An Application-Level Network Mapper

Arnaud Legrand[1] Frédéric Mazoit[2] Martin Quinson[1].

1: ID – UMR 5132 (CNRS – INPG – INRIA – UJF), Grenoble, France.
2: LIP – UMR 5668 (CNRS – ENS-Lyon – UCBL – INRIA), Lyon, France.

IPDPS 2005

Outline

Introduction (1/2)

Motivation

- Modern platforms (Grid, P2P systems) heterogeneous and dynamic.
- Distributed applications have to be reactive and network-aware.

- Quantitative information (bandwidth) well studied [NWS, RPS, ganglia].
- Qualitative information (topology) seldom known, but needed for:
  - Host siting and automatic configuration
  - Group communication

Definitions of topology

Almost as many as layers in the OSI model.

- Physical interconnexion map (wires in the walls)
- Routing infrastructure (path of network packets, from router to switch)
- Application level (focus on effects – bandwidth, latency – not causes)

Introduction (2/2)

Our context is at application level

Grid or P2P systems = multi-organization platforms.

- System heterogeneity ⇒ cannot rely on specific system feature
- Trust issue ⇒ no privileges for grid administrators ("root" or other)

Our Goal is...

Discover What Applications can Expect from the Platform

Given 4 hosts (a, b, c, d), determine whether $a \longrightarrow b$ impact $c \longrightarrow d$ (perfs).

Intuition: if they share a link, they share the bandwidth.

Our goal is not...

- Discover network bottleneck and configuration issues
- Discover packet paths

Topology discovery methodologies

State of the art

Method	Restriction	Focus	Routers	Notes
SNMP	authorized	path	all	passive, LAN
traceroute	ICMP	path	all	level 3 of OSI
pathchar	root	path	all	link bandwidth, slow
Other tomography	no	path	$d_{in} \neq d_{out}$	tree bipartite [Rabbat03]
ENV	no	interference	some	tree only
ALNeM	no	interference	?	complete graph

Outline

Model used

Definition: routed graph $G = (V, E, r)$

Non-oriented graph with *routing function* $(r : V \times V \to V)$.

$\left(u \underset{G}{\longrightarrow} v\right)$ is the path (set of vertices encountered in the graph G).

Definition: (ab) interfere with (cd) in G

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

Symmetric relation: $(ab) \between_G (cd) \Leftrightarrow (cd) \between_G (ab)$

Routing not symmetric: $(ab) \between_G (cd) \not\Leftrightarrow (ab) \between_G (dc)$

Definition: (ab) does not interfere with (cd) in G

$$(ab) /\!/_G (cd) \iff \neg\Big((ab) \between_G (cd)\Big)$$

Measurement methodology

Notation

$bw(ab)$: bandwidth on $a \longrightarrow b$.

$bw_{/\!/ cd}(ab)$: bandwidth on $a \longrightarrow b$ when $c \longrightarrow d$ is saturated.

Definition of the **measured interference**

$(ab) \asymp_{mes} (cd) \iff \frac{bw_{/\!/ cd}(ab)}{bw(ab)} < 0.7$; $(ab) /\!/_{mes} (cd)$ if ratio > 0.9

Not symmetric: $a \xleftrightarrow{\text{10 Mo/s}} c \xleftrightarrow{\text{100 Mo/s}} b \xleftrightarrow{\text{100 Mo/s}} d$ $(ab) \asymp_{mes} (cd)$ and $(cd) /\!/_{mes} (ab)$.

Definition of the **"real" interference** (to reintroduce symmetry)

$$(ab) \asymp_{rl} (cd) \iff \begin{cases} (ab) \asymp_{mes} (cd) \\ (cd) \asymp_{mes} (ab) \end{cases} (or) \iff \neg (ab) /\!/_{rl} (cd)$$

Problem statement

Notations

$\mathcal{H}$: set of nodes

Interference matrix $I(\mathcal{H}, \between_{rl})$:

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

Definition

InterferenceGraph: Given $\mathcal{H}$ and $I(\mathcal{H}, \between_{\tilde{G}})$, find a routed graph $G = (V, E, r)$ such that:

$$\begin{cases} \mathcal{H} \subset V \,; \\ I(\mathcal{H}, \between_{\tilde{G}}) = I(\mathcal{H}, \between_{G}) \,; \\ |V| \text{ is minimal.} \end{cases}$$

