Distributed Algorithms for Mobile Networks

Sébastien Tixeuil

LIP6 - CNRS 7606 - UPMC Sorbonne Universités
Sebastien.Tixeuil@lip6.fr
Thanks to

• Quentin Bramas
• Arnaud Casteigts
• Arnaud Labourel
• Giuseppe Prencipe
• Eric Ruppert
• Nicola Santoro
This Lecture is **NOT** about

- **Centralized** algorithms
- Distributed **heuristics**
- **Mobility-unrelated** dynamic networks
Two Domains
Passively Mobile Networks
Actively Mobile Networks
Outline

• **Passively mobile networks**
  • Static Algorithms for Dynamic Networks
  • Dynamic Network Models and Algorithms

• **Actively mobile networks**
  • Agents vs. Robots
Three Approaches
Complexity-driven
Complexity-driven

Complexity

Lower Bound

Model A

Problem 1
Model-driven

Model A

Problem 1

Problem 2

Problem 3
Problem-driven
- Landing gear’s frozen
- I lost my radar
- I’m out of fuel
- Lost a wing
- Lost the other one

- looking good
- right for line-up
- doing fine
- Ok you can do it
A Map of Models

Model A

Model B

Model C

Model F

Model D

Model E
A Map of Models

Model A

Model B

Model C

Model D

Model E

Model F
A Map of Models

Model A

Model B

Model C

Model D

Model E

Model F
A Map of Models

Model B

Model C

Model F

Model A

Model D

Model E
A Map of Models

- Model B
- Model C
- Model F
- Model A
- Model D
- Model E
Passively Mobile Networks
Mobility-induced Dynamic Networks
Mobility-induced Dynamic Networks
Mobility-induced Dynamic Networks
Static Algorithms for Mobile Networks
Link Lifetime
Link Lifetime
Link Lifetime
Additionally affected by the increase in transmission power was set to automatically adjust the transmit power, the radio throughput increases and decreases sharply. Since the radio communicating vehicles. This is visible from the graph where the (e.g. other vehicles) in-between and around the two communication was reached with the vehicles 0 m from each other, i.e. at the maximum peak bandwidth recorded was 34.5 Mbps occurring was transferred in a representative 100 seconds period. The test was conducted on a highway with a measured average absolute vehicle speed of 113 km/h. The average results of the setup was conducted in an urban environment.

At various stages of the experiment there were obstacles (e.g. trees, buildings) in-between the vehicles. This reduced the bandwidth available for transmission. The results for these scenarios are shown in Figs. 4c and 4d respectively. The results represent an out of 1.88 ms and an average of 51.7 MB data transferred per run taken over the period of established contact, average jitter and the average maximum communication range was found to be 70 m. The link lifetime is shown in the graph below.

On the use of WiMAX and Wi-Fi to provide in-vehicle connectivity and media distribution. Lerotholi S. Mojela ; Marthinus J. Booysen, Industrial Technology (ICIT), 2013 IEEE International Conference on
On the use of WiMAX and Wi-Fi to provide in-vehicle connectivity and media distribution. Lerotholi S. Mojela ; Marthinus J. Booysen, Industrial Technology (ICIT), 2013 IEEE International Conference on

Fig. 4b shows a graph of the two vehicles traveling in opposite directions at an average relative speed of 64 km/h in an urban area. The average contact time recorded was 33s and the average maximum communication range was found to be 302 m with an average bandwidth of 13.7 Mbps per test run taken over the period of established contact, average jitter of 1.88 ms and an average of 51.7 MB data transferred per contact period. The maximum peak bandwidth of 31.7 Mbps
<table>
<thead>
<tr>
<th></th>
<th>Barcelona</th>
<th>Paris</th>
<th>Tokyo</th>
<th>Toronto</th>
<th>Washington</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amsterdam</td>
<td>44.173ms</td>
<td>11.876ms</td>
<td>242.819ms</td>
<td>96.619ms</td>
<td>93.342ms</td>
</tr>
<tr>
<td>Auckland</td>
<td>295.323ms</td>
<td>280.111ms</td>
<td>241.541ms</td>
<td>199.854ms</td>
<td>211.48ms</td>
</tr>
<tr>
<td>Berlin</td>
<td>49.71ms</td>
<td>20.041ms</td>
<td>275.013ms</td>
<td>103.027ms</td>
<td>113.909ms</td>
</tr>
<tr>
<td>Copenhagen</td>
<td>43.745ms</td>
<td>25.574ms</td>
<td>272.459ms</td>
<td>117.751ms</td>
<td>100.623ms</td>
</tr>
<tr>
<td>Dallas</td>
<td>143.543ms</td>
<td>114.941ms</td>
<td>145.957ms</td>
<td>76.173ms</td>
<td>34.194ms</td>
</tr>
<tr>
<td>London</td>
<td>35.431ms</td>
<td>4.542ms</td>
<td>240.519ms</td>
<td>85.102ms</td>
<td>75.484ms</td>
</tr>
<tr>
<td>Los Angeles</td>
<td>157.289ms</td>
<td>141.484ms</td>
<td>112.265ms</td>
<td>58.689ms</td>
<td>71.881ms</td>
</tr>
<tr>
<td>Moscow</td>
<td>74.806ms</td>
<td>55.67ms</td>
<td>299.067ms</td>
<td>137.308ms</td>
<td>144.68ms</td>
</tr>
<tr>
<td>New York</td>
<td>130.6ms</td>
<td>77.551ms</td>
<td>171.641ms</td>
<td>17.78ms</td>
<td>47.161ms</td>
</tr>
<tr>
<td>Paris</td>
<td>31.085ms</td>
<td>—</td>
<td>260.569ms</td>
<td>103.015ms</td>
<td>107.643ms</td>
</tr>
<tr>
<td>Stockholm</td>
<td>59.785ms</td>
<td>27.894ms</td>
<td>285.462ms</td>
<td>123.358ms</td>
<td>115.657ms</td>
</tr>
<tr>
<td>Tokyo</td>
<td>267.753ms</td>
<td>268.811ms</td>
<td>—</td>
<td>172.841ms</td>
<td>186.818ms</td>
</tr>
</tbody>
</table>
Mobility vs. Global State
Mobility vs. Global State
Mobility vs. Global State
Mobility vs. Global State

