An Application-Level Network Mapper

Arnaud Legrand, Frédéric Mazoit, Martin Quinson

Laboratoire de l'Informatique du Parallélisme

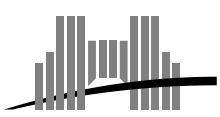
École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 5668

An Application-Level Network Mapper

Arnaud Legrand,
Frédéric Mazoit,
Martin Quinson

February 2003

Research Report N° 2003-09



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr

An Application-Level Network Mapper

Arnaud Legrand, Frédéric Mazoit, Martin Quinson

February 2003

Abstract

This paper presents a tool to automatically discover the network topology. The goal is to evaluate the performance of concurrent transfers (for example to improve collective communications) and not to discover the physical machines interconnection scheme (for administration purposes). The problems encountered, preliminary algorithms to solve them, as well as theoretical proofs of their validity (under some conditions) are presented.

Keywords: Grid computing, simulation, network topology, communication performance prediction

Résumé

Cet article présente un outil de découverte automatique de la topologie du réseau. Son objectif est d'évaluer les performances de transferts concurrents (par exemple pour améliorer des communications collectives) et non de découvrir le schéma d'interconnexion physique des machines (ce qui pourrait servir à diagnostiquer des problèmes de configuration réseau). Les problèmes rencontrés, des algorithmes préliminaires pour les résoudre, ainsi que la preuve de leur validité (sous certaines conditions) sont présentés.

Mots-clés: Calcul distribué à grand échelle, simulation, topologie de réseaux, prédiction de performances de communications

1 Introduction

Metacomputing consists in federating distributed computational resources in order to aggregate their power. The Grid [FK98] denotes the resulting virtual computer formed by a large-scale set of machines sharing their local resources with each other. Usually, Grid testbeds are constituted by several organizations providing access to several of their machines, resulting in a wide area constellation of local area networks.

Using the resulting platform is known to be a challenging task due to its heterogenous, dynamic and shared nature. We need to get not only the available bandwidth and latency of all end-to-end host pairs, but also the interconnection topology. Such knowledge can be used to simulate or predict performance of collective communications, or to determine the optimal placement of services like web caches or computational servers.

Even if we would like to get information about communications between all possible host pairs, monitoring everything would reduce drastically the scalability of the system. Moreover, monitoring all links would imply to deploy sensors on routers, which is simply impossible. That is why we must predict the performance on an unmonitored path by aggregating measurements done on its parts.

According to [Pax97], it is very difficult to determine the paths followed by the packets on wide area networks and it would be almost impossible to deduce the communication performance and the interaction of a data stream on another. That is why our goal is to acquire a macroscopic point of view of the network, rather than a raw microscopic one.

This paper presents a tool called ALNeM (*Application-Level Network Mapper*), which is designed to gather network topology informations useful to network-aware applications. Its goal is not to help administrators to monitor their network and diagnostic bottlenecks or failure points, but rather to give other applications the ability to predict the network performance of several interacting data streams.

This paper is organized as follows: Section 2 presents some methodologies existing in the literature to gather such informations, including the one used by ALNeM. Section 3 focuses on what informations about the network topology we are looking for, depending on the planned use. The main part of this article comes with Section 4, which sketches the ALNeM algorithm, from the experiment plans to the reconstruction of the searched topology, as well as the proof (under some assumptions) of the validity of the reconstruction algorithm. Section 5 concludes this paper and presents future work.

2 State of the art

Most of the previous work focused on collecting information about each separate host, or host-to-host communication [WSH99, Din02], and little attention has been given to gathering informations about the global network topology, and its impact on parallel data transfers. Most often, Grid systems rely on manual configuration [CD98], which is very error prone.

Some tools [dBKB02, BCW] rely on the `traceroute` tool to get informations about the network topology. This program computes the network path to a remote host by sending packets with increasing TTL (time to live). Most routers give their address in the error message they return when a packet dies, allowing `traceroute` to list the hosts traversed by packets on the path.
This approach suffers of several drawbacks: First, since routers can return different addresses,

combining the paths can be non-trivial. Then `traceroute` gives no information on how concurrent transfers interact when sharing a given link, neither even the available bandwidth. Finally, the informations given by `traceroute` are at level 3 of the OSI standard, while the applications are on level 4 and upper. Thus, the reality captured by this program may differ from the one experienced by applications.

`pathchar` uses the same methodology as `traceroute`, changing not only the packets TTL, but also their size. If enough measurements are done, it can statistically compute the latency and bandwidth of each link in the path [Dow99], which would almost fulfill our purposes. Sadly, this tool is not usable in our context for several reasons. First, `pathchar` takes a very long time to get the characteristics of a given link (some of our experiments on a path constituted of 4 links took up to one hour), which decreases the usability of this tool on very large platforms. Then, in order to forge the packets needed for its experiments, `pathchar` needs to be given the super-user privileges on the machines where it runs, which is clearly unacceptable in Grid context. Finally, it only gives the bandwidth of encountered links and not how they will be shared between several competing data streams.

In LAN management context, SNMP (Simple Network Management Protocol [BJ00]) is often used to gather information about the network. On the other hand, BGP (Border Gateway Protocol [RL95]) is used to exchange routing informations between autonomous systems of the Internet, and could be used to get information about WAN. The Remos [DGK+01] tool uses the SNMP protocol to construct the local network topology, plus some simple benchmarks to gather informations about WAN [MS00]. But we believe that none of these approaches is satisfying in our context. The main problem is that the gathering of informations with these protocols is almost always restricted to authorized users. This is due to two major reasons: security, since it is possible to conduct Deny Of Service attacks by flooding the network of requests, and privacy, since ISP generally do not like to publicly expose the possible bottlenecks of their networks.
Moreover, even when these informations are available, we believe that they do not fulfull our expectations, because they give no information about how a given network resource will be shared when two streams try to use it.