Outline

Mathematical tools: Total interference and separators

Definition of the **total interference**

$$a \perp b \Longleftrightarrow \forall (u,v) \in \mathcal{H},\ (au) \between_{rl} (bv)$$

Lemma (Separation)

$a \perp b \Longleftrightarrow \exists \rho \in \widetilde{V} \ \Big/\ \forall z \in \mathcal{H} : \rho \in (a \to z) \cap (b \to z)\,.$

Such a ρ is said to be a separator of a and b.

Theorem: $\perp$ **is an equivalence relation** **(under some assumptions)**

Moreover, $\forall$ equivalence class, $\exists$ common separator for all pair of elements.

Theorem (Representativity)

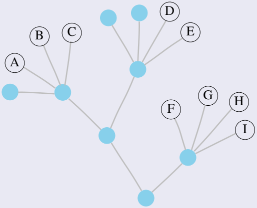
Let $\mathcal{C}$ be an equivalence class for $\perp$ and ρ a separator of its elements.

$$\forall a \in \mathcal{C},\ \forall b,u,v \in \mathcal{H},\ (a,u) \between_{rl} (b,v) \Leftrightarrow (\rho,u) \between_{rl} (b,v)$$

Reconstructing algorithm

Handling trees and cliques of trees

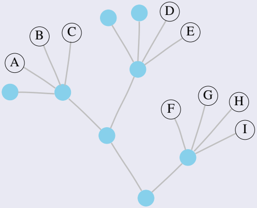
Equivalence class ⇒ greedy algorithm *eating* the leaves.

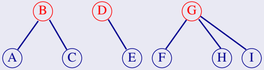


Reconstructing algorithm

Handling trees and cliques of trees

Equivalence class ⇒ greedy algorithm *eating* the leaves.

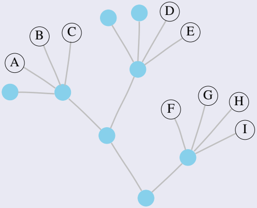


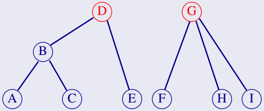


Reconstructing algorithm

Handling trees and cliques of trees

Equivalence class ⇒ greedy algorithm *eating* the leaves.

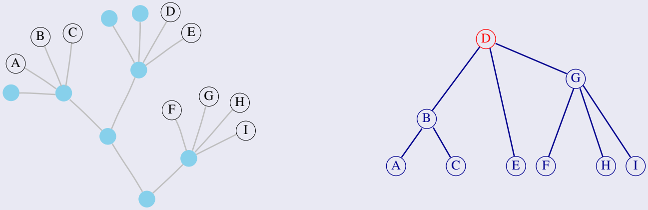




Reconstructing algorithm

Handling trees and cliques of trees

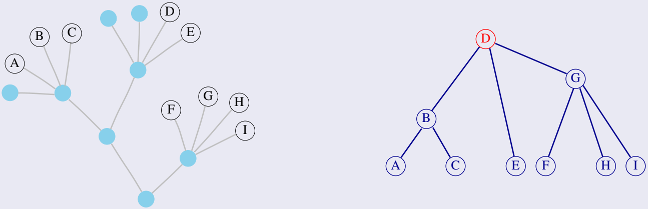
Equivalence class ⇒ greedy algorithm *eating* the leaves.



Reconstructing algorithm

Handling trees and cliques of trees

Equivalence class ⇒ greedy algorithm *eating* the leaves.

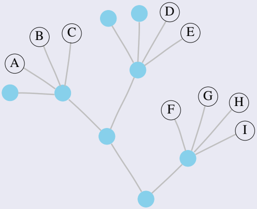


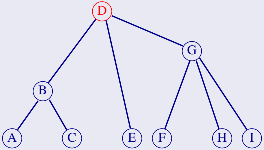
Theorem: When $|C_{\text{inf}}| = 1$, the graph built is a solution.
Theorem: If a tree being a solution exists, $|C_{\text{inf}}| = 1$.
Remark: The graph built is optimal (wrt $|V|$ since $V = \mathcal{H}$)

Reconstructing algorithm

Handling trees and cliques of trees

Equivalence class ⇒ greedy algorithm *eating* the leaves.





