Arigatoni: Overlaying Internet via Low Level Network Protocols

Didier Benza, Michel Cosnard, Luigi Liquori, Marc Vesin

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Arigatoni: *Overlaying Internet via Low Level Network Protocols*

Didier Benza — Michel Cosnard — Luigi Liquori — Marc Vesin

N° 5805

Janvier 2006

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--5805--FR+ENG

INRIA
SOPHIA ANTIPOLIS

Arigatoni: Overlaying Internet via Low Level Network Protocols

Didier Benza, Michel Cosnard, Luigi Liquori , Marc Vesin

Thème COM — Systèmes communicants
Projets Mascotte

Rapport de recherche n° 5805 — Janvier 2006 — 27 pages

Abstract: We design a lightweight communication model, called Arigatoni, with related architecture, that is suitable to deploy the *Global Computing Paradigm* over the Internet. Communication over the behavioral units of the model are performed by a simple communication protocol on top of TCP or UDP protocol. Basic global computers can communicate by first registering to a brokering service and then by mutually asking and offering services, in a way that is reminiscent to Rapoport's *tit-for-tat* strategy of co-operation based on reciprocity. In the model, resources are encapsulated in the intranet in which they reside, and requests for resources located in another intranet traverse a broker-2-broker negotiation using classical PKI mechanisms. The model is suitable to fit with various global scenarios from classical P2P application, like file sharing, or band-sharing, to more sophisticated GRID application, like remote and distributed big (and small) computations, until possible, futuristic real migration computations, in the vein of the programming language Obliq by Luca Cardelli.

Key-words: Overlay Networks, Resource Discovery, Grid

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 —Télécopie : +33 4 92 38 77 65

Arigatoni: Un Réseau d'Overlay qui utilise des Protocoles à Bas Niveau

Résumé : Nous proposons l'architecture du système de communication Arigatoni, qui permet d'implémenter dans l'Internet le paradigme du *Global Computing*. La communication entre les différentes unités du modèle s'effectue par le biais d'un protocole de communication simple, qui opère directement au dessus des protocoles TCP ou UDP. Les ordinateurs globaux de base communiquent en s'enregistrant tout d'abord à un service de courtage, ils peuvent ensuite proposer ou demander des services de manière interchangeable, selon une stratégie similaire au modèle de coopération *tit-for-tat* de Rapoport. Dans notre modèle, les ressources sont encapsulées dans l'intranet dans lequel elles résident, et les demandes pour des ressources qui se trouvent dans d'autres intranets sont effectuées au moyen d'une négociation courtier-à-courtier, en utilisant les mécanismes classiques de PKI. Ce modèle permet de réaliser divers scénarios d'informatique globale, depuis les applications classiques pair-à-pair telles que le partage de fichier, ou encore le partage de bande, jusqu'à des applications plus sophistiquées d'informatique en grille, telles que les gros (ou petits) calculs distribués à distance. Nous pouvons même envisager des scénarios réels de migration de calculs, dans l'esprit du langage de programmation Obliq, développe par Luca Cardelli.

Mots-clés : Réseau d'Overlay, Découverte de Ressources, Grille

1 Introduction

This paper presents the first design of a light-weight architecture called, informally, Arigatoni[1][2] that is suitable to deploy, via the Internet the *Global Computing Communication Paradigm*, *i.e.* computation via a seamless, geographically distributed, open-ended network of bounded resources by agents acting with partial knowledge and no central coordination. In *freetalian* network-jargon, when you align some rigatoni

then you make an (high-speed) network connection that allows two or more units to communicate in a point-to-point fashion.

The Arigatoni model aims at designing, implementing, testing, and simulating a new light-weight architecture that in principle is suitable to deploy the *Global Computing Communication Paradigm (*GC*)* using a minimalist infrastructure and a *Global Internet Protocol (*GIP*)* built within the UNIX operating system and over the protocols TCP or UDP. It will be used by the units of the Arigatoni architecture to communicate.

Compared to OGSA-based middlewares [1] (*e.g.* Globus [2]), the Arigatoni model is much simpler and exploit the lower levels of the OSI stack. In principle, it could be deployed firstly in an intranet and further from intranet to intranet by overlapping an *Overlay Network* on the top of the *actual network*. For this we could consider the Arigatoni model, with related middleware, as one prototypical example of *overlay network*. In other words the Arigatoni slogan could be: "programming a *collaborative Global Internet* over the actual Internet". Recall that an *Overlay Network* is an abstraction that can be implemented on top of a global network to yield another global network. Overlay examples are resource discovery services (notion of resource sharing in distributed networks), search engines (abstraction of information repository) or systems of trusted mobile agents (notion of autonomic, exploratory behavior) [3].

The main ingredients in the Arigatoni model are one protocol, the *Global Internet Protocol,* GIP, and three main units:

- A *Global Computer Unit,* GCU, *i.e.* the basic peer of the Global computer paradigm; typically it is a small device, like a PDA, a laptop or a PC, connected with any IP network, unrelated to the media used, wired or wireless, etc.
- A *Global Broker Unit,* GBU, is the basic unit devoted to register and unregister GCUs, to receive service queries from client GCUs, to contact potential servants GCUs, to

[1] "Arigatou" in Japanese means (informally) *Thank-you*, while "Rigatoni" are one of the most commonly used pasta in Southern and Central Italy. Rigatoni, a wide, ridged, tube-shaped pasta, have holes large enough to capture pieces of meat or vegetables in sauces. They have ridges which allow them to hold more sauce.

[2] The Arigatoni model, protocol and middleware, is copyrighted by Luigi Liquori and the INRIA under the CECIL License.

negotiate with the latter the given services, to trust clients and servers and to send all the informations useful to allow the client GCU, and the servants GCUs to be able to communicate. Every GCU can register to only one GBU, so that every GBU controls a *colony* of collaborating Global Computers. Hence, communication intra-colony is initiated via only one GBU, while communication inter-colonies is initiated through a chain of GBU-2-GBU message exchanges. In both cases, when a client GCU receives an acknowledgment for a request service (with related trust certificate) from the proper GBU, then the client will enjoy the service directly from the servant(s) GCU, *i.e.* without a further mediation of the GBU itself.

- A *Global Router Unit,* GRU is a simple basic unit that is devoted to send and receive packets of the Global Internet Protocol and to forward the payload to the units which is connected with this router. Every GCU and every GBU have one personal GRU. The connection between router and peer is ensured via suitable API.

Effective use of computational grids via P2P systems requires *up-to-date* information about widely-distributed resources. This is a challenging problem for very large distributed systems particularly taking into account the continuously changing state of the resources. Discovering dynamic resources must be scalable in number of resources and users and hence, as much as possible, fully decentralized. It should tolerate intermittent participation and dynamically changing status/availability.

Many resource discovery algorithms and protocols have been proposed recently. As example, in [4], a P2P approach to resource discovery in grid environments is proposed. More precisely, the authors present a framework that guides the design of any resource discovery architecture. In [5], non-uniform information dissemination protocols are used to efficiently propagate information to distributed repositories, without requiring flooding or centralized approaches. Results indicate a significant reduction in the overhead compared to uniform dissemination to all repositories. In [6], a semantic resource discovery in the GRID is proposed using a P2P network to distribute and query to the resource catalog. Each peer can provide resource descriptions and background knowledge, and each peer can query the network for existing resources.