Diagram showing a network of states with arrows indicating transitions between them.
Mobility vs. Global State
Mobility vs. Global State
Stateless Algorithms
Statelessness

HTTP

UDP  TCP

IP

RIP  OSPF  BGP

Lower layers
Statelessness

HTTP

UDP  TCP

IP

RIP  OSPF  BGP

Lower layers
A routing algorithm is **stateless** if it is designed such that devices store *no information* about messages *between transmissions*. It is **stateful** otherwise.
Flooding
Stateless Flooding
Stateless Flooding
Stateless Flooding
Stateless Flooding
Stateless Flooding
Stateless Flooding
Stateless Flooding
Stateless Flooding
Stateless Flooding
Stateless Flooding
Stateless Flooding
Stateless Flooding
Stateless Flooding
Stateless Flooding
Flooding v2
TTL Flooding
TTL Flooding
TTL Flooding
TTL Flooding

Diagram with nodes and edges labeled with the number 2.
TTL Flooding
TTL Flooding
TTL Flooding
Flooding v3
Stateful Flooding
Stateful Flooding
Stateful Flooding
Stateful Flooding
Stateful Flooding
Stateful Flooding
Stateful Flooding
Geometric Routing

- Each node is aware of its *coordinates* (and those of its neighbors)
- The message contains the coordinates of the destination
- **Goal**: deliver the message to the destination *without routing tables*

The notion of progress was used to define the MFR rule [84], which chooses the neighbor with the most forwarding progress within the transmission radius (see Figure 2). Similar to this rule, the greedy method in [19] selects the neighbor that minimizes the distance to the destination, which is equivalent to maximizing the advance. The angular criterion is used in Compass routing (CR) [48], where the neighbor is selected that minimizes the angle separation with respect to the destination. Usually, the next hop in greedy forwarding is chosen among the neighbors with a positive progress (right of the dashed line in Figure 1) or with a positive advance (shaded area in Figure 1, also called greedy area). Selecting the next hop by the minimum distance or the maximum progress (MFR, greedy) gives an inherently loop-free forwarding rule independent of the unit disk graph assumption. Compass routing, which is based on the direction, is not loop-free [82].

Motivated by the observation that energy consumption can be reduced when using short links, required that the transmission range can be adjusted, the NFP (nearest with forwarding progress [38]) and NC (nearest closer [83]) criteria have been proposed. They select a neighbor which is closest to the forwarding node among all neighbors, but closer to the destination than the forwarding node itself, using distance or progress.
Which Criterion?

- **MFR**: most forwarding progress
- **CR**: minimize angular criterion
- **Greedy**: minimize distance to destination
- **NC**: nearest closer
- **NFP**: nearest with forwarding progress
2.2 Advanced Strategies

Greedy forwarding has one important drawback: it fails in local minimum situations where the forwarding node has no other neighbors closer to the destination. In some cases, a sophisticated strategy is necessary to recover from this situation; in other cases, a simple backward step is sufficient to be able to resume the greedy strategy successfully.

The GEDIR [82] method is such a greedy strategy with backward steps. Whenever a message has reached a local minimum, the packet is sent back to the previous hop, which applies the greedy rule again while excluding previously visited dead end nodes from the selection. This strategy is also loop-free.

Further improvements of the basic strategies Greedy, MFR and CR can be achieved if 2-hop information is available [82]. In this case, the forwarder selects a suitable neighbor out of the 2-hop neighborhood and forwards the packet to the direct neighbor that is connected to the selected node. Note, that 2-hop information has to be distributed, which requires a higher message overhead.

