Meta-communications in Component-base Communication Frameworks for Grids

Alexandre Denis

▶ **To cite this version:**

Alexandre Denis. Meta-communications in Component-base Communication Frameworks for Grids. Cluster Computing, 2007, 10 (3), pp.253-263. 10.1007/s10586-007-0036-5 . inria-00410993

HAL Id: inria-00410993

https://inria.hal.science/inria-00410993v1

Submitted on 25 Aug 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Meta-communications in Component-based Communication Frameworks for Grids

Alexandre Denis
INRIA/LaBRI
351 cours de la Libération
33405 TALENCE CEDEX
FRANCE
Email: Alexandre.Denis@labri.fr

Abstract

Applications are faced with several network-related problems on current grids: heterogeneous networks, firewalls, NAT, private IP addresses, non-routed networks, performance problems on WAN. Moreover, the requirements concerning communications are varied and the acceptable tradeoffs highly depends on the applications. A solution to reach the flexibility regarding communication on grids is the use of a component-based communication framework. The users then compose their own protocol stacks by assembling building blocks in the way they want. However, a truly flexible and dynamic component-based communication framework needs a *meta-communication* channel for its out-of-band communications required by dynamic component assembly in a consistent way on multiple nodes. The meta-communication channel is useful for some "brokered" communication methods, too, and in particular those designed to cross firewalls. The meta-communication channel has often been the "*weakest link*" of component-based communication frameworks: bottleneck for the performance, back-door from the security point of view, and limited connectivity.

In this article, we present an architecture for a meta-communication channel that suffers from none of the aforementioned limitations. It exhibits good properties regarding connectivity, security and performance. Thus, the gain in flexibility brought by software components may be fully exploited without trading anything against flexibility.

1 Introduction

The goal of grid computing is to aggregate the computing power of multiple clusters of PCs and parallel machines scattered throughout multiple sites. Undoubtly, network communications play a critical role to reach this purpose.

Communication management on grids is different from a lot of other applications involving networking. The main characteristic of networks in grids is heterogeneity. The networking technologies are various, ranging from high-performance networks between nodes of clusters (SAN) through wide area networks (WAN) with a latency of multiple tenths or hundreds of milliseconds and a random bandwidth. These multiple levels bring each their own issues, thus an application for grids is faced with all of them. We consider in particular the following problems on modern grids:

Connectivity — To protect their machines from intruder attacks, many site administrators have drastically restricted the connectivity to the Internet. Many sites are using firewall routers, non-routed private networks [24], or hide their machines via *Network Address Translation* (NAT) [13]. As a consequence, plain TCP/IP is not sufficient to get a full connectivity, from every node to every other on the grid. NAT and firewalls introduce non-symmetry in the topology. Some nodes are hidden and not visible from the Internet. This is quite unusual for people used to parallel computing where it is traditional to have an all-to-all communication channel with no restriction.

Security — As WAN connections between sites cross the Internet, they are vulnerable to attackers. Thus, many application require authentication of communication peers and privacy based on encryption. The widespread solution to authentication and encryption is the use of the Transport Security Layer (TLS) [12], a successor of the Secure Sockets Layer (SSL).

Performance — Since most applications on grids expect high performance, performance is a critical aspect of network communication. Different network have different performance properties. Even a given network may exhibit different performance results depending on the protocol used. For example, plain TCP can hardly exploit the bandwidth capacity of WAN connections. One solution to improve TCP performance in WANs is to use multiple TCP streams in parallel. The Globus implementation of GridFTP [5] is probably the best-known tool implementing this approach. Alternatively, WAN performance can be improved using data compression, as implemented, e.g., in the AdOC library [19].

In this paper, we will use two different metrics for evaluating performance; we will consider separately the link utilization performance (characterized by the bandwidth and latency), and the connection establishment performance (characterized by the connection establishment delay).

The problems to overcome are very different and influence each other, e.g. usually improving security degrades performance, thus tradeoffs have to be made. However, the applications that may benefit from a deployment on grids are varied with very different requirements regarding security and performance from one application to another. There won't be any best tradeoff suitable for any application.

A communication framework for grids has to be able to utilize a very large spectrum of networking technologies, must be flexible enough to be adapted to the requirements of various applications, and must overcome the main problems of communication on grids, namely connectivity, security and performance. One solution to reach such a flexibility in a communication framework is the use of a component-based approach. The user is offered the ability to assemble itself the building blocks he/she wants to get a custom service. For a good flexibility and adaptability, we will see that it is welcome that the communication framework implements an overlay network for out-of-band communications, that we call a *meta-communication channel*. The meta-communication channel is often the weakest link of a component-based communication framework. It may introduce security holes, performance bottlenecks, or connectivity restrictions.

This paper presents on-going work on a component-based approach for the meta-communication channel itself, in order to solve all the aforementioned limitations at the same time.

