Weaving Arigatoni with a graph topology

Michel Cosnard, Luigi Liquori

Weaving Arigatoni with a graph topology

Michel Cosnard and Luigi Liquori

Abstract — **Arigatoni is a structured multi-layer overlay network that provides various services with variable guarantees, and promotes an intermittent participation in the overlay because peers can appear, disappear, and organize themselves dynamically. Arigatoni provides a very powerful decentralized, asynchronous, and scalable *resource discovery mechanism* within an overlay with a *dynamic network topology*. In the first version of Arigatoni, the network topology was tree- or forest-based. This paper makes a significant step by weaving the network topology with general *dynamic graph* properties. As an immediate consequence, the Arigatoni protocols must be reconsidered in order to take into accounts *routing loops* when updating *routing tables*, for dealing with *resource overbooking*, and *resource discovery loops*.**

Index Terms — Overlay networks, resource discovery, virtual organizations, dynamic graphs, peer-to-peer, global computing, grid computing.

I. Introduction

THE explosive growth of the Internet gives rise to the possibility of designing large overlay networks and virtual organizations consisting of Internet-connected global computers, able to provide a rich functionality of services that makes use of aggregated computational power, storage, information resources, etc. Arigatoni [1] is a structured multi-layer overlay network which provides resource discovery with variable guarantees in a virtual organization where peers can appear, disappear and organize themselves dynamically. In a nutshell, the main units in Arigatoni are:

- A *Global Computer Unit, GC*, i.e. the basic peer of the global computing paradigm; it is typically a small device, like a PDA, a laptop, or a PC, connected through IP in various ways (wired, wireless, etc.).
- A *Global Broker Unit, GB*, i.e. the basic unit devoted to subscribe and unsubscribe GCs, to receive service queries from GC-clients, to contact potential GC-servers, to negotiate with them services, to authenticate clients and servers, and to send all the information necessary to allow the client and the servers to communicate. Every GB controls a colony of collaborating GCs. Hence, communication intra-colony is initiated via only one GB, while communication inter-colonies is initiated through a chain of GB-2-GB message exchanges whose security is guaranteed via PKI mechanisms. In both cases, when a client GC receives an acknowledgment of a service request from the direct leader GB, then the GC is served directly by the server(s) GC, i.e. without a further mediation of the GB, in a pure peer-to-peer fashion. Registrations and requests are performed via a simple query language *à la* SQL and a simple orchestration language *à la* LINDA or BPEL.
- A *Global Router Unit, GR*, i.e. the basic unit devoted to send and receive packets, using the *Arigatoni resource discovery protocol* [2,3], and to forward the "payload" to the units which are connected with this router. The connection GB-GR-GC is ensured via a suitable API.
- A *Colony* is a simple virtual organization composed of exactly one leader GB and a set (possibly empty) of individuals. Individuals are GCs, or sub-colonies. The two main characteristics of a colony are:
  1. a colony has exactly one leader GB and at least one individual (the GB itself);
  2. a colony contains individuals (GC's, or other sub-colonies).

The main challenges in Arigatoni lie in the management of an overlay network with a dynamic topology, the routing of queries, and the discovery of resources in the overlay. In particular, resource discovery is a non-trivial problem for large distributed systems featuring a discontinuous amount of resources offered by global computers and an intermittent participation in the overlay.

Thus, Arigatoni features two protocols: the *virtual intermittent protocols (VIP)*, and the *resource discovery protocol (RDP)*. The VIP protocol deals with the dynamic topology of the overlay, by allowing individuals to login/logout to/from a colony. This implies that the routing process may lead to *failures*, because some individuals have logged out, or are temporarily unavailable, or because they have been logged out by the broker, because of their poor performance or greediness.

The total decoupling between GCs in space (GCs do not know each other), time (GCs do not participate in the interaction at the same time), and synchronization (GCs can issue service requests and do something else, or may be doing something else when asked for services) is a major feature of the Arigatoni overlay network. Another important property is the *encapsulation* of resources in colonies. Those properties play a major role in the scalability of Arigatoni's RDP.