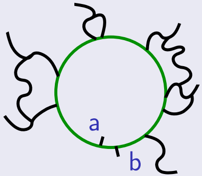
Theorem: When $|C_{\text{inf}}| = 1$, the graph built is a solution.
Theorem: If a tree being a solution exists, $|C_{\text{inf}}| = 1$.
Remark: The graph built is optimal (wrt $|V|$ since $V = \mathcal{H}$)

Theorem: When no interferences in I, clique of C_i is valid solution
Remark: It is also optimal

Reconstructing algorithm: Extension for cycles

Idea

- Find a and b close to each other on a cycle;
- cut the cycle in between;
- iterate previous algorithm;
- reintroduce the cycle: reconnect (a, b).



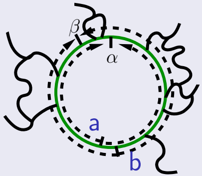
Finding out how to cut

a, b: nodes with the most interferences (*i.e.*, maximizing $\{u, v : au \between bv\}$)

Reconstructing algorithm: Extension for cycles

Idea

- Find a and b close to each other on a cycle;
- cut the cycle in between;
- iterate previous algorithm;
- reintroduce the cycle: reconnect (a, b).



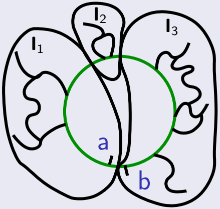
Finding out how to cut

a, b: nodes with the most interferences (*i.e.*, maximizing $\{u, v : au \between bv\}$)

Reconstructing algorithm: Extension for cycles

Idea

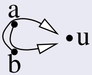
- Find a and b close to each other on a cycle;
- cut the cycle in between;
- iterate previous algorithm;
- reintroduce the cycle: reconnect (a, b).



Finding out how to cut

a, b: nodes with the most interferences (*i.e.*, maximizing $\{u, v : au \between bv\}$)

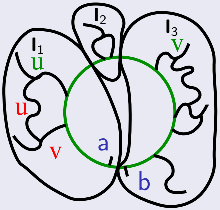
$$\begin{cases} I_1 = \{u \in \mathcal{C}_i : a \in (b \rightarrow u) \text{ and } b \notin (a \rightarrow u)\} \\ I_2 = \{u \in \mathcal{C}_i : a \notin (b \rightarrow u) \text{ and } b \in (a \rightarrow u)\} \\ I_3 = \{u \in \mathcal{C}_i : a \notin (b \rightarrow u) \text{ and } b \notin (a \rightarrow u)\} \\ I_4 = \{u \in \mathcal{C}_i : a \in (b \rightarrow u) \text{ and } b \in (a \rightarrow u)\} \end{cases}$$

$I_4 = \{a; b\}$ or 

Reconstructing algorithm: Extension for cycles

Idea

- Find a and b close to each other on a cycle;
- cut the cycle in between;
- iterate previous algorithm;
- reintroduce the cycle: reconnect (a, b).



Finding out how to cut

a, b: nodes with the most interferences (*i.e.*, maximizing $\{u, v : au \between bv\}$)

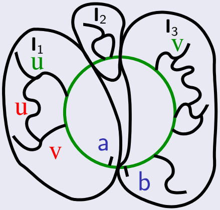
$$\begin{cases} I_1 = \{u \in \mathcal{C}_i : a \in (b \to u) \text{ and } b \notin (a \to u)\} \\ I_2 = \{u \in \mathcal{C}_i : a \notin (b \to u) \text{ and } b \in (a \to u)\} \\ I_3 = \{u \in \mathcal{C}_i : a \notin (b \to u) \text{ and } b \notin (a \to u)\} \\ I_4 = \{u \in \mathcal{C}_i : a \in (b \to u) \text{ and } b \in (a \to u)\} \end{cases}$$

v \ u	a … α	α … β	β … b	
a … α	1	1	1	} I_1
α … β	0	0\1	1	} I_2
β … b	0	0	1	} I_3

Reconstructing algorithm: Extension for cycles

Idea

- Find a and b close to each other on a cycle;
- cut the cycle in between;
- iterate previous algorithm;
- reintroduce the cycle: reconnect (a, b).