The remaining of this paper is divided as follows: the second section analyzes component-based communication frameworks for grids and their needs and requirements for a meta-communication channel. Section 3 explains our proposal for managing such a meta-communication channel. Section 4 describes and evaluates our implementation of our proposal in the PadicoTM communication framework. Section 5 discusses related work, and section 6 draws conclusions and directions for further work.

2 Component-based Communication Frameworks

In this section, we study the principles and operation of component-based communication frameworks, and exhibit their need for a *meta-communication* channel.

2.1 Motivation and Principles

The most challenging part to manage communication on grids is the heterogeneity of the resources and the variety of applications —and thus their requirements for a communication sub-system. The networks are ranged from high-performance networks in cluster to wide-area networks between sites. Not only their properties are different, but their protocol, communication methods and programming interface are different. Moreover, the requirements for the communication sub-system depends on the application; the performance *v.s.* security tradeoff largely depends on the nature of the application and may not be hard-coded in a communication framework.

Such a needed flexibility may hardly be reached by the usual two-layer portability model based on an abstraction layer and drivers for each supported resource. Considering the variety of cases to deal with, on grids it would be highly

welcome to have communication methods be *assembled* by the users depending on the application and the kind of network and protocols involved. For example, a user may want to add compression or encryption on the fly to any communication method; another user may want no encryption at all to get the best achievable performance with non-critical data.

To reach such a flexibility, it has been proposed [8, 18] to manage communications with a freely and dynamically assembled protocol stack made of several simple *building blocks*. Such a technique is nowadays commonly used in all fields of software development and is known under the name of *software component*. In the remaining of this paper, we will call "component-based" a communication framework based on freely assembled building blocks.

Such a flexible assembled protocol stack based on "building blocks" has been implemented in particular in x-kernel [18], Globus XIO [8], NetIbis [7] and PadicoTM [10].

2.2 Need for a Meta-communication Channel

In this section we introduce the concept of *meta-communication* and its motivations. Formally, we distinguish two classes of network communications: *meta*-communications and *data* communications.

- **Data communications** are communications carrying data from the upper levels —middleware or application— which are using the communication framework. These communications are controlled through the API of the communication framework.
- **Meta-communications** are communications used *internally* by the communication framework or one of its components. They are sometimes called *service links*, *control channel* or *out-of-band* communications in some other communication frameworks.

In the next paragraphs we explain why the meta-communications are welcome in a communication framework for grids.

Controlling the assembly. As a result of the component-baseness of a communication framework, various assembly schemes of building blocks may be selected to adapt to the requirement and networking resources. However, it introduces the problem of choosing the appropriate assembly and ensuring that all peers (i.e. client and server) are using the same assembly. For example, if a server uses *zip* compression over plain TCP and a client uses directly plain TCP, they are not likely to understand each other. At least two approaches are possible to solve this problem:

Static component stack — This is the approach used in Globus XIO [8]. Client and server know in advance the protocol stack to use. Once the server is bound to a protocol stack, clients must use the same protocol stack. This approach is simple but suffers from a lack of flexibility; the

servers must know in advance where the requests will come from. As a consequence, for a given server, all clients have to use the same protocol stack. However, the user may want to use a different protocol stack for example for connection coming from nodes on the same cluster reachable through a high-performance network and for connections that cross an insecure and slow WAN.

Dynamically assembled component stack — This is the approach used in PadicoTM [10] and NetIbis [7]. Both parties agree on the fly on the protocol stack to use. Therefore a server is not required to know in advance where the requests will come from, and different clients to the same server may even use different protocol stacks. The *dynamically assembled component stack* strategy uses the following algorithm: when a client requests a new connection establishment, the communication framework first selects the assembly scheme to use according to configuration rules and depending on the nodes involved. Then the framework sends an *assembly request* to the framework of the server node; this request asks the server node to create an instance of the selected protocol stack (on the server side). In the meantime, the client creates its own instance of the selected protocol stack. Finally, the client uses the usual connection mechanisms on his stack, and is sure that the server is already listening with the same protocol stack.

The main obstacle to dynamic assembly is that there must be a way of sending the *assembly request* to the framework on the server node even though the connection is not established yet. Thus we need a pre-existing framework-to-framework communication channel to send meta-data. This is precisely the role of the meta-communication channel. Dynamic assembly using a control channel (*meta-communication channel*) is depicted in Figure 1: on step 1, node B does not know the assembly that will be used (actually, it does not even know that a connection will be estbalished from node A); node A sends a connection request with the assembly description embedded in he request. On step 2, node B builds locally the requested assembly then sends an acknowledgement with connection information (the port number) to node A. On step 3, node A establishes the data connection through the selected component stack.

We should notice that we have restricted our study to the case of client–server connection establishment. However, some other connection schemes are possible. For example, PadicoTM has the notion of *circuit* which is composed of a set of nodes (roughly similar to an MPI communicator). It is possible to apply the same algorithm to a larger set of nodes than two as in client–server, but we will only consider the case of client–server in the remaining of this paper to avoid useless overcomplexification.