Contrary to the SNMP- or BGP-based approach, we decided not to get the network configuration from where it is expressed by administrators, but rather to conduct some experiments to *feel* its effects. Our approach differs from the `traceroute` one, because we do not focus on the path followed by level 3 packets, but on interactions between application-level streams. Finally, contrary to `pathchar`, we do not try to get all possible informations about the underlying network, but only the ones we need to predict application-level performance. This allows us to map a given network more quickly, and without any special authorization or privilege.

This approach is not completely new, and was first presented in ENV [SBW99]. Although if very innovating, the ENV project suffers from a number of drawbacks. First, to reduce the number of tests needed, ENV does not try to give a complete map of the network, but only the view a given machine (the so-called *master*) has. It is well adapted to the master/slave paradigm, but is clearly not general enough. Mapping completely the network using the ENV methodology may require N^4 tests, with N being the number of hosts in the network, and each test during up to a minute, which is not really scalable. Moreover, ENV requires a recent version of the python scripting language to be deployed on all machines of the testbed, which is a too strong requirement for a Grid application. Finally, ENV is very sensible to external

network load during its experimentations.

3 Which topology for which use

Before presenting how ALNeM tries to gather the information, this section presents more in details the information that we are trying to obtain, and why. Even if we hope this tool to be useful for other purposes, we designed it to help us for two projects in which we are involved.

First, the SimGrid [CLM03] grid platform simulator needs something like a platform catalog, which the user could choose, and run his simulated application onto. It would naturally be possible to generate random platforms, but since the *realism* of a given graph as network platform is so hard to qualify [PF97, CDZ97, FFF99], a tool to capture real platforms is welcome.

Then, the goal of the FAST [Qui02] library is to provide grid middleware schedulers the informations they need about the platform, both in term of routine requirements (computational time, memory space, communication amount, *etc.*), and platform availability (CPU load, free memory, bandwidth, *etc.*). It relies on the Network Weather Service [WSH99] to obtain the information about the network. Moreover, when each part of a network path is monitored by NWS but not the complete path, FAST tries to approximate the path characteristics by aggregating the values measured for each part. For this approach to work, the deployment of NWS is naturally crucial.
Moreover, it is very important for their accuracy that NWS experiments never collide. If two probe packets are sent on the same link at the same time, each of them will only report half of the available bandwidth. That is why NWS implements a clique protocol to ensure that only one host from a machine set sharing a common resource will conduct a network experiment at the same time [WGT00]. Naturally, this clique protocol is not completely scalable, and it would be a bad idea to run only one clique containing all hosts of the testbed.

For these two reasons, it is a very difficult and error prone task to choose which link will be monitored and which can be recomputed from subparts as well as to determine which experiments set have to conducted under the clique protocol to avoid conflicts. Doing this automatically suppose a very accurate knowledge of the network topology, and its effects on application-level data-streams.

To fulfill both project requirements, we need to get network performance between the machines participating to the grid platform, *i.e.* on which the grid middleware can launch jobs (such machines are called *nodes* in this article). But knowing the end-to-end performance is not enough if we do not know how several data streams will interact with each other. To solve this problem, we need a graph representing the network. Each *node* has to be represented as vertex of this graph, but since the performance of routers and switches are not important at all for us, some routers may be omitted, or some fake routers may be added if it simplifies the process. There is clearly more that one graph fulfilling these criteria, and we search one of them.

For example, if it was possible to run `pathchar` between each pair of nodes and to aggregate the results in a graph, this would be sufficient for us. But as we discussed above, running this program is not possible on a Grid platform. That is why we have to search for such a graph using other methods.

4 ALNeM algorithm

4.1 Overview

Our method is decomposed in two phases. The first one consists in measuring the possible interferences between data streams on the platform while the second builds a graph inducing the effects measured during the first step. Most of the technical difficulties reside in the first phase, since it requires a lot of optimizations. The second phase is theoretically difficult, because it involves solving a difficult graph-theoretic problem.

4.2 Gathering the informations

4.2.1 Presentation and notations

The first step of the algorithm consists in gathering informations about the platform by conducting interference experiments. Given four nodes a, b, c and d, we measure if a data stream from a to b does interfere with another data stream from c to d. This is done by measuring the available bandwidth on (ab) when (cd) is quiet (noted $\boldsymbol{bw(ab)}$), and comparing it to the available bandwidth on (ab) when a data stream saturates (cd) (noted $\boldsymbol{bw_{/\!/\,cd}(ab)}$). If the ratio of the two measured bandwidth for (ab) is equal to 0.5, it means that (ab) and (cd) share a resource ruled by a fair-sharing algorithm. If this ratio equals 1, (cd) have no influence on (ab). In the real world, we have to use arbitrary thresholds since this ratio is rarely equal to 1 or 0.5.

Definition 1. We say that (ab) and (cd) do not interfere if and only if $\frac{bw_{/\!/\,cd}(ab)}{bw(ab)} > 0.9$. In that case, we use the notation $(ab) \,/\!/_{rl}\, (cd)$ (rl stands for real life).

Definition 2. If this ratio is below 0.7, we say that both transfer do interfere, and we use the notation $(ab) \between_{rl} (cd)$.

Note that a ratio between 0.7 and 0.9 is supposed to denote an experimental error.