However, all these papers propose high level mechanisms or algorithms and do not address the overlaying Internet low level network protocols as we intend in this paper. From this point of view, it is worth mentioning reference [7] which investigates the applicability of a structured overlay network for the discovery of GRID resources based on the P-GRID overlay network and presents experimental results from a large-scale deployment on PlanetLab [8].

We do believe that our approach is complementary to this overlay network in the sense that it provides the necessary basic infrastructure necessary to a real deployment of the overlay network itself. Moreover, our work abstract on *which kind of resource* the overlay network is playing with; pragmatically speaking, this work could be useful for GRID, or for distributed file/band sharing, or for more evolved scenarios like mobile and distributed object-oriented computation in the style of the programming language Obliq [9].

Summarizing, the original contribution of the paper are:

- a simple distributed communication model that is suitable to make resource discovery transparent;
- a Global Internet Protocol that allows Global Computers to negotiate resources;
- a *complete independence* of the classical scenarios of the arena, *i.e.* GRID, file/band sharing, web services, etc. This domain independence is a key feature of the model and of the protocol, since it allows a complete abstraction from any given scenario.

We hope that the Arigatoni model could represent a little step toward a natural integration of different scenarios/area under the common "shield" of the Global Computing paradigm.

Road Map

The paper is structured as follows: Section II describes in an high level fashion, the Arigatoni model and its functional units. Section III presents one possible semantics of the three units (via one "reference" implementation). Section IV describes the protocol used by all the units to communicate. Section V concludes with some further work. In Appendix, the referee can find a real scenario handled in the Arigatoni model, some issues related to research discovery and security and and some keynotes on the social model underneath Arigatoni.

2 Arigatoni Units: Informal Description

An informal description of all the functional units of the Arigatoni model follows.

2.1 Global Computer Unit

In the Arigatoni model, a *Global Computer Unit (*GCU*)* is a cheap computer device composed by a small RAM-ROM-HD memory capacity, a modest CPU, a ≥ 20 keystrokes keyboard, a $\geq$ 1.5 inch screen, an IP connection, an USB port, and very few programs installed inside (one simple editor, on or two compilers, a `mail` client, a mini browser, a GSM module, etc). The operating systems installed in the GCU is not important. The computer should be able to work in *Standalone Local Mode* for all the tasks that it could do locally or in *Global Mode*, by first registering itself in the Arigatoni architecture, and then by making a global request to the Overlay Network induced by the architecture (that we call, ArigatoNet). Figure 1 shows the Arigatoni model. The GCU must be able to perform the following tasks:

- Discover, upon the physical arrival of the GCU in a new colony, the address of a GBU, representing the *leader* of the colony;
- Register/Unregister on the GBU which manages the colony;
- Request some services to its GBU, and respond to some requests from the GBU;

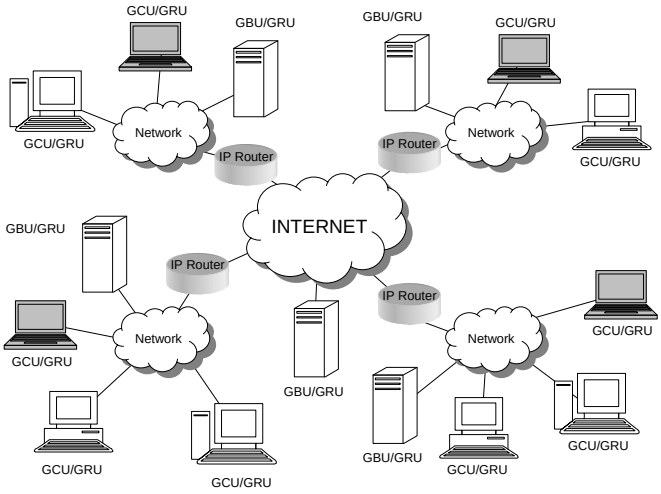

Figure 1: ArigatoNet

- Upon reception from a GBU of a positive response to a request, be able to connect directly with the servant(s) GCU in a P2P fashion, and offer/receive the service.

Since the Arigatoni model is P2P, it is worth noticing that a GCU can also be a resource provider (or play both roles). Hence, a GCU can also be a supercomputer, an high performance parallel cluster, a large database server, an high performance visualizer (*e.g.* connected to a virtual reality center), or any particular resource provider, that is linked to Internet. This symmetry is another key feature of the Arigatoni model.

Typically, a GCU could ask for big computational power, *e.g.* the GRID, or ask for a particular piece of software, *e.g.* classical peer-to-peer systems for file sharing, or ask for memory space, *e.g.* hosting web pages, or, more interestingly, ask to transfer one non completed local run in another GCU saving the partial results, under the case of a catastrophic scenario, like, *e.g.* fire, terrorist attack, earthquake etc, *e.g.* truly mobile ubiquitous computations.

2.2 Global Broker Unit

The *Global Broker Unit (*GBU*)* performs the following tasks

- Discover, the address of another *super* GBU, representing the *superleader* of the *supercolony*, where the GBU's colony is embedded[3]. We assume that every GBU comes with its proper PKI certificate. The policy to accept or refuse the registration of a broker with a different PKI are left open to the level of security requested by the Global Computers Colony (see Subsection B.2);

[3]One example, in a real life scenario, could be that the leader (Pope Benedict XVI GBU) of Vatican's colony, asks to the leader (President Barroso GBU) of the European Community's supercolony (composed

$\{\mathsf{GBU}\}$	is a (small) colony
$\{\mathsf{GBU}_1, \mathsf{GCU}_1 \ldots \mathsf{GCU}_m\}$	is a colony
$\{\mathsf{GBU}_1, \mathsf{GCU}_1 \ldots \mathsf{GCU}_m, \overbrace{\{\mathsf{GBU}_2, \mathsf{GCU}_{m+1} \ldots \mathsf{GCU}_{m+n}\}}^{subcolony}\}$	is a colony (it contains a subcolony)
$\{\mathsf{GBU}_1, \mathsf{GCU}_1 \ldots \mathsf{GCU}_m, \mathsf{GBU}_2, \mathsf{GCU}_{m+1} \ldots \mathsf{GCU}_{m+n}\}$	is not a colony (two GBUs)
$\{\mathsf{GBU}_3, \overbrace{\{\mathsf{GBU}_1, \mathsf{GCU}_1 \ldots \mathsf{GCU}_m\}}^{subcolony}, \overbrace{\{\mathsf{GBU}_2, \mathsf{GCU}_{m+1} \ldots \mathsf{GCU}_{m+n}\}}^{subcolony}\}$	is a colony (with 2 subcol.)
$\{\mathsf{GBU}_1, \{\mathsf{GBU}_1, \mathsf{GCU}_1 \ldots \mathsf{GCU}_m\}, \{\mathsf{GBU}_2, \mathsf{GCU}_{m+1} \ldots \mathsf{GCU}_{m+n}\}\}$	is a colony (GBU_1 is *elected* as the common leader)
$\{\overbrace{\{\mathsf{GBU}_1, \mathsf{GCU}_1 \ldots \mathsf{GCU}_m\}}^{subcolony}, \overbrace{\{\mathsf{GBU}_2, \mathsf{GCU}_{m+1} \ldots \mathsf{GCU}_{m+n}\}}^{subcolony}\}$	is not a colony (no leader in the top level colony)