Finding out how to cut

a, b: nodes with the most interferences (*i.e.*, maximizing $\{u, v : au \between bv\}$)

$$\begin{cases} I_1 = \{u \in \mathcal{C}_i : a \in (b \to u) \text{ and } b \notin (a \to u)\} \\ I_2 = \{u \in \mathcal{C}_i : a \notin (b \to u) \text{ and } b \in (a \to u)\} \\ I_3 = \{u \in \mathcal{C}_i : a \notin (b \to u) \text{ and } b \notin (a \to u)\} \\ I_4 = \{u \in \mathcal{C}_i : a \in (b \to u) \text{ and } b \in (a \to u)\} \end{cases}$$

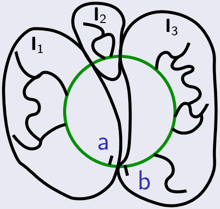
v \ u	a–α	α–β	β–b	
a–α	1	1	1	} I_1
α–β	0	0\1	1	} I_2
β–b	0	0	1	} I_3

Topological sort on the graph associated to the matrix slice gives I_1, I_2, I_3

Reconstructing algorithm: Extension for cycles

Idea

- Find a and b close to each other on a cycle;
- cut the cycle in between;
- iterate previous algorithm;
- reintroduce the cycle: reconnect (a, b).



Finding out how to cut

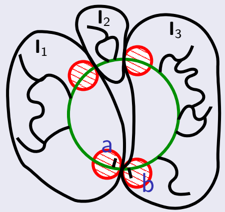
. . .

How to connect parts afterward

Reconstructing algorithm: Extension for cycles

Idea

- Find a and b close to each other on a cycle;
- cut the cycle in between;
- iterate previous algorithm;
- reintroduce the cycle: reconnect (a, b).



Finding out how to cut

...

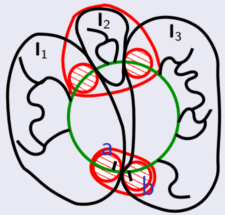
How to connect parts afterward

First step on $I_1 \longrightarrow$ Finds 2 classes I_{1_a} and I_{1_α}; $a \in I_{1_a}$.
First step on $I_3 \longrightarrow$ Finds 2 classes I_{1_b} and I_{1_β}; $b \in I_{1_b}$.

Reconstructing algorithm: Extension for cycles

Idea

- Find a and b close to each other on a cycle;
- cut the cycle in between;
- iterate previous algorithm;
- reintroduce the cycle: reconnect (a, b).



Finding out how to cut

. . .

How to connect parts afterward

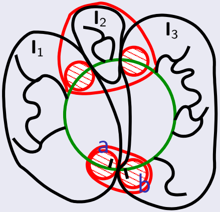
First step on $I_1 \rightarrow$ Finds 2 classes I_{1_a} and I_{1_α}; $a \in I_{1_a}$.
First step on $I_3 \rightarrow$ Finds 2 classes I_{1_b} and I_{1_β}; $b \in I_{1_b}$.

Reconnect I_{1_a} and I_{1_b} ; Reconnect I_{1_α} and I_{1_β}.

Reconstructing algorithm: Extension for cycles

Idea

- Find a and b close to each other on a cycle;
- cut the cycle in between;
- iterate previous algorithm;
- reintroduce the cycle: reconnect (a, b).



Finding out how to cut

...