Brokered communication. Some communication methods need to exchange information prior to establishing connections. Plain TCP is the best known example of this. To establish a TCP connection on an ephemeral port, the port

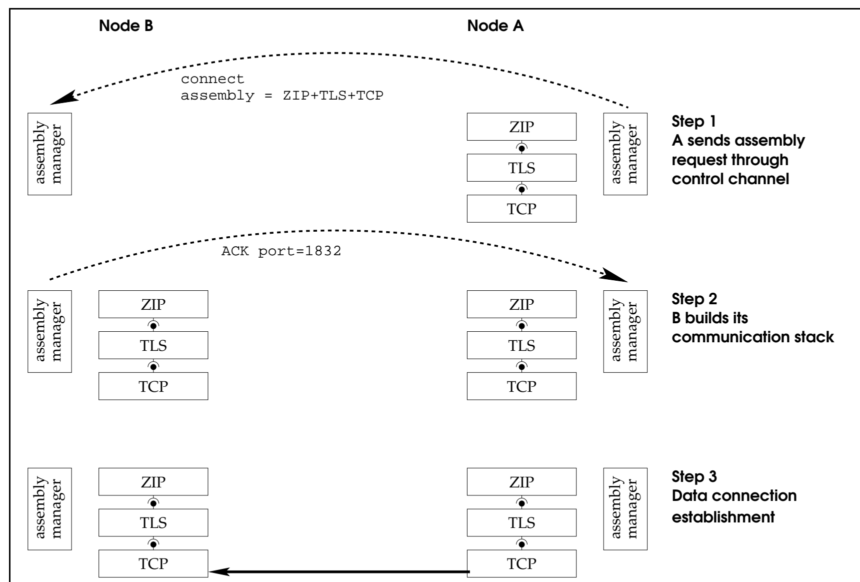

Figure 1: Dynamically assembled communication stack from node A through node B, using ZIP compression over TLS over TCP/IP; node B does not know in advance the component assembly.

number has first to be transmitted from the server to the client before the client can connect to this port. There are various methods to solve this problem:

- listen on a well-known (fixed) port instead of an ephemeral port;
- use a third party that plays the role of directory (or "name service");
- send the port number through a meta-communication channel, pre-existing before the data connection is attempted.

The first solution may not work in case the chosen port is busy, and does not supports multiple instances. The second solution supposes that all nodes are able to communicate with a third party; this means that actually the third party establishes an indirect route for meta-communications between nodes.

Other communication methods that plain TCP can benefit from a meta-communication channel. For example, in case a server is behind a firewall that drops incoming packets but not outgoing packets (common case), or behind a NAT [13] gateway, we establish connections in the outgoing way; this is the so-called *reverse connection* method. Clients send a request to the server so that it connects to them. Another technique for crossing firewalls is TCP splicing (also called "*simultaneous SYN*" or "*simultaneous initiation*" in [23]): both end-points needs to exchange port numbers, and need to synchronize themselves to succeed in *simultaneous* connection. Both *reverse connection* and *TCP splicing*

need to exchange meta-data. The availability of a meta-communication channel which allows component-to-component communications facilitates the use of plain TCP over dynamic ports and enables connection methods that wouldn't be available without it. Thus the meta-communication channel is a *must*, especially for communication methods designed to overcome the connectivity issues typically encountered in a grid environment.

To dynamically assemble component stack and to use brokered communication methods, we need a *meta-communication channel* which allows framework-to-framework and component-to-component communications. We define the meta-communication channel as a communication channel that:

- allows communication from every node to every other node;
- exists *before* any data connection is attempted;
- exists *implicitly*, i.e. it is created without any explicit action from the user, as soon as the processes are started.

2.3 Requirements of a Meta-communication Channel

In this section, we define and analyze the requirements that a meta-communication channel must fulfill to be used in a communication framework for grids. Actually, the meta-communication channel is often designed with little care and is usually the "*weakest link*" of a communication framework. However, the whole communication framework cannot have a better connectivity, security, and performance than its meta-communication channel.

Therefore, the meta-communication channel has the same requirements as the data communication channels, namely: connectivity, security, and performance.

Connectivity. Nodes not reachable through the meta-communication channel are not able to use dynamically assembled component stacks since there is no way to send an assembly request. Moreover, without a meta-communication channel, no brokered communication method can be used. As a consequence, if a node is not reachable through plain TCP (because of firewalls, NAT, etc.) and is not reachable through a meta-communication channel, then it is not reachable for any data connection. Therefore, for a communication framework based on dynamically assembled component stacks —the most flexible model—, the set of nodes reachable for data connection is a subset of the nodes reachable through the meta-communication channel.

Security. Since the protocol stack is decided by clients, any intruder able to send meta-communication messages to a node may send forged assembly requests. Therefore, such an intruder may request unauthenticated and/or unencrypted connections to a server. A world-accessible meta-communication channel is undoubtly a back-door through which an intruder may change the security

policy used for its own connection attempts. As a consequence, the security level of the whole communication framework cannot be higher than the security level of the meta-communication channel.