A greedy-based algorithm that goes beyond using 2-hop information is SPEED [34], which is designed to increase the relay speed. It uses an additional “backpressure” heuristic to avoid congested areas and void regions. The protocol relies on beaconing, extended by on-demand beacons for delay estimation and backpressure information. The forwarding works as follows: Nodes from the greedy area, whose relay speed is above a certain threshold, are selected probabilistically. The higher the relay speed the higher the probability to be selected. If no neighbor meets the relay speed requirement, the node drops the packet with a certain probability that depends on the failure ratio of packet forwarding to the neighbors. The necessary information to derive the failure ratio is gained from the neighbors by backpressure.
Delivery Guarantee?
Planar Graph Routing

beacons, which are sent in case of congestion or in a local minimum situation. This method can alleviate local minima problems in case of small void regions, but it cannot guarantee delivery in general.

3 Planar Graph Routing and Recovery Strategies

Planar graph routing is a geographic routing strategy that is able to overcome the local minimum problem of greedy forwarding. Local minima exist at the border of void regions, where a node cannot find a neighbor closer to the destination than itself. Such nodes are also called dead-end nodes or concave nodes. Planar graph routing is a key concept for recovery from a local minimum situation. It is based on the idea that the network links form a communication graph, and a message can be routed along a sequence of faces in this graph. Routing along a face means that the nodes of a face pass the message along the incident edges by locally applying the left-hand or right-hand rule (see Figure 3). This rule is well-known from maze problems: One can find a way out of every simply connected maze when having the right-hand always in touch of the wall while walking. Applying the right-hand or left-hand rule to network graphs means to find a successor node in (counter-)clockwise order after the predecessor. This results in a traversal of a face of the communication graph. For a successful application of this rule, the underlying graph has to be planar.

3.1 Face Routing

The first face routing strategies were proposed in 1999 by Kranakis et al. [48] and Bose et al. [10] under the names “Compass Routing II” and “Face-2”, respectively.

Face Routing

Compass Routing II traverses a sequence of adjacent faces until reaching the destination as shown in Figure 3. Each face is traversed completely in order to determine the edge that intersects the $st$-line and is closest to the target. Then the message is passed to an endpoint of this edge, where the face is changed and the traversal of the next face continues. Face-2 [10] also visits a sequence of faces, but it avoids the complete traversals and performs the face change before crossing the $st$-line, as depicted in Figure 4. On each face traversal, a node $u$ checks whether the edge to the next node $(u, u_0)$ intersects the $st$-line. If this is the case, then $u$ changes the face and continues traversing the next face. A detailed description of face change rules can be found in [22]. Face routing has the advantage that it guarantees delivery on planar graphs while the nodes use only local position-based rules and do not need to keep state information.

The planarity of the underlying network graph is required for assuring delivery guarantees, because crossing links as shown in Figures 5 and 6 can cause detours or routing loops. Therefore, an arbitrary unit disk graph has to be transformed into a planar graph first. This can be done locally by removing crossing edges using geometric criteria. The removal of edges however can increase the hop count, which makes face routing steps less efficient than greedy routing. Therefore, Bose et al. proposed the Greedy-Face-Greedy algorithm (GFG) [10], a combination of the efficient greedy forwarding and face routing on a planar subgraph to recover from local minima. A variant of this algorithm is known as GPSR [43].

Figure 4: Face routing by FACE-2. A face change takes place at nodes $u$, $v$, and $w$.

Figure 5: Crossing links causing a detour (starting from node $u$).
Planar Graphs!

Figure 6: Crossing links causing a face routing failure

GFG and GPSR use greedy forwarding as long as possible. If greedy routing fails, a face traversal starts until the greedy strategy can be resumed. When starting recovery, the distance of the first node to the target $d_r$ and the first edge $e_r$ have to be stored in the packet header. If the first edge $e_r$ is visited again for the second time, then the destination is not reachable and the packet is dropped. The distance $d_r$ is used to check whether the next hop on the face traversal is closer to the destination than the node entering recovery mode. If such a node is found, greedy forwarding can be resumed instead of continuing the traversal until crossing the $s$-$t$-line (this is known as the sooner-back procedure [14]). Pseudo-code for such a combined greedy and face routing algorithm is presented in the following. An example is shown in Figure 7.

A Combined Greedy/Face-Routing Algorithm (GFG with sooner-back procedure [15])

Variables: previous hop $p$, current node $u$, target $t$, first edge in recovery mode $e_r$ and distance to target $d_r$ in rec. mode

if packet in greedy mode
   select next hop $v$ according to the greedy rule
   if no such neighbor exists
      select next hop $v$ in ccw. direction from $u, t$
   endif
   switch packet to recovery mode
   store current distance to the destination $d_r$ and $e_r(u, v)$ in the packet header
else
   (packet is in recovery mode)
   if there is a neighbor $v$ with $||v-t|| < d_r$
      switch packet to greedy mode
   else
      select next hop $v$ in ccw. direction from $u, p$
      (using only nodes of a GG or RNG subgraph)
      if $(u, v)$ equals the first edge $e_r$ in recovery mode
         drop packet; return
      endif
   endif
endif
forward packet to $v$
GFG and GPSR use greedy forwarding as long as possible. If greedy routing fails, a face traversal starts until the greedy strategy can be resumed. When starting recovery, the distance of the first node to the target $d_r$ and the first edge $e_r$ have to be stored in the packet header. If the first edge $e_r$ is visited again for the second time, then the destination is not reachable and the packet is dropped. The distance $d_r$ is used to check whether the next hop on the face traversal is closer to the destination than the node entering recovery mode. If such a node is found, greedy forwarding can be resumed instead of continuing the traversal until crossing the $s$-$t$-line (this is known as the sooner-back procedure [14]). Pseudo-code for such a combined greedy and face routing algorithm is presented in the following. An example is shown in Figure 7.