Remark 1. *$\between_{rl}$ is a symmetric relation (i.e. $(ab) \between_{rl} (cd) \Leftrightarrow (cd) \between_{rl} (ab)$ holds for all a, b, c, d). It comes from the fact that each side is possible if and only if (ab) and (cd) share a network element and if the use of this element by one stream limits the other stream.*

Remark 2. *The equivalence $(ab) \between_{rl} (cd) \Leftrightarrow (ba) \between_{rl} (cd)$ is wrong in the general case, because network routes are not always symmetric on Internet.*

We implicitly supposed that running two streams were enough to detect all interferences, but this is not the case since some network resources may become limiting (thus creating the interference) only when shared between 3 or more streams. This leads us to enhance our notion of interference: a *N-interference* is an interference is revealed by the use of N simultaneous streams. When unspecified, we speak of 2-interference in this paper.

The information we want to gather can be stored in a 4-dimensional matrix defined as:

Definition 3. Let $I(V, \between_{rl})$ be the interference matrix between all elements of a set V as induced by the interference relation $\between_{rl}$:

$$I(V, \between_{rl})(a, b, c, d) = \begin{cases} 1 & \text{if } (ab) \between_{rl} (cd) \\ 0 & \text{otherwise} \end{cases}$$

Remark 3. *The matrix I is symmetric due to the remark 1 and thus only contain* $\frac{N^4}{2}$ *useful cells.*

4.2.2 Naive algorithm

The simplest way to gather this information will take N^4 steps, each of them consisting for (a, b, c, d) in V^4:

1. measuring $bw(ab)$
2. measuring $bw_{/\!/cd}(ab)$.
3. computing the ratio.

Both steps (1) and (2) have to be long enough to let the network stabilize: in (1), we have to wait until the previous experiment really ends to avoid a possible perturbation, and then, we have to wait until (cd) saturate the link until we can go for step (2). It seems reasonable to expect that each step can be run in half a minute or a minute, which allows the complete algorithm to run in $\frac{N^4}{2}$ minutes, or $\frac{N^4}{4}$ minutes. If $N = 20$, this represents 25 or 50 *days*.

4.2.3 ALNeM algorithm

In order to speed things up, ALNeM does some of the tests in parallel. The idea here is to consider independent edges. Since they do not interfere with each other, we can conduct saturation experiments on them in parallel. That way, each time we start a new transfer, we do not test it against only one other edge, but against all already started experiments.

If starting a experiment on one new edge leads to interferences with an already running experiment, we indicate this interference in the computed matrix, remove the lastly added edge, and continue. If the new edge does not interfere with already running ones, we try adding a new one. This process may even discover some N-interference that way, but we only conduct the needed experiment to search 2-interferences.

The more independent edges we start at the same time, the better parallelism we achieve. In order to guess which edges should be independent, we run `traceroute`s between all nodes and merge them as a graph. Note that the resulting graph will contain errors, for example because of Virtual LANs which are not detected by this tool. That is to say that it may report two edges as probably independent when they actually interfere with each other. Anyway, this a good guess about independent edges to lead our experiments.

The obtained speedup naturally depends on the graph characteristics. The worst case is when all edges interact with all others. In that case, no test can be done in parallel, and we still need N^4 steps. The best case is when there is no interaction at all. In that case, we can run in $2N^2$ steps. Since the target platform is a constellation of local networks, which in turn are often trees, the expected gain of this parallelization is high.

Since this algorithm needs a centralized clock to run well, the efforts of sensors are coordinated by a special node called *maestro*. The position of this node has no influence on the resulting mapping. The pseudo code for sensors and maestro is given in Appendix A.

4.3 Reconstructing a graph from these informations

4.3.1 Problem presentation and formalization

In order to model the interferences, we decided to search for a hybrid graph, counting two different kinds of vertices: *nodes*, being machines participating to the Grid platform, and *separators*, representing the contention spots of the network. That way, our algorithm can focus on vertices, and if they are connected or not. All edges have the same signification and no associated value since the network characteristics are ported by the separator edges. Moreover, separator characteristics can capture not only the link abilities, but also the delay encountered in routers and switches.

The problem is thus to construct a graph containing all nodes and some separators which would induce the interference effects measured during the first phase. To formalize this problem further, some additionnal notations are needed:

Let $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ be the graph representing the idealistic real topology, with $\widetilde{V}$ being the set of all real machines (nodes, routers and switches) and $\widetilde{E}$ the existing links between them.

Let $\mathcal{H}$ be the set of nodes (without routers or switches).

Definition 4. Let $u, v \in \widetilde{V}$: we use the notation $\left(u \underset{\widetilde{G}}{\longrightarrow} v\right)$ to denote the ordered set of *vertices* belonging to the route from u to v (u and v are respectively the first and the last elements of this set). This notation represents *the* routing on the graph $\widetilde{G}$.

Definition 5. Let $a, b, c, d \in \mathcal{H}$: we define $\curlywedge_{\widetilde{G}}$ by

$$(ab) \curlywedge_{\widetilde{G}} (cd) \iff \left(a \underset{\widetilde{G}}{\longrightarrow} b\right) \cap \left(c \underset{\widetilde{G}}{\longrightarrow} d\right) \neq \emptyset$$

Remark 4. *The previous definition does not match exactly the experimental Definition 2 since the capacity of routers and links are not taken into account. Using a more idealistic definition should help to gain some intuition on the problem.*

The problem is then the following:

Definition 6. InterferenceGraph: Given $\mathcal{H}$ and $I(\mathcal{H}, \widetilde{G})$, find a graph $G(V, E)$ and a routing in this graph such that:

$$\begin{cases} \mathcal{H} \subset V \\ I(\mathcal{H}, \curlywedge_G) = I(\mathcal{H}, \curlywedge_{\widetilde{G}}) \\ |V| \text{ is minimal.} \end{cases} .$$

As time of writing, we still don't know whether this problem is NP-complete or not. Our feeling is that it is NP-complete in the general case, but that good approximation schemes exist and that it is solvable in a polynomial time under some assumptions about the underlying $\widetilde{G}$ graph and the routing it uses.