Manuscript received 20s Mai 2007. This work was supported in part by FP6 FET Global Computing: *"Algorithmic Principles for Building Efficient Overlay Computers"* and by a *"Colors"* INRIA Sophia Antipolis grant.

Luigi Liquori is with INRIA Sophia Antipolis (corresponding author: e-mail: Luigi.Liquori@inria.fr).

Michel Cosnard is with INRIA Rocquencourt Headquarters (e-mail: Michel.Cosnard@inria.fr).

```
While true do
MetaData = Listen()
case MetaData.OPE is
 SREQ(rid,[S_i]^{i=1..n}) from CardB
  if loop(rid,ReqList)
  then SRESP(rid,MyCard,REJ) to CardB
  else [S_i]^{i=1..k} = match([S_i]^{i=1..n},Services)
       SRESP(rid,MyCard,[S_i]^{i=1..k}) to CardB
 SRESP(rid,X,Y) from CardB
  case Y of
    REJ: no P2P action possible
      _: start a P2P action with X offering Y
  endcase
endcase
```

Fig. 1. RDP V4: Main loop for global computer

II. Resource Discovery Protocol V4

In what follows, and according to [1], `MyCard`, `CardB`, `CardC`, and `CardS`, are the *identification triples* `(IP,PORT,PKI)` of the current unit/broker/client/server, respectively. Let `rid` be, for each kind of messages, its unique request identifier.

A. Main loop for global computer

The pseudocode is shown in Figure 1. A synthetic explanation follows:

- `MetaData=Listen()`: wait for a RDP-packet from one of its direct global broker (a global computer can log to many brokers).
- `SREQ(rid,[S_i]^{i=1..n}) from CardB`: a service request, identified by its unique request identifier `rid`, comes from the direct broker `CardB`.
- `loop(rid,ReqList)`: loop detection in case a cycle in the registration list is detected: an `SRESP(MyCard,REJ)` `with rid to CardB` is sent back to the broker.
- `[S_i]^{i=1..k = match([S_i]^{i=1..n},Me.Services)`: the requested services are a (possibly empty) subset of the list of services (`k≤n`) declared by the global computer itself: a `SRESP(rid,MyCard,[S_i]^{i=1..k}) to CardB` is sent back to the broker.
- `SRESP(rid,X,Y) from CardB: Y` can be `REJ` or a list of services provided by a list `X` of servers. Eventually, the peer-to-peer negotiation can start.

B. Main loop for global broker

The pseudocode is shown in Figure 2. A synthetic explanation follows (the pseudocode in common with the one of Figure 1 is not commented):

- `SREQ(rid,[S_i]^{i=1..n}) from CardC`: a service request identified by its unique request identifier `rid` comes from a client `CardC`. The client can be either a global computer or a global broker.
- `ReqList ←(rid,(CardC,[S_i]^{i=1..n}))`: the associative list `ReqList` is updated with the binding of the request identified `rid` with the pair (client, list of services), denoted by `(CardC,[S_i]^{i=1..n}))`.
- `([S_i]^{i=1..k},CardS_i^{i=1..k})=match([S_i]^{i=1..n},RouteTabl`

```
While true do
MetaData = Listen()
case MetaData.OPE is
 SREQ(rid,[S_i]^{i=1..n}) from CardC
  if loop(rid,ReqList)
  then SRESP(rid,MyCard,REJ) to CardC
  else ReqList ←(rid,(CardC,[S_i]^{i=1..n}))
       ([S_i]^{i=1..k},CardS_i^{i=1..k})=
                 match([S_i]^{i=1..n},RouteTable)
       if empty([S_i]^{i=1..k} )
       then SREQ(rid,[S_i]^{i=1..n}) to CardB
       else
       SRESP(rid,CardS_i^{i=1..k},[S_i]^{i=1..k}) to CardC
 SRESP(rid,X,Y) from CardS
  (Card,_) = ReqList(rid)
  SRESP(rid,X,Y) to Card
Endcase
```

Fig. 2. RDP V4: Main loop for global broker

`e)`: the requested services are filtered against the routing table `RouteTable`, owned by the global broker. The result of this matching operation is a (possibly empty) subset of the list of services `[S_i]^{i=1..k}` (`k≤n`), declared in the routing table `RouteTable`, with their corresponding list of servers `CardS_i^{i=1..k}`.