Figure 2: Some Colony's Examples

- Register/Unregister the proper colony on the *leader* GBU which manages the super-colony;
- Register/Unregister clients and servants GCU in its local base of Global Computers. We assume by definition that every GCU can register to *at most* one GBU;
- Acknowledge the request of service of the client GCU;
- Discover the resource(s) that satisfies the GCU's request in its local base (local colony) of GCU;
- Delegate the request to another GBU governing another colony;
- Perform a combination of the above two actions;
- Deal with all PKI intra- and inter-colony policies;
- Notify to the client GCU or to a delegating GBU the servant(s) GCUs that have accepted to serve its request, or notify a *failure* of the request.

Every GCU in the colony sends its request to the GBU which is the leader of the colony. There are different scenarios concerning the demanded resource for service discovery, namely:

by 25 colonies) to join the European colony, instead of working in standalone mode. If accepted, then the Vatican colony can ask, always via the superleader to access to some services in the other EU colonies (*e.g.* Italy, France, etc.). Another hybrid, real life inspired, example could be the leader of the United Kingdom's colony (Prince Charles GBU) that ask to President Barroso to switch in the Euro-money-mode and leave the Pound-standalone-mode...

1. The broker finds all the resource(s) needed to satisfy the requested services of the GCU client locally in the intranet. Then it will send all the information necessary to make the GCU client able to communicate with the GCU servants. This notification will be encoded using the GIP protocol. Then, the GCU client will directly talk with GCU servant(s), and the latter will manage the request, as in classical P2P systems;

2. The broker did not find all the resource(s) in the local intranet. In this case it will forward and delegate the request to another broker. To do this, it must first register the whole colony to another supercolony;

3. A combination of steps 1 + 2 could be envisaged depending on the capability of the GBU to combine resources that it manages and resources that come from a delegate GBU;

4. After a fixed *timeout period*, or when all delegate GBUs failed to satisfy the delegated request, the broker will notify to the GCU client the *refusal of service* requested by the GCU client.

2.3 Global Router Unit

The last unit in the Arigatoni model is the *Global Router Unit (*GRU*)*. The GRU implements all the low level network routines, those which really have access to the IP network. It is the only unit which effectively runs the GIP protocol. The GRU can be implemented as a small daemon which runs on the same device as a GCU or a GBU, or as a shared library dynamically linked with a GCU or a GBU. The GRU is devoted to the following tasks:

- Upon the initial startup of a GCU it helps to register the unit with one GBU;
- It checks the well-formedness and forwards GIP packets across the ArigatoNet toward their destinations. GIP packets encode the requests of a GCU or a GBU in the Arigatoni network;
- Upon the initial startup of a GBU it helps the unit with several other GBUs that it knows or discovers.

2.4 Discovery

The are three kinds of *Resource Discovery* in Arigatoni, namely:

- the process of a GCU to discover a GBU, upon physical insertion in a GCUs colony;
- the process of a GBU to discover other *friend* GBU, upon physical insertion in the ArigatoNet network;
- the process of a GBU to find and negotiate a resource to serve a GCU's request in its own colony.

While the latter kind of discovery is discussed in the GBU's Subsections 2.3 and 3.4, the first two kinds will be discussed in detail in Subsection B.1.

3 Arigatoni's Units: Formal Description

This section describes a prototype implementation of the three units of the Arigatoni model. As any pseudocode, this encoding does not bring into light all the details which are usually swept under the carpet. We try to get the encoding as clean and compact as possible, and to do this, we will abstract as much as possible on all "bureaucracy" concerning synchronization between processes.

In what follows, everything in italic denotes a constant; in particular, *MyId* denotes the name of the current unit (like, *e.g.* `this` in object-oriented languages), and *MyGRU* denotes the name of the Global Router which is uniquely attached, via API to *MyId*, and *MyPKI* denotes my security certificate, and *MyRes* denotes the set of resource that the individual can offer to the community. Those values are packaged in a record (the calling card) called *MyCard*. The **inparallel**...**with**...**endinparallel** control structure allows two or many processes to execute concurrently and independently [10, 11].

3.1 Colony

A colony is a simple virtual organization composed by exactly one leader and some individuals. Individuals are Global Computers (think it an *Amoeba*), or (sub)colonies (think it as a *Protozoa*). A formal definition of colony is given using the BNF syntax

$$colony \ ::= \ \{\mathsf{GBU}\} \mid colony \cup \{\mathsf{GCU}\} \mid colony \cup \{\, colony \,\}$$

Rules are:

1. every colony has *exactly* one leader GBU and at least one individual (the GBU itself);
2. every colony contains individuals (some GCU's, other colonies).

Some examples of colonies are shown in Figure 2.

3.2 GCU's Semantics

The semantics of the GCU is described in the pseudo-code in Figure 3. It is composed by four processes, running in parallel whose intuitive behavior is as follows:

1. **Registering/Unregistering process** implements the (un)registration of a GCU to a GBU leader of a given colony;
2. **Basic shell process** is the classical read/eval/print loop. In the case of a local failure of a request, if the GCU is working in global mode, then the same request is forwarded to the GBU leader of the colony;

```
inparallel
while true do                    // Registration loop
 GBU = Discover(MyCard)
 case (GlobalMode,RegMode) is
  (true,false):
   ServiceReg(MyCard,GBU,LOGIN)
  (false,true):
   ServiceReg(MyCard,GBU,LOGOUT)
  otherwise:                     // Do nothing
 endcase
endwhile
with
 while true do                   // Shell loop
  Data     = ListenLocal()
  Response = LocalServe(Data)
  case (Response,GlobalMode,RegMode) is
   (login,_,_):                  // Open global mode
    GlobalMode = true
   (logout,_,_):                 // Close global mode
    GlobalMode = false
   (fail,true,true):             // Ask to the GBU
    MetaData = PackScenario(Data)
    ServiceRequest(MyCard,GBU,MetaData)
   otherwise: LocalReply(Response)
  endcase
 endwhile
with
 while RegMode do     // Global GBU listening
  MetaData = ListenGBU()
  case MetaData.OPE is
   SREG:               // GBU responds if it accepts my registration
    if   CanJoin(MetaData)
    then RegMode = true
    endif
    if   CanLeave(MetaData)
    then RegMode = false
    endif
   SREQ:   // GBU is asking for some resources
    if   CanHelp(MetaData)
    then ServiceResponse(MyCard,GBU,ACC)
    else ServiceResponse(MyCard,GBU,REJ)
    endif
   SRESP:                        // GBU responds if it has found some resources
    if   CanServe(MetaData)
    then Peers    = GetPeers(MetaData)
         Response = GlobalServe(MyCard,
                                Peers,MetaData)
         ServiceResponse(MyCard,GBU,DONE)
         LocalReply(Response)
    else LocalReply(fail)
    endif
  endcase
 endwhile
with
 while RegMode do     // Global GCU listening
  MetaData = ListenGCU()
  if   Verify(MetaData)
  then Data     = UnPackScenario(MetaData)
       Response = LocalServe(Data)
       if   Response == fail
       then ServiceResponse(MyCard,GBU,ERR)
       else ServiceResponse(MyCard,GBU,DONE)
            SendResult(MyCard,GCU,Response)
       endif
  else ServiceResponse(MyCard,GBU,SPOOF)
  endif
 endwhile
endinparallel
```

Figure 3: GCU pseudocode

3. **Global GBU Listening process** listens for any communication (service request or service response) from the GBU;

4. **Global GCU Listening process** deals with the (P2P like) interaction between GCUs. Normally this interaction takes place after a clear phase of negotiation with the GBU.