The rest of this section presents some mathematical tools that help designing a polynomial algorithm to find a small graph whose interference matrix is equal to $I(\mathcal{H}, \curlywedge_{\widetilde{G}})$ under some hypothesis.

We will suppose in the rest of the paper that $\widetilde{G}$ respect the following hypotheses:

Hypothesis 1 (Routing hypothesis). $\forall (a,b,c) \in \widetilde{V}$:

$$c \in \left(a \underset{\widetilde{G}}{\longrightarrow} b\right) \Rightarrow \left(a \underset{\widetilde{G}}{\longrightarrow} b\right) \cap \left(a \underset{\widetilde{G}}{\longrightarrow} c\right) = \left(a \underset{\widetilde{G}}{\longrightarrow} c\right)$$

This hypothesis states that the routing used in $\widetilde{G}$ does not present weird inconsistencies. If a node c is encountered on the path from a to b, packets routed from a to c will follow the same beginning path than when traveling from a to b. The contrary would imply for example that a use a machine ρ as gateway, and that ρ routes packets for b through c while a can connect to c directly using another network and not the one offered by ρ.

Hypothesis 2 (Symmetry hypothesis). $\forall (a,b) \in \widetilde{V} : \left(a \underset{\widetilde{G}}{\longrightarrow} b\right) = \left(b \underset{\widetilde{G}}{\longrightarrow} a\right)$

This hypothesis states that the network routes are symmetric.

We know that these hypothesis are violated in the reality of Internet, since any imaginable routing inconsistency do exist [Pax97]. But even if the current version of our algorithm cannot deal with these problems, it can detect them, allowing us to propose some special handling in future versions.

4.3.2 Notations and preliminary lemma

Definition 7. For sake of simplicity in the following proof, we introduce the following notation:

$$\boldsymbol{a \perp b} \Longleftrightarrow \forall (u,v) \in \mathcal{H} \setminus \{a,b\}, (au) \curlywedge_{\widetilde{G}} (bv)$$

We trivially have $a \perp b \Leftrightarrow b \perp a$.

Lemma 1 (Separation). *For all $a, b \in \mathcal{H}$, we have:*

$$a \perp b \textit{ iff there exists } \rho \in \widetilde{V} \textit{ such that } \forall z \in \mathcal{H} : \rho \in \left(a \underset{\widetilde{G}}{\longrightarrow} z\right) \cap \left(b \underset{\widetilde{G}}{\longrightarrow} z\right).$$

Proof. For sake of clarity we will denote by $(v \rightarrow w)$ the set $\left(v \underset{\widetilde{G}}{\longrightarrow} w\right)$.

$\boxed{\Rightarrow}$ Suppose that $a \perp b$. Let u be in $\mathcal{H} \backslash \{b,a\}$ and u_a be the first node distinct of a, appearing in $(a \rightarrow u) \cap \mathcal{H}$.

Let ρ be the first vertex in $(a \rightarrow u_a) \cap (b \rightarrow u_a)$. Note that $\rho \in \widetilde{V}$ (see Figure 1).

Let's show that u_a is the first node distinct of a and b, in $\mathcal{H}$, appearing in $(b \rightarrow u_a)$.

Proof. Using Hypothesis 1, we have the following relations:

$$\begin{cases} (a \rightarrow u_a) = (a \rightarrow \rho) \cup (\rho \rightarrow u_a) \\ (b \rightarrow u_a) = (b \rightarrow \rho) \cup (\rho \rightarrow u_a) \end{cases}$$

- If $\rho = a$ then there cannot be any $c \in \mathcal{H} \backslash \{b,a\}$ such as $c \in (b \rightarrow a)$ (otherwise, we would have $bc \mathbin{/\!\!/_{\widetilde{G}}} au_a$). Thus u_a is the first node of $\mathcal{H}$ distinct of a appearing in $(b \rightarrow u_a)$.

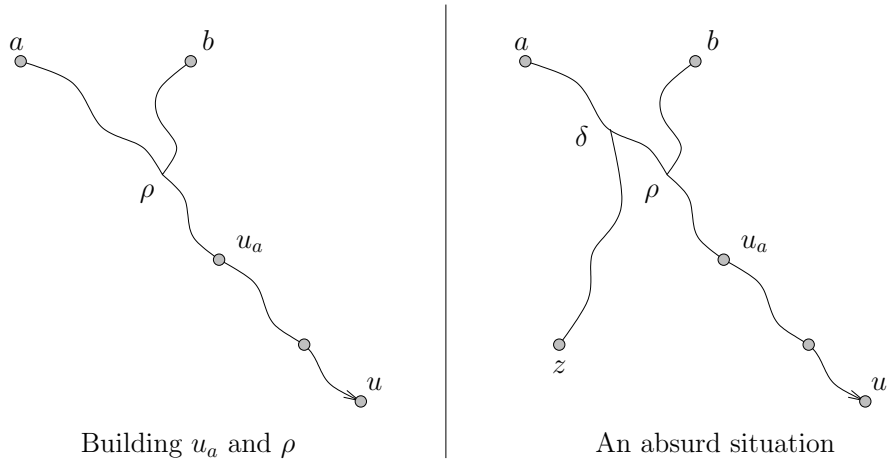


Figure 1: Sketch of the proof of Lemma 1

– If $\rho \neq a$, then we have $(\rho \rightarrow u_a) \cap \mathcal{H} = \{u_a\}$ thanks to the definition of u_a. There cannot be any $c \in \mathcal{H} \backslash \{b\}$ such as $c \in (b \rightarrow \rho)$. Indeed, otherwise we would have $b \rightsquigarrow c \rightsquigarrow \rho \rightsquigarrow u_a$, $a \rightsquigarrow \rho \rightsquigarrow u_a$. And as $(b \rightarrow \rho) \cap (a \rightarrow u_a) = \{\rho\}$ (definition of ρ) we would have $bc \not\parallel_{\bar{G}} au_a$, which is absurd. u_a is then the first node distinct of a and b, in $\mathcal{H}$, appearing in $(b \rightarrow u_a)$. □

Now, let's prove that for all z : $\rho \in (a \rightarrow z)$.

