Best-effort Group Service in Dynamic Networks

B. Ducourthial\textsuperscript{1}, S. Khalfallah\textsuperscript{1}, F. Petit\textsuperscript{2}

\textsuperscript{1} Université de Technologie de Compiègne, France
CNRS Heudiasyc (UMR 6599)

\textsuperscript{2} Université Pierre et Marie Curie, Paris, France
CNRS LiP6 (UMR 7606) & INRIA Regal

June 2010
Plan

1. Dynamic networks howto
2. Best-effort algorithms
3. Group service for dynamic networks
4. Best-effort algorithm for the group service
5. Conclusion
1 Dynamic networks howto
   Definition
   Metric
   Examples
   Building applications in dynamic networks

2 Best-effort algorithms

3 Group service for dynamic networks

4 Best-effort algorithm for the group service

5 Conclusion
Dynamic networks howto

Basic definitions

- **Network**
  - computing nodes
  - communication links
  - topology ~ graph $G(V, E)$

- **Ad hoc network**
  - uniformity of the nodes:
    all equivalent, all routers

- **Wireless network**
  - mutual exclusion in the neighborhood:
    a message sent by a neighbor is received only if no other neighbor is sending
  - message passing or register model

- **Wireless ad hoc networks**
Dynamic networks howto

Definition

- Dynamic network: the topology is not fixed
  → sequence of graphs
  \[ G_0(V_0, E_0), G_1(V_1, E_1), G_2(V_2, E_2), G_3(V_3, E_3), \ldots \]

1. Dynamic links
   - adding links
   - deleting links temporarily or definitively

2. Dynamic nodes
   - adding nodes
   - deleting nodes temporarily or definitively
   - \( \sim \) adds and deletes links

3. Moving nodes
   - \( \neq \) deleting node + adding node elsewhere
     memory of the node
   - \( \sim \) adds and deletes links temporarily

4. Dynamic and moving nodes...
Dynamic networks how to
Metric of the dynamic

- Percentage of nodes or links affected
- Mean percentage of a neighborhood affected
- Frequency of changes Unit of time?
- Frequency of changes vs. efficiency of the com. Nodes could move very fast without impact on the algorithm if the communication protocol is efficient.

- Algorithmic metric
  - $\delta$-dynamic system: any node that experiments a neighborhood change is able to send a message to all its neighbors until $\delta$ hops before the next topology change
  - 1-dynamic system: a minimal requirement for allowing local exchange in a dynamic network
Dynamic networks howto
Metric of the dynamic

- Percentage of nodes or links affected
- Mean percentage of a neighborhood affected
- Frequency of changes Unit of time?
- Frequency of changes vs. efficiency of the com.
  Nodes could move very fast without impact on the algorithm if the communication protocol is efficient.

- Algorithmic metric
  - $\delta$-dynamic system: any node that experiments a neighborhood change is able to send a message to all its neighbors until $\delta$ hops before the next topology change
  - 1-dynamic system: a minimal requirement for allowing local exchange in a dynamic network
Dynamic networks howto
Metric of the dynamic

- Percentage of nodes or links affected
- Mean percentage of a neighborhood affected
- Frequency of changes Unit of time?
- Frequency of changes vs. efficiency of the com.
  Nodes could move very fast without impact on the algorithm if the communication protocol is efficient.
- Algorithmic metric
  - $\delta$-dynamic system: any node that experiments a neighborhood change is able to send a message to all its neighbors until $\delta$ hops before the next topology change
  - 1-dynamic system: a minimal requirement for allowing local exchange in a dynamic network
Dynamic networks howto
Examples of dynamic networks

- Large networks are generally dynamic
- Social networks
- Peer-to-peer networks
- Network of laptops
  IEEE working group MANET: Mobile Ad hoc NETworks
- Network of pedestrian with personal devices
- Network of embedded computers
  - Robots networks
  - Vehicular networks
• **Backbones, spanning trees, clusters...**
  - using such structures as in non-dynamic networks
  - updating the structures when the topology changes

• **But:**
  - require control messages to be updated
  - when the dynamic increases, too much control messages are required
  - diverge

• **Thus:**
  - useful only when the dynamic is low
Dynamic networks howto
Building applications: virtual structures

- **Backbones, spanning trees, clusters...**
  - using such structures as in non-dynamic networks
  - updating the structures when the topology changes

- **But :**
  - require control messages to be updated
  - when the dynamic increases, too much control messages are required
  - diverge

