Layer Discovery in RINA networks

Eleni Trouva *, Eduard Grasa*, John Day** and Steve Bunch§

*i2CAT Foundation, Jordi Girona, Barcelona, Spain, eleni.trouva@i2cat.net, eduard.grasa@i2cat.net
**Computer Science, Boston University, Massachusetts, USA, day@bu.edu
§TRIA Network Systems, LLC, Illinois, USA, steve.bunch@ieee.org

***Abstract*—In the course of generalizing the Inter-Process Communication (IPC) model from one system, two, to N systems directly connected, the necessity of a function to figure out via which "wire" or interface the requested application is available became apparent. In a Recursive InterNetwork Architecture-enabled system this is equivalent to asking on which of the Distributed IPC Facilities (layers) the requested application is available. We name this distributed application, the Inter-DIF Directory (IDD). In this paper we explain why the IDD is fundamental in networks, allowing the discovery of applications that belong to layers other than the one the requested application is on. Moreover, there are two phases of the IDD function: first, finding the requested application and then, creating a DIF that supports the communication. We describe the actions taken in each phase. Finally, we give a simple example of an IDD that makes use of hierarchical names.**

I. Introduction

It is well accepted in traditional networks, such as the Internet, that for an application to find another application, they must both have access to the same address space. In other words, they both should have access to the same layer. In exploring the implications of the Inter-Process Communication (IPC) model during the development of the Recursive InterNetwork Architecture (RINA) [1], [2], [3], we found that this commonly accepted view is not the case for a complete network architecture. Although it is the case that for two applications to communicate they must belong to the same layer, there is no requirement for discovering the requested application to be in the same layer. What discovering the requested application does require is finding which layers make it available, and then choosing an existing layer to join or creating a new one to enable the communication. A simple case that illustrates the problem under research is given in Fig. 1. In the scenario shown the web browser (application A) residing in host H3 requests to communicate with the web server (application B) on host H2. The question is which layers support application B and which one should be chosen for the communication.

How is this problem tackled in the current Internet architecture? The current architecture does not provide names for layers. Since there are no names for layers, there is no way to describe on which layers a requested application is on. In addition, neither a directory with mappings of application names to layers exists. Domain Name System (DNS) is the only directory-like service in the current Internet, which provides IP addresses where an application protocol might be found if the requesting application knows the well-known port to connect to. If different applications are available using the same application protocol at the same IP address, the application protocol must be able to determine how to find them, e.g. http, SIP. In the Internet, applications must belong to the same layer. Even though there are distinct IP address spaces, there are no names for these layers. Hence, there is no means to find applications that use them. As a consequence, all discoverable applications are assumed to be accessible via the same layer instantiation, the same IP address space, whether a private network or the public Internet.

In RINA, a layer is recognized as a distributed application and hence has a name like any other application. In the course of constructing the IPC model, at one stage a function is required to hide the different interfaces from the application, so that it sees a consistent API and does not have to know what interface the requested application is available on. This function generalizes for layers of arbitrary scope. If the requested application can not be found on any layer the requesting system is a member of, it should simply ask its peers. We have called the application responsible for this function Inter-DIF Directory (IDD). Once the requested application is found, the IDD can create a DIF either by joining other DIFs or creating a new one, that gives the two applications a common address space to use for communication.

II. The Inter-DIF Directory

The IDD is a distributed application, or as called in RINA, a Distributed Application Facility (DAF), which is a collection of two or more cooperating application processes in one or more processing systems, which exchange information using IPC and maintain shared state. Each processing system has an instance of the IDD, an application process that enables the system to make available its applications and discover other applications. The application processes that are members of the same DAF are called Distributed Application Processes (DAPs). The IDD DAPs that belong to the same DAF (also called "peer IDDs") are able to exchange messages for the discovery of applications. Fig. 2 shows an example of an IDD DAF that consists of five DAPs. As every other DAF the IDD is supported by one DIF or a set of underlying DIFs that provide to the distributed application IPC services. The IDDs of any two systems that share a common DIF are called "nearest neighbors".

