Distributed Consensus and Intelligent Transportation Systems

Gérard Le Lann

INRIA

Paris-Rocquencourt, France

Contents

Part I: **Basics in Distributed Consensus for Static Networks ***

Part II: **Distributed Consensus in Mobile Networks and Intelligent Transportation Systems**

* See Proc. of ACM PODC, IEEE ICDCS, …

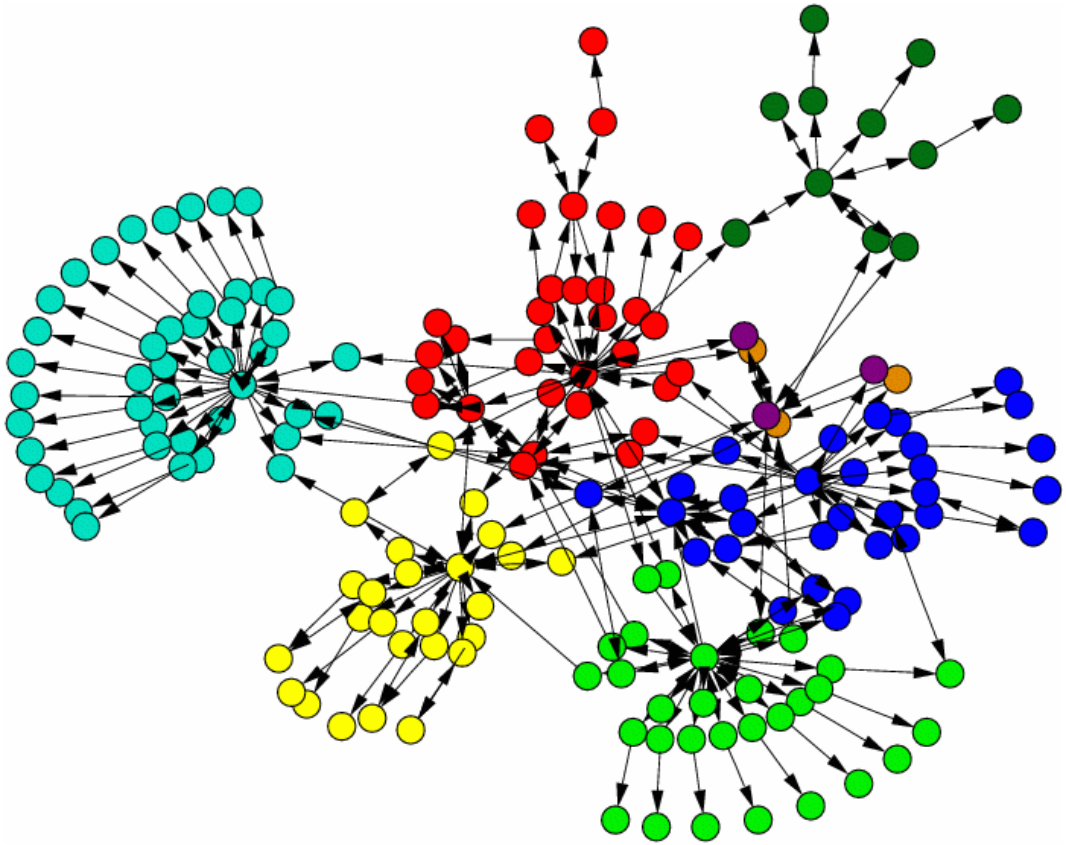
Part I

- Distributed system & failure models
- The DC problem (and Uniform DC)
- A synchronous DC algorithm—Results for synchronous systems
- The FLP impossibility result
- A non synchronous DC algorithm—Results for non synchronous systems

Distributed System?

- Multiple entities
- Interconnection structure (message passing over a network of nodes and links)
- An entity, a node, a link, may experience failures
- No unique statically designated controlling entity
- Global state not “naturally” available
- Bounds on message latency, on processing times?
- Global time (synchronized clocks)?