- **Thus :**
  - useful only when the dynamic is low
Dynamic networks howto
Building applications: redundancy

- **Important data are replicated**
  - critical system
  - A node’s disappearance is then supported

- **But:**
  - replicated data should be coherent
  - pessimistic replication requires consensus
    - consensus is unsolvable in unreliable asynchronous networks [FLP85]
    - alternative: failure detectors [CT96]
  - optimistic replication will eventually converge
    - working with non up-to-date data

- **Thus:**
  - strong conditions on the network
  - or weak conditions on the replicas
Dynamic networks howto
Building applications: redundancy

• Important data are replicated
  • critical system
  • A node's disappearance is then supported

• But:
  • replicated data should be coherent
  • pessimistic replication requires consensus
    • consensus is unsolvable in unreliable asynchronous networks [FLP85]
  • alternative: failure detectors [CT96]
  • optimistic replication will eventually converge
    • working with non up-to-date data

• Thus:
  • strong conditions on the network
  • or weak conditions on the replicas
Dynamic networks howto
Building applications: self-stabilization

- **Self-stabilizing algorithms**:  
  - recover after a transient fault affecting a memory, a message  
  - neighborhood change  
    \[\leadsto\] some memories are not up-to-date  
  - topology change \[\leftrightarrow\] transient fault  

- **But**:  
  - duration of the convergence phase vs. dynamic  
  - the system doesn’t know whether the stabilized phase is reached or not  

- **Thus**:  
  - useful only when the dynamic is low  
  - and for non critical applications
Dynamic networks howto
Building applications: self-stabilization

- **Self-stabilizing algorithms**:  
  - recover after a transient fault affecting a memory, a message  
  - neighborhood change  
    ~ some memories are not up-to-date  
  - topology change ↔ transient fault  

- **But**:  
  - duration of the convergence phase vs. dynamic  
  - the system doesn’t know whether the stabilized phase is reached or not  

- **Thus**:  
  - useful only when the dynamic is low  
  - and for non critical applications
1 Dynamic networks howto

2 Best-effort algorithms
   Principle
   Self-stabilization ?
   Best-effort
   Related work

3 Group service for dynamic networks

4 Best-effort algorithm for the group service

5 Conclusion
• The dynamic affects the algorithms
  When the dynamic increases, it becomes illusory to expect that an application continuously ensures the service for which it has been designed.
    • impossibility results?
    • weak specifications?
    • conditions on the dynamic

• What we can only expect from the distributed algorithms is to behave as "the best" as possible, the result depending on the dynamic.

• A best effort algorithm fulfills its specifications if the dynamic of the network allows it, and fulfills them few time after the network allows it, otherwise.
• Self-stabilization can help face to the dynamic neighborhood change
  ~ some memories do not reflect the neighborhood
  ~ similar to a transient failure
Self-stabilization can help face to the dynamic Neighborhood change
\[\sim\] some memories do not reflect the neighborhood
\[\sim\] similar to a transient failure

However, it is implicitly assumed that the convergence time is smaller than the delay between two topology changes

General case: self-stabilization property must be completed
Best-effort algorithms

Definition

- **Continuity predicate** $\Pi_C$ on successive config. :
  False if the “quality of successive outputs” decreases
  depends on the problem

- **Topological predicate** $\Pi_T$ on successive config. :
  False if the topological change is “important”
  depends on the problem

- **Best-effort requirement** : $\Pi_T \Rightarrow \Pi_C$
  While the system is converging to a correct behavior, the result is better and better, as long as the dynamic allows it.
Best-effort algorithms

Related work

- How to complete self-stabilization?
  - Fault-containing network protocols
    Gosh, Gupta, Pemmaraju, SAC ’97
  - Stabilizing time adaptive protocols
  - Superstabilizing protocols for dynamic distributed systems
    Dolev, Herman. PODC 1995

- Superstabilization
  - self-stabilization + passage predicate
  - after a legitimate state is reached, if a single topology change occurs, the predicate passage holds until a new legitimate state is reached
  - but:
    - what before stabilization?
    - important in a dynamic system
How to complete self-stabilization?