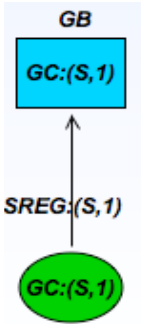
- `SREQ(rid,[S_i]^{i=1..n}) to CardB`: delegate the same request to one or of all the direct superbrokers (`CardB`) in case the result of filtering the routing table is empty.
- `SRESP(rid,CardS_i^{i=1..k},[S_i]^{i=1..k})to CardC`: respond to the client `CardC` that a list of servers `CardS_i^{i=1..k}` accepted to serve `[S_i]^{i=1..k}`.
- `SRESP(rid,X,Y) from CardS`: when a service response arrives from `CardS` (GC or GB), the response is "routed back" (`SRESP(rid,X,Y) to Card`) by recovering the sender `Card`, using the associative list `ReqList`.

III. Virtual Intermittent Protocol V2

This section presents the various kinds of possible topologies induces by service registrations in the Virtual Intermittent Protocol V2, and shows the pseudocode of the main loop for the global broker.

A. Monolog

Suppose a GC registers to one GB and declares its availability to offer one "unit" of a service S. This is the most common form of registration where every GC registers to only one GB. The resource tables are updated in the GB that receives the registration. Every service request for S, received by the GB, will be forwarded directly to the GC.

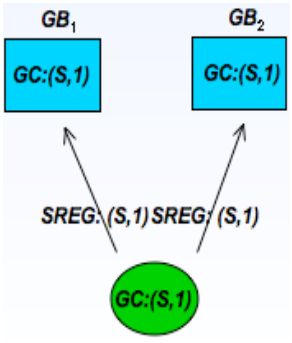


B. Dialog

Suppose a GC registers to two GBs, by declaring the same resource S twice: this situation can happen very frequently and has the drawback that the resource S offered by the GC is counted *twice* (a sort of *resource overbooking*). This pheno-

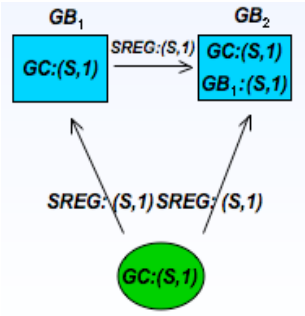
menon is well known in the telecommunications industry, such as in the "frame relay" world.

Overbooking in telecommunications means that a telephone company has sold access to too many customers which basically flood the telephone company lines, resulting in an inability for some customers to use what they purchased. Other examples of overbooking can be found in the domain of transportations and hotel reservations (sources from Wikipedia).



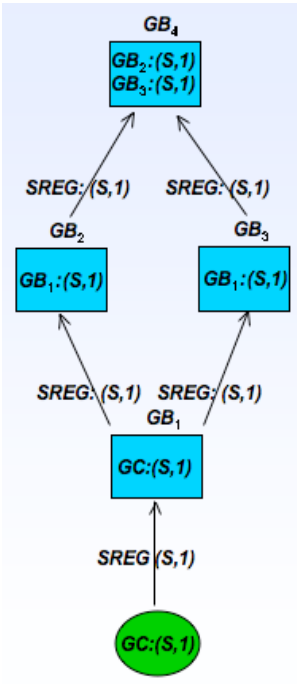
C. Pyramid

The pyramid topology is a dialog topology plus a service registration of one of the two GBs that registers to the other GB. As for the dialog topology, an overbooking of resource S is generated.