A distributed system (entities perform different functions)



Focus of this tutorial: Collective Distributed Agreement

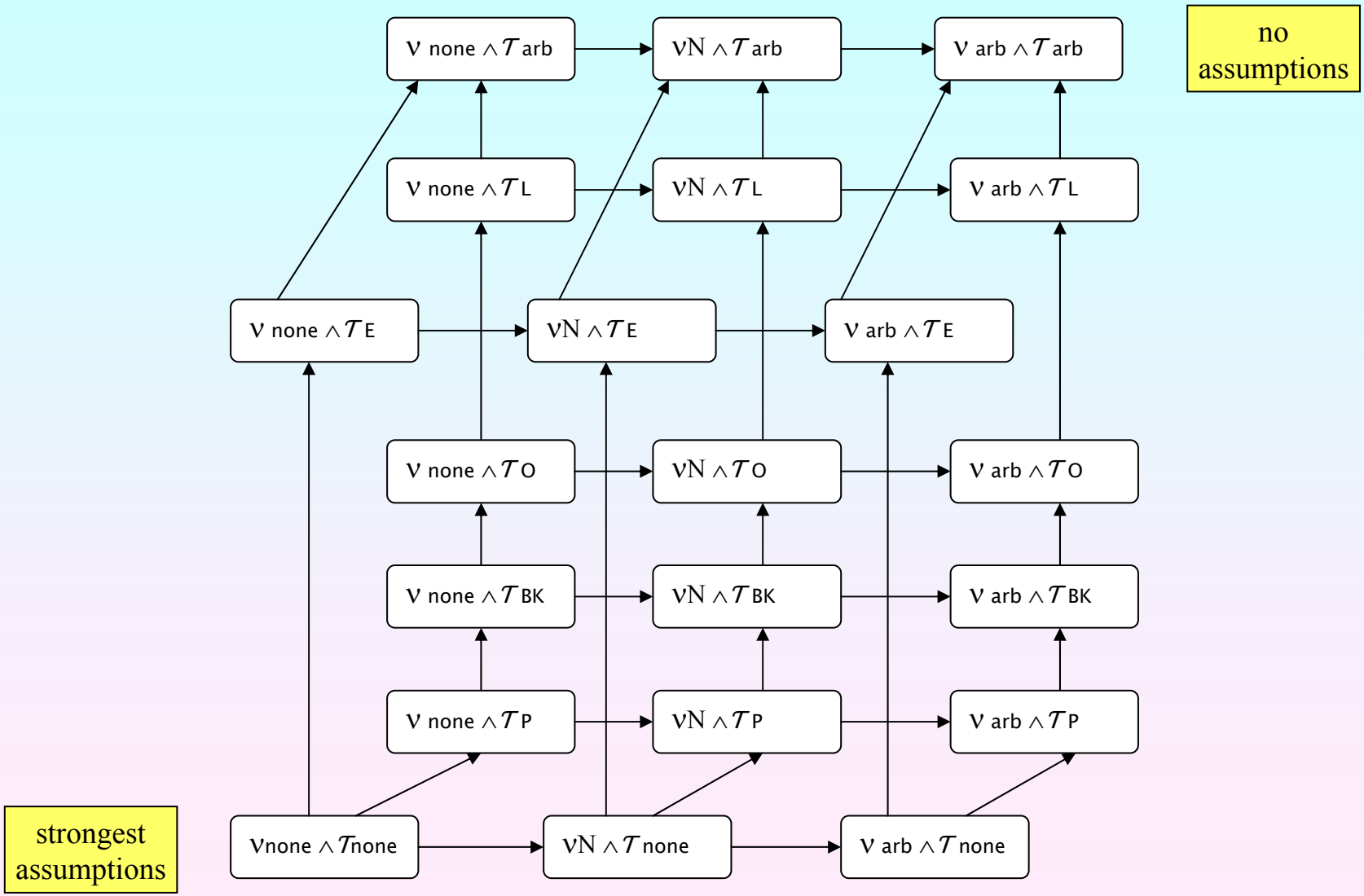
Exact Agreement ≡ Consensus

Essential models (assumptions)