- Fault-containing network protocols
  Gosh, Gupta, Pemmaraju, SAC ’97
- Stabilizing time adaptive protocols
- Superstabilizing protocols for dynamic distributed systems
  Dolev, Herman. PODC 1995

Superstabilization

- self-stabilization + passage predicate
- after a legitimate state is reached, if a single topology change occurs, the predicate passage holds until a new legitimate state is reached

but:

- what before stabilization?
- important in a dynamic system
• How to complete self-stabilization?
  • Fault-containing network protocols
    Gosh, Gupta, Pemmaraju, SAC ’97
  • Stabilizing time adaptive protocols
  • Superstabilizing protocols for dynamic distributed systems
    Dolev, Herman. PODC 1995

• Superstabilization
  • self-stabilization + passage predicate
  • after a legitimate state is reached, if a single topology change occurs, the predicate passage holds until a new legitimate state is reached
  • but:
    • what before stabilization?
    • important in a dynamic system
1 Dynamic networks howto

2 Best-effort algorithms

3 Group service for dynamic networks
   Requirements
   Example
   Specifications

4 Best-effort algorithm for the group service

5 Conclusion
Groups service
Requirements: groups in vehicular networks

- Intelligent Transport Systems
  - infrastructure oriented applications
  - vehicle oriented applications
  - driver oriented applications
  - passenger oriented applications
- Some services are based on collaboration
  - driving, diagnostic, perception, infotainment...
  - collaboration \(\sim\) group
- Vehicular networks: a kind of dynamic networks
Groups service

Requirements: constraints on the groups

- Maintaining the running service
  - the aim is not to optimize the partition of the vehicles into groups
  - it is much more important to not split existing groups
  - $\rightsquigarrow$ keeping the existing groups as long as possible

- Diameter constraint
  - delay vs. number of hops
  - no collaboration with far vehicles
    - either useless (driving, diagnostic, perception...)
    - or inefficient (chat, games...)
  - $\rightsquigarrow$ bound on the diameter depending on the applications
Groups service for inter-vehicles applications

Example
Groups service for inter-vehicles applications

Example
Groups service for inter-vehicles applications

Example
Groups service for inter-vehicles applications

Example

- Dynamic networks
- Definition
- Metric
- Examples
- Building app.
- Best-effort algorithms
- Principle
- Self-stab.
- Best-effort
- Related work
- Group service
- Requirements
- Example
- Specifications
- Algorithm
- Overview
- Sketch of proof
- Conclusion
Groups service for inter-vehicles applications

Example
Groups service for inter-vehicles applications

Example
Groups service for inter-vehicles applications

Example
Groups service for inter-vehicles applications

Example
Groups service for inter-vehicles applications

Example
Groups service for inter-vehicles applications

Example
Groups service for inter-vehicles applications

Example
Groups service for inter-vehicles applications

Example
Groups service for inter-vehicles applications

Specifications

- **Groups**: disjoint subgraphs of $G(V, E)$
  - subgraphs $H_i(V_i, E_i)$ with $V_i \cap V_j = \emptyset$
    - $V_i \subset V$ and $\forall (u, v) \in E, (u, v \in V_i) \Rightarrow (u, v) \in E_i$
  - view$_v^c$: knowledge of $v$ about its own group at configuration $c$

- **Agreement** $\Pi_A(c)$: views define groups
  - $(u \in V_i$ and $v \in V_i) \Leftrightarrow$ view$_u^c = view_v^c = V_i$
  - $\Omega_v^c = view_v^c$ if $\Pi_A(c)$ holds, $\emptyset$ else

- **Safety** $\Pi_S(c)$: groups are well formed connected and bounded diameter
  - $(d_{\Omega_v^c}$: distance in $\Omega_v^c)$
  - $\forall v \in V, \max_{x, y \in \Omega_v^c} d_{\Omega_v^c}(x, y) \leq D_{\text{max}}$

- **Maximality** $\Pi_M(c)$: groups cannot merge more
  - $\forall u, v \in V$ with $\Omega_u^c \neq \Omega_v^c$
  - $\exists x, y \in \Omega_u^c \cup \Omega_v^c$ such that $d_{\Omega_u^c \cup \Omega_v^c}(x, y) > D_{\text{max}}$
Groups service for inter-vehicles applications

Best-effort specification

- **Topological predicate** $\Pi_T(c_i, c_{i+1})$:
The distance between members of a group will remain smaller than $D_{\text{max}}$

$$\forall v \in V, \max_{x, y \in \Omega_v^c} d_{\Omega_v^c}^{c_{i+1}}(x, y) \leq D_{\text{max}}$$

- **Continuity property** $\Pi_C(c_i, c_{i+1})$:
No node disappears from a group

$$\forall v \in V, \Omega_v^c \subseteq \Omega_v^{c_{i+1}}$$

- **Best-effort specification**:
$\Pi_T(c_i, c_{i+1}) \Rightarrow \Pi_C(c_i, c_{i+1})$