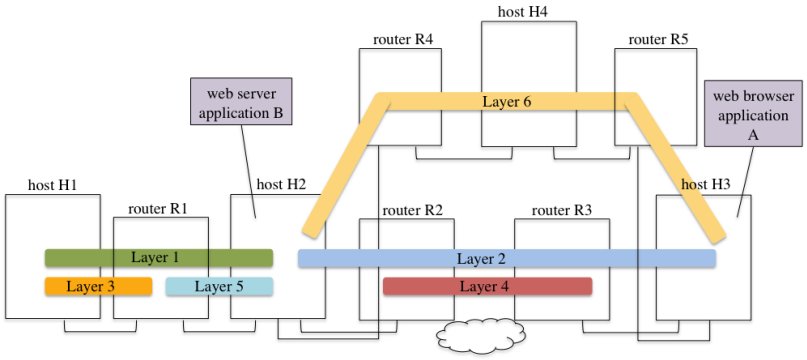

Fig. 1. Application discovery involves discovering which layers support the requested application.

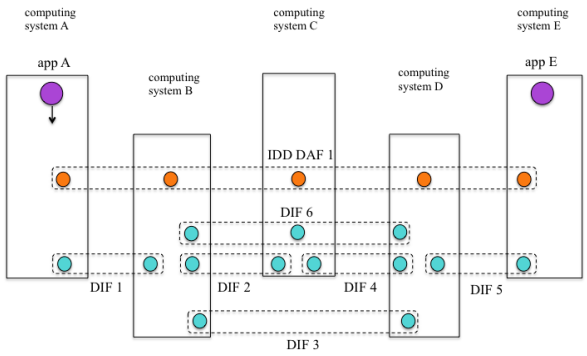

Fig. 2. Example of an IDD DAF.

Communication in RINA implies that the two systems that the communicating applications are residing share a DIF which provides IPC service. When an application wants to communicate with another application, first all the DIFs available to the source system are examined to see if the requested application is available through them. If none of the available DIFs returns a positive result, the IDD is responsible to first find the requested application and then attempt to create a supporting DIF between the two systems for the communication.

The DAPs that comprise the IDD will forward a request from one to another asking for the requested application. Forwarding begins at the IDD DAP of the source system and will be based on the forwarding policies between the members of the DAF. The forwarding continues until the IDD DAP of the system in which the requested application resides is found or when some other predefined termination condition is met to prevent infinite searching.

In the case that the destination system is reached, the IDD will have first to verify that the requested application is still there and then that the requesting application is allowed to communicate with it. If both checks are positive, the next action for the IDD is to find a DIF that will support the communication between the two applications. Since a common DIF between the source and the destination systems does not exist, a new one will have to be created by either expanding (joining) an existing DIF or creating a new one from scratch. The creation of the supporting DIF involves choosing a path from the source to the destination and coordination with the intermediate IDD DAPs for authorization control and creation and initialization of the new IPC application processes. Once the supporting DIF is in place, communication between the two applications can start.

Given the description of the IDD functionality in the previous paragraphs, we can divide the process into two phases:

- **Discovery of the application** Forwarding of the request between the peer IDDs until the destination application is found or a pre-defined termination condition is met. Confirmation that the requested application is available in the destination system and authorization check that the requesting application has the rights to access it take place in this phase too.
- **Creation of the supporting DIF** A DIF supporting the communication between the two user applications has to be found. This either involves creating a new DIF from scratch or expanding (joining) an existing one so that it spans from the source to the destination system.

To support these operations certain information is associated with each member of the IDD DAF. Each DAP has **naming information** that has to be unique amongst peer IDDs. Except from their names, the DAPs might have synonyms internal to the DAF that facilitate operations such as searching or routing. The synonyms might be flat or might make use of available heuristics schemes such as hierarchical names. Using hierarchical names the IDD DAPs can be clustered according to a chosen distance function and at the same time keeping the routing tables for routing amongst the DAPs small in size.