We assume, among others, the following variables shared by the four processes, via classical semaphores *à la* Dijkstra:

- `GBU` holds all the security and network informations of the leader of the colony;
- `GlobalMode` is *true* if and only if the `GCU` works in global mode;
- `RegMode` is *true* if and only if the `GCU` has been registered in a given colony. This variable controls the while loops in the global mode (*i.e.* while it is registered, the `GCU` must keep the dialog with the `GBU`).

Below we list, in a nutshell the key functions of the algorithm:

- `Discover(`*`MyCard`*`)` is devoted to discover the only `GBU`, denoted by `GBU`, which is the leader of the colony, where the `GCU` is going to connect;
- `ServiceReg(`*`MyCard`*`,GBU,LOGIN)` tries to register the `GCU` to `GBU` on the local colony he is trying to connect. The registration can fail depending of different parameters (like the fact that the `PKI` is not trustful, or that the `GCU` will offer insufficient resources to the colony, etc.); this function will set `RegMode` to `true`;
- `ServiceReg(`*`MyCard`*`,GBU,LOGOUT)` unregisters the `GCU` to *`MyGRU`*, leader of the local colony he is actually connected; the `GCU` will now work in local standalone mode; this function will set `RegMode` to `false`;
- `ListenLocal()` awaits a request coming from local `API`;
- `LocalServe(Data)` executes the `Data` on the local machine. It can fail;
- `PackScenario(Data)` encodes the scenario request with the `Data` to be sent, in the payload part of the `GIP` protocol, within the service request;
- `ServiceRequest(`*`MyCard`*`,GBU,MetaData)` sends a request of service to `GBU`;
- `LocalReply(Response)` forwards locally `Response`;
- `ListenGBU()` awaits a request coming from `GBU`;
- `CanHelp(MetaData)` analyzes if the request can be served;
- `ServiceResponse(`*`MyCard`*`,GBU,COMMAND)` responds positively/negatively to the `GBU` concerning the requested service;
- `CanServe(MetaData)` analyzes if the request can be served;
- `GetPeers(MetaData)` gets the peers that `GBU` found in his colony;
- `GlobalServe(`*`MyCard`*`,Peers,Data)` forwards the request to the peers that the `GBU` found in his colony. The request will be processed remotely;
- `CanJoin(MetaData)` checks if the `GCU` can join the colony;

- `CanLeave(MetaData)` checks if the GCU can leave the colony;
- `ListenGCU()` awaits a request coming from GCU;
- `Verify(MetaData)` verifies if the request is well formed. Many security checks can be performed by this routine (for example it verifies the PKI of the GCU, or it checks if the demanded service was previously asked by GBU, etc.);
- `UnPackScenario(MetaData)` decodes the scenario request from the `Data` received in the payload part of the GIP protocol, within the service request;
- `SendResult(`*`MyCard`*`,GBU,Response)` sends the results of the request to the requesting GCU;

```
while true do
 inparallel
  GIPacket = ListenLocal()                                  // Local listening
  Route(MyCard ,MyPeerCard ,GIPacket)
 with
  GIPacket = ListenGlobal()                                 // Global listening
  if GIPacket.TTL != 0
  then GIPacket.TTL --
       Deliver(MyCard ,MyPeerCard ,GIPacket)
  endif
 endinparallel
endwhile
```

Figure 4: GRU pseudocode

3.3 GRU's Semantics

The semantics of the GRU is described in the pseudo-code in Figure 4. Let *`MyPeerCard`* denotes the name of the GCU (resp. GBU) which is uniquely attached, via a suitable API to the GRU, denoted by *`MyCard`*. This unit is the only units that *de facto* understands the GIP protocol; it will deals with resource discovery (function `Discover()` of the GCU (resp. GBU). To simplify the pseudo code, all details concerning resource discovery will be treated in Subsection B.1.

The new `TTL` slot in a GIP packet will be used to *count* the maximum number of *hops* from one unit to another: this value is useful to limit the number of request forwarded from one GBU to another one. This field help the GRU to discard some packets (typically service request) that "surfs" the ArigatoNet looking for some charitable GCU that could help him. Below we list, in a nutshell some key functions of the algorithm:

- `ListenLocal()` awaits a request coming from the local API;

- `ListenGlobal()` awaits a request coming from ArigatoNet;
- `Route(`*`MyCard`*`,`*`MyPeerCard`*`,GIPacket)` routes a GIP packet to its destination (defined in the `GIPacket`);
- `Deliver(`*`MyCard`*`,`*`MyPeerCard`*`,GIPacket)` unpacks and delivers a GIP packet to the peer (GCU or GBU) to which the GRU is uniquely attached.

3.4 GBU's Semantics

The semantics of the GBU is described in the pseudo-code in Figure 5. It is composed by five processes, running in parallel whose intuitive behavior is as follows:

1. **Registering/Unregistering process** implements the (un)registration of a GBU to a leader-GBU of a given supercolony;
2. **Basic shell process** is the classical read/eval/print loop. The GBU itself can work in local standalone mode (*i.e.* it do not forward any requests to other brokers), or in global mode (any request that cannot be completely served intra-colony is forwarded to the leader-GBUof the supercolony);
3. **Spool process** spool an associative list compose by an unique identifier of a service request and a list of potential GCUs that have accepted to serve the task associated with the identifier;
4. **Intra-colony Listening process** listens for any communication (service request or service response) from the local colony;
5. **Inter-colony Listening process** deals with the interaction between the leader-GBU of the colony and the superleader-GBUof the supercolony where the colony is registered: normally this interaction takes place after a clear phase of negotiation between both leaders of colonies.