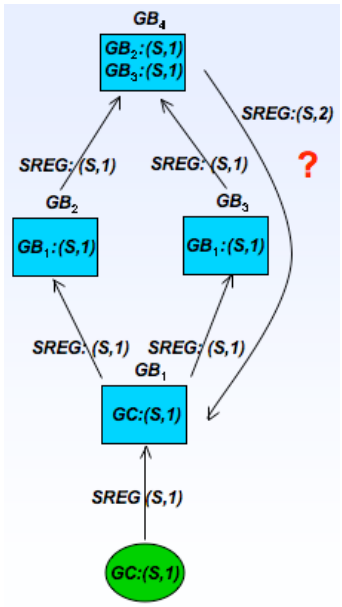
D. Diamond

The diamond topology is essentially a dialog topology plus two service registrations of brokers GB_2 and GB_3 to GB_4. This topology is quite similar to the one we can found in object-oriented languages with multiple inheritance. Also this topology has an overbooking phenomenon, since GB_4 has the unique resource S provided by GC in two items. While this topology presents the same overbooking problems as the previous one, it is quite important because it shows the "minimum number" of GBs to set up the, so called, *loops in routing tables update*, obtained when a tree becomes a graph because of a cycle in a broker service registration.



E. Cycle-diamond

A cycle-diamond is a diamond topology plus an attempt of a service registration by broker GB_4 to the GB_1's colony. As it is clearly shown in the picture on the right, this registration will generate a cycle in the GBs graph. Hence his graph is quite dangerous, and must be detected as soon as possible. The problem lies in updating the routing tables; accepting the registration of GB_4 to GB_1 colony will produce, via a simple service update, an *infinite multiplication* of the unique resource S, as the table can show. Since the routing tables in GBs only contain the local information of the direct colony members, cycles cannot be detected unless we put the global informations of all the members in all GBs. Unfortunately, this solution will cause the explosion of the GB routing tables, it breaks the encapsulation and the locality of the Arigatoni model, and makes the whole routing system *untractable* and *not scalable*. As such, we must either reject this registration, or at least conditionally accept it under condition of verifying, in a short time, that a loop is not created. Loop detection over dynamic registration will be the subject of the next section.



GB	2S	6S	18S	54S	162S	∞
GB	1S	3S	9S	27S	81S	∞
GB	1S	3S	9S	27S	81S	∞
GB	1S	3S	9S	27S	81S	∞

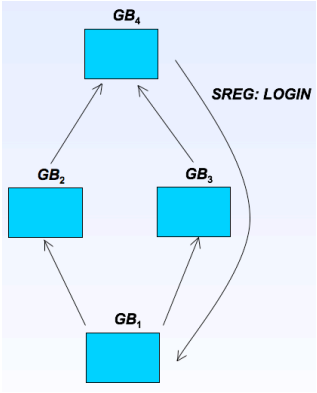
F. Loop detection

We propose a simple mechanism that allows to discover registration loops in presence of "non selfish" GBs and at the cost of a "fake" service update. The cost of this and of other possible solutions are discussed at the end of this subsection. In a nutshell, when GB_4 tries to log to GB_1, as depicted in the previous figure, we do as follows:

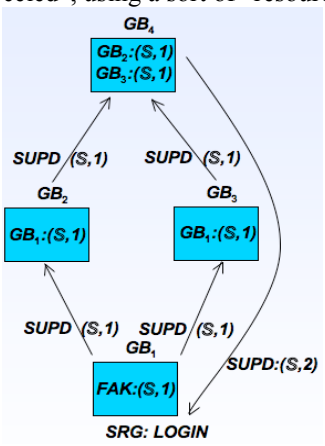
1. Accept a conditional registration of GB_4 to GB_1: the routing tables of GB_1 and of all its "Super-brokers" are not service updated;
2. Create a "fake-unique" resource S = *gen*(GB_4,GB_1), where *gen* is an `keygen`-like function depending on the two GBs involved in this registration;
3. Perform service updates in the routing tables with S;
4. Wait for a fixed time T;
5. If the fake S comes back as routing table update, then the registration is withdrawn, else the registration becomes permanent.