Additionally, each DAP has two tables to facilitate the discovery of applications, the Neighbor Table and the Search Table. Both tables contain forwarding information. The Neighbor Table contains local forwarding information, while the Search Table information that allows forwarding within the IDD DAF for finding the next directory to look for an application name. The **Search Table** maps requested application process names to list of next peer IDDs (that can be IDD names or synonyms), replying to the question “which peer IDD should I ask next for this application”. This table does not always point to nearest neighbors peer IDDs. The **Neighbor**

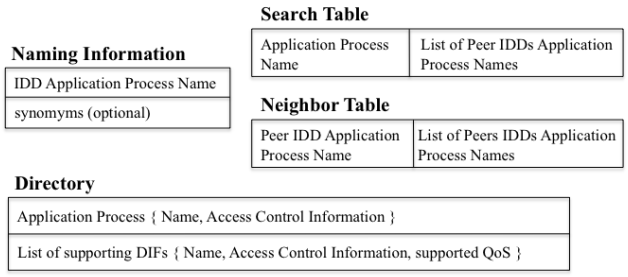

Fig. 3. Information maintained in an IDD DAP.

Table maps peer IDDs to list of next peer IDDs, attempting to maintain consistent, up-to-date routing information from each IDD DAP to every other IDD DAP in the DAF. This table always points to nearest neighbors peer IDDs, replying to the question "which of my nearest neighbor IDDs should I ask next to reach this destination peer IDD". Both tables might contain "default entries", which reflect the default IDD DAP to route to unless it is specified differently. Moreover, aggregation strategies on the application names or IDD synonyms might be used to minimize the search time within the DAF. Caching algorithms that optimize further the search might also be used. When changes in the graph of the DAF occur the IDD DAPs propagate updates throughout the DAF in order to maintain a consistent network view. Routing table updates are periodically transmitted in order to maintain table consistency.

Each IDD DAP has also a **Directory**. The Directory maintains information for the application processes running in a processing system. It contains records that have mappings of application processes information to list of DIFs information, providing the DIFs that support a requested application residing in this system. The application process information stored in the Directory is the application's name and access control information. The DIF information maintained is the DIF names, the access control information and the Quality of Service (QoS) classes provided. Fig. 3 summarizes all the information maintained in an IDD DAF member.

A. Application discovery

As we already mentioned, IDD DAPs exchange messages for discovering applications. Specifically, an IDD DAP can ask a peer IDD for a particular application by sending an **IDD-Request**. The request contains the destination's IDD DAP name, the source's IDD DAP name, the requested application's name, access control information, the requested QoS for the connection and a termination condition that is checked in each step to avoid infinite search. For example, a possible termination condition might be keeping the forwarding steps under a certain number. This can be done by setting a value of a hop counter before sending the request and decreasing it by one in each step. The hop counter is checked at each IDD DAP receiving it, ensuring the termination of the search after the chosen number of hops.

Upon receiving an IDD-Request, a peer IDD will first validate it, checking whether the request is in a valid format. If the request is not well-formed, an error will be returned to the sender. Differently, if the request is valid, the IDD will first examine the Destination IDD Name in the request and:

- If the Destination IDD is not itself, then it will look up the destination IDD in the Neighbor Table and see to which neighbor IDD it should forward the request to reach this destination IDD.
- If the Destination IDD is itself or if it is unknown, it will next look up the requested application process name in the Search Table and get the next IDD DAP that the request should be forwarded for the particular application name. The returned IDD might be itself, a neighbor IDD or any other IDD member of the DAF. Next, according to the returned IDD:

(a) *If the returned IDD is not itself*, the IDD will set the Destination IDD name in the request as the returned IDD, decrease the hop count and forward the request to a neighbor IDD according to what the Neighbor Table says for the destination IDD.