1. Failure models

- Fail-stop ("clean crash")
  - correct behavior until stop, eternal silence thereafter.
- Byzantine
  - no assumptions (arbitrary behavior).
- Others "in between"
  - omission (e.g., message not sent, not received),
  - value (incorrectly valued variable or message),
  - timing (early or late state transition),
  - .../…

A lattice of failure models



Distributed Consensus (DC)

- A set of n processes in a distributed system want to reach agreement, e.g., about:
  - The value of a sensor reading,
  - Whether to accept/reject the results of a computation,
  - …/…
  - Abstractly, on a value in some set V.
- Each process starts with some initial value in V, and processes want to decide on a unique value in V.
- Properties:

  Validity: Any value decided by a process is a value proposed
  Agreement: No two correct processes decide differently
  Termination: Every correct process eventually decides

The twist: ***A (presumably small) number of processes (f) might be faulty, and might not participate correctly in the algorithm.***

Uniform DC

- With DC, a faulty process is free to decide whatever it "wants".
- Might be problematic for certain applications, notably safety-critical ones.
- Uniform DC prevents such scenarios, replacing the agreement property with its Uniform counterpart (Validity and Termination remain unchanged):

  Uniform Agreement: No two (correct or not) processes decide differently

Uniform DC cannot be solved in the presence of Byzantine failure(s).

Why?

(Uniform) DC

- Stated as a generic problem, i.e. does not depend on application or operational semantics (e.g., the "meanings" of values in set V)
- May be viewed as a system-level problem (e.g., middleware level)
- Therefore, is not equivalent to "agent-based" or AI or control theory problems (e.g., sensor fusion, planning, etc.)

➔ In distributed systems, a (Uniform) DC algorithm is an essential "building block" that:

(1) supports any kind of system-wide decision process (semantics-free),

(2) handles/masks "undesired" events (accidental failures, variable delays, malicious "attacks") that cannot be dealt with by application level solutions.

Essential models (assumptions)

1. System models

- Synchrony
  - message latency is bounded, bounds are known,
  - processing times are bounded, bounds are known,
  - processes have synchronized clocks.
  - ➔ A DC algorithm can use these bounds (via timers) and/or global time
- Asynchrony
  - no assumptions.
  - ➔ A DC algorithm can only rest on occurrence of events (incoming messages)
- Partial synchrony or augmented asynchrony
  - some assumptions.

Lock-Step Round Model

- A DC algorithm comprises a sequence of rounds.
- Round # r at process p consists of:
  - Send/multicast a message containing $v_p(r-1)$ to other processes, $v_p(r-1)$ being the value computed by p at the end of round # r-1,
  - Receive messages from current round,
  - Do some local computation (either $v_p(r)$ or Decide (and Halt)).

✧ Synchronous model: A round is "communication-closed", i.e. messages sent within a round are received within that round by all correct processes.

✧ Asynchronous model: Rounds are not "communication-closed".

Types of Round Models
(communication patterns)

❶ n-to-n (peer-to-peer)

❷ n-to-1 & 1-to-n (2 successive rounds), explicit "rotating coordinator"

Typically: process *k mod n* is round *k*'s coordinator.

❸ n-to-1 & 1-to-n (2 successive rounds), implicit "rotating coordinator"

Eventual identical "discovery" of which process is the current coordinator (unchanged unless it is suspected of having failed).

DC Algorithms

◈ **Symmetric**: all rounds are type ❶

◈ **Asymmetric**: all rounds are type ❷ or type ❸

◈ **Hybrid**: rounds of different types alternate

❍ ❍ ❍ ❍ ❍

Three common **<u>"choice" schemes</u>** for selecting a value in a vector:

- Most frequent
- Simple functions (min, max, …) on ordered sets
- Reliance on a (rotating) "coordinator"

An example of symmetric DC assuming synchrony (round duration Δ) and crash failures (f < n)

Set V is totally ordered. Round number denoted r.

Process p: $v_p \equiv$ initial value, $V_p \equiv$ decision value (final).