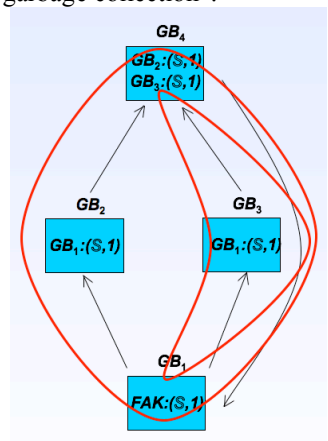
The process can be represented by the following figures.

Figure on the right shows the attempt of GB_4 to login to GB_1. Since neither GB_1 nor GB_4 know each other (the respective routing tables register only neighbors of distance 1 in the broker's graph), the cycle can be detected only dynamically, using the feature of the VIP protocol that updates their respective routing tables, according to new registrations.



As said before, when a conditional registration is accepted, it does not imply an immediate update of the routing tables. A "fake" unique resource is created. The next figure on the left shows the propagation, via successives VIP service updates, of the fake resource S in the overlay, until an update of the fake S *comes back* to GB_1. The cardinality of the units of the fake S (2) indicates the *number* of detected cycles, as shown in the same figure on the right. At this point, the two cycles are dynamically detected, the conditional registration is canceled, and either the fake resource S is canceled (via a VIP service update), or it is left in the routing tables and later "auto-canceled", using a sort of "resource garbage collection".





G. *Main loop for Global Broker*

Figure 3 shows the main loop for the VIP V2 protocol. A synthetic explanation follows:

- `SREG(LOGIN,[S_i]^{i=1..n}) from Card:` a service registration login comes from the individual `Card` offering $[S_i]^{i=1..n}$ resources.
- `fakeS = gen(MyCard,Card)`: a fake unique resource is created.
- `SUPD([fakeS]) to CardB:` a service update propagating the fake resource is sent to all direct superbrokers.
- `for i=0 to T do MetaData = Listen()`: the broker listen for `T` units of time.
- `If then else`: if, during this time `T`, the broker receives a service update (`SUPD([fakeS]) from CardS`) for the fake resource (`fakeS`), then the registration is rejected (`SREG(REJ) to Card`), otherwise the registration is accepted (`SREG(ACC) to Card`), and the routing tables are updated (`RouteTable ←(Card,[S_i]^{i=1..n})`) and propagated to the direct superbrokers (`SUPD([S_i]^{i=1..n}) to CardB`), accordingly.
- `SREG(LOGOUT) from Card:` a service registration logout is acknowledged only if the individual `Card` has no pending queries in RDP (`some(Card,ReqList)`), otherwise the individual is "kindly" requested to stay in the colony.

`SUPD([S_i]^{i=1..n}) from CardS:` for each service update, the broker's routing table is updated (`RouteTable ←(CardS,[S_i]^{i=1..n})`).

```
While true do
MetaData = Listen()
case MetaData.OPE is
 SREG(LOGIN,[S_i]^{i=1..n}) from Card
  fakeS = gen(MyCard,Card)
  SUPD([fakeS]) to CardB
  for i=0 to T do MetaData = Listen()
  if MetaData.OPE = SUPD([fakeS]) from CardS
  then SREG(REJ) to Card
  else RouteTable ←(Card,[S_i]^{i=1..n})
       SREG(ACC) to Card
       SUPD([S_i]^{i=1..n}) to CardB
 SREG(LOGOUT) from Card
  if some(Card,ReqList)
  then SREG(STAY) to Card
  else SREG(LOGOUT) to Card
 SUPD([S_i]^{i=1..n}) from CardS
  RouteTable ←(CardS,[S_i]^{i=1..n})
Endcase
```

Fig. 3. VIP V2: Main loop for global broker

IV. Conclusions

Until now the original network topology was tree or forest-based. The idea of turning the topology into graphs is a small step for an overlay network but a momentous step for a network computer based on the Arigatoni programmable overlay network. As an immediate consequence, all Arigatoni's protocols must be reconsidered in order to take into accounts loops when updating routing tables, resource's overbooking, loops during resource discovery; this is the most important contribution of the paper.