Performance. The meta-communication channel is used for assembly request and brokered communication methods when a data connection is attempted. It means that the meta-communication channel is on the critical path for data connection establishment. In other words, the data connection establishment performance is impacted by the meta-communication channel performance. Depending on the application, the data connection establishment delay may or may not be critical for overall performance. On the other side, the meta-communication channel connection establishment only affects the process initialization time.

3 An Approach for a Flexible Meta-communication Channel

In this section, we describe our approach for a meta-communication channel suitable for a communication framework for grids.

As seen in the previous section, the meta-communication channel itself has roughly the same requirements as data communications: connectivity, security, and performance. We propose thus to use a similar solution to a similar problem; indeed, following the study of section 2.1, we propose the idea that the meta-communication channel might be implemented with dynamically assembled protocol stacks of software components. The remaining of this section explains such an approach where the meta-communication channel itself reaches a good flexibility and fulfills its requirements through a component-based architecture.

3.1 Overall architecture: two-step bootstrap

The main difficulty raised by the idea of a meta-communication channel following itself a component-based architecture is that it needs its own meta-communication channel —or rather: meta^2-communication channel. However, the requirements for such a meta^2-communication channel are not as high as for the meta-communication channel since it is used only at bootstrap time to build only the (primary) meta-communication channel. From now on, we will call this meta^2-communication channel the *bootstrap channel*.

Undoubtly, the bootstrap channel has the same connectivity and security requirements as the meta-communication and data channels. However, the constraints on performance may be relaxed. The performance of the bootstrap channel only impacts the performance of the meta-communication connection establishment that takes place at process start-up. We choose to neglect this one-time initialization delay. As a consequence, the requirements and constraints for the bootstrap channel are:

- full connectivity (every node to every node);
- secure communications;
- uses no meta-communication channel (no meta^3-communication channel);
- performance requirements are low.

With these hypothesis, we conclude that for the bootstrap channel, *static component stack* is mandatory since no meta-communication is possible for a dynamically assembled stack. This is no problem since a "one size fits all" approach is possible at the bootstrap channel level: we can guarantee security with an authenticated/encrypted communication method; we can bring the full connectivity through *routing* done by the communication framework on top of the encrypted transport. The performance of such a systematically routed and encrypted communication system is likely to be suboptimal, but it fulfills our requirements for a bootstrap channel.

Following this scheme, the sequence of initialization and data communication establishment is as follows:

1. start processes;
2. each process opens its bootstrap channel;
   Each node has an initial basic connectivity to other nodes.
3. processes open meta-communication channel towards other nodes, using the bootstrap channel for meta^2-communications;
   Each node has a meta-communication channel to other nodes.
4. upon data connection establishment attempt, an assembly request is sent to the other node through the meta-communication channel.

The internals of the bootstrap channel, the meta-communication channel, and various optimizations are detailed in the following sections.

3.2 Bootstrap channel

The goal of the bootstrap channel is to reach a basic initial full connectivity. This implies *resource discovery*, and basic messaging towards every known node. For scalability reason, we use a two-level hierarchical approach based on clusters of nodes.

3.2.1 Bootstrap channel architecture

The overall architecture of the bootstrap channel is depicted in Figure 2.

We define a *node* as a process involved in the considered application; there may be several nodes per hosts. We define a *cluster* as a set of nodes which are implicitly connected through an underlying native communication subsystem. A typical cluster is for example a set of nodes connected with a vendor-MPI

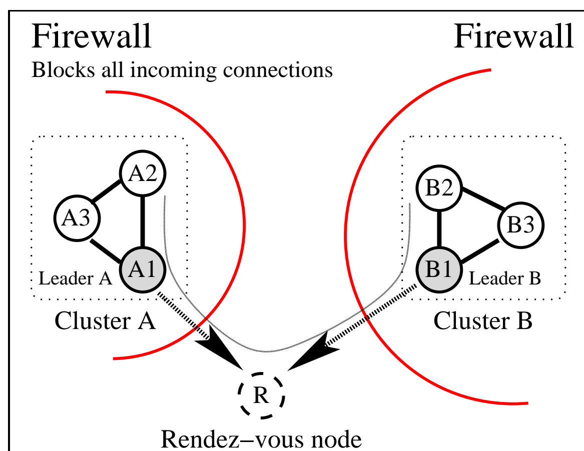

Figure 2: The bootstrap channel uses a relayed protocol through a rendez-vous node. The route from node A2 to B2 goes through A1, rendez-vous node (R), and B1.

on a parallel machine, or nodes connected through the Madeleine [6] communication library. Usually, the native intra-cluster communication subsystem is high-performance, non-TCP, and unsecure but isolated from the outside.

In each cluster, we distinguish a particular node that we call the *leader*. It should be able to connect to the internet with plain TCP, and be able to communicate with every node of the cluster with the native communication subsystem of the cluster. A typical example of cluster leader choice is the front-end of the cluster.