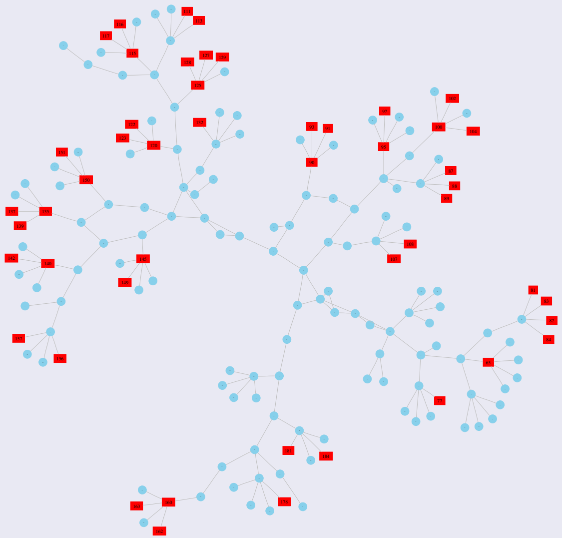
How to connect parts afterward

First step on $I_1 \longrightarrow$ Finds 2 classes I_{1_a} and I_{1_α}; $a \in I_{1_a}$.
First step on $I_3 \longrightarrow$ Finds 2 classes I_{1_b} and I_{1_β}; $b \in I_{1_b}$.

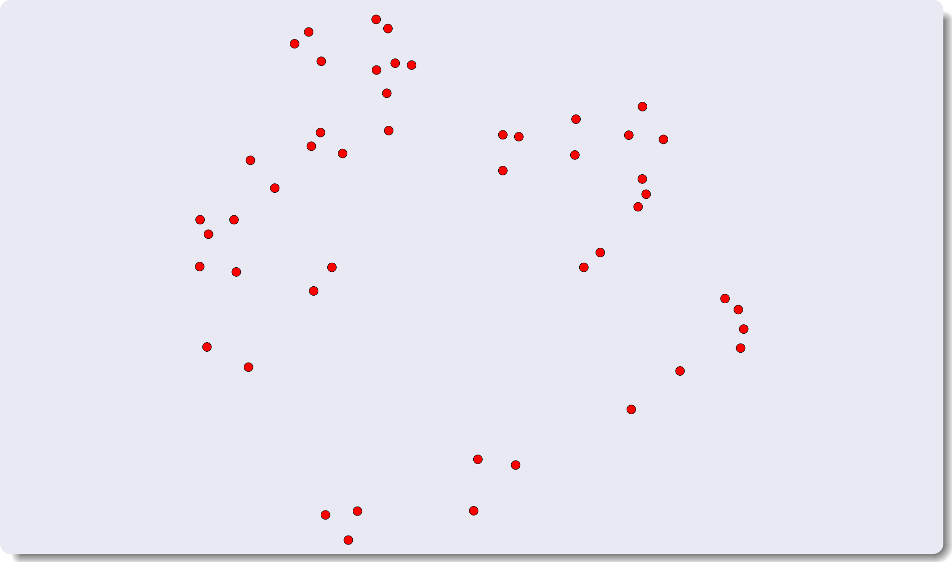
Reconnect I_{1_a} and I_{1_b} ; Reconnect I_{1_α} and I_{1_β}.

No demonstration of this...

