an efficient alternative because profiling is carried out only once while matching algorithm might need to be run at many places. Further this approach decouples the naming (type IDs) from discovery similar to the conventional naming systems like DNS do, and hence improves the scalability (in both naming and discovery stages).

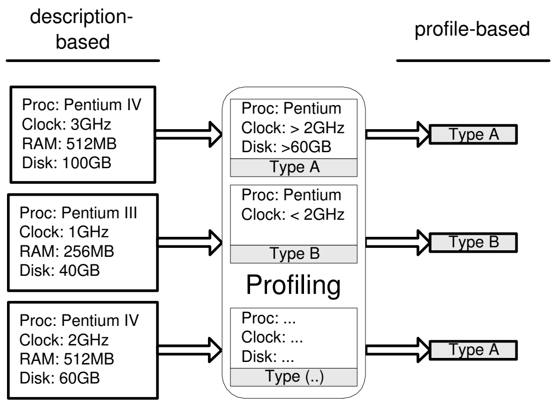

Figure 4: Profiling the resources.

Figure 5 shows the initial steps at the time of a new resource joining the system. When a resource wants to join the Galaxy, it first contacts the *Galaxy service provider* (GSP) which provides with the profiling and galaxy bootstrapping codes [2]. The profiling code probe the resource for a specific set of resource attributes and from the values it extracted it profiles the resource into one of the RAN-recognized types. Once profiled, the resource is identified by the profile name.

The resource request in RAN is allowed to be detailed, specifying list of attribute-value pairs of the requested resource. Upon reaching the RAN, the request goes through the same profiling process as a new

[2]this can be through a secure web registration interface similar to joining a KaZaA network

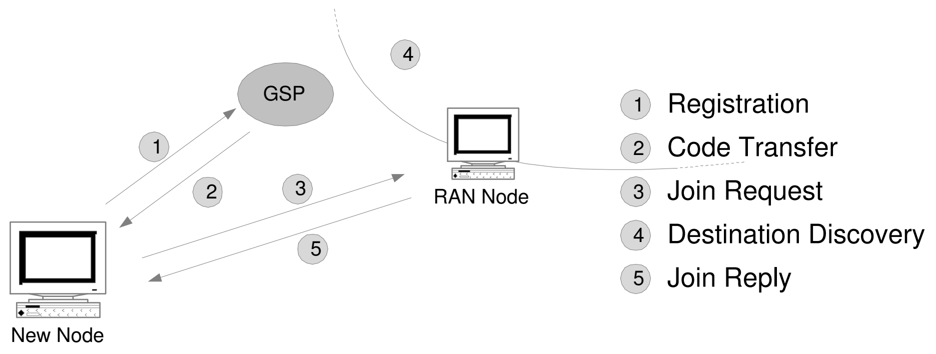

Figure 5: New node join process.

resource does to be classified into a recognized type. In the case of a request that includes wildcards (“don’t care”) in its attribute-value description, it can be profiled into more than one resource type. The discovery is carried out for this type or types.

4.1. Creation of Type IDs as Hilbert Indices

The design of type ID in RAN serve two purposes: (a) any resource/query description to be mapped onto one of the RAN-recognized profiles; (b) to integrate security with type IDs in the form of *self-certifying type names*.

The profiling process of RAN define an *attribute template* that list a set of resource attributes to be considered in the process. The process considers the resource descriptions in an Cartesian space of which axis is formed by the the attributes defined in the attribute template. Now, any attribute-value tuple of a resource or query can be mapped on a point in this *profile space*. In other words, each profile is a point in this space. However, in order to implement this procedure, some type of attribute values such as processor type or OS type, have to be manually enumerated.

When there is a coordinate space, there can be a SFC defined that follows through the regions of the space. Therefore, with a Hilbert curve used as the SFC, each profile can be denoted with a Hilbert index. By restricting the approximation at different levels at different regions in the profile space (Section 3), the number and resolution of the recognized profiles can be controlled. For example, one recognized profile can denote a description of {*CPU=Pentium; Memory=(0–256MB)*} while another denotes {*CPU=Pentium; Memory=(256MB–5GB)*} (note the non-uniform memory ranges). Any point in the profile space belongs to a particular region, and it means that every single resource description can be mapped on to a recognized profile. Wildcards (the “don’t-care” values) in attribute-value pairs denotes a region (instead of a point) in the profile space. This region can be sometimes covered by a single recognized profile or in other cases covered by multiple adjacent profiles. The locality property property of the SFC (Section 3) ensure that each description is mapped onto a closest profile if not exact.