A particular node is dedicated to the directory management. We call this node the *rendez-vous* node. The rendez-vous node should be visible from the internet —or at least from all the cluster leaders, in case of a private grid. Typically, the rendez-vous node will be located on a gateway, outside of any firewall, and with a public IP address. The rendez-vous node manages a directory of nodes comprised in the current session. More precisely, it manages a table of node entries; each entry is composed of a node ID (actually an UUID [21]), and the ID of the leader or a reference to the connection if the node is a leader. The rendez-vous node listens for incoming connections from the internet on a fixed port number, using a secure (e.g. SSL/TLS [12]) communication method.

3.2.2 Discovery phase

The initial reference of the rendez-vous node is supplied to every node. When a process starts, it initializes its bootstrap connections. A standard node (non-leader) sends its ID to its leader. A leader node connects to the rendez-vous node with the secure communication method, using the supplied bootstrap initial reference; it sends its ID and the list of IDs of the nodes in its cluster. The

rendez-vous node registers the IDs and the route to reach every known node. Then, it broadcasts the ID of new nodes to every already known leader, so as every node knows the list of currently running nodes. In case of a broken connection between a leader and the rendez-vous node, it unregisters the given leader and all the nodes of its cluster, and broadcasts the information to the other leaders.

The communication method used between the rendez-vous node and the leaders may be configured. For example, as an optimization one may want not to use authentication at all on a private grid. However, all leaders and the rendez-vous node must use the same configuration for a given session. The initial reference of the rendez-vous node is given similarly as a configuration parameter. It is not expected to change very often.

3.2.3 Messaging on bootstrap channel

Once the bootstrap channel is connected (i.e. the nodes are connected to their leader, and the leaders connected to the rendez-vous node), the messages on the bootstrap channel are *routed*, as depicted on example shown in figure 2. Since the topology of the bootstrap channel is a tree rooted in the rendez-vous node, the routing algorithm is straightforward. To send a message, a node sends it to its cluster leader. If the final recipient is in the same cluster, then the leader sends the message directly, else it forwards it to the rendez-vous node. Following its routing table, the rendez-vous node sends the message to the appropriate cluster leader, which finally forwards the message to its final recipient. The properties of such a bootstrap channel are:

- full connectivity, from every node to every other node;
- as secure as the chosen underlying transport layer;
- low performance, due to routing and the bottleneck in the rendez-vous node. However, every route is no longer than 4 hops;
- static protocol stack, does not require a meta-communication channel.

These properties fulfill the requirements for a bootstrap communication channel.

3.3 Meta-communication channel

The goal of the meta-communication channel is to provide the framework and the components with fast and secure connections from every node to every other node. The meta-communication channel is based on dynamically assembled protocol stacks. It has at its disposal the bootstrap channel.

The meta-communication may use the straightforward approach introduced in section 3.1: just after bootstrap, open all-to-all connections for the meta-communication channel. However, optimizations are highly welcome to overcome two main drawbacks: opening n^2 connections at the same time (n being the number of nodes) is likely to be a superfluous overload on the bootstrap

channel —the rendez-vous node is a bottleneck—, and describing all the protocol stacks for every node to every other node is a tedious job.

Lazy connections. To solve the problem of the bootstrap channel flood, reduce startup time, and save on resources wasted by unneeded connections, the meta-communication channel uses lazy connection establishment. All nodes of the session are known as a result of the resource discovery phase, but it is not necessary to immediately open meta-communication connections to every known node. Therefore, it is lighter to open meta-communications connections *on-demand*, on the first message sent to a given node on the meta-communication channel.

Default configuration schemes. The assembly patterns used for protocol stacks are configured by the user as a set of rules defining which assembly pattern to use to reach which node. This is very powerful and may be used to describe the protocol stacks for any topology supported by the communication framework. However, the targeted topology are not random, thus one can want to optimize the configuration process for commonly encountered network topologies. It also saves the user's time by reducing the configuration complexity.

Basically, a configuration can be described as a default configuration strategy, and a list of exceptions. The default configuration is a sensible default scheme, for example: *open direct TCP connections* from every node to every other node (typically for small single-site, firewall-less, multi-cluster configurations); *use native intra-cluster* communication method for intra-cluster, establish direct connections between leaders, and route messages (max.: 3 hops). It can save a long distance round-trip if the rendez-vous node is far from both leaders; *use bootstrap channel* as meta-communication channel —a last resort option, but works everywhere. These default configurations are a basis upon which more advanced configurations are built in adding rules describing only exceptions.

4 Implementation and Evaluation

In this section, we describe our implementation of our meta-communication channel model in the PadicoTM [10] communication framework for grids.

4.1 The PadicoTM communication framework

PadicoTM [10] is a component-based communication framework for grids. PadicoTM is designed to be as flexible as possible. It supports a wide range of networks, from high-performance networks to wide area networks. Moreover, several middleware systems —MPI, various CORBA implementations, Java RMI, SOAP implementations, HLA, ICE, DSM systems, JXTA— have been ported on top of PadicoTM thanks to its flexible personality layer that enables a seamless integration of existing code.