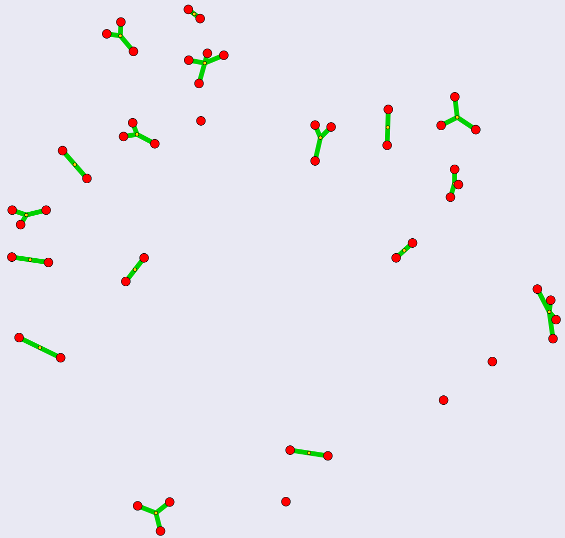
Example of reconstruction



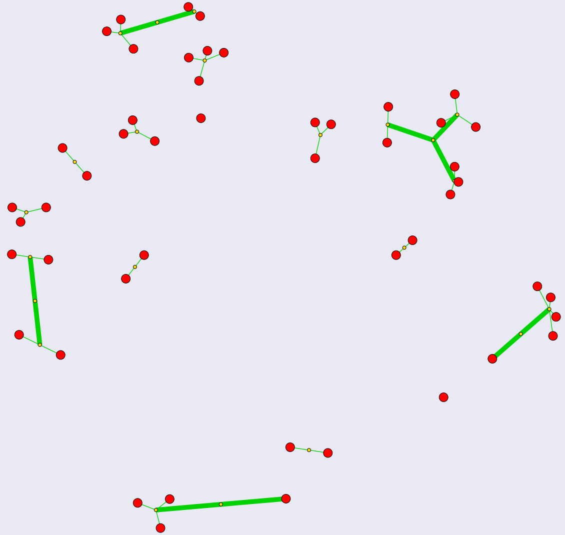
Example of reconstruction



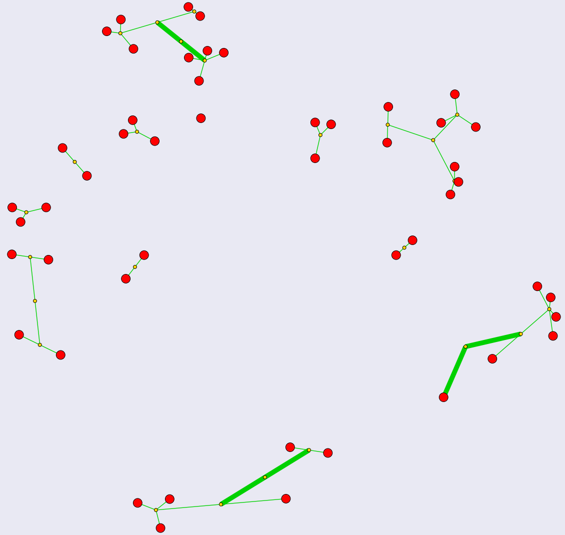
Example of reconstruction



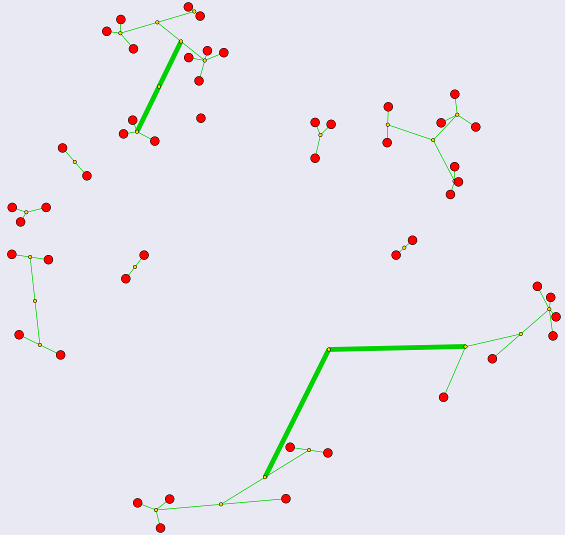
Example of reconstruction



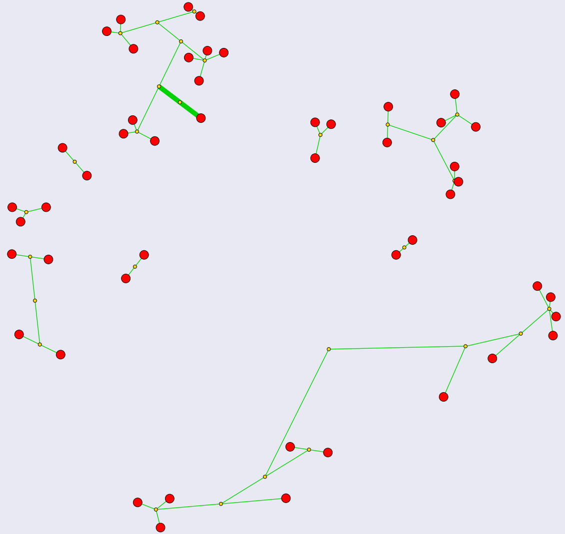
Example of reconstruction



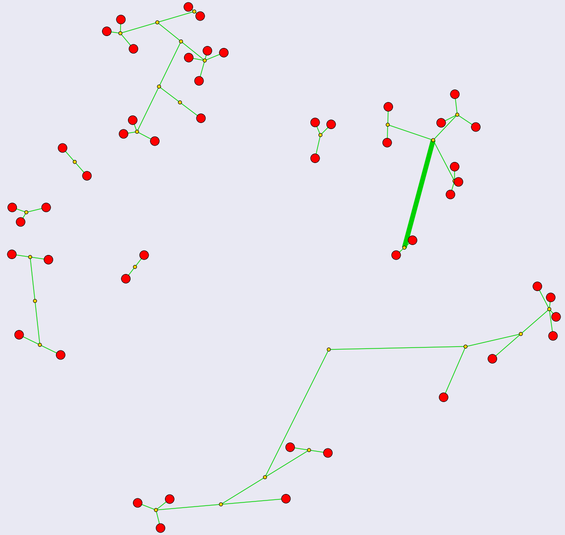
Example of reconstruction



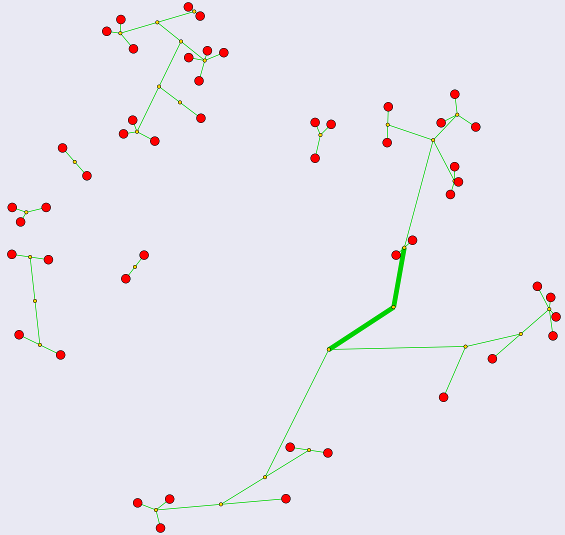
Example of reconstruction



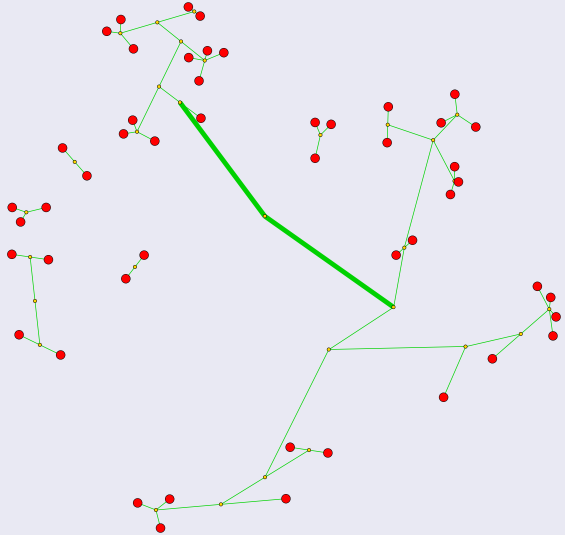
Example of reconstruction



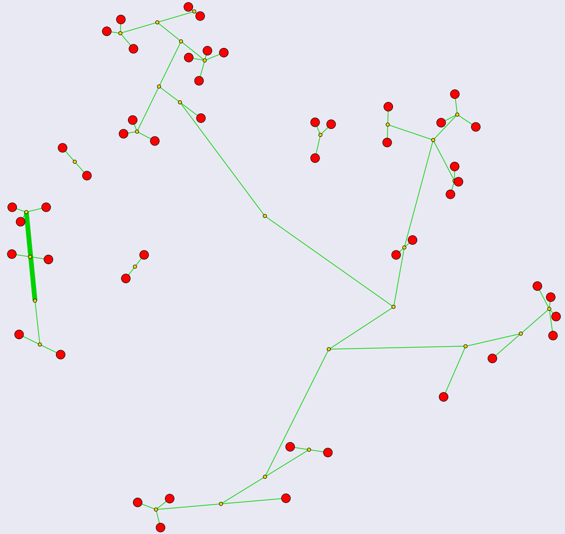