The profile adaptation introduced in Section 2 and described in Section 4.2 is basically to decide the level of approximation of the SFC (recall that the approximation level decides the number of Hilbert indices). If a particular region in the profile space receives much hit, then the parent region is considered to be refined into the next higher level of approximation so that the high-hit region owns a separate Hilbert index.

4.2. Profile Adaptation

As mentioned in Section 4, only a certain number of profiles are recognized by RAN. The profiling mechanism is designed such that any resource description will fall into one RAN-recognized profile. Wildcards are allowed in resource description, which results in mapping the resource description to possibly more than one profile. In these cases, the reply to the query will provide a list of resources from which the request originator can select the most appropriate. Therefore restricting the number of recognized profiles in the system can result in reduced performance as the post-process on the result may become heavy. On the other hand increasing the number of recognized profiles increases the routing table size (Section 7) and number of messages.

RAN employs a *profile adaptation* mechanism (Figure 6) to efficiently trade-off performance to reduced system overhead and fast response. It is a feedback system which is basically tries to keep the number of recognized types to the lowest. The feedback from the system extract the trend in the queries to capture the popular resource at the time. When the number of queries for a unrecognized type increases beyond a design-specific threshold the system includes the type within its recognized types; similarly, the profiles that lost their popularity below another threshold are removed from the recognized list. Feedback in profile adaptation consider only the query descriptions not the resource descriptions as it is the unsuccessful queries that are considered as the degraded performance.

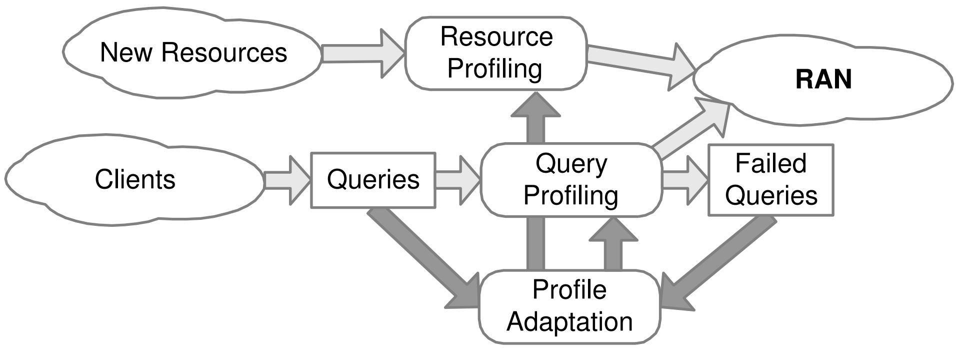

Figure 6: Profile adaptation scheme.

The details of the profile adaptation mechanism is still to be explored further

Profile adaptation consider only the popularity of a particular resource type. In the future we can develop a revenue weighted adaptation: There can be some queries which occur in low frequency but, if served, provide high revenue. It is better to acknowledge the profiles that allow this kind of high revenue queries.

4.3. Profile Adaptation and Type IDs

This details in this section might have to be changed depending on the exact implementation of the profile adaptation engine (Section 4.2).

Profile adaptation described above is basically adding or removing a Hilbert index into the list of recognized profiles. When there is a new profile is to be recognized, it is always the case that a new type ring is born from an existing type ring because conceptually it is bringing up the approximation level of the Hilbert curve to the next level. The nodes in the parent type ring go into a re-profiling phase to refine their membership. If any resource finds itself belongs to the new type, it sends a time-stamped announcement message along the parent ring. When a node of the new type receives an announcement message from another node of the new type, it neglects it if the time-stamp is newer than its own, otherwise it selects the other node as the authoritative member of the new type. The authoritative member then creates the self certifying type name and send partition message through the parent ring. This message in effect informs each node of the new type about its preceding neighbor in the to-be-formed new type ring. These nodes advertise themselves to their predecessors so that the ring pointers for the new type ring can be formed. Once the new ring pointers are formed, the new ring gracefully separate from its parent by each node sending a leave message in the parent ring. As the location of the nodes are unchanged, old location based naming can be still used making the process lightweight.