A Combined Greedy/Face-Routing Algorithm (GFG with sooner-back procedure [15])

Variables: previous hop $p$, current node $u$, target $t$, first edge in recovery mode $e_r$ and distance to target $d_r$ in rec. mode

if packet in greedy mode
   select next hop $v$ according to the greedy rule
   if no such neighbor exists
      select next hop $v$ in ccw. direction from $(u, t)$
      switch packet to recovery mode
      store current distance to the destination $d_r$ and $e_r(u, v)$ in the packet header
   endif
else (packet is in recovery mode)
   if there is a neighbor $v$ with $||v - t|| < d_r$
      switch packet to greedy mode
   else
      select next hop $v$ in ccw. direction from $(u, p)$ (using only nodes of a GG or RNG subgraph)
      if $(u, v)$ equals the first edge $e_r$ in recovery mode
         drop packet; return
      endif
   endif
endif
forward packet to $v$
Figure 7: Combined greedy/face routing: After reaching local minimum $u$ in greedy mode (dashed arrow), a face traversal is started (solid arrow) until a node $v$ is found that is closer to the target than $u$.

When using such combined algorithms, the greedy part can be performed using all links of the unit disk graph, while face routing needs a local planar subgraph. We will see in the next section how a planar subgraph can be constructed.

3.2 Planarization

In their paper on face routing, Bose et al. [10] proposed a local planar subgraph construction based on the so-called Gabriel graph (GG) [27]. The Gabriel graph of a given point set contains an edge $uv$ if Thales' circle on $uv$, i.e. the circle having $uv$ as diameter, is empty. This circle is also called Gabriel circle over $uv$ within this context. The Gabriel graph is known to be planar and connected.

This construction rule can be applied locally to a node's 1-hop neighborhood in order to extract a planar subgraph. The Gabriel graph construction and the so-called relative neighborhood graph (RNG) [86, 39] are the two most prominent local planarization schemes. Planarization using the GG criterion removes an edge $uv$ if Thales' circle on $uv$ contains another node $w$. Following the RNG criterion, an edge $uv$ is eliminated, if the intersection of two circles with radius $|uv|$ centered at $u$ and $v$ contains another node $w$ (see Figure 8). Applying the GG or RNG criterion to a unit disk graph yields a planar and connected graph, if the unit disk graph is connected.
Self-stabilization
Self-stabilization 101
Example

\[ U_0 = a \]

\[ U_{n+1} = \frac{U_n}{2} \text{ if } U_n \text{ is even} \]

\[ U_{n+1} = \frac{3U_n+1}{2} \text{ if } U_n \text{ is odd} \]
Example

\[ U_0 = a \]

\[ U_{n+1} = \frac{U_n}{2} \text{ if } U_n \text{ is even} \]

\[ U_{n+1} = \frac{3U_n+1}{2} \text{ if } U_n \text{ is odd} \]

<table>
<thead>
<tr>
<th>( n )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>( U_n )</td>
<td>7</td>
<td>11</td>
<td>17</td>
<td>26</td>
<td>13</td>
<td>20</td>
<td>10</td>
<td>5</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Example

\[ U_0 = a \]

\[ U_{n+1} = \frac{U_n}{2} \text{ if } U_n \text{ is even} \]

\[ U_{n+1} = \frac{3U_n + 1}{2} \text{ if } U_n \text{ is odd} \]
Example
Self-stabilization

Stabilization Time

Stabilized

Configurations

"Correct"

Time
Self-stabilization

- Legitimate
  - f1
  - f2

- Arbitrary
Self-stabilization
Distributed Systems

- **Configuration**: product of the local states of system components
- **Execution**: interleaving of the local executions of the system components
Distributed Systems

- **Classical**: Starting from a particular initial configuration, the system immediately exhibits correct behavior.

- **Self-stabilizing**: Starting from any initial configuration, the system eventually reaches a configuration from which its behavior is correct.
Distributed Systems

• **Self-stabilizing**: Starting from any initial configuration, the system eventually reaches a configuration from which its behavior is correct.