$prev(p) \leftarrow \perp \quad V_p \leftarrow v_p$

when r = 1, 2, …, f+1 do

begin_round

if $\{prev(p) \neq V_p\}$ then foreach $j \neq p$: send V_p to j endif;

$prev(p) \leftarrow V_p$;

record values received during r in set *rec_from(r)* until timeout Δ;

if $\{rec\_from(r) \neq \emptyset\}$ then $V_p \leftarrow \min [\{V_p\} \cup rec\_from(r)]$ endif;

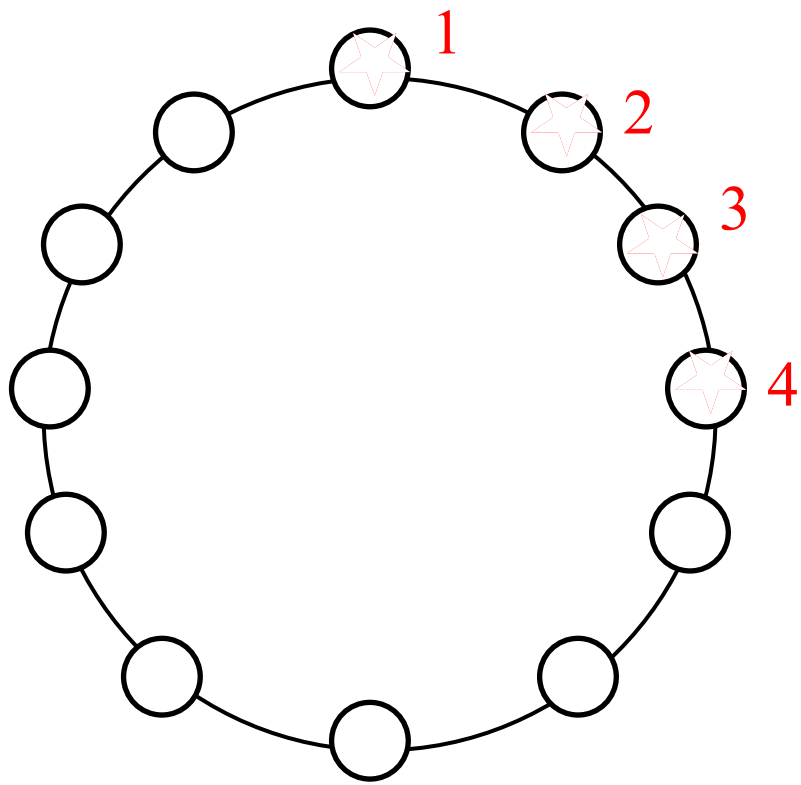
end_round

return V_p

Solving Consensus using a type ❷ Rotating Coordinator Algorithm assuming asynchrony

Works for up to $f < n/2$ crashes

- Processes are numbered 1, 2, …, n
- They execute asynchronous *rounds*
- In round r , the *coordinator* is process (r mod n)



- In round r, the coordinator:
  - tries to impose its estimate as the consensus value
  - succeeds if it does not crash and it is not suspected

Essential results

Synchrony and reliable communications

Time complexity (lower bound): f+1 rounds

- DC under crash/omission failures: feasible if $f < n$
- Uniform DC under crash failures: feasible if $f < n$
- Uniform DC under omission failures: feasible if $f < n/2$
- Weak DC under Byzantine failures: feasible if $f < n/3$

(Weak Validity: if all correct processes propose the same value v, then v is the value decided by every correct process)

DC and synchronous models

Why is it easy?

➔ Physical time can be used to know when to stop waiting for (missing) messages (re. variable Δ in the example algorithm).

- Bad news:

  "It is impossible or inefficient to implement the synchronous model in many types of distributed systems", Nancy Lynch, 1996.

- Stated differently:

  ➢ Coverage of assumptions regarding ***upper bounds*** on delays (computations, communications) is not high enough in many real settings ➔ probability of violating Validity and/or Agreement is too high.

  ➢ Assuming synchrony usually leads to poor performance.

➔Hence the relevance of other models …

FLP Impossibility Result

- [Fischer, Lynch, Paterson, J. of ACM, 1985] proved the following: There is no deterministic solution for DC in asynchronous systems in the presence of one faulty process.