Proof. Suppose that there exists a z such that $\rho \notin (a \rightarrow z)$. Using Hypothesis 1, we have the following relation:

$$(a \rightarrow z) \cap (\rho \rightarrow u_a) = \emptyset$$

Then let δ be the first node in $(a \rightarrow z) \cap (a \rightarrow u_a)$. We know that $\delta \in (a \rightarrow \rho)$ and $(\delta \rightarrow z) \cap (\rho \rightarrow u_a) = \emptyset$ (using Hypothesis 1).

Thanks to the definition of ρ, we have $\delta \notin (b \rightarrow \rho)$. Therefore, $(a \rightarrow z) \cap (b \rightarrow u_a) = \emptyset$, which is absurd since $a \perp b$. □

Using the same arguments, we can show that $\rho \in (b \rightarrow z)$, hence the result:

$$\forall z \in \mathcal{H} : \begin{cases} \rho \in (a \rightarrow z) \\ \rho \in (b \rightarrow z) \end{cases}$$

$\boxed{\Leftarrow}$ This implication is obvious. Let $\rho \in \widetilde{V}$ such as

$$\forall z \in \mathcal{H} : \begin{cases} \rho \in (a \rightarrow z) \\ \rho \in (b \rightarrow z) \end{cases}$$

Then for all $u, v \in \mathcal{H} : \rho \in (a \rightarrow u) \cap (b \rightarrow v)$, therefore we have $a \perp b$. □

Remark 5. *The proof of Lemma 1 does not rely on Hypothesis 2.*

Definition 8. For a and $b \in \mathcal{H}$, a node ρ such as the one given by Lemma 1 is called a *separator* of a and b.

Lemma 2 (Transitivity).

$$\forall a, b, c \in \mathcal{H} : a \perp b \text{ and } b \perp c \Rightarrow a,b \text{ and } c \text{ have the same separator and then } a \perp c.$$

Proof. Let a, b, c be in $\mathcal{H}$ and suppose that $a \perp b$ and $b \perp c$. Let $u \in \mathcal{H}$ and u_a be the first node distinct of a, in $\mathcal{H}$, appearing in $(a \rightarrow u)$. Let ρ be the first node in $(a \rightarrow u_a) \cap (b \rightarrow u_a)$ and σ be the first node in $(b \rightarrow u_a) \cap (c \rightarrow u_a)$.
ρ is a separator of a and b and σ is a separator of b and c (using Lemma 1). Then we have $\rho \in (b \rightarrow c)$ and $\sigma \in (b \rightarrow c)$. Suppose that $\rho \neq \sigma$.

- if $\rho \in (b \rightarrow \sigma)$ then let $d \in \mathcal{H}$ be different from a, b and c. Let's show that $(b \rightarrow a) \not\parallel_{\tilde{G}} (c \rightarrow d)$. We need to show that $(b \rightarrow \rho \rightarrow a) \cap (c \rightarrow \sigma \rightarrow d) = \emptyset$. Using Hypothesis 2, we have $(\rho \rightarrow a) = (a \rightarrow \rho)$ and $(\sigma \rightarrow c) = (\sigma \rightarrow c)$, therefore $(\rho \rightarrow a) \cap (c \rightarrow \sigma) = \emptyset$ and $(b \rightarrow \rho) \cap (c \rightarrow \sigma) = \emptyset$ (there is no loop in a path: Hypothesis 1). If we had $(a \rightarrow \rho) \cap (\sigma \rightarrow d) \neq \emptyset$, then $(a \rightarrow d)$ would not contain ρ. If we had $(\rho \rightarrow b) \cap (\sigma \rightarrow d) \neq \emptyset$, then $(b \rightarrow d)$ would not contain σ. Therefore, we have $(b \rightarrow a) \not\parallel_{\tilde{G}} (c \rightarrow d)$, which is absurd.
- if $\rho \in (\sigma \rightarrow c)$ then $(c \rightarrow u_a) = (c \rightarrow \rho \rightarrow u_a)$ and $\sigma \notin (c \rightarrow u_a)$, which is absurd.

Then we have $\rho = \sigma$ and ρ is then a separator for a and c. □

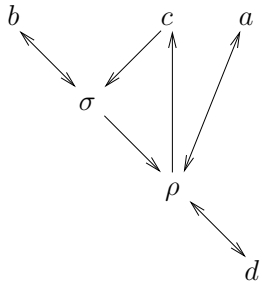


Figure 2: Lemma 2 counter-example when Hypothesis 2 is not verified.

Remark 6. *Lemma 2 does not hold if the Hypothesis 2 is not verified. Figure 2 depicts a situation where $a \perp b$ and $b \perp c$ but $a \not\perp c$.*

Remark 7. *$\perp$ is an equivalence relation.*

Lemma 3 (Leader). *Let $\mathcal{C}$ be an equivalence class for $\perp$ and ρ a separator for these nodes.*

$$\forall a \in \mathcal{C}, \forall b, u, v \in \mathcal{H} : (a, u) \curlywedge_{\tilde{G}} (b, v) \Leftrightarrow (\rho, u) \curlywedge_{\tilde{G}} (b, v)$$

Proof. Let ρ be a separator for the nodes of $\mathcal{C}$ (using Lemma 2). Let $a \in \mathcal{C}$ and $b, u, v \in \mathcal{H}$.