PadicoTM is based on a three-layer approach [11]: the lowest layer does multiplexing and arbitration between concurrent accesses to a given network, and between accesses to different networks (e.g. TCP/Ethernet and Myrinet) on the same machine; the middle layer is the abstraction layer, based on dynamically assembled components; the higher layer, or personality layer, adapts the API to the expectations of applications. The meta-communication channel is needed only for the abstraction layer, where the dynamic component assembly takes place.

4.2 Communication methods implemented

Various communication methods have been implemented in PadicoTM. Each communication method is provided in its own component and may be freely used in any assembly for supplying communication to any middleware system (MPI, CORBA, etc.). The supported communication methods are:

Plain TCP — This is the usual vanilla TCP connection, with access to some configuration parameters such as window size.

Madeleine — We use the Madeleine [6] communication library for access to high-performance networks in clusters. Supported networks are: Myrinet (through MX, GM or BIP), SCI, Quadrics QsNet, VIA.

Shmem — A shared memory communication component offers low-latency high-bandwidth inter-process communication on SMP hosts.

TCP derivatives for WAN — A large set of communication methods derived from TCP are implemented to overcome connectivity and performance problems specific to WAN. These methods are: *TCP splicing* (*aka* simultaneous connect) for crossing firewalls with no performance drop; *one-way connection* to always establish connections in the same direction, to cross firewalls when only one side is firewalled; SOCKS [22] *proxy*; connection through *SSH tunnels*; *parallel streams* to improve TCP performance, as implemented in GridFTP [5].

Data filters — Some data filters are proposed. These filters may be composed atop any other communication method. The implemented filters in PadicoTM are: compression —LZO, BZIP2, and AdOC [19] (adaptive ZIP)—, and Gnu TLS for authentication/encryption.

Last resort — A last resort communication method is proposed. It performs tunneling through the meta-communication channel. The performance is likely to be low, but this solution works in *any* case where a meta-communication channel is established.

	Latency	Establishment delay (direct)	Establishment delay (with basic meta-comm. ch.)	Establishment delay (comp.-based meta-comm. ch.)
Myrinet	10 μs	30 μs	400 ms	50 μs
TCP/Ethernet	100 μs	300 μs	400 ms	500 μs
TCP/WAN	100 ms	300 ms	700 ms	500 ms

Table 1: Typical latency and connection establishment delay on various networks.
(The presented figures are *orders of magnitude*)

4.3 Meta-communication channels in PadicoTM

In PadicoTM, the concept of *cluster* is guided by Madeleine. The rendez-vous node is a dedicated process that can be started on any accessible host. Three schemes are available for the bootstrap channel:

- rendez-vous node on an internet-visible host, connections from leaders to rendez-vous node through TLS over TCP. This closely follows the model described in section 3.2.
- rendez-vous node on the machine of the user who launches processes, connections from leaders to rendez-vous node through SSH tunnels. The advantage is that it does not require an Internet-visible host and works even if some leaders have no access to the public Internet.
- rendez-vous node on some random machine, connections through plain TCP. This avoids unnecessary TLS certificates mangling when deploying on a private network.

Bootstrap connections from cluster nodes to cluster leaders are done through Madeleine. The implementation of the meta-communication channel is quite straightforward following the model described in section 3.3.

4.4 Evaluation

We have evaluated our component-based approach of meta-communication channel on various grid configurations.

Connectivity analysis. We deployed PadicoTM on multiple sites of Grid'5000 [1] and some sites outside Grid'5000. Grid'5000 as a whole is a private network without routing towards the outside Internet (private IP address without NAT) except one gateway per site allowed to connect to the outside. Most sites outside Grid'5000 are themselves protected by stateful firewalls.

In all cases, we were able to establish a bootstrap channel from every node to every other node and thus reach a full connectivity for the meta-communication channel and data links. When there are nodes inside a private network without NAT and nodes outside, there is no choice but to use proxies or SSH tunnels

for the bootstrap channel. This is made possible by the fact that our bootstrap channel uses a configurable component assembly (even though it is *static* for the bootstrap channel). We should notice that getting even basic connectivity on such a topology is not possible for most communication frameworks, even component-based ones such as NetIbis [7].

Security analysis. Both the bootstrap and the meta-communication channels are built as component stacks for which the default is either TLS or a private intra-cluster network. Our approach introduces no world-accessible unsecured TCP server, unless explicitly asked by a user willing to trade security against performance in a controlled environment.

Performance analysis. We have measured the quantitative impact of our approach for a component-based meta-communication channel. The performance of the meta-communication channel impacts the data connection establishment. Table 1 shows typical connection establishment performance.

The first column shows transmission latency, which is the latency of a meta-communication channel using the given network. The second column shows connection establishment delay with a three-way handshake (native for TCP/Ethernet and TCP/WAN, at application level for Myrinet) and a static protocol stack (no meta-communication channel). The third column shows connection establishment delay with dynamic protocol stacks and meta-communication routed through a relay located in a site 100 ms apart from the hosts (not uncommon on large-scale grids). Finally, the fourth column shows connection establishment delay with dynamic protocol stacks and component-based meta-communication channel (our proposed architecture).