  • Defined by *Dijkstra* in 1974

  • Advocated by *Lamport* in 1984 to address fault-tolerant issues

• Stale states due to **mobility** can be recovered!
Configurations

```c
int x = 0;
...
if( x == 0 ) {
    // code assuming x equals 0
} else {
    // code assuming x does not equal 0
}
```
Configurations
Configurations
Hypotheses
Atomicity

• A «stabilizing» sequential program

```c
int x = 0;
...
while( x == x ) {
    x = 0;
    // code assuming x equals 0
}
```
Atomicity

• A «stabilizing» sequential program

```
0  iconst_0
1  istore_1
2  goto 7
5  iconst_0
6  istore_1
7  iload_1
8  iload_1
9  if_icmpeq 5
```

Problem
Communications
Communications
Example

- **Shared memory**: in one atomic step, read the state of all neighbors and write own state

- **Guarded command**

  \[ \text{Guard} \rightarrow \text{Action} \]

  - Predicated on the states of the neighborhood
  - Executed if Guard is true
true → Distance_i := \text{Min}_{j \in \text{Neighbors}_i} \{\text{Distance}_j + 1\}
true → Distance_i := Min_{j ∈ Neighbors_i} \{Distance_j + 1\}
true → \( \text{Distance}_i := \min_{j \in \text{Neighbors}_i} \{ \text{Distance}_j + 1 \} \)
true → Distance_i := \min_{j \in \text{Neighbors}_i} \{Distance_j + 1\}
Example

true → Distance\(_i\) := Min\(_{j \in Neighbors_i}\) \{Distance\(_j\) + 1\}
true \rightarrow \text{Distance}_i := \min_{j \in \text{Neighbors}_i} \{\text{Distance}_j + 1\}
Example

true \rightarrow \text{Distance}_i := \min_{j \in \text{Neighbors}_i} \{\text{Distance}_j + 1\}
true → \( Distanc_{e_i} := \min_{j \in Neighbors_i} \{ \text{Distance}_j + 1 \} \)
true \rightarrow \text{Distance}_i := \min_{j \in \text{Neighbors}_i} \{\text{Distance}_j + 1\}
true → \( Distance_i := \min_{j \in \text{Neighbors}_i} \{ Distance_j + 1 \} \)
Example

$true \rightarrow Distance_i := \min_{j \in \text{Neighbors}_i} \{Distance_j + 1\}$
true → Distance\textsubscript{i} := Min_{j \in Neighbors\textsubscript{i}}\{Distance\textsubscript{j} + 1\}
true → Distance_i := Min_{j ∈ Neighbors_i} \{Distance_j + 1\}
true \rightarrow Distance_i := \min_{j \in \text{Neighbors}_i} \{Distance_j + 1\}
Scheduling

- **Scheduler** (a.k.a. **Daemon**): the daemon chooses among activatable processes those that will execute their actions

  - can be seen as an **adversary** whose role is to prevent stabilization
Spatial Scheduling

\[ true \rightarrow \text{color}_i := \text{Min}\{\Delta \setminus \{\text{color}_j | j \in \text{Neighbors}_i\}\} \]

\[ \Delta = \{ \text{red}, \text{blue}, \text{blue}, \text{yellow}, \text{green} \} \]
Spatial Scheduling
Temporal Scheduling

token $\rightarrow$ pass token to left neighbor with probability $\frac{1}{2}$

token = $\bullet$  no token = $\circ$

[Diagram of a network with tokens and arrows showing the passing of tokens]
Temporal Scheduling

token → pass token to left neighbor with probability $\frac{1}{2}$

token = ● ● no token = ○ ○
Temporal Scheduling
Temporal Scheduling

Unfair → Weak → Strong → Global

∞ → 3 → 2 → 1
A Map of Daemons

Unfair ← Weak ← Strong ← Global
A Map of Daemons

1

2

3

∞

Unfair

Weak

Strong

Global
A Map of Daemons
Self-stabilization
Population Protocols
Population Protocols

Population Protocols
Population Protocols
Population Protocols

- **Definition**

  - A *Population Protocol* is a 6-tuple \((X, Y, Q, I, O, T)\)
    - \(X\): Set of inputs
    - \(Y\): Set of outputs
    - \(Q\): Set of states
    - \(I\): Input mapping function, \(X \rightarrow Q\)
    - \(O\): Output mapping function, \(Q \rightarrow O\)
    - \(T\): Transition function, \(Q \times Q \rightarrow Q \times Q\)
Example 1b
Example 2

- Inputs: • Inputs: 
- Outputs: • Outputs: 
- # • < # • ?
Example 2
Example 3

- **Inputs:** 0 1 2 3
- **Outputs:** 0 1 2 3
- **Sum mod 4?**
Example 3
Population Protocols
Dynamic Graphs
Time-varying Graphs

- A *time-varying graph* (TVG) is a 5-tuple \((V, E, T, p, l)\)
  
  - **\(V\)**: set of nodes
  
  - **\(E\)**: (labelled) set of edges
  
  - **\(T\)**: lifetime, \(T \subseteq \mathcal{T}\)
  
  - **\(p\)**: presence function, \(E \times T \rightarrow \{0, 1\}\)
  
  - **\(l\)**: latency function, \(E \times T \rightarrow \mathcal{T}\)
Time-varying Graphs

- A *time-varying graph* (TVG) is a 5-tuple \((V, E, T, p', l')\)
  - \(V\): set of nodes
  - \(E\): (labelled) set of edges
  - \(T\): lifetime, \(T \subseteq T\)
  - \(p'\): *node* presence function, \(V \times T \rightarrow \{0, 1\}\)
  - \(l'\): *node* latency function, \(V \times T \rightarrow T\)
Time-varying Graphs