The profile updation might sometimes require a new attribute that is not in the profile template to be considered in the profiling, or might require an attribute that is in the profile template to be removed (when it is not of anybody's interest anymore). Then the profile template has to be modified. If there is a need for this, which is a highly infrequent event, the new templates are passed to all the nodes. Even though the profile-names (Hilbert indices) change with the change of profile-template, it can be expected most of the nodes remains in the same ring (but with different type name). If any of the resources finds out that it belongs to another type ring according to the modified profile template it gracefully leaves the old ring and joins the one appropriate. In the case of a change profile template, the creation of the new type ring will not be like dividing from an already existing ring, but building it up as a completely new type joining the RAN.

5. Landmark Aided Positioning

The QoS aware resource discovery in RAN is achieved by the *landmark aided positioning* (LAP) of the resources. Here, the delay between resources/network points is considered to be the fundamental QoS metric. The idea of LAP is not new ([NgZ02, XuM03, CoD03]). [XuM03] is much closer to our design in RAN in which LAP is combined with SFCs. However, the LAP described here differs in introducing a hierarchical location-based naming (Section 6) to LAP scheme.

The LAP works as follows: The design assumes a d-dimensional Cartesian coordinate space to place the resources. Distances in the space are mapped to network delays. At least $(d + 1)$ persistent nodes with known locations (coordinates) are selected as *landmarks*. Here "persistent" means that the landmark nodes stay in the network quite a long time and with low variant network conditions. When a new node is to be positioned in the space, it pings the landmarks and forms a *distance vector* $\mathbf{L} = \{l_0, l_1, \ldots, l_n\}$, where l_i denotes the delay between the node and the landmark i and n $(\geq (d + 1))$ is the total number of landmarks. With the coordinates of the landmarks and the measured $\mathbf{D}$, finding the coordinates of the new node is a geometrical problem. The rest of this draft assumes a coordinate space of $d = 2$ and $n = 3$.

Theoretically this geometrical problem should look like the Figure 7 and we should be able to apply a simple geometric solution algorithm. But, as the pings never travel in straight lines, the actual problem would look like the one shown in Figure 8. Now the solution can not result in exact solution, but must try to find an optimal values for the coordinates of the new node which minimize the difference between the measured distance and calculated distance from the calculated coordinates [NgZ02]. Instead of running the heavy simplex algorithm to solve this problem as it is in [NgZ02], RAN uses the *spring algorithm* proposed in [CoD03] to solve this problem. This algorithm incrementally refines the coordinates of the new node until the error falls within a design specific tolerance. Appendix A outlines this algorithm.

6. Location-Based Naming

location-based naming is the process of assigning hierarchically structured names to the nodes according to their position in the space described in Section 5. The algorithm to generate the positional names is evolved from the SFC creation algorithm [But71] so that the positional names exploit the characteristics of the SFC as described in Section 3.

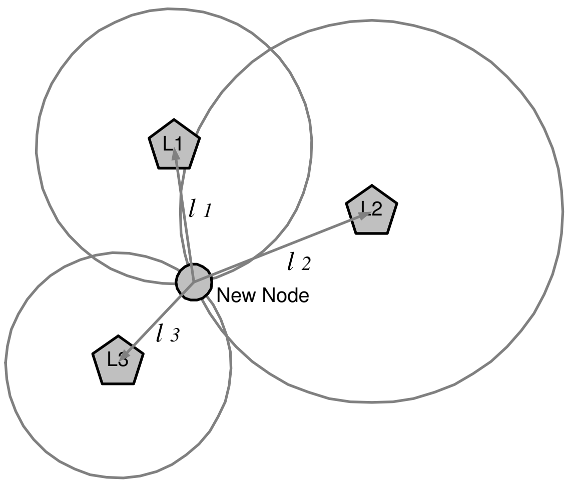

Figure 7: Landmark positioning at ideal conditions.

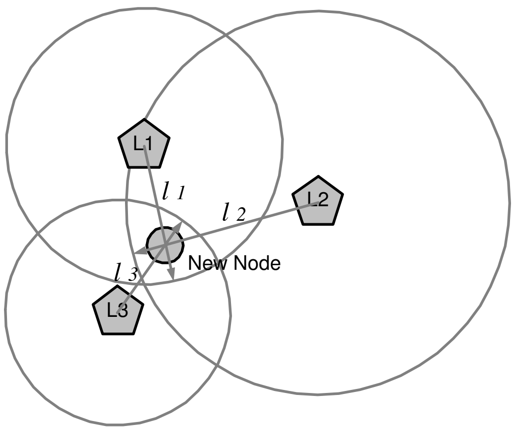

Figure 8: Landmark positioning at real conditions.