We assume, among others, the following variables shared by the five processes, via classical semaphores *à la* Dijkstra:

- `Colony` is the set of *individuals* belonging to the colony;
- `Peers4Id` is a dictionary of the shape `[(Id,Peers)]*` denoting, for each service request labeled by `Id`, the list of potential `Peers` that have accepted to serve `Id`;
- `History` is a dictionary of the shape `[(Id,MetaData)]*`, where `MetaData` contains all the informations about the kind of request (GCU, GRU, PKI, etc.).
- `GlobalMode` is *true* if and only if the GCU works in global mode; is *false* otherwise;

```
inparallel
while true do                    // Registration loop
 GBU = Discover(MyCard)
 case (GlobalMode,RegMode) is
  (true,false):
   ServiceReg(MyCard,GBU,LOGIN)
  (false,true):
   ServiceReg(MyCard,GBU,LOGOUT)
  otherwise:                     // Do nothing
 endcase
endwhile
with
 while true do                   // Shell loop
  Data     = ListenLocal()
  Response = LocalServe(Data)
  case (Response,GlobalMode,RegMode) is
   (login,_,_):                  // Open global mode
    GlobalMode = true
   (logout,_,_):                 // Close global mode
    GlobalMode = false
   (fail,true,true):             //Ask for you
    MetaData = PackScenario(Data)
    ServiceRequest(MyCard,MyCard,MetaData)
   otherwise: LocalReply(Response)
  endcase
 endwhile
with
 while true do                   // Intra-colony listening
  MetaData = ListenPeer()
  PushHistory(MetaData)
  case MetaData.OPE is
   SREG: //A GCU is asking for (un)registration
    Update(Colony,MetaData)
   SREQ:  // A GCU is asking for some request
    SubColony = SelectPeers(Colony,MetaData)
    if SubColony == {}           // Broadcast inter
    then
      ServiceRequest(MyCard,GBU,MetaData)
    endif
    foreach Peer in SubColony do //B. intra
     ServiceRequest(MyCard,Peer,MetaData)
    endforeach
   SRESP:       // A GCU responds to a request
     Sort&PushPeers4Id(MetaData)
  endcase
 endwhile
with
 while true do                   // Spooling Peers4Id
  foreach (Id,Peers) in Peers4Id do
   if   Timeout(Id)
   then ServiceResponse(MyCard,{},NOTIME)
   else if Satisfy(Peers,History(Id))
        then
         ServiceResponse(MyCard,
                         GetBestPeers4Id(Id),DONE)
        endif
   endif
   PopPeers4Id(Id)
  endforeach
 endwhile
with
 while RegMode do                // Inter-colony listening
  MetaData = ListenGBU()
  PushHistory(MetaData)
  case MetaData.OPE is
   SREG:                         // Registration inter GBU
    case MetaData.ROLE is
     LEADER://A GBU register in a leader GBU
     if   CanJoin(MetaData)
     then RegMode = true
     endif
     if   CanLeave(MetaData)
     then RegMode = false
     endif
     INHABITANT:       // A GBU (un)register
      Update(Colony,MetaData)
   SREQ:
     ... as for SREQ intra-colony
   SRESP:// A leader GBU responds to a request
    Sort&PushPeers4Id(MetaData)
     endcase
  endcase
 endwhile
endinparallel
```

Figure 5: GBU pseudocode

- `RegMode` is *true* if and only if the `GCU` has been registered in a given colony; it holds *false* otherwise; this variable control the while loops in the global mode (*i.e.* unless unregistered, the `GCU` must keep the dialog with the `GBU`;
- `GBU` holds all the security and network informations of the leader of the colony.

Below we list, in a nutshell some new functions of the algorithm:

- `Discover(MyCard)` discovers the leader-`GBU` unit, upon physical/logical insertion in the ArigatoNet network;
- `ListenPeer()` wait for a request coming from an individual of the colony (*i.e.* a `GCU`, or a `GBU` leader of a subcolony;
- `PushHistory(MetaData)` push the pair `(Id,MetaData)` on the `History` dictionary (`Id` is contained in `MetaData` as well);
- `SelectPeers(Colony,MetaData)` perform a *static analysis* about the possibility to fully satisfy the service request *inside* the local colony, *i.e.* without forwarding the request out of the colony; if the function returns `{}` then the request *a priori* cannot be satisfied internally;
- `Sort&PushPeers4Id(MetaData)` inserts and sort in the list of peers identified by `Peers4Id(GetId(MetaData))` the new peers, calculated by the function `GetPeers(MetaData)`: sorting is done following *ad hoc* criteria w.r.t. the resources requested for a given scenario;
- `Update(Population,MetaData)` log and delog one `GCU` (resp. `GBU`), whose coordinates are contained in `MetaData`, from the colony (denoted by `Population`); the criteria of logging/delogging can be arbitrarily complex, depending on which security policy the colony has adopted;
- `Timeout(Id)` is true when a service request, labeled with a given `Id`, oversize a fixed time of waiting;
- `Satisfy(Peers,History(Id))` checks, for a given service request `Id` in the `History` dictionary, the `Peers` capabilities;
- `GetBestPeers4Id(Id)` selects the "best" peers for the request with `Id` key, from a list of potential peers: the selection criteria depends, among others, on the peculiar scenario we are dealing with;
- `PopPeers4Id(ID)` pops the pair `(ID,PEERS)` in the `Peers4Id` dictionary;
- `CanJoin(MetaData)` checks if the `GBU` can join the colony; this function also verifies that the registration does not induce *cycles* in the colony the `GBU` he is trying to join.

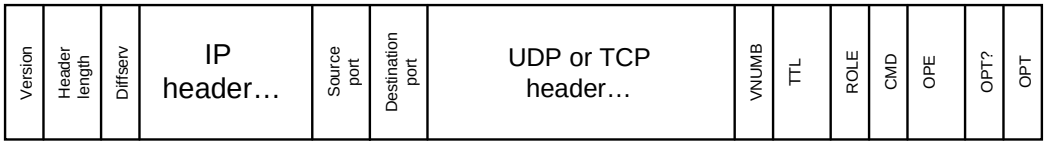

Figure 6: A GIP packet on UDP or TCP

4 The GIP Protocol

This section deals with the packet description of the GIP protocol. For obvious lack of space, many details are left implicit. As shown in the Figure 6, the GIP packet resides in the payload of a UDP datagram, or eventually of a TCP packet. We assume the following common datatypes, like *BYTE*, *INT*, *BOOL*, *SET*, etc. plus the *Variable-Length* (recursive)-type *VLT* defined as follows:

Definition 4.1 (*VLT* Type) *Any element of type VLT has the following fields:*

- LENGTH *: INT is the length of the* Payload *field in bytes;*
- PAYLOAD *: VLT contains the data to be interpreted.*

4.1 GIP s Fields Description

The fields of the GIP protocol are:

- VNUMB : *INT* is the version number of the GIP protocol. Currently the version is 1;
- TTL : *INT* is the *time to live* of the packet protocol. This value is used to avoid that packets "lives" too much in the ArigatoNet jumping from one hop to another hop;
- ROLE : *BOOL* indicates the role of the sender of the packet, either a LEADER or INHABITANT;
- CMD : *2BYTE* is the command carried by the packed; it is composed by the two subfields SERVICE : *BYTE* and VALUE : *BYTE*;
- OPE : *VLT* describes, for each command, a particular operation and its parameters;
- OPT? : *BOOL* indicates that options are present at the end of the GIP packet;
- OPT : *VLT* describes the optional fields.

For each command described in the CMD field, the OPE field contains, in its payload field, all data necessary to perform the command. For lack of space, we only describe the CMD and the OPE fields.