- Suppose that $(a \to u) \cap (b \to v) \neq \emptyset$ and $(\rho \to u) \cap (b \to v) = \emptyset$. Then we have $(a \to \rho) \cap (b \to v) \neq \emptyset$. Let θ be the first node of $((a \to \rho) \setminus \{\rho\}) \cap (b \to v)$. The path from a to v is then $(a \to \theta \to v)$ and ρ does not belong to it, which is absurd. Therefore we have:

$$(a \to u) \cap (b \to v) \neq \emptyset \Rightarrow (\rho \to u) \cap (b \to v) \neq \emptyset$$

- Suppose that $(a \to u) \cap (b \to v) = \emptyset$. Then we have $(a \to \rho \to u) \cap (b \to v) = \emptyset$ and therefore $(\rho \to u) \cap (b \to v) = \emptyset$.

Thus we have:

$$(a, u) \curlywedge_{\widetilde{G}} (b, v) \Leftrightarrow (\rho, u) \curlywedge_{\widetilde{G}} (b, v)$$

□

Remark 8. *The proof of Lemma 3 does not rely on Hypothesis 2. It only requires that there is a common separator for a group of nodes and that these nodes are all* $\curlywedge$.

4.3.3 Reconstruction algorithm

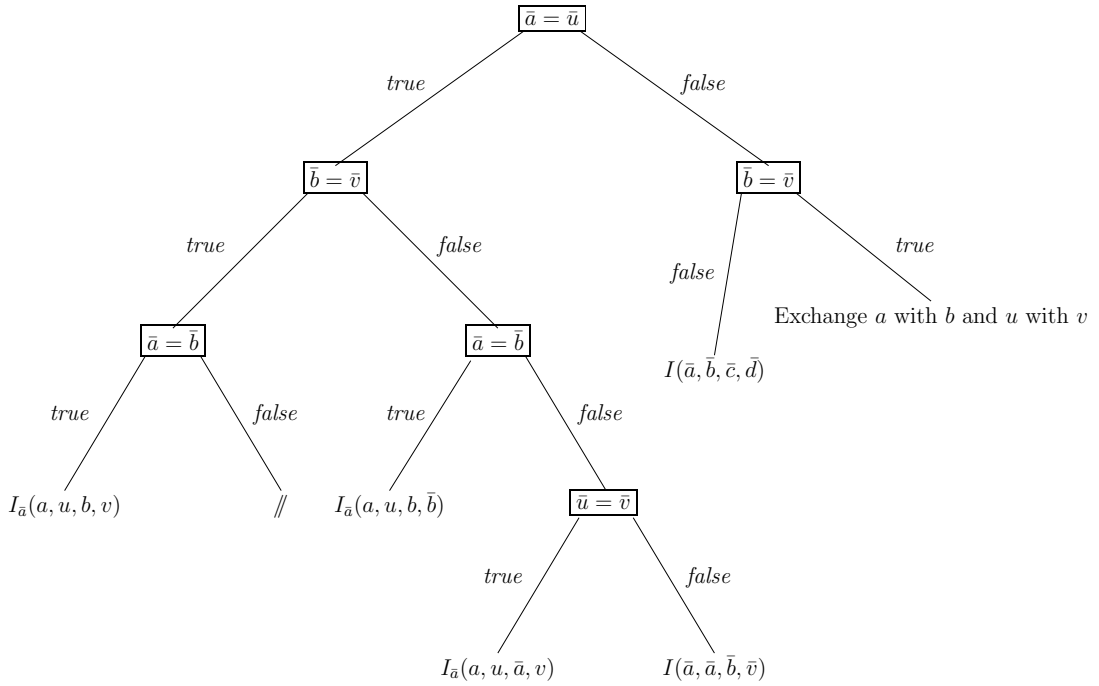
We will now present an algorithm building a graph inducing the same interference matrix I than the one of $\widetilde{G}$. This algorithm is not guaranteed to find such a graph in any case, and we will discuss later the conditions in which it succeeds and the conditions in which it fails.
TREE$(V, I(V, \curlywedge_{\widetilde{G}}))$

1. $\mathcal{C}_0 \leftarrow V$ and $i \leftarrow 0$.
2. $E_0 \leftarrow \emptyset$; $V_0 \leftarrow \emptyset$
3. Let $h_1, \ldots, h_p$ be the equivalence classes of $\curlywedge_{\mathcal{C}_i}$. They can easily be computed (with a greedy algorithm) from $I(\mathcal{C}_i, \curlywedge_{\widetilde{G}})$. For each h_i, select a leader l_i. $\mathcal{C}_{i+1} \leftarrow \{l_1, \ldots, l_p\}$ and $I(\mathcal{C}_{i+1}, \curlywedge_{\widetilde{G}})$ is computed from $I(\mathcal{C}_i, \curlywedge_{\widetilde{G}})$.
4. $V_{i+1} \leftarrow V_i$; $E_{i+1} \leftarrow E_i$
   For all $h_j \in \mathcal{C}_i$: $\begin{cases} \forall v \in h_j : E_{i+1} \leftarrow E_{i+1} \cup \{(v, l_j)\} \\ V_{i+1} \leftarrow V_{i+1} \cup \{v\} \end{cases}$
5. Repeat step 3 and 4 until $\mathcal{C}_i = \mathcal{C}_{i+1}$.

Theorem 1. *When the algorithm does finish with one leader, the resulting graph* G *satisfies* $I(\mathcal{H}, \curlywedge_G) = I(\mathcal{H}, \curlywedge_{\widetilde{G}})$ *with a shortest path routing.*

Proof. Let $a, u, b, v \in V$ be four distinct nodes.