(b) *If the returned IDD is itself*, this means it is in the final destination system in which the requested application resides.

The forwarding process between the IDD DAPs continues until the system in which destination application resides is reached except if the termination condition terminates earlier the search. If the latter the IDD visited last will return a negative indication to the source IDD DAP notifying that the search has finished and the requested application was not found.

If the destination application is found the IDD will look up the requested application process name in the Directory. The IDD will perform the following actions:

- Confirm that an instance of the requested application is executing in the system.
- Verify that the requesting application is allowed to access the requested application using the access control information in the IDD-request and the access control information returned in the Directory record.

In case any of the above checks returns a negative result, the process will not continue. The destination IDD should return a negative indication to the source IDD DAP with an appropriate error message. If both checks turn positive, the process will continue to the next phase.

B. Creation of the supporting DIF

During this second phase the IDD will have to create a DIF that will be used to support the communication between the requesting and the requested applications. For the supporting DIF there are two possible options: a) Use an existing DIF and expand it to span from the source to the destination or b) Create a new DIF from scratch between the source and the destination systems. The example of Fig. 4 shows the possible cases for application A to communicate with application B. DIF 5 is a DIF created from scratch to support

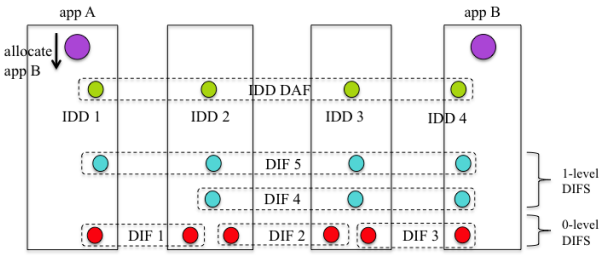

Fig. 4. There are two possible choices for the IDD in creating a supporting DIF for the communication: a) expand (join) an existing DIF (e.g. DIF 4) or b) create a new DIF acroos all the systems in the path (e.g. DIF 5).

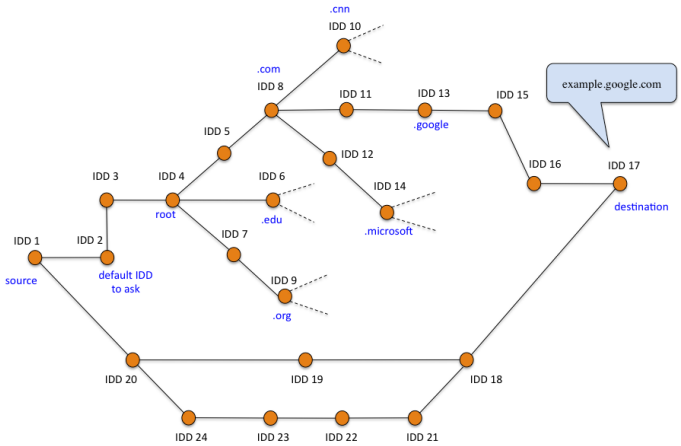

Fig. 5. An example of the IDD DAF in which the search during the first phase of the IDD function is organized according to the hierarchical organization of the application namespace. Neighbor IDDs are connected to each other (in reality a hop in the IDD graph may be several hops away).

the communication by creating and initializing accordingly IPC application processes in all the systems along the path. The existing DIF, DIF 4 can be also used if a new IPC application process is created in the first system in which application A is residing and join DIF 4.

Using the information returned by the Directory for the requested application process, the IDD will:

- Confirm that the supporting DIFs returned for this application exist and this system has an IPC process member of these DIFs. If this is not the case, stale records exist in the Directory that were not updated and do not reflect the current situation.
- Examine whether the source system is allowed to access any these DIFs. If none of them is allowed to be accessed, efforts should be focused towards the creation of a new DIF.
- Try to match the request's QoS requirements with the QoS classes supported by these DIFs. If none of the supporting DIFs can provide the requested quality in the communication, again the efforts should be focused towards the creation of a new DIF.