As long as the diameter of a group remains smaller than $D_{\text{max}}$, the algorithm should ensure that no node will disappear

$\leadsto$ An application can work with the current knowledge of the group ($\text{view}$) even if the convergence did not happen
• **Topological predicate** $\Pi_T(c_i, c_{i+1})$:
The distance between members of a group will remain smaller than $D_{\text{max}}$

$$\forall v \in V, \max_{x,y \in \Omega^c_i} d^c_{\Omega^c_i}(x, y) \leq D_{\text{max}}$$

• **Continuity property** $\Pi_C(c_i, c_{i+1})$:
No node disappears from a group

$$\forall v \in V, \Omega^c_v \subseteq \Omega^c_{v+1}$$

• **Best-effort specification** :
$\Pi_T(c_i, c_{i+1}) \Rightarrow \Pi_C(c_i, c_{i+1})$

As long as the diameter of a group remains smaller than $D_{\text{max}}$, the algorithm should ensure that no node will disappear

$\leadsto$ An application can work with the current knowledge of the group ($\text{view}$) even if the convergence did not happen
Groups service for inter-vehicles applications

Best-effort specification

- **Topological predicate** $\Pi_T(c_i, c_{i+1})$:
The distance between members of a group will remain smaller than $D_{max}$
\[ \forall v \in V, \max_{x,y \in \Omega_v^c_i} d_{\Omega_v^c_i}^{c_{i+1}}(x, y) \leq D_{max} \]

- **Continuity property** $\Pi_C(c_i, c_{i+1})$:
No node disappears from a group
\[ \forall v \in V, \Omega_v^c_i \subseteq \Omega_v^c_{i+1} \]

- **Best-effort specification**:
$\Pi_T(c_i, c_{i+1}) \Rightarrow \Pi_C(c_i, c_{i+1})$
As long as the diameter of a group remains smaller than $D_{max}$, the algorithm should ensure that no node will disappear
\[ \leadsto \text{An application can work with the current knowledge of the group (view) even if the convergence did not happen} \]
Summary

- **Self-stabilizing algorithm for** $\Pi_A \land \Pi_S \land \Pi_M$  
  - $\mathcal{C}$ : set of all the configurations  
  - $\mathcal{L} \subset \mathcal{C}$ : set of configurations $c$ satisfying $\Pi_A(c) \land \Pi_S(c) \land \Pi_M(c)$  
  - prove, on a fixed topology, that  
    - $\mathcal{L}$ is an attractor for $\mathcal{C}$  
    - $\mathcal{L}$ is close

- **Best-effort requirement**  
  - assuming a dynamic network  
    sequence of graphs  
  - considering two consecutive configurations $c_i, c_{i+1}$ in any execution, prove that $\Pi_T(c_i, c_{i+1}) \Rightarrow \Pi_C(c_i, c_{i+1})$
1. Dynamic networks howto

2. Best-effort algorithms

3. Group service for dynamic networks

4. Best-effort algorithm for the group service
   - Overview
   - Sketch of proof

5. Conclusion
• **Candidates for a group**: neighbors up to $D_{\text{max}}$

• **Lists of close nodes**:  
  • diffusion at timer expiration  
  • merging of the received lists  
  • lists of nodes ordered by the distance:  
    $$\{(d), \{b\}, \{a, c\}\}$$  
  • lists truncated to $D_{\text{max}}$

• **Lists filtering**:  
  • only symmetric links  
    Three-way handshake by marking nodes: $v$
  • malformed lists ignored  
  • arrival list accepted only if the diameter remains smaller than $D_{\text{max}}$ after the merge  
  • if merging is impossible, the neighbor is double-marked ($\overline{v}$)
Algorithm
Overview: diffusion of lists

- **Candidates for a group**: neighbors up to Dmax
- **Lists of close nodes**:
  - diffusion at timer expiration
  - merging of the received lists
  - lists of nodes ordered by the distance: 
    \( (\{d\}, \{b\}, \{a, c\}) \)
  - lists truncated to Dmax

- **Lists filtering**:
  - only symmetric links
  - Three-way handshake by marking nodes: \( v \)
  - malformed lists ignored
  - arrival list accepted only if the diameter remains smaller than Dmax after the merge
  - if merging is impossible, the neighbor is double-marked (\( v \))
Algorithm