Positional name of a node is the Hilbert index (in hierarchical notation) of the node's coordinates at some level of SFC approximation. The approximation level is fixed to avoid running the Hilbert index finding routine again and again. Still as the distribution of the nodes in the landmark space is not even, the SFC for location-based naming is non-uniform as in Figure 3(d). For example the Hilbert index for the node A in the figure can be $1.3.2$ (if the fixed approximation level is 3), but as there is no other node in the region at the third approximation level, node A reigns the whole region of 1.3.

7. Routing in RAN

Routing in RAN is an application level routing process using an application level routing table. The building up of this routing table (Section 8) is incremental and merged with the routing algorithm. Therefore the routing in RAN is explained in this section before describing the route creation process. The objective of RAN routing is to find the IP address of a machine of the requested type which is at or closest to the requested location.

7.1. Routing Tables in RAN

The routing table in a RAN node is two layered: one for routing along the profile space to reach the type ring of the requested type and the next for routing along the type ring to reach the requested location. The entries in each section map location-based or profile-based names to IP addresses.

Type Ring Routing Table:

Table 1 shows a type ring routing table at a node with the locational name $2.3.3.1.0$. This type ring routing table has two sections: one containing *ring pointers* and another for *jump pointers*. Ring pointers point to the (left and right) neighbors of the node in the type ring and therefore they point to the two closest nodes. Jump pointers provides the capability of "jumping" to nodes in different regions than the current node. (Compare the Table 1 and Figure 3 for a clear understanding of regions). It is well structured such that the number of hops a discovery message should pass through is in the order of $O \log_4 n$ where the n is the total number of nodes in the system. The jump pointer section has 4 columns as the location-based names are derived from a Hilbert curve and it has 4 primary regions (Figure 3(a)). The section has m number of rows where m is the level of approximation of the Hilbert curve used for location-based naming. In the example shown $m = 5$. Ring pointer section can not be empty, but the jump pointer section can be. Jump pointer entries get filled up with the life time of the system.

Ring Pointer Section			
Left Pointer	IP Address	Right Pointer	IP Address
2.3.3.0.0	...	2.3.1.2.0	...

Jump Pointer Section							
0		**1**		**2**		**3**	
Name	IP Address	Name	IP Address	Name	IP Address	Name	IP Address
0.2.1.0.0	...	1.2.3.3.3	...	▨		3.2.2.0.1	...
2.0.2.3.2	...	2.1.3.1.2	...	2.2.3.2.0	...	▨	
–	–	2.3.1.2.0	...	2.3.2.0.1	...	▨	
2.3.3.0.0	...	▨		–	–	–	–
▨		–	–	–	–	–	–

Table 1: A sample type ring routing table at type a node with locational name of 2.3.3.1.0.

Connection Ring Routing Table:

The *connection pointers* are used for routing along the profile space. The routing table for connection pointers has the similar structure as the type ring routing table. similar to the type ring routing table, connection pointer table also need not be complete, but atleast the neighbor pointer section should be. This essentially makes up a *connection ring* and guarantees the reachability of any type ring from any other type ring (refer Figure 2). The neighbors in the connection ring are the ones next in the profile-based Hilbert indices (Section 4). While the entries in the type rings point to nodes in different locations, connection pointers point to nodes in different type rings. Here, it has to be remembered, as a range of unpopular resource descriptions are mapped to the closest recognized type, those profiles do not form seperate type rings. Therefore, unlike the entries in type ring table, many connection ring entries can point to a single IP address (i.e to a single node).

As described in Section 4, profile-based naming also uses a Hilbert curve similar to the location-based naming scheme. However, the dimension of the coordinate spaces in each case is different. The location-based naming which maps nodes onto a space of network delay uses a 2-D space and therefore the number of primary regions in Hilbert curve is 4. But in profile-based naming, the number of dimension is equal to the number of attributes in the attribute template used. For a 5-attribute template, the dimension of the coordinate space is 5 and consequently the number of primary regions is 32 (2^d). Therefore the number of columns in the routing table for connection pointers differs from that in the jump pointer section. Also the number of rows need not be the same as both of them are uncorelated design parameters.

As there are many nodes in each type ring, every entry in the connection ring table has the possibility of pointing to one of those nodes. However, the route buildup procedure (Section 8) makes sure that the entry points to the node that is closest in terms network delay. This design augments the location-aware routing within the type ring and produces a system-wide QoS aware resource discovery.