If all checks return a positive result for at least one of the application's supporting DIFs, the process of joining one of these DIF might continue. Differently, the creation of a new DIF between the requesting and the requested applications is the way to go. Whatever strategy is followed, both involve the creation of new IPC processes. Access control information in the request should be used to determine whether an IPC application process is allowed to be created and initiated in the processing systems along the path of the requestor and the requested application.

Note that the path followed during the first phase for the discovery of the application might not be the only path connecting the source and the destination systems in which the requesting and the requested applications reside. More importantly, it might not be the optimal path between the two applications. Having discarded first the paths that access is not allowed, the choice of the optimal path amongst the remaining paths on which the supporting DIF will span depends on the evaluation according to a chosen metric. The metric that helps to determine the optimal path might be a QoS measurement, might incorporate more than one QoS parameters or be something simple such as the path with minimum number of hops. As we have mentioned, optionally the IDD DAPs might be assigned with synonyms internal to the DAF that facilitate routing. The choice of these synonyms might be such that help determine the path that leads from the source to the destination (or vice versa).

III. An example of an IDD DAF

In this section to further explain how search in an IDD DAF works, we describe a simple example of an IDD DAF. Fig. 5 shows a possible configuration of an IDD DAF. The orange circles denote IDD DAPs in different processing systems that form a single DAF. IDD DAPs connected together with a link are neighbor DAPs, meaning that a DIF exists common to the processing systems that the IDD DAPs reside. An application residing in the same system with IDD 1, for example, a web browser, requests to communicate with an application that resides in the same system with IDD 17, named *example.google.com* . We assume that the IDD DAF serves a very large number of applications so that some aggregation strategy is required to speed up the search time. In this particular example an hierarchical namespace is chosen, in the same way it is used in DNS today.

The source IDD DAP, IDD 1 determines that the requested application is not reachable via any DIF it has. It then looks up the requested application name in the Search Table and finds the next IDD DAP to ask. IDD 1 is configured to always forward its IDD requests to IDD 2, meaning that IDD1 is an IDD with the minimum functionality. So, IDD 1 fields an IDD request in which the requested application name is *example.google.com*, destination IDD name is IDD 2, source IDD name is IDD 1, access control information, QoS requested for the connection and sets a termination condition for the search.

The request is now forwarded from IDD-DAP 1 to IDD-DAP 2. IDD 2 checks the destination IDD in the request and sees it is itself. So, it will look up the the application name in the Search Table and find out that the request should next be forwarded to IDD DAP 4. IDD DAP 4 is the top-level zone in the hierarchical namespace, as it is the root in DNS. So, IDD 2 changes the destination IDD name of the request to IDD 4 and looks in its Neighbor Table for IDD 4 and forwards the request accordingly to IDD 3. IDD 3 will check if it is the destination in the IDD Request and since it is not, it will consult its Neighbor Table for IDD 4 to see to which neighbor IDD it can forward the request to reach the destination IDD 4. Following this procedure IDD 4 will forward the request to IDD 5, which forwards to IDD 8, which forwards to IDD 11. The same is repeated and the request passes from IDD 13, IDD 15, IDD 16 and IDD 17.

Eventually, when IDD 17 receives the IDD request with a destination itself and requested application name example.google.com. It looks in the Search Table and finds that the returned IDD DAP to forward the request is itself. This means that the request has reached the destination system in which the requested application resides. The IDD DAP looks in its Directory and retrieves the record for the requesting application name *example.google.com* and gets back a list of supporting DIFs, access control information of the requested application and the DIFs and the QoS supported by the returned DIFs. The forwarding process and the contents of the Search and Neighbor tables at each step are shown in Fig. 6.