4.2 The CMD Field

Version 1 of the GIP allows for the three services, namely SREG, SREQ, and SRESP. For each service, a number of answers are possible.

- (SREG : *BYTE*,VALUE : *BYTE*), *Service Register*, is used either for the registration of a GCU in local mode to a GBU, or for the registration of one GBU (leader of a subcolony working in local mode) to a GCU leader of another colony that physically (or logically) contains the subcolony. Registration must be acknowledged by both units involved. Possible kinds of values are:
  - LOGIN applies when a GCU wants to register to a GBU, or when a GBU (representing a subcolony) wants to register to another GBU. No additional fields are used by this operation;
  - LOGOUT applies when a GCU wants to unregister to a GBU (*i.e.* switch in local mode) or when a GBU (representing a subcolony) wants to unregister to another GBU. No additional fields are used by this operation;
  - LOGGED applies when a GBU notifies a successful registration to an individual. No additional fields are used by this operation;
  - UNLOGGED applies when a GBU notifies a failed registration to an individual. No additional fields are used by this operation.
- (SREQ : *BYTE*,VALUE : *BYTE*), *Service Request*, is initially sent by a GCU, working in global mode, to request a service to its GBU. A GBU working in global mode, can forward this request to another GBU, leader of a another colony, in case it did not find in its own colony all the needed resources to serve the request. A GBU also can sends this request to every registered inhabitant of his colony, namely GCUs or GBUs leader of some subcolonies. Every bit of the VALUE represents any possible distributed resource that can be asked, *i.e.*:
  - (bit 0) CPU applies when we ask for computational power (*e.g.* mips);
  - (bit 1) MEM applies when we ask for memory space;
  - (bit 2) DATA applies when we ask for some (distributed) files;
  - (bit 3) BAND applies when we ask for some bandwidth (the GCU is usually an ISP);
  - (bit 4) WEB applies when we ask for web services;
  - (bit 5) RUN applies when we ask to abort a run, pack everything (complete dump of the registers, stack, etc.) in a closure and migrate somewhere the computation;
  - (bit 6) left for future use;
  - (bit 7) parity bit.

  Of course a combination of different requests can be done, like the following one:

```
CPU MEM DATA BAND WEB RUN --- PAR
 1   1   1    1    0   0   0   0    Ask for CPU, Memory, Data, and Bandwidth
```

- (SRESP : *BYTE*,VALUE : *BYTE*), *Service Response*, is sent by a GCU to a GBU, to answer a received SREQ. Is is also exchanged between two GBUs or from a GBU to a GCU, following the reverse path of the SREQ. It indicates whether or not the individual may process the request of the leader GBU. A service response is also exchanged between two GCU when one servant GCU processes the request of a client GCU, *i.e.*:
  - To acknowledge the reception of the request;
  - To inform the client that it has to wait since the request is still processing on the servant;
  - To send the result or informations on how to retrieves the result to the client.

  Possible kinds of values are:
  - ACC applies when the request is accepted. Sent by a GBU to the individual which transmitted the request;
  - REJ applies when the request cannot be processed. Sent by a GBU to the individual which transmitted the request;
  - DONE applies when the request has been processed. Sent by an individual to the GBU, leader of the colony;
  - ERR applies when the request has been processed, but some errors occurs (*i.e.* a `core dump` in a run). Sent by an individual to the GBU, leader of the colony;
  - SPOOF applies when the request cannot be processed, because of some problems, *e.g.* in the authentication, refused PKI certificates, etc. Sent by an individual to the GBU, leader of the colony;
  - NOTIME applies when the request has expired it timeframe. Sent by the GBU to individuals of its colony;
  - RES applies when the request is processed and the result is going to be transmitted. Sent by an individual to the GBU, leader of the colony.

4.3 The OPE Field

According to the CMD field, the OPE field of type *VLT* is used to encode in its payload part all the information necessary to execute the command. We briefly review some of this payloads for command SREQ and SRESP:

- For an SREQ command:

  - ID :*4BYTE* contains the Unique ID identifying the request carried by this command. This field is created by the original individuals which emitted the request and is left unchanged by all the nodes forwarding the request;
  - CARD: *VLT* contains all the informations necessary for the exchange between the client and a servant (*i.e.* Protocol, IP Address, Port number, PKI, etc.);
  - REQNUMB:*INT* indicates the number of request units follow in the packet. This number *must not* be equal to zero;
  - REQDATA: *VLT**[4] describes all informations necessary to deal with a simple request.

- For an SRESP command:
  - ID :*4BYTE* contains the Unique ID identifying the request carried by this command;
  - CARD: *VLT* contains all the informations necessary for the exchange between the client and a servant (*i.e.* Protocol, IP Address, Port number, PKI, etc.);
  - RET: *VLT* contains the result of the request.

5 Conclusions, and Future Work

5.1 Conclusion

We presented Arigatoni, a light-weight formal model and a communication network called ArigatoNet that is suitable to deploy the Global Computing Paradigm over the Internet. We defined a simple but very efficient communication protocol, called Global Internet Protocol, GIP on top of TCP or UDP protocol [12].

Basic global computers and colonies of global computers can communicate by first registering to a brokering service and then by mutually asking for, or offering services. In the model, resources are encapsulated in the colony in which they reside, and requests for resources located in another colony traverse a broker-2-broker negotiation using a P2P overlay network. The model is suitable to fit with various global scenarios from classical P2P applications, like file sharing, or band-sharing, to more sophisticated GRID applications, like remote and distributed big (and small) computations.

Undergoing work includes the development of a first prototype, the implementation on an experimental platform and the use in real scenarios like the ones described within this paper.

Future works will be mainly focused on security issues as for example using many PKI instead of a unique PKI, the study of trust models based on reputation and more advanced security models and techniques.

[4] List of *VLT*.

Acknowledgment

The authors ack Aeolus FP6-2004-IST-FET Proactive.

References

[1] H. Kishimoto and T. Maguire, "Open Grid Services Architecture Working Group," https://forge.gridforum.org/projects/ogsa-wg.

[2] Globus Alliance, "Globus Home Page," http://www.globus.org/.

[3] V. Sassone, "Global Computing II: A New FET Program for FP6," Talk, Bruxelles, 4/6/04, http://www.cogs.susx.ac.uk/users/vs/research/paps/gc2InfoDayPres.pdf.

[4] A. Iamnitchi, I. T. Foster, and D. Nurmi, "A Peer-to-Peer Approach to Resource Location in Grid Environments," in *Proc. of High Performance Distributed Computing, HPDC*, 2002, p. 419, full version in http://www.cs.uchicago.edu/files/tr_authentic/TR-2002-06.pdf.

[5] V. Iyengar, S. Tilak, M. J. Lewis, and N. B. Abu-Ghazaleh, "Non-Uniform Information Dissemination for Dynamic Grid Resource Discovery," in *Proc. of Network Computing and Applications, NCA*. IEEE, 2004.

[6] F. Heine, M. Hovestadt, and O. Kao, "Towards Ontology-Driven P2P Grid Resource Discovery," in *Proc. of International Workshop on Grid Computing, GRID*. IEEE/ACM, 2004, pp. 76–83.