7.2. Routing in RAN

A resource query always contains the profile of the required resource and a location of interest. Converting the human specified query description into a location of interest and profile is the responsibility of the RAN kernel at the node of request origination. If there is no location of interest specified in the query, the location of the originating node itself is passed as such.

When a discovery request reaches a node, it checks whether query is for a resource belongs to its type ring (note that nodes in a type ring need not be exactly of same type). If it is, the query will be restrained to the type ring; if not, the node checks its connection ring table to reach the requested type and routes the request to a node of the requested type. If no pointer found for the exact match to the requested profile, the request is passed to the next closest type ring according to the profile-based Hilbert index. This type of routing ensures the request reaches the type ring that represents the requested resource profile. Once the request has reached the required type ring, then the location based routing starts off. It checks the location of interest where the queries resource should be located and uses the type ring jump pointers to reach the location with a minimum number of hops possible. When a node can not find a jump pointer to route the request towards the requested location, it uses the ring pointer that points towards that particular direction.

In the case of the resource request includes wildcards (“dont cares”) in the resource description, the request can be mapped into multiple resource profiles. In this case, discovery procedure can be carried out quickly as a resource of any of those matched profiles can be returned as the result.

The resource request message can also specify a tolerance value, in which case the routing procedure need not look for the exact or closest match to the destination, but can return with a node that is within a specified tolerance distance from the destination. Also the number of resources required can be specified in the request. Other than acquiring multiple resources, this is helpful in post-processing the result to find a resource with the exact required resource description.

8. Route Creation

The routing tables at each RAN node are created at the time of its join and grow with its life with RAN.

As mentioned in Section 2, when a new node wants to join RAN, it has to send a join request into the system through a already existing RAN node (*entry node*) (Figure 5). The join request is treated as a regular resource request message, where the location of the new node is considered to be the required destination location. Hence the join request is routed to a destination node as described in the Section 7. Here there can be two cases:

1. The destination node is belongs to the same type ring as the new node. This can lead to two sub-cases:
   (a) The destination node matches exactly with the new node’s location-based name: It is a very rare occurrence as the approximation level of the locational Hilbert curve is selected such that every resources can have their own RAN region. However, single site nodes can fall into this fate. In such a case, the new node copies the routing tables from the existing node and the existing node keeps a *buddy-pointer* to the new node. Other nodes in the system need not know about the new node, unless the already existing node of that location departs the system. In which case, the newly joined node captures the place.
   (b) The destination node is the closest match: the destination node decides on which side the new node should be inserted. It updates its ring pointer of that side with the pointer to the new node and informs its exiting neighbor on that side about the arrival of the new node so that the neighbor also can update its ring pointer. The new node creates its routing table by copying the jump pointers and connection pointers from the destination node and by creating the ring pointers of which one points to the destination node and another to its ex-neighbor.
2. The destination node and the new node do not belong to the same type ring: The routing mechanism

is such that the destination node is always belongs to the type ring of the new node. If it is not the case, it is only because the new node is the first node for that type ring. In this case, the destination node will be a node from the closest type ring. In this situation, the new node copies the connection ring table from the destination node and creates proper ring pointers. The destination node updates its ring-pointers to include the new type ring. It sends an ring pointer update message along its type ring so that the other nodes in the ring can update their ring pointer entries. Also the destination node informs its ex-neighbor in the connection ring about the new insertion, which in turns updates its connection ring pointers and sends and update message to the other nodes in its ring.

As the entry node and the new node happen to know each other in the procedure they can create jump pointer entries that point to each other in their routing tables. This jump pointer entries will be in the type ring table if they belongs to the same type ring or in the connection ring table if they do not.

It should be noted here that the above described procedure does not ensure that the entries in the connection ring point to the closest nodes as mentioned in Section 7.1. The connection pointers are updated later to that state as the node stays in operation with RAN (*lazy update*).

Once the routing table is initialized, the node keeps on sniffing the RAN messages go through it. Each RAN message have a "source" section which provides the information of the origin of the message. This section contains the location-based name, type, and the IP address of the source. When a node see a message passing through it, it can lead to the following route updates:

1. The source node belongs to the same type ring as the current node: If the current node does not have a pointer to the source location in its jump pointer section, it uses the source to create the required entry.
2. The source node and current nodes belongs to different type rings:
   (a) If the current node does not have a connectoin pointer for the source type ring, it creates one using the source.
   (b) If it already has one pointer for the source type ring, it compares the location of the source and the node in the existing entry and if the source found to be closer than the node in the exisiting entry, the entry is updated with the source. It is the *lazy update* of the connection pointers to keep also the connection ring location-aware.