Since the complexity of the RDP V4 protocol remains the same w.r.t. the version V3 [2,3,4] (basically the only difference is a test to check service request loops via the unique service registration identifier), and since complexity of the VIP V2 protocol is increased of a fixed time factor T (needed to "flood" the fake VIP service update in routing tables) this paper does not need *ad hoc* simulation experiences, and the reader can refers to [2,3,4,5] for the simulation results we conducted in both protocols.

Acknowledgment

Luigi Liquori thanks Jean-Claude Bermond for unvaluable discussions and suggestions about various dynamic graph topologies in overlays networks during a in a car-trip from Siena (Italy) to Nice (France).

References

[1] D. Benza, M. Cosnard, L. Liquori, M. Vesin, Arigatoni: A Simple Programmable Overlay Network, in: Proc. of John Vincent Atanasoff International Symposium on Modern Computing, IEEE, 2006, pp. 82–91.

[2] R. Chand, M. Cosnard, L. Liquori, Resource Discovery in the Arigatoni Overlay Network, in: I2CS: International Workshop on Innovative Internet Community Systems, LNCS, Springer, 2007, to appear. Also available as RR INRIA 5928.
[3] R. Chand, M. Cosnard, L. Liquori, Improving Resource Discovery in the Arigatoni Overlay Network, in: ARCS: International Conference on Architecture of Computing Systems, LNCS 4415, Springer, 2007, pp. 98—111.
[4] R. Chand, M. Cosnard, L. Liquori, Powerful resource discovery for Arigatoni overlay network, in: Future Generation Computing Systems, 2007.
[5] M. Cosnard, L. Liquori, R. Chand, Virtual Organizations in Arigatoni, DCM: International Workshop on Developpment in Computational Models. ENTCS 171 (3), Elsevier 2007, pp. 55-75.
[6] E. Zegura, K. Calvert, S. Bhattacharjee, How to Model an Internetwork, in: Proc. of INFOCOM, IEEE, 1996, pp. 594–602.
[7] AEOLUS, Deliverable D2.1.1: Resource Discovery: State of the Art Survey and Algorithmic Solutions, Tech. rep., By Evangelia Pitoura, University of Ioannina, http://aeolus.ceid.upatras.gr (2006).
[8] H. Jagadish, B. Q. Vu, BATON: A Balanced Tree Structure for Peer-to-
[9] Peer Networks, in: Proc. of VLDB, ACM, 2005, pp. 661–672.
[10] K. Aberer, P-Grid: A Self-Organizing Access Structure for P2P Information Systems , in: Proc.of CoopIS, no. 2172 in LNCS, Springer, 2001, pp. 179–194.
[11] P. Ganesan, P. Krishna, H. Garcia-Molina, Canon in G-ma jor: Designing, DHTS with Hierarchical Structure, in: Proc. of ICDCS, IEEE, 2004, pp. 263–272.
[12] M. J. Freedman, D. Mazieres, Sloppy Hashing and Self-Organizing Clusters, in: Proc. of IPTPS, no. 2735 in LNCS, Springer, 2003, pp. 45–55.

Michel Cosnard, MS 1975 Cornell University, Doctorat d'Etat 1983 Universite de Grenoble, served as Professor at ENS Lyon and from 1997, as director of the INRIA Research Unit in Lorraine. In 2001, he has been nominated director of the INRIA Research Unit in Sophia Antipolis and served as Professor at the University of Nice - Sophia Antipolis. In 2006, he has been appointed Chairman and CEO of INRIA. Michel Cosnard served as Editor in Chief of PPL, and editor of IEEE TPDS, Parallel Computing, Mathematical Systems Theory. He received a prize from the French Academy of Science, the IFIP Silver Core and IPDPS Babbage award.

Luigi Liquori, MS 1990 Udine University, Ph.D. 1996 University of Turin, served as Lecturer at the Ecole des Mines of Nancy from 1999. Since 2001, he is a senior researcher of INRIA. Luigi Liquori research's fields range from logics and foundations of mechanical proof assistants, to semantics of object oriented programming languages, until foundations of Overlay Networks and Pervasive Computing.