Overview: diffusion of lists

• **Candidates for a group**: neighbors up to Dmax

• **Lists of close nodes**:
  - diffusion at timer expiration
  - merging of the received lists
  - lists of nodes ordered by the distance:
    \[ \{d\}, \{b\}, \{a, c\} \]
  - lists truncated to Dmax

• **Lists filtering**:
  - only symmetric links
  - Three-way handshake by marking nodes: \( v \)
  - malformed lists ignored
  - arrival list accepted only if the diameter remains smaller than Dmax after the merge
  - if merging is impossible, the neighbor is double-marked \( (\overline{v}) \)
• \( S \): set of lists of nodes’ sets
  \[ (\{d\}, \{b\}, \{a, c\}) \in S \]

• Operator \( \oplus \) on \( S \) that merges two lists while deleting useless members :
  \[ (\{d\}, \{b\}, \{a, c\}) \oplus (\{c\}, \{a, e\}, \{b\}) = (\{d, c\}, \{b, a, e\}, \{a, c, b\}) = (\{d, c\}, \{b, a, e\}) \]

• Endomorphism \( r \) of \( S \), that inserts an empty set at the beginning of a list
  \[ r(\{d\}, \{b\}, \{a, c\}) = (\emptyset, \{d\}, \{b\}, \{a, c\}) \]

• \( l_1 \triangleleft l_2 = l_1 \oplus r(l_2), \; l_1, l_2 \in S \)

strictly idempotent \( r \)-operator
• $\mathbb{S}$ : set of lists of nodes’ sets
  $\{\{d\}, \{b\}, \{a, c\}\} \in \mathbb{S}$

• Operator $\oplus$ on $\mathbb{S}$ that merges two lists while deleting useless members:
  $\left(\{d\}, \{b\}, \{a, c\}\right) \oplus \left(\{c\}, \{a, e\}, \{b\}\right) = \left(\{d, c\}, \{b, a, e\}\right)$

• Endomorphism $r$ of $\mathbb{S}$, that inserts an empty set at the beginning of a list
  $r(\{d\}, \{b\}, \{a, c\}) = (\emptyset, \{d\}, \{b\}, \{a, c\})$

• $l_1 \bowtie l_2 = l_1 \oplus r(l_2)$, $l_1, l_2 \in \mathbb{S}$

strictly idempotent r-operator
Algorithm
Overview: merging lists

- $\mathcal{S}$: set of lists of nodes’ sets
  $\{(d), \{b\}, \{a, c\}\} \in \mathcal{S}$

- Operator $\oplus$ on $\mathcal{S}$ that merges two lists while deleting useless members:
  $\{(d), \{b\}, \{a, c\}\} \oplus (\{c\}, \{a, e\}, \{b\}) = (\{d, c\}, \{b, a, e\}, \{a, c, b\}) = (\{d, c\}, \{b, a, e\})$

- Endomorphism $r$ of $\mathcal{S}$, that inserts an empty set at the beginning of a list
  $r(\{d\}, \{b\}, \{a, c\}) = (\emptyset, \{d\}, \{b\}, \{a, c\})$

- $l_1 \triangleleft l_2 = l_1 \oplus r(l_2)$, $l_1, l_2 \in \mathcal{S}$
  strictly idempotent $r$-operator
Algorithm
Overview: merging lists

- $\mathcal{S}$: set of lists of nodes’ sets
  $\left(\{d\}, \{b\}, \{a, c\}\right) \in \mathcal{S}$

- Operator $\oplus$ on $\mathcal{S}$ that merges two lists while deleting useless members:
  $\left(\{d\}, \{b\}, \{a, c\}\right) \oplus \left(\{c\}, \{a, e\}, \{b\}\right) = \left(\{d, c\}, \{b, a, e\}\right) = \left(\{d, c\}, \{b, a, e\}\right)$

- Endomorphism $r$ of $\mathcal{S}$, that inserts an empty set at the beginning of a list
  $r\left(\{d\}, \{b\}, \{a, c\}\right) = (\emptyset, \{d\}, \{b\}, \{a, c\})$

- $l_1 \triangleleft l_2 = l_1 \oplus r(l_2)$, $l_1, l_2 \in \mathcal{S}$

strictly idempotent $r$-operator

<table>
<thead>
<tr>
<th>self-stabilizing system</th>
<th>registers communications</th>
<th>message passing communications</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>composite atomicity</td>
<td>read-write atomicity</td>
</tr>
<tr>
<td></td>
<td>partially ordered</td>
<td>totally ordered</td>
</tr>
<tr>
<td>strictly idempotent $r$-semi-group</td>
<td></td>
<td>totally ordered</td>
</tr>
</tbody>
</table>