- Proof works even for very limited failures:
  - Communications are reliable.
  - At most one process ever fails, and everyone knows this.
  - The process may simply stop, without warning.*

Two ways out:

- Weaken the DC problem → stochastic variations
- Consider partial synchrony, or augmented asynchrony

* possibly in the midst of broadcasting …

Non Synchronous DC

- Round-based algorithms. Rounds are not communication-closed: a message received in round r may have been sent in round r' < r
- Round r terminates (no more waiting) when n - f messages timestamped #r have been received.
- Duration of a round is not determined solely by message transit delays, depends on failure models to be tolerated.
- Notion of "stabilization": during a stable period, assumptions stated in a DC specification are not violated; namely, failures (that may occur) are "no worse" than those postulated.
- A system may alternate between stable periods and unstable periods, or may initially enter an unstable period, up to some time t (GST), after which it enters a stable period forever.
- Time complexity (lower bound): f+1 "stable" rounds (e.g., possibly after GST).

◈ **Limited information on upper bounds** (e.g., duration of computations performed by some processes) is available:

- Time bounds hold from time 0, or
- Time bounds hold only after GST, GST unknown.

Crash, omission, timing, process failures are accommodated.

Asymmetric (type ❸) algorithms ([Dwork, Lynch, Stockmeyer, 1988], [Lamport, 1989]):

- ➢ Keep trying to choose a leader, who tries to coordinate agreement.
- ➢ Many attempts can fail.
- ➢ Once system stabilizes (possibly more than once), unique leader is chosen, coordinates agreement.

The tricky part:

Ensuring failed attempts don't lead to inconsistent decisions.

◈ **Ratio upper bound/lower bound Θ is known for some delays**

➔ the Θ–Model [Le Lann, Schmid, 2003]

Crash, omission, timing, process failures are accommodated.

Interesting feature with many systems: Θ is not violated when (postulated) upper bound is violated.

Of interest for safety-critical applications.

◈ **Unreliable Failure Detectors** [Chandra-Hadzilacos-Toueg, 1996]

An FD is a distributed oracle that provides hints about the operational status of processes. ➔ LS: list of suspected processes at every process. However, hints may be incorrect.

Crash process failures are accommodated.

FD defined after abstract properties

- Strong Completeness
  - Eventually, every process that crashes is permanently suspected by ***every*** correct process
- Weak Completeness
  - Eventually, every process that crashes is permanently suspected by ***some*** correct process

- Strong Accuracy
  - No process is suspected before it crashes
- Weak Accuracy
  - **Some** correct process is never suspected
- Eventual Strong Accuracy
  - There is a time after which correct processes are not suspected by any correct process
- Eventual Weak Accuracy
  - There is a time after which **some** correct process is never suspected by any correct process.

Failure Detectors Classes

Completeness	Accuracy			
	Strong	Weak	Eventual Strong	Eventual Weak
Strong	*Perfect* *P*	*Strong* *S*	*Eventually Perfect* ◊*P*	*Eventually Strong* ◊*S*
Weak	*Q*	*Weak* *W*	◊*Q*	Eventually Weak ◊W

◊S: Weakest FD for solving DC

Solving DC using ◊S

Code for process p_k $1 \leq k \leq n$

$est_k \leftarrow$ initial value v_k;
for r from 1 to f+1 **do**
if (r = k) **then** % p_k is the coordinator of round k %
    multicast <est_k>
else wait until <est_r> is received from p_r or $p_r \in LS(p_k)$
    % p_r suspected by p_k %
    if <est_r> received from p_r **then** $est_k \leftarrow est_r$
    endif
endif
endfor
decide est_k

Challenges
New frontiers in distributed computing theory

- Transient failure models

  Lower bounds for permanent failure models to be revisited (too much pessimistic).

- Timeliness (real-time) properties

  Upper bounds (physical time) for executing DC fully

  ➢ "*Decision must be made by some strict deadline*" replaces "*Eventual Termination*".