We note $\bar{x}$ the leader of the connected component in (V, E_i) containing x. One can prove by recurrence that, at step i, $I(V, \curlywedge_{\widetilde{G}})(a, u, b, v)$ can be determined using the following decision tree (where $I_{\bar{a}}(x, y, z, t) = I(V, \curlywedge_{(V, E_i)})(x, y, z, t)$):



Thus, when (V_i, E_i) is connected, the resulting interference matrix is equal to the original one. □

4.3.4 Working out an example

In this section, we run the algorithm on a graph generated with Tiers [Doa96] (see Figure 3).

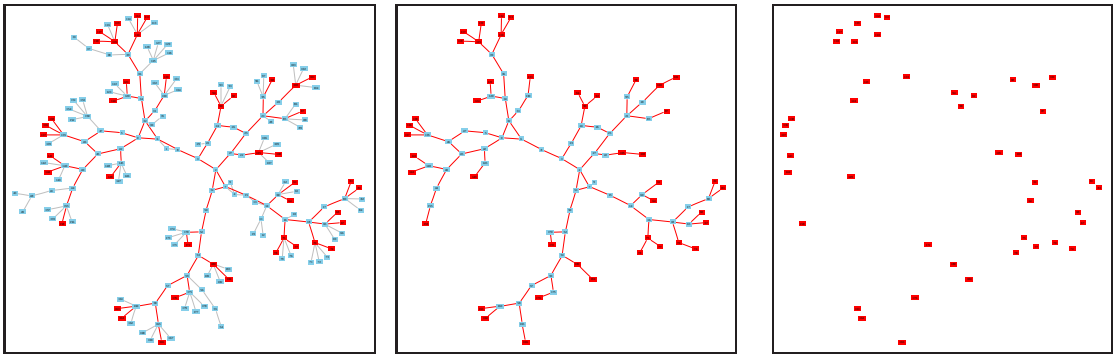
Hypothesis 3 (Graph structure hypothesis). $\widetilde{G}$ is a tree, or a constellation of trees where all root tree are fully connected and do not interfere one with another (*ie*, are connected by a clique).

This hypothesis seem acceptable for most of the Grid testbeds deployed in the reality, and we are working on relaxing this in order to allow $\widetilde{G}$ to be a more general graph.

Some graph structures seem to be easy to identify, only according to the interference matrix. Trees can be detected, using the $\curlywedge$ relation. Cliques can be detected using the following relation : $a /\!\!/_T b$ iff $\forall u, v \in T : au /\!\!/_{\widetilde{G}} bv$. There is little hope to prove strong relations on $/\!\!/_T$ like we did on $\curlywedge$ because this relation is much less constraining. Nevertheless, a set T such as $\forall a, b : a /\!\!/_T b$ can easily be conceived as a clique. Once a clique has been detected, the gateway (if it is unique) can be identified using $\curlywedge$ and the whole clique can the be replaced by the gateway. That is why on graphs fulfilling hypothesis 3's requirements, InterferenceGraph can easily be solved by alternating tree detection and clique detection. Detection and handling of cycles is subject of future work.

5 Conclusion

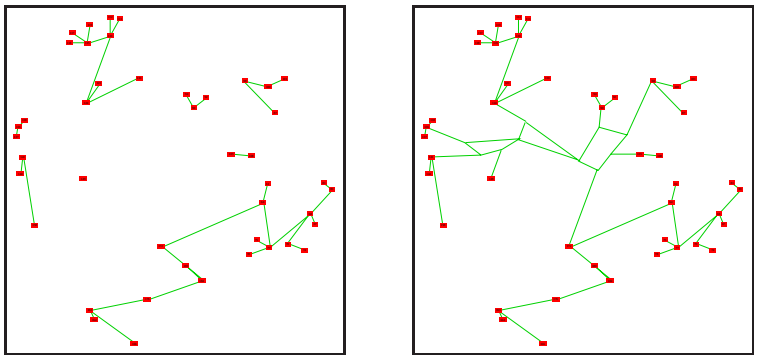
All the work presented in this paper is still ongoing, and deserves more efforts to be fully satisfying. We plan to tackle the following problems in the future:



Original graph

Machine nodes and involved routers

Input of the reconstruction algorithm (along with the interference matrix)



Output of the current reconstruction algorithm

Output of the ideal reconstruction algorithm (cycles have been identified)

Figure 3: An example of graph reconstruction

First, we would like to limit even further the number of needed tests during the information gathering phase. We could for example try to deduce that a whole set of related information given by `traceroute` is right by testing only some of them and not all.

Then, we would like to handle more generic underlying graphs $\widetilde{G}$ than the ones specified in hypothesis 3. We have some ideas to detect and handle properly cycles in the graphs, but this point definitively deserves some more theoretical work. We would also like to improve the detection of routing inconsistency violating hypothesis 1, and provide a better handling than just informing the user that this problem prevents us to map the network. Another possible relaxation to our hypotheses would be to consider the case that streams use QoS and are therefore not equally served by resources.

We also plan to extend this algorithm to an incremental version. Indeed, nodes of a grid platform can fail and quit the testbed while new nodes can join the platform after the beginning of the experiment. In such a case, it would be pleasant to not have to recompute the whole interference graph each time. If possible, we would also like to develop a version of these methods based only on locally available knowledge, and remove the need of a central maestro orchestrating the experiments. Even if very appealing, this last extension seems however very difficult.

Naturally, we also plan to actually implement this algorithm, both on top of the SimGrid simulator, to test it on some special cases, and in the real life to test it in real condition and prove its usability. We would like to use the simulator to compare the behavior of some classical communication patterns on the real topology and on the topology discovered by ALNeM, and experimentally show that this tool captures the right informations for our purpose.

References

[BCW] H. Burch, B. Cheswick, and A. Wool. Internet mapping project. http://www.lumeta.com/mapping.html.