[7] M. Hauswirth and R. Schmidt, "An Overlay Network for Resource Discovery in Grids," in *Proc. of International Workshop on Database and Expert Systems Applications, DEXA*, 2005, pp. 343–348.

[8] Planet Lab Consortium, "Planet Lab Home Page," http://www.planet-lab.org/.

[9] L. Cardelli, "A Language with Distributed Scope," *Computing Systems*, vol. 8, no. 1, pp. 27–59, 1995, also in Proc. of POPL, 1995.

[10] R. Milner, *A Calculus of Communicating Systems*. Springer, 1980.

[11] C. A. R. Hoare, *Communicating Sequential Processes*. Prentice-Hall, 1985.

[12] D. Benza, M. Cosnard, L. Liquori, and M. Vesin, "Arigatoni: Overlaying Internet via Low Level Network Protocols," INRIA, Tech. Rep., November 2005.

[13] ——, "Arigatoni: 6 possible Scenarios," 2005, manuscript.

[14] A. Gulbrandsen, P. Vixie, and L. Esibov, "Rfc2782, a dns rr for specifying the location of services (dns srv)," IETF, Tech. Rep., 2000.

[15] S. Bradner and V. Paxson, "Iana allocation guidelines for values in the internet protocol and related headers," IETF, Tech. Rep., 2000.

[16] S. Alexander and R. Droms, "Rfc2132, dhcp options and bootp vendor extensions," IETF, Tech. Rep., 1997.

[17] D. Meyer, "Rfc2365, administratively scoped ip multicast," IETF, Tech. Rep., 1998.

[18] J. Linn, "Rfc 2743 - generic security service application program interface version 2, update 1," IETF, Tech. Rep., 2000.

[19] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, "Rfc3489, stun - simple traversal of user datagram protocol (udp) through network address translators (nats)," IETF, Tech. Rep., 2003.

A Examples

The Arigatoni model is, by construction, independent from any given scenario. We could envisage at least the following scenarios to be completely full-fitted in our model (list not exhaustive):

1. Ask for computational power (*i.e.* the GRID);
2. Ask for memory space;
3. Ask for bandwidth (*i.e.* VoIP);
4. Ask for file retrieving (*i.e.* P2P);
5. Ask for web service (*i.e.* Google);
6. Ask for a computation migration (*i.e.* ask to transfer one non completed local run in another GCU saving the partial results, under the case of a catastrophic scenario, like, *e.g.* fire, terrorist attack, earthquake etc, *e.g.* truly mobile ubiquitous computations);
7. Ask for a *Human Computer Interaction*;
8. ...

These scenarios will be described in a companion paper [13]. In what follows, we will put Arigatoni@work in a GRID arena.

A.1 A GRID scenario for Seismic Monitoring

John, chief engineer of the SeismicDataCorp Company, Taiwan, on board of the seismic data collector ship, has to decide on the next data collect campaign. For this he would like to process the 100 TeraBytes of seismic data that have been recorded on the data mass recorder located in the offshore data repository of the company to be processed and then analyzed.

He has written the processing program for modeling and visualizing the seismic cube using some *parallel library* like *e.g.* `MPI/PVM`: his program can be distributed over different machines that will compute a chunk of the whole calculus;

However, the amount of computation is so big that a supercomputer and a cluster of PC has to be *rented* by the SeismicDataCorp company. John will ask also for *bandwidth* in order to get rid of any bottleneck related to the big amount of data to be transferred.

Aftermath, the processed data should be analyzed using a *Virtual Reality Center,* VRC based in Houston, U.S.A. by a specialist team and the resulting recommendations for the next data collect campaign have to be sent to John.

Hence he would like the following scenario to happen:

- John logs on the Arigatoni overlay network in a given colony in Taiwan, and sends a quite complicated service request in order for the data to be processed using his own code. Usually the GBU leader of the colony will receive and process the request;
- If the resource discovery performed by the GBU succeeds, *i.e.* a supercomputer and a cluster and an ISP are found, then the data are transferred at a very high speed and processed;
- John will ask also to the GCU containing the seismic data to dispatch suitable chunks of data to the supercomputer and the cluser designated by the GBU to perform some pieces of computation;
- John will ask also to the global supercomputer unit the task of collecting all intermediate results so calculating the final result of the computation (*i.e.* it will play the role of *Maestro di Orchestra*);
- The processed data are then sent from the supercomputer, via the high speed ISP to the Houston center for being visualized and analyzed;
- Finally, the specialist team's recommendations have to be sent to John's laptop.

This scenario is pictorially presented in Figure 7 (we suppose a number of subcolonies with related leaders GBU, all registered as individuals to a superleader-GBU (for example the John's GBU could be elected as the superleader). All GBU's are trusted[5], making *de facto* in common all resources of their colonies.

[5] As a simpler approximation *à la* Globus, all GBU s share the same PKI.

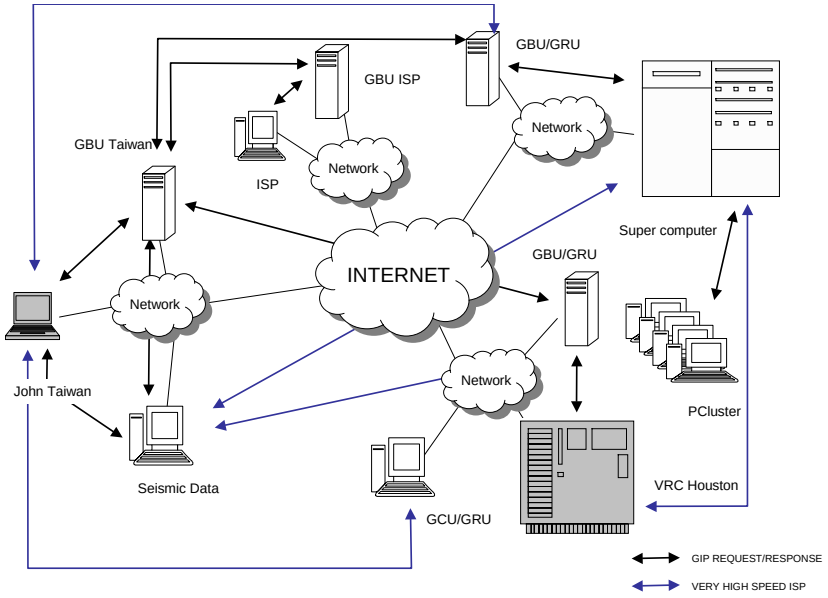

Figure 7: A GRID Scenario for Seismic Monitoring

B Resource Discovery, Security and Social Issues

B.1 Resource Discovery

In Arigatoni resource discovery takes place in two different contexts:

- for a GCU to discover a GBU, upon physical insertion in a GCUs colony;
- for a GBU to discover other *friend* GBU, upon physical insertion in the ArigatoNet network.

Discovery is delegated by a node to its proprietary GRU. The GRU comes with a set of network endpoints[6] that are entries to other instances of the GRU on the Internet.

When a node is first connected to the network, then it needs the discovery process to complete successfully before it can send a request. Successful completion of discovery is preliminary for a GRU instance to be fully functional, and provide service to the units (GCU or GBU) it serves.