Example of reconstruction



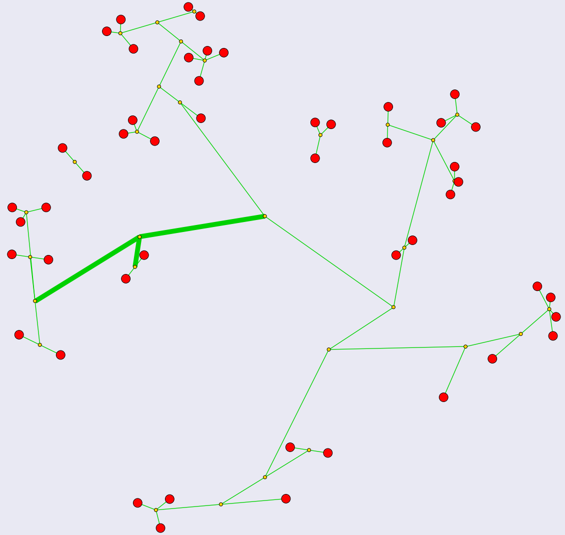
Example of reconstruction



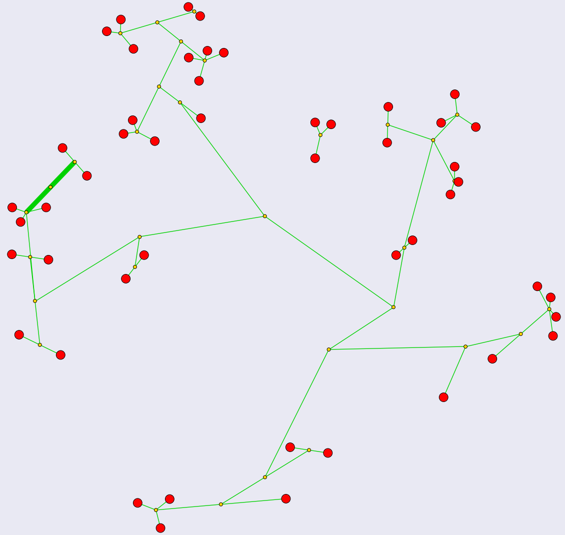
Example of reconstruction



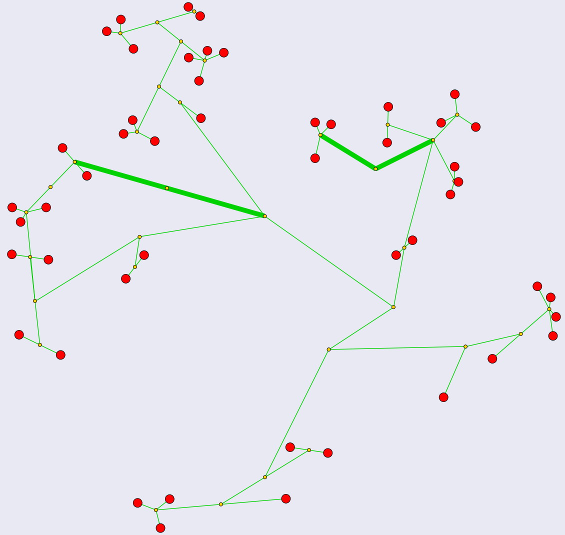
Example of reconstruction



Example of reconstruction



Example of reconstruction



Outline

Data collection

Intuitive algorithm

1. Measure the bandwidth on (ab) ;
2. Measure the bandwidth on (ab) when the link (cd) is saturated ;
3. Compute the ratio.

N^4, 30s. per step $\Rightarrow$ 50 days for 20 hosts.

Speeding things up

Using traceroute or other tomography

- Independent tests in parallel
- Validation of information sets

Refinement of existing graph?

Outline

Conclusion

Contributions

- Retrieve the interference-based topology from direct measurements
- Strong mathemathical basements (optimal for cliques of trees)
- More generic than ENV (partial cycle handling)
- Based on GRAS (development of distributed applications on simulator)

Future work

- NP-hardness
- Experimentation on real platform (measurements optimization)
- Iterative algorithm (modification detection)
- Couple measurement and reconstruction phases
- Integration within the NWS (auto-configuration; provide information)