[BJ00] A. Bierman and K. Jones. Physical topology mib. RFC2922, september 2000.

[CD98] H. Casanova and J. Dongarra. Using Agent-Based Software for Scientific Computing in the Netsolve System. *Parallel Computing*, 24:1777–1790, 1998.

[CDZ97] Kenneth L. Calvert, Matthew B. Doar, and Ellen W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997. Available at http://citeseer.nj.nec.com/calvert97modeling.html.

[CLM03] Henri Casanova, Arnaud Legand, and Loris Marchal. Scheduling distributed applications : the SimGrid toolkit. In *Proceedings of the IEEE Symposium on Cluster Computing and the Grid (CCGrid'03)*. IEEE Computer Society, May 2003. Accepted for publication.

[dBKB02] Mathijs den Burger, Thilo Kielmann, and Henri E. Bal. TOPOMON: A monitoring tool for grid network topology. In *International Conference on Computational Science (ICCS 2002)*, volume 2330, pages 558–567, Amsterdam, April 21-24 2002. LNCS.

[DGK+01] P. Dinda, T. Gross, R. Karrer, B Lowekamp, N. Miller, P. Steenkiste, and D. Sutherland. The architecture of the remos system. In *10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10)*, August 2001.

[Din02] Peter Dinda. Online prediction of the running time of tasks. *Cluster Computing*, 5(3), 2002.

[Doa96] Matthew B. Doar. A better model for generating test networks. In *Globecom '96*, Nov 1996. Available at http://citeseer.nj.nec.com/doar96better.html.

[Dow99] Allen B. Downey. Using pathchar to estimate internet link characteristics. In *Measurement and Modeling of Computer Systems*, pages 222–223, 1999.

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999. Available at http://citeseer.nj.nec.com/faloutsos99powerlaw.html.

[FK98] I. Foster and C. Kesselman. The Globus project: A status report. In *Proceedings of the Heterogeneous Computing Workshop*, pages 4–18, 1998.

[MS00] Nancy Miller and Peter Steenkiste. Collecting network status information for network-aware applications. In *INFOCOM'00*, pages 641–650, 2000.

[Pax97] Vern Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California, Berkeley, 1997. Available at ftp://ftp.ee.lbl.gov/papers/vp-thesis/dis.ps.gz.

[PF97] Vern Paxon and Sally Floyd. Why we don't know how to simulate the internet. In *Proceedings of the Winder Communication Conference*, December 1997. Available at http://citeseer.nj.nec.com/article/floyd99why.html.

[Qui02] Martin Quinson. Dynamic performance forecasting for network-enabled servers in a metacomputing environment. In *International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS'02)*, April 15-19 2002.

[RL95] Y. Rekhter and T. Li. A border gateway protocol 4 (bgp-4). RFC1771, March 1995.

[SBW99] Gary Shao, Francine Berman, and Rich Wolski. Using effective network views to promote distributed application performance. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, June 1999. Available at http://apples.ucsd.edu/pubs/pdpta99.ps.

[WGT00] Rich Wolski, Benjamin Gaidioz, and Bernard Tourancheau. Synchronizing network probes to avoid measurement intrusiveness with the Network Weather Service. In *9th IEEE High-performance Distributed Computing Conference*, pages 147–154, August 2000.

[WSH99] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computing Systems, Metacomputing Issue*, 15(5–6):757–768, Oct. 1999.

A Pseudo-code of the first phase

Let E be the set of all possible host pairs. For sake of simplicity, we denote $\left(a \underset{\widetilde{G}}{\longrightarrow} b\right)$ by $(a \to b)$.

SensorCode()
do forever:
wait request from maestro
switch on request:
case "traceroute to node a":
Do this traceroute and send the result back to the maestro
case "test connectivity to node a during n seconds":
Try to saturate the link to a during n seconds
Compute the achieved bandwidth during that time and send it to the maestro

MaestroCode(E)
Traceroute step:
do for each host pair a and b
Ask a the traceroute to b, and b the traceroute to a
wait all the traceroute results
Initialization:
$T \leftarrow \emptyset$. *(Tested pairs. By construction,* $\forall a, b \in T, a \not\parallel_{rl} b$*)*
$C \leftarrow \emptyset$. *(Conflicting pairs,* $\forall c \in C, \exists t \in T / c \not\asymp_{rl} t$*)*
$A \leftarrow E$. *(Available pairs, ie not currently tested)*
Iteration:
while Some info are missing
(search an experiment to start)
Let $H = \left\{ (a \to b) \in A \;\middle/\; \begin{array}{l} a \text{ and } b \text{ are not an endpoint of any pair in } T \\ (a \to b) \text{ is not known to interfere with any pair in } T \\ \text{we need more info about } (a \to b) \end{array} \right\}$
if $H \neq \emptyset$
Choose $h \in H$ so that according to traceroute, $\forall t \in T, h \not\parallel_{rl} t$
If no such h exists in H, take any h in H
$A \leftarrow A \setminus h$; $T \leftarrow T \cup h$
(Make the experiment)
$\forall (a \to b) \in T$, ask a the connectivity to b for n seconds ; Gather sensors' answers
(Conclude from measurements)
if $\exists t \in T /$ the bandwidth of t decreased since last iteration
$T \leftarrow T \setminus h$; $C \leftarrow C \cup h$
for each such t, do remember that it banned on (more) pair to C *(see below)*
for all $t \in T$, **if** t had a stable bandwidth, **then** $t \not\parallel_{rl} (a \to b)$ **else** $t \not\asymp_{rl} (a \to b)$
else
Let t be the element of T which banned the most elements to C
$T \leftarrow T \setminus \{t\}$; $A \leftarrow A \cup \{t\}$
Move from C to A all pairs which were banned by t