$t$ in $[2,5]$

latency $= 3$
Evolving Graphs
Example

Porquerolles

La Tour Fondue

Nice

Marseille

Boat

Taxi

Bus

Train
Example

A

B

C e2 e2 e2 D

e1 e2 e3
Example

\[ (5,6) \cup (7,8) \]

\[ [2,5) \quad [2,5) \]

\[ (1,3) \]

\[ (1,3) \]
Evolving Graphs
Evolving Graphs
Evolving Graphs

[Diagram of evolving graphs with nodes A, B, C, and D, and edges and time intervals labeled.]
Evolving Graphs
Evolving Graphs

[Diagram of evolving graphs showing nodes A, B, C, D with edges and time intervals (5,6) U (7,8), (2,5), (1,3), T1, T2, T3, T5]
Evolving Graphs

A

B

C

D

A

B

C

D

T1

T2

T3

T5
Evolving Graphs

[Image of evolving graph with nodes A, B, C, D, and timestamps T1, T2, T3, T5, T6]
Evolving Graphs
Journeys from C to A
Shortest Journey
Foremost Journey
Fastest Journey
Condition for Broadcast?

T1

T2

T3

T4

T5
Condition for Broadcast?
Condition for Broadcast?

There exists a node (C) from which a journey reaches every other node.
Condition for Election?

A --- B --- C --- D

T1: A --- B

T2: B --- C

T3: C --- D

T4: B --- C --- D

T5: C --- D
Condition for Election?

T1: A → B → C
T2: A → B → C
T3: A → B → D
T4: B → C → D
T5: C → D

Conditions for the election are met in T1, T2, and T5, but not in T3 and T4.
Condition for Election?

There exists a node (C) such that there exists a journey from every other node to it.
Condition for Global Calculus?

X \rightarrow Y \Rightarrow X+Y \Rightarrow X+Y

X \rightarrow Y \Rightarrow Y \Rightarrow Y

Condition for Global Calculus?

\[
\begin{align*}
X & \rightarrow Y \quad X + Y \\
Y & \rightarrow X \quad Y + X
\end{align*}
\]
There exists a node (Center) such that there exists a journey from every other node to it and back.
Connectivity Classes

- There exists a node $r$ from which a journey reaches every other node $1 \rightsquigarrow \ast$
- There exists a node $r$ such that there exists a journey from every other node to it $\ast \rightsquigarrow 1$
- There exists a node $r$ such that there exists a journey from every other node to to and back $1 \rightsquigarrow \ast$

More Classes

• There exists a journey between any two nodes $* \leadsto *$

• There exists a roundtrip journey between any two nodes $* \leadsto * \leadsto *$

• There exists a journey between any two nodes infinitely often $* \leadsto *$ \(\mathcal{R}\)

• Every edge appears infinitely often $\bullet \_ \_ \bullet$ \(\mathcal{R}\)

More Classes

- Every edge appears infinitely often, and there is an upper bound between between two occurrences.
- Every edge appears infinitely often with some period $p$. 

More Classes

- At any time, the graph is connected
- Every spanning subgraph lasts at least $T$ time units
- Every edge appears infinitely often, and the underlying graph is a clique

---

A Classification
A Classification

\[ \mathcal{R} \xrightarrow{\cdot \sim \cdot} \mathcal{R} \xrightarrow{\ast \sim \ast} \ast \leftrightarrow \ast \xrightarrow{1 \sim \ast} 1 \sim \ast \]

\[ \ast \sim \ast \xrightarrow{1 \sim \ast} 1 \sim \ast \]

\[ \ast \sim \ast \xrightarrow{1 \sim \ast} 1 \sim \ast \]

\[ \ast \sim \ast \xrightarrow{1 \sim \ast} 1 \sim \ast \]
A Classification
A Classification

[Diagram showing a classification of broadcast protocols and their relationships]

Reliable communication despite Byzantine failures in dynamic networks

Alexandre Maurer, Sébastien Tixeuil, Xavier Défago: Communicating Reliably in Multihop Dynamic Networks Despite Byzantine Failures. SRDS 2015: 238-245
Context

Information broadcast in multi hop networks
Example

S -> Blue -> D
Example

S said Blue
Example

S said Blue
Example

S → Blue → D
Example

S said Red
Example

S said Red
Information Broadcast

Diagram showing a network with nodes S and D connected by arrows representing the broadcast process.
Information Broadcast
Objective

- **Broadcast** algorithms resilient to **Byzantine** Failures
  - No **false** message ever accepted
  - **Correct** messages always received
Local Vote
Vote on Multiple Paths
Condition for reliable communication in static networks

• $k =$ number of Byzantine nodes

• **Condition**: $2k+1$ node-disjoint paths between the source and the destination
Enter Dynamic Networks
Menger’s Theorem
Menger's Theorem
Menger’s Theorem
Condition in Dynamic Networks

• $k =$ number of Byzantine nodes

• **Condition:** $2k + 1$ nodes must be removed to cut all dynamic paths
Necessary Condition
Sufficient Condition

• Send the message through *all* journeys

• Register the journeys

• When a set of journeys that cannot be cut by $2k$ nodes is collected, accept the message
Why not Cryptography?
PKI
Example
Example