The connection establishment delay on a dynamic component-based software with a basic meta-communication channel is bounded by the performance of the meta-communication channel. If the performance of the meta-communication channel is poor, e.g. caused by relaying through a WAN, then connection establishment is slow even if the remote machine is theoretically reachable through Myrinet. In contrast, our proposed architecture (rightmost column) gets performance results close to direct connection (delay $\sim +60\%$). This is made possible by the use of an appropriate communication method by the meta-communication channel itself. We can see that our approach greatly reduces the connection establishment delay and makes the overhead of using a dynamically assembled protocol stack acceptable.

Regarding *scalability*, it should be noted that the rendez-vous node may look like a bottleneck. However, only cluster leaders are connected to the rendez-vous node and very little communication goes through the rendez-vous node (actually, only *bootstrap* communications).

In conclusion, our proposed architecture for a meta-communication channel enables connectivity in cases where most communication frameworks cannot even get basic connectivity, and gets better performance than other meta-

communication-channel based approaches where it can compare, without compromising security.

5 Related Work

Many researchers are working on communication management for grids. Most of the works rely on the difference between intra-cluster high-performance communication and inter-cluster TCP communication, but only a few actually uses a component-based architecture for a flexibility pushed further than the binary intra-/inter-cluster approach.

Globus XIO [8] is becoming a *de facto* standard for communication on grids. Its main concept is the driver stack which is an assembly of building blocks very similar to software components. However, its *static* driver stack approach, with no meta-communication channel, defeats most of the purpose of software components in communication frameworks. In particular, a server must know in advance the driver stack that clients will use, which limits the flexibility of the communication framework.

A widely used grid programming model is MPI. The most popular implementation for grids is MPICH-G2 [20], an MPI implementation over Globus. However, WAN communications methods in MPICH-G2 are rudimentary; it does not cross firewalls nor NAT. The only communication methods that MPICH-G2 is able to utilize are vendor-MPI for intra-cluster communication and plain TCP for inter-cluster communication. PACX-MPI [15] is an implementation of MPI that has been designed from scratch for grids. For each site, PACX-MPI uses a dedicated gateway node for relaying messages across the WAN. This static configuration solves some of the connectivity problems. However, it does not solve all problems caused by firewalls and introduce a performance penalty because of relaying. GridMPI [2] is another implementation of MPI designed from scratch for grids. It solves some connectivity problems but supports only vendor-MPI communications, plain TCP, and routing on top of these. OpenMPI [25] is becoming a major MPI implementation and is built with software components. However, software components are used as an engineering tool to ease development by independant people and not as a tool to reach flexibility and dynamicity. Components in OpenMPI are statically assembled by the end-user. None of these MPI implementations is as flexible as PadicoTM with dynamic protocol stack and brokered communication methods (splicing, reverse connections, etc.).

NetIbis [7] is another component-based communication framework for grids. It features dynamically assembled protocol stacks and brokered communication methods. Actually, this advances in NetIbis are our own work [9]. Our present work transposes these concepts in PadicoTM and goes further with a two-step bootstrap for better performance of the meta-communication channel and a hierarchical bootstrap channel.

Finally, Project JXTA [16] is an alternative to solving connectivity problems in WAN with application-level relaying building an overlay network. This is very

similar to our bootstrap channel. However, JXTA is targeted towards peer-to-peer and very volatile nodes rather than grid computing. It will presumably not be suitable for high-performance communication [17].

6 Conclusion and Future Work

Applications are faced with connectivity and security problems in current grids. Moreover, the requirements concerning communications and the acceptable tradeoffs highly depends on the applications. A solution to reach the flexibility regarding communication on grids is the use of a component-based communication framework. The users are then completely free to configure and assemble the building block in the way they want. However, we have seen that a truly flexible and dynamic component-based communication framework needs a meta-communication channel for its out-of-band communications required by consistency and dynamic adaptability. The meta-communication channel is useful for some "brokered" communication methods, in particular those designed to cross firewalls. The meta-communication channel has often been the "weakest link" of component-based communication frameworks: bottleneck for the performance, back-door from the security point of view, and limiting connectivity to nodes reachable by plain TCP.

We proposed in this article an architecture for a meta-communication channel that suffers from none of the aforementioned limitations. It exhibits good properties regarding connectivity, security and performance. Thus, the gain in flexibility brought by software components may be fully exploited without trading anything against flexibility. The proposed architecture has been successfully implemented in the PadicoTM communication framework which is available [4] as open source software.

The following steps in our work are in multiple directions. The first direction is quite short term and consists in adding support for more communication methods, and in particular for the ubiquitous Globus Security Infrastructure (GSI) [14]. The second direction consists in investigating precisely the *scalability* of our approach for thousands of nodes, and our envisaged solution with a *federation of rendez-vous nodes*. Finally, *fault-tolerance* which was not taken very much into account in our present study, will be investigated for very large scale experiments.