Several discovery techniques are described hereafter. All these techniques should not be supported by all the implementations and the Arigatoni model does not recommend any of

[6]Omitted in the GRU pseudocode for the sake of simplicity. A network endpoint is defined by a protocol type and an address. For example: the IPv4 protocol, the IP address, the UDP protocol, an UDP port.

them in particular. Of course, the implementation of all these methods in a node increases its chances to find peers.

Techniques for a GRU proper of a GCU to find the GRU of the GBU leader of the colony (resp. for a GRU proper of a GBU to find other GRU of the GBU leader of other colony) includes:

- Reading File `/etc/ghosts`[7]. A device may use a static pre-configured list of potential peers. This list may be stored in a file. In this case, finding the peers is just reading the file. Protocol type and port can either be well-known registered values or be specified in the this file;

- Using the DNS service. The DNS SRV Resource Record (RR) is used for specifying the location of services. It is specified in [14] and allows a client to retrieve information for a service (including IP address and protocol port), on a per-DNS-domain basis. Service is defined by "service identifier", protocol and DNS domain. As recommended in [15], a service identifier would once be registered for Arigatoni peers at the IANA[8]

  This technique could even improve the discovery algorithm for discovering close colonies by the GBU. For example, in addition to current domain, several domains may be parsed for SRV RR resource. These domains could be chosen on various criteria to build an efficient GBU mesh. For example, criteria may be "proximity" (search SRV RR from upper-level domain in DNS; use network statistics to determine top partner networks) or "distance" (widen the scope of peers by searching SRV RR from an unrelated DNS-domain)

- Using the DHCP protocol. The DHCP protocol provides a framework for passing configuration information to devices, as specified in [16]. Length and format of those DHCP options are option-specific. In this case:

  1. a DHCP option type for Arigatoni peers would once be specified and registered at IANA, and
  2. DHCP administrators would reference one or more GBU in their DHCP server configuration.

  When connecting to a network and acquiring an IP address via DHCP, a device would then learn one or more potential peer IP address through the Arigatoni DHCP option. Protocol type and port would be Arigatoni protocol well-known registered values;

- Using *Caching* techniques. A GRU may cache peer location information found in previous runs, and read the cache in subsequent run;

- Using *Multicast* and *Broadcast* techniques. The use of multicast (either administratively scoped multicast as specified in [17] or "TTL-scoped" multicast) is also an option,

[7] By analogy with the Unix `/etc/hosts` file.
[8] Internet Assigned Numbers Authority.

and the use of broadcast too. Both have implications on the Arigatoni protocols that have not been measured yet: discovery is active, one-to-many discovery.

B.2 Security Issues

B.2.1 Trust through Public Key Infrastructure

In order to work securely, the Arigatoni model needs to be able to offer the following guarantees to its components:

- The communication between two nodes must be secured;
- The "role" played by a node (*i.e.* client GCU, servant GCU or GBU) must be certified by a third party trusted by the nodes which have to communicate with this particular node.

A way to implement those constraints is to use a PKI. A *Certification Authority* delivers certificates, and couples of keys[9] for GCUs and GBUs which attest of their distinctive roles. The whole mechanisms involved by a PKI are not described here, but good use of the PKI model and an implementation compliant with [18] can provide all the necessary security:

- Trustfulness on the identity of the peers;
- Trustfulness of all the transmitted data, *i.e.* secrecy, authenticity, and integrity.

B.2.2 Extending the trust model

In addition to PKI a more "liquid" trust model could be built, based on *reputation*. Reputation represents the amount of trust an entity in the model has in another entity based on its partial view of ArigatoNet:

- Each node maintains a reputation score for each node it knows;
- Each node maintains a reputation score for each resource it serves;
- Exchanges between nodes update dynamically each others scores.

The computation of the reputation score and the way nodes exchange scores are beyond the scope of this document but could be precised in the future.

[9] One key is private and is kept by the node, the other is public and is communicated to all the correspondants.

B.2.3 Firewalls and NAT traversal

- *Firewalls* are equipments typically placed on the border side of a campus or enterprise network which control all the incoming connections. Unknown UDP flows are always considered as suspects by those equipments and dropped;
- NAT is a technique applied by a router to mask to the outside world the real IP Addresses of inside hosts. It consists in modifying each packet that goes through the NAT equipment. The NAT router changes the IP source address when a packet *goes to* outside, it changes the IP destination address when a packet *come from* outside.

The usage of these mechanisms is very frequent on the Internet and they are barriers that can prevent connections between *inside* and *outside* nodes. The implementation of [19] could be used to thwart this problem.

B.3 The Social Model underneath Arigatoni

The Arigatoni model defines mechanisms for devices to interoperate, by offering services, in a way that is reminiscent to Rapoport's *tit-for-tat* strategy of co-operation based on reciprocity. This way to understand common behavior of virtual organization has some theoretical basis on Game Theory. Classical results from Game Theory are based on the assumption that a basic shared currency connectivity (*i.e.*, different resources as CPU, Memory, Bandwidth, Data, etc.) is available and then the task is to design truthful mechanisms where users have an incentive to collaborate.

For simplicity's sake, we described the Arigatoni model with each GCU registering to one unique GBU. This GBU is the leader of a tight *colony*. But the Arigatoni model can be scaled up to a more general model where each GCU may simultaneously be registered to several GBU, and where a *colony* is just one possible *social scheme*.

This means that Arigatoni fits with motivations and cooperation behavior of different communities using ArigatoNet. It tries to be *policy neutral*, leaving policy choices for each node at the implementation or configuration level, or at the community or organization level. Policy domains can overlap (one node can define itself as belonging "much" to colony *foo* and "a little bit" to colony *bar*). This denotes a decentralized non-exclusive policy model.

One question now arise: who is Arigatoni designed for? We believe the model is flexible enough to serve a mix of different social structures:

- Independent end-user connecting through his ISP or migrating from hot-spot to hot-spot;
- Cooperative communities of disseminated people;
- More regulated or hierarchical communities (maybe a better picture of the corporate network);
- Cooperative or competitive resource providers.

The Arigatoni model can be extended to support various trust models, such as the extended trust model proposed in Subsection B.2.2 Moreover, reputation score could be expanded to a multi-dimensional value, for example adding a score for quality of the service offered by a node. However, Arigatoni encourages cooperation and enables gratuitous resource offering. But it may also suit for business extensions:

- A servant can sell resource usage, creating a resource business;
- A GBU can sell research service, creating a brokering business ("I point you to the best resources, more quickly than anyone else").

Among others, some Arigatoni extensions may define:

- How to create and call third party services for on-line payment of services;
- How to exchange digital cash for payment of services;
- How to negotiate service conditions between client and servant, including price and quality of service.

The one-to-many nature of the SREQ GIP protocol request are of particular interest in this case. An Arigatoni extension may define how to join a third party auction server. Candidate servants for a SREQ would contact the auction server and make their bid. The trusted auction server chooses the elected candidate and service conditions based on auction terms. The client would then contact the auction server and get this information.

Those extensions may take advantage of the GIP options field, for example to transmit location and parameter information to call a third party system.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399