S said Red

S → Red → D
Example

S → [Red] → D

I can't believe it
Trusted third party
Trusted keys

gemalto
security to be free
Trusted Software
Condition in Dynamic Networks with Cryptography

• $k =$ number of Byzantine nodes

• **Condition**: $k+1$ nodes must be removed to cut all journeys
Necessary Condition with Cryptography
Sufficient Condition with Cryptography

- Send the message through all journeys
- When a cryptographically acceptable message arrives, accept it
Case Studies

- Participants in a conference
- Agents in the subway
Participants Interacting in a Conference

Percent of pairs communicating within 10 minutes

- **Unreliable multihop communication**
- **Reliable multi hop communication**
- **Direct communication**
Participants Interacting in a Conference

- Cryptographic multihop communication
- Non-Cryptographic multi hop communication
- Direct communication
Paris Subway Users
Paris Subway Users

Mean communication time (minutes)

Unreliable multihop communication
Reliable multi hop communication
Direct communication
Conclusion

- Classical node-disjoint paths cannot be used as « if and only if » condition
- Falling back to (dynamic) Min-cut works
IF vs. IFF

Percent of pairs communicating within 10 minutes

- Min-cut multihop communication
- Node-disjoint multihop communication
- Direct communication

Hour of beginning of the communication
Actively Mobile Networks
Mobile Agents
Mobile Agents
Mobile Agents
Problems to Solve

• **Exploration** (perpetual or with stop)

• Mapping

• Rendez-vous

• Black hole search

• Capturing an intruder
Models

- **Network** (anonymous vs. ID based)
- **Agents** (anonymous vs. ID based)
- **Synchrony**
- **Initial** (structural) **knowledge**
- **Communications** (none, peebles, whiteboards)
- **Agent** **memory** (infinite, bounded, constant)
Mapping
Rendez-vous

- Two (or more) mobile agents must meet in a graph
- They start on distinct locations
- Computability?
- Complexity?
Rendez-vous in ID Graphs
DFS to find Smallest ID Node
Rendez-vous in A
Rendez-vous in Anonymous Graphs
Rendez-vous in Anonymous Graphs
Anonymous Graphs with Known ID (1,2) Agents
Anonymous Graphs with Known ID (1, 2) Agents
Anonymous Graphs with Known N, ID Agents
Anonymous Graphs with Known N, ID Agents
Anonymous Graphs
with Known N, ID Agents
Black Hole Search
Black Hole Search
Black Hole Search

- A **single** black hole in the graph
- The black hole **does not disconnect** the graph
- Identify each **adjacent edge**
- **Minimize** #agents, #moves
Synchronous Agents
Synchronous Agents
Asynchronous Black Hole Search
Asynchronous Black Hole Search
Asynchronous Black Hole Search
Asynchronous Black Hole Search
Mobile Agents
Mobile Robots
Mobile Robots
Mobile Robots
Mobile Robots

- **Autonomous** (no central control)
- **Homogeneous** (run same algorithm)
- **Identical** (indistinguishable)
- **Silent** (no explicit communication)
Robot Life Cycle

Use sensors to observe the world, get a **snapshot**
Robot Life Cycle

- Look
- Compute
- Move
- Sleep

Execute the algorithm, get a destination point
Robot Life Cycle

- Look
- Compute
- Move
- Sleep

Use motors to move toward the destination point
Robot Life Cycle

- Look
- Compute
- Move
- Sleep

remain **idle** for a while
Visibility
Visibility
Visibility
Limited Visibility
Visibility

- Limited
- Full
Multiplicity Detection

How many robots do you see?

- No detection
- Weak multiplicity detection
- Strong multiplicity detection
Multiplicity

No → Weak → Strong
Multiplicity

No → Local Weak → Local Strong → Global Strong → Global Weak → Local Weak → No
Memory

Algorithm

Persistent Memory

Volatile Memory
Oblivious Robot Memory

Algorithm

Persistent Memory

Volatile Memory
Oblivious Robot Life Cycle

remain **idle** for a while, forget about the past
Memory

Oblivious → Finite → Infinite
Scheduling

Look → Compute → Move

FSYNC

\[
\begin{array}{cccccccc}
\text{r}_1 & 1 & 2 & 3 & 4 & 5 & 6 \\
\text{r}_2 & & & & & & \\
\text{r}_3 & & & & & & \\
\end{array}
\]
Scheduling

Look → Compute → Move

SSYNC

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>$\bigcirc$</td>
<td>$\bigcirc$</td>
<td>$\bigcirc$</td>
<td>$\bigcirc$</td>
<td>$\bigcirc$</td>
<td>$\bigcirc$</td>
</tr>
<tr>
<td>$r_2$</td>
<td>$\bigcirc$</td>
<td>$\bigcirc$</td>
<td>$\bigcirc$</td>
<td>$\bigcirc$</td>
<td>$\bigcirc$</td>
<td>$\bigcirc$</td>
</tr>
<tr>
<td>$r_3$</td>
<td>$\bigcirc$</td>
<td>$\bigcirc$</td>
<td>$\bigcirc$</td>
<td>$\bigcirc$</td>
<td>$\bigcirc$</td>
<td>$\bigcirc$</td>
</tr>
</tbody>
</table>
Scheduling