Acknowledgements

This work has been funded by the project "LEGO" [3] (contract number ANR-CICG05-11) from the French National Agency for Research (ANR).

Experimentations were performed on the Grid'5000 [1] platform funded by the ACI GRID from the French Ministry of Research.

References

[1] The grid'5000 project. `http://www.grid5000.fr/`.

[2] GridMPI. `http://www.gridmpi.org/`.

[3] Lego: League for efficient grid operation. Website: `http://graal.ens-lyon.fr/LEGO/`.

[4] The PadicoTM web site. `http://runtime.futurs.inria.fr/PadicoTM/`.

[5] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Meder, and S. Tuecke. GridFTP Protocol Specification. GGF GridFTP Working Group Document, 2002.

[6] Olivier Aumage, Luc Bougé, Alexandre Denis, Lionel Eyraud, Jean-François Méhaut, Guillaume Mercier, Raymond Namyst, and Loïc Prylli. A portable and efficient communication library for high-performance cluster computing. *Cluster Computing*, 5(1):43–54, January 2002.

[7] Olivier Aumage, Rutger Hofman, and Henri Bal. Netibis: An efficient and dynamic communication system for heterogeneous grids. In *Proc. of the Cluster Computing and Grid 2005 Conference (CCGrid 2005)*, Cardiff, UK, May 2005. ACM/IEEE. 8 pages.

[8] John Bresnahan. The eXtensible Input Output library for the Globus Toolkit (tm). `http://www-unix.globus.org/developer/xio/`.

[9] Alexandre Denis, Olivier Aumage, Rutger Hofman, Kees Verstoep, Thilo Kielmann, and Henri Bal. Wide-area communication for grids: An integrated solution to connectivity, performance and security problems. In *Proc. of the Thirteenth IEEE International Symposium on High-Performance Distributed Computing (HPDC'13)*, Honolulu, Hawaii, June 2004. IEEE.

[10] Alexandre Denis, Christian Pérez, and Thierry Priol. PadicoTM: An open integration framework for communication middleware and runtimes. *Future Generation Computer Systems*, 19(4):575–585, May 2003.

[11] Alexandre Denis, Christian Pérez, and Thierry Priol. Network communications in grid computing: At a crossroads between parallel and distributed worlds. In *18th International Parallel and Distributed Processing Symposium (IPDPS2004)*, page 95a, Santa Fe, New Mexico, April 2004. IEEE Computer Society.

[12] T. Dierks and C. Allen. The TLS Protocol Version 1.0. Request for comments 2246, IETF, January 1999. `http://www.ietf.org/rfc/rfc2246.txt`.

[13] K. Egevang and P. Francis. The IP Network Address Translator (NAT). Request for comments 1631, IETF, May 1994. http://www.ietf.org/rfc/rfc1631.txt.

[14] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *5th ACM Conference on Computer and Communications Security Conference*, pages 83–92, 1998.

[15] E. Gabriel, M. Resch, and R. Rühle. Implementing MPI with Optimized Algorithms for Metacomputing. In *Message Passing Interface Developers and Users Conference (MPIDC)*, pages 31–41, Atlanta, March 1999.

[16] Li Gong. Project JXTA: A technology overview. Technical report, Sun Microsystems, Palo Alto, USA, October 2002. http://www.jxta.org/project/www/docs/TechOverview.pdf.

[17] Emir Halepovic and Ralph Deters. JXTA performance study. In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, Victoria, B.C., Canada, August 2003. IEEE Computer Society.

[18] N. C. Hutchinson and L. L. Peterson. The x-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, 1991.

[19] Emmanuel Jeannot, Björn Knutsson, and Mats Björkman. Adaptive Online Data Compression. In *11th International Symposium on High-Performance Distributed Computing (HPDC11)*, pages 379–388, Edinburgh, Scotland, July 2002. IEEE Computer Society.

[20] N. Karonis, B. Toonen, and I. Foster. MPICH-G2: A grid-enabled implementation of the message passing interface. *Journal of Parallel and Distributed Computing (JPDC)*, 63(5):551–563, May 2003.

[21] P. Leach, M. Mealling, and R. Salz. A UUID URN Namespace. Internet-draft, IETF, December 2004. http://www.ietf.org/internet-drafts/draft-mealling-uuid-urn-05.txt.

[22] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. SOCKS Protocol Version 5. Request for comments 1928, IETF, March 1996. http://www.ietf.org/rfc/rfc1928.txt.

[23] Jon Postel. Transmission Control Protocol. Request for comments 793, IETF, September 1981. http://www.ietf.org/rfc/rfc0793.txt.

[24] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. Request for comments 1918, IETF, February 1996. http://www.ietf.org/rfc/rfc1918.txt.

[25] Jeffrey M. Squyres and Andrew Lumsdaine. The component architecture of open MPI: Enabling third-party collective algorithms. In Vladimir Getov and Thilo Kielmann, editors, *Proceedings, 18th ACM International Conference on Supercomputing, Workshop on Component Models and Systems for Grid Applications*, pages 167–185, St. Malo, France, July 2004. Springer.