[SSS’07, JACIC’06, TCS03, DC01]
- **Conflict**:  
  - deciding between far nodes in a too large list

- **Node priority**:  
  - oldness of the node in its group  
  - local logical clock increased until the node belongs to a group

- **Quarantine**:  
  - waiting for $D_{max}$ timers before entering into the view  
  - allowing to broadcast its identity in the whole group, and then resolve conflicts **(using priorities)**
Algorithm

Sketch of proof - self-stabilization

1. lists bounded
2. lists contain only existing nodes
3. propagation until double-marked edges
4. if $d(u, v) > D_{\text{max}}$, each path from $u$ to $v$ contains a double-marked edge
5. if $d(u, v) > D_{\text{max}}$, $u$, $v$ not in the same subgraph
6. Agreement: convergence to similar views inside each subgraph $\rightsquigarrow$ groups
7. Safety: group’ diameters smaller than $D_{\text{max}}$
8. The number of external edges does not increase
9. The number of external edges decrease
10. Maximality: if new merge, safety is false
11. Best-effort: a node leaves a group only if the safety becomes false
Algorithm

Sketch of proof - self-stabilization

1. lists bounded
2. lists contain only existing nodes
3. propagation until double-marked edges
4. if $d(u, v) > D_{\text{max}}$, each path from $u$ to $v$ contains a double-marked edge
5. if $d(u, v) > D_{\text{max}}$, $u, v$ not in the same subgraph
6. Agreement: convergence to similar views inside each subgraph \(\sim\) groups
7. Safety: group’ diameters smaller than $D_{\text{max}}$
8. The number of external edges does not increase
9. The number of external edges decrease
10. Maximality: if new merge, safety is false
11. Best-effort: a node leaves a group only if the safety becomes false
Algorithm
Sketch of proof - self-stabilization

1. lists bounded
2. lists contain only existing nodes
3. propagation until double-marked edges
4. if $d(u, v) > D_{\text{max}}$, each path from $u$ to $v$ contains a double-marked edge
5. if $d(u, v) > D_{\text{max}}$, $u$, $v$ not in the same subgraph
6. Agreement: convergence to similar views inside each subgraph $\sim$ groups
7. Safety: group’s diameters smaller than $D_{\text{max}}$
8. The number of external edges does not increase
9. The number of external edges decrease
10. Maximality: if new merge, safety is false
11. Best-effort: a node leaves a group only if the safety becomes false
Algorithm

Sketch of proof - self-stabilization

1. lists bounded
2. lists contain only existing nodes
3. propagation until double-marked edges
4. if $d(u, v) > D_{\text{max}}$, each path from $u$ to $v$ contains a double-marked edge
5. if $d(u, v) > D_{\text{max}}$, $u, v$ not in the same subgraph
6. Agreement: convergence to similar views inside each subgraph $\sim$ groups
7. Safety: group’s diameters smaller than $D_{\text{max}}$
8. The number of external edges does not increase
9. The number of external edges decrease
10. Maximality: if new merge, safety is false
11. Best-effort: a node leaves a group only if the safety becomes false
Algorithm

Sketch of proof - self-stabilization

1. lists bounded
2. lists contain only existing nodes
3. propagation until double-marked edges
4. if \( d(u, v) > D_{\text{max}} \), each path from \( u \) to \( v \) contains a double-marked edge
5. if \( d(u, v) > D_{\text{max}} \), \( u \), \( v \) not in the same subgraph
6. Agreement: convergence to similar views inside each subgraph \( \sim \) groups
7. Safety: group’s diameters smaller than \( D_{\text{max}} \)
8. The number of external edges does not increase
9. The number of external edges decrease
10. Maximalit y: if new merge, safety is false
11. Best-effort: a node leaves a group only if the safety becomes false
1 Dynamic networks howto
2 Best-effort algorithms
3 Group service for dynamic networks
4 Best-effort algorithm for the group service
5 Conclusion
Conclusion

- Dynamic ad hoc network:
  - a next step in distributed computing?
  - How to build distributed applications?
- **Best-effort approach**:
  - algorithms do their best
  - the result depends on the dynamic
- **Best-effort algorithm**:
  self-stabilizing + continuity in the outputs depending on the dynamic

- **Application**: group service in vehicular networks

Code and videos:
http://www.hds.utc.fr/~ducourth/airplug