Look → Compute → Move

ASYNC

1  2  3  4  5  6

r₁   Look   Compute   Move   Look
r₂   Look   Compute   Move   Look   Compute
r₃   Look   Compute   Move
Scheduling

ASYNC → SSYNC → FSYNC
Two Axes
Direction and Orientation
One Axis
Direction and Orientation

[Diagram of one axis direction and orientation]
Two Axes
Direction
Chirality
No Agreement
Overview
Scattering
Scattering

No two robots should occupy the same position

- No deterministic solution

- No termination without multiplicity detection
Scattering

1 toss

$O(\log(n) \log\log(n))$ rounds

Scattering

$n$ robots: $2n^2$ destinations

$O(1)$ rounds

How Many Tosses?

Minimum?

Influence of multiplicity detection?

Relationship with scattering speed?
Optimal Speed

**With** strong multiplicity detection:

Algorithm with optimal \#tosses terminates in $O(1)$ rounds

**Without** strong multiplicity detection:

$O(1)$ rounds scattering of $n$ robots is impossible

How fast can we go?
Gathering
Gathering
Gathering
Gathering

Impossible for two robots

A bivalent configuration
Gathering vs. Convergence

- **Gathering**: robot must **reach** the same point in finite time

- **Convergence**: robots must get **closer** at time goes by
Center of Gravity

\[ \bar{c}[t] = \frac{1}{n} \sum_{i=1}^{n} \mathbf{r}_i[t] \]
Center of Gravity

\[ \vec{c}[t] = \frac{1}{n} \sum_{i=1}^{n} \vec{r}_i[t] \]
Center of Gravity

\[ \overline{c}[t] = \frac{1}{n} \sum_{i=1}^{n} \overrightarrow{r}_i[t] \]
Center of Gravity of Positions

\[ \bar{c}[t] = \frac{1}{p} \sum_{i=1}^{p} \vec{p}_i[t] \]
FSYNC Gathering

\[
\bar{c}[t] = \frac{1}{p} \sum_{i=1}^{p} \bar{p}_i[t]
\]
FSYNC Gathering

\[ \bar{c}[t] = \frac{1}{p} \sum_{i=1}^{p} \bar{p}_i[t] \]
SSYNC Gathering?
SSYNC Gathering?
### Convergence & Gathering

<table>
<thead>
<tr>
<th></th>
<th>Convergence</th>
<th>2-Gathering</th>
<th>n-Gathering</th>
<th>n-Gathering +MD</th>
<th>n-Gathering +MD+WF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FSYNC</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SSYNC</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>ASYNC</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>?</td>
</tr>
</tbody>
</table>
### Convergence & Gathering

<table>
<thead>
<tr>
<th></th>
<th>Convergence</th>
<th>2-Gathering</th>
<th>n-Gathering</th>
<th>n-Gathering +MD</th>
<th>n-Gathering +MD+WF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FSYNC</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SSYNC</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>ASYNC</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>?</td>
</tr>
</tbody>
</table>

# Convergence & Gathering

<table>
<thead>
<tr>
<th></th>
<th>Convergence</th>
<th>2-Gathering</th>
<th>n-Gathering</th>
<th>n-Gathering +MD</th>
<th>n-Gathering +MD+WF</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSYNC</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SSYNC</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ASYNC</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>?</td>
</tr>
</tbody>
</table>

### Convergence & Gathering

<table>
<thead>
<tr>
<th></th>
<th>Convergence</th>
<th>2-Gathering</th>
<th>n-Gathering</th>
<th>n-Gathering +MD</th>
<th>n-Gathering +MD+WF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FSYNC</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SSYNC</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>ASYNC</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Convergence</th>
<th>2-Gathering</th>
<th>n-Gathering</th>
<th>n-Gathering +MD</th>
<th>n-Gathering +MD+WF</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSYNC</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SSYNC</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ASYNC</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Convergence</th>
<th>2-Gathering</th>
<th>n-Gathering</th>
<th>n-Gathering +MD</th>
<th>n-Gathering +MD+WF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FSYNC</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SSYNC</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>ASYNC</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>?</td>
</tr>
</tbody>
</table>

## Convergence & Gathering

<table>
<thead>
<tr>
<th></th>
<th>Convergence</th>
<th>2-Gathering</th>
<th>n-Gathering</th>
<th>n-Gathering +MD</th>
<th>n-Gathering +MD+WF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FSYNC</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SSYNC</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>ASYNC</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>?</td>
</tr>
</tbody>
</table>

Mobile Robots
Conclusion
Mobility as an Adversary

- Can corrupt the distributed state of a network
- Can reduces communication capacity
- Can increase uncertainty
- Can increase protocol complexity
Mobility as a Friend

- Mobility can be the solution to the problem
- Mobility can improve efficiency
- Mobility can promote simplicity
Mobile Networks

Diagram:
- Complexity
- Model
- Problem

The diagram illustrates the relationship between mobile networks, complexity, model, and problem.
Thank You