The above described procedure ensure that the routing table of a node is got filled up as it stays with RAN in operation continously.

9. Galaxy Service Provider

The *Galaxy service provider* (GSP) is a separate entity of the Galaxy system, where "separate" means that their middleware component is completely different from the rest of the Galaxy nodes. GSP need not be a single node, but probably is a network of inter operating GSP nodes. The primary function of the GSP is to assist the resource join procedure (Figure 5). The resource to join the Galaxy contacts the GSP through a secure web registration interface. Once the registration is complete, the resource is asked to download the Galaxy code and install it. The Galaxy code essentially have the RAN code and, depending on the policy creation procedure at the time of installation, some additional components from the upper layers of the Galaxy middleware (Figure 1) are installed on demand basis.

The GSP should maintain a database that includes the followings: (a) list of recognized profiles; (b) list of candidate landmark nodes; and (c) list of nodes from every recognized profile. These information is used by the Galaxy code at the time of new resource join. GSP randomly selects one node as the contact point of each resource join which ensure the jump pointers are created (Section 8 really in a random manner (which is the required characteristics to ensure the $\log n$ hops complexity).

If the profile adaptation procedure modifies the list of recognized profiles, the super-super node contacts the GSP to update the profile information.

10. Fault Tolerance in Galaxy

This section still has to be explored.

11. Conclusion

This is a design draft created to assist the coding stage of RAN. Therefore, it does not include any evaluation work on the theory/algorithm developed here.

In summary RAN provides two basic functionalities: naming and discovery. The naming used in RAN is two-tier: profile-based naming partition the available resource into type rings, within which landmark/SFC aided location-based naming is used to pin-point a specific resource. The SFC based hierarchical location-based naming enables an efficient routing scheme which is of $O(\log n)$ hops complexity and provides a low per node state cost (size of the routing table). RAN is fully decentralized with no "administrative" nodes and it is self-organizing and and self-healing.

References

[But71] A. R. Butz, "Alternative algorithm for hilbert's space filling curve," *IEEE Transactions on Computers*, April 1971.

[CoD03] R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris, "Practical, distributed network coordinates," *Proceedings of the Second Workshop on Hot Topics in Networks (HotNets-II)*, Nov. 2003.

[NgZ02] T. S. Eugene Ng and H. Zhang, "Predicting internet network distance with coordinates-based approaches," *The Proceedings of INFOCOM 2002*, June 2002.

[Sag94] H. Sagan, *Space-filling curves*, 1994.

[ShC02] S. Shekhar and S. Chawla, *Spatial Databases : A Tour*, 1 Edition, 2002.

[XuM03] Z. Xu, M. Mahalingam, and M. Karlsson, "Turning heterogenity to an advantage in overlay routing," *The Proceedings of INFOCOM 2003*, April 2003.

A. Spring Algorithm

```
inputs: $d$: dimension of the space;
        $n \geq (d+1)$: number of landmarks;
        $\mathbf{C} = \{\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_n}, \}$: landmark coordinate set
        where $\mathbf{c_i} = (x_{i1}, x_{i2}, \ldots, x_{id})$: coordinates of the landmark $i$
        $\mathbf{L} = \{l_1,\ l_2,\ \ldots\ l_n,\ \}$: distance vector;
        $conv\_limit$: limit for convergence

outputs: $\mathbf{coords} = \{y_1, y_2, \ldots, y_d, \}$: coordinates of the new node

Algorithm:
$\mathbf{coords} =$ random coordinates
$\delta = 1.0$
do {
    $\mathbf{old\_coords} = \mathbf{coords}$
    for each landmark $i$ {
        $\mathbf{u} = \frac{(\mathbf{c_i} - \mathbf{coords})}{|\mathbf{c_i} - \mathbf{coords}|}$    /*unit vector*/
        $dist\_diff = |\mathbf{c_i} - \mathbf{coords}| - l_i$
        $\mathbf{displacement} = \mathbf{u} * dist\_diff * \delta$
        $\mathbf{coords} = \mathbf{coords} - \mathbf{displacement}$
    }
    $\delta = \delta - 0.025$
    $\delta = \max(delta, 0.05)$
    $change = \max(\mathbf{old\_coords} - \mathbf{coords})$
} while ($change > conv\_limit$)
```