After confirming that an instance of the requested application *example.google.com* is still executing in the processing system, using the access control information carried in the IDD-Request, it will next examine whether the requestor has is allowed to access this instance. If it is allowed the next task for the IDD is to find a supporting DIF between the source and the destination systems. This either means to create a new DIF or use an existing one.

We assume that synonyms have been assigned to the IDD DAPs of the DAF to facilitate routing. For example, the synonyms might indicate that according to a chosen metric (e.g. number of hops, available bandwidth, etc) the path IDD17 - IDD18 - IDD19 - IDD20 - IDD1 is the optimal path to use for routing between the source and the destination systems. Now a negotiation procedure between these IDD DAPs has to take place to decide whether to create a new DIF or expand an existing one is the best way to proceed in creating the supporting DIF for the communication.

IV. Conclusions, On-going and Future Work

We are exploring and testing a theory, nominally referred to as RINA, to increase our understanding of networks. In this paper we explored how layer discovery can be done in RINA networks. We first gave some basic guidelines which an IDD should follow and then described an example of a specific IDD implementation that is applicable to real world networks. Our work is novel as it is the first effort to explore the properties

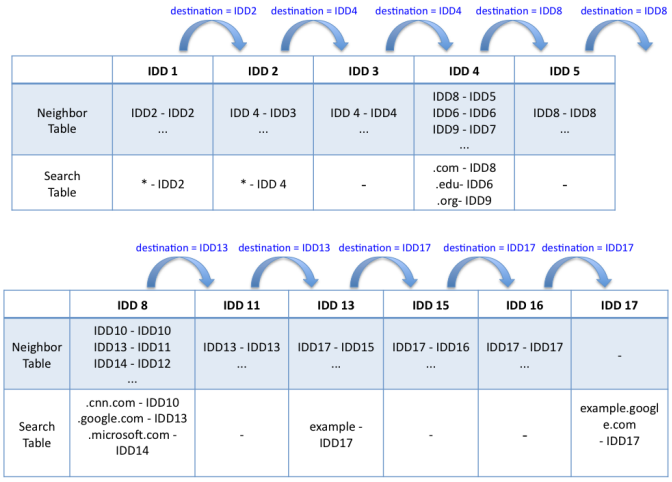

Fig. 6. The forwarding between IDD DAPs during the first phase of the IDD functionality and the relevant Search and Neighbor Tables contents of each IDD DAP. Entries with an asterisk denote default forwarding unless it is differently specified.

of the IDD function in RINA. As we explained, there is no a comparable function to the IDD in the Internet today. In the Internet, applications must belong to the same layer (DIF). As we saw, the IDD DAF might span across a sequence of RINA layers, allowing the discovery of all the applications registered in the IDD DAF. The implications of the IDD in network architectures seem profound. So far it appears that it eliminates the need for layers with large address spaces and creates greater security by better compartmentalization without impairing reachability. We expect other implications to emerge as we further explore the IDD's properties.

Our current work involves the development of a prototype that implements the RINA architecture, including an IDD implementation. Further research is on-going on exploring how existing DIFs between the source and the destination systems can be used to our advantage and in which cases expanding an existing DIF is preferred over creating a new one from scratch. In addition, we are investigating the architectural similarities of name look up across multiple layers and within layers, which appears to be the same operations at different scales. Future work includes experimentation, measurements and performance comparisons using the developed IDD implementation under a choice of different routing, updating and caching strategies.

Acknowledgment

This work has been partially supported by the Spanish Government, MICINN, under research grant TIN2010-20136-C03.

References

[1] J. Day, *Patterns in Network Architecture: A Return to Fundamentals (paperback)*. Prentice Hall, Jan. 2008.

[2] J. Day, I. Matta, and K. Mattar, "Networking is ipc: a guiding principle to a better internet," in *Proceedings of the 2008 ACM CoNEXT Conference*. ACM, 2008.

[3] Pouzin Society http://www.pouzinsociety.org/.