- Mobile wireless networks

  Much worse behaved than traditional wired networks:

  - No one knows who the participating processes are.
  - The set of participants may change.
  - Mobility models?
  - Unreliable communications.

DC and mobile networks

Some early results:

[DLS 1988], [Lamport 1989], assume that eventually there is a connected majority of processes.

[Santoro, Widmayer, 1989/90]: DC is impossible if as few as n-1 of the n^2 possible messages sent in a round can be lost.

❖ ❖ ❖ ❖ ❖

➲ Wireless ad hoc networks

➲ Wireless networks with common space/time referential (GPS, …)

➲ Hybrid networks:
wireless (ad hoc) nets interconnected via a wired (backbone) net.

The same categorization applies to VANETs and ITS.

⇨ **DC solutions for such systems apply accordingly**.

Part II

II.1. Some recent results on DC in mobile networks

II.2. DC in VANETs/ITS

(3 safety-critical scenarios)

II.1 – DC in mobile networks

1) Wireless ad hoc networks

➢ **Virtual Mobile Nodes** [Dolev et al., 2004]:

- Derived from the idea of "compulsory" protocols, which require a subset of mobile nodes to move in a specific manner.
- Apply this constraint to abstract nodes (emulated by real nodes) ➔ virtual mobile nodes (VMNs) that move in a predetermined, predictable manner, possibly completely uncorrelated with the motion of real nodes.

➔ a VMN network may be connected even when the network of mobile nodes is disconnected.

- Hence, group communication is feasible within a network of VMNs ➔ DC can be solved (under conditions valid with wired networks, if so desired—depends on imposed VMN motion).

2) Wireless networks with geographical referential

➢ **Collision Detectors** [Chockler et al., 2005]:

- Single-hop MAC layer collision detectors (synchrony assumptions), crash failures (except while broadcasting)
  ➔ single-hop DC solved in up to 5 rounds of 2 phases each (propose, veto).
- Multi-hop DC: non-overlapping grid squares (global knowledge), each node knows its approximate location in the grid; step 1: single-hop DC is conducted in each grid square;

  step 2: all nodes gossip the grid square consensus values;

  it is proved that every correct node eventually receives a value for every grid square ➔ decision is possible.

3) Wireless (ad hoc) networks

➢ **Overlays, clustering, dominating sets**

- Network of dynamic cells, each cell "controlled" by a mobile node ("leader"), with companion backups.
- A node can move within a cell or between two cells.
- Any node—hence a leader—may fail or become unreachable ➔ leader (and backups) election required within a cell.
- DC achievable:
  - ✧ within a cell (leader is the ("rotating") coordinator),
  - ✧ across cells (among cell leaders): DC based on knowledge of current virtual topology (ensuring that the network of leaders remains connected).

4) Hybrid networks (wired/wireless)

➢ **Overlays, clustering, dominating sets**

- Network of dynamic cells, each "controlled" by a base station.
- A node can move within a cell or between two cells.
- Possible assumption: a base station does not fail ➔ redundant (diversified) design & implementation & proofs required.
- Other possible assumption: a base station may fail ➔ group membership is needed among base stations.*
- DC achievable:
  - ✧ within a cell (base station is the (non "rotating" or "rotating") coordinator),
  - ✧ across cells (among base stations ➔ strictly identical to DC in wired networks).

* Group membership is as hard as DC.

II.2 – DC in VANETs/ITS

Some essential features

- Motion of vehicles is constrained (parkings, streets, roads, highways)
- Location awareness, positioning, global time referential (GPS, Galileo, ground-based infrastructures)
- Trajectories of vehicles can be monitored and enforced with high precision (via on-board or roadside equipments).

  Technologies are progressing constantly.

II.2 – DC in VANETs/ITS

ESA, 2007: *Future missions involving formation flying will require very accurate measurements of the distances between spacecraft. An ESA-funded study of high-precision optical metrology has developed prototype systems capable of relative positional accuracies of the order of 0.1 millimeter over distances of tens or hundreds of meters.*

➲ For most safety-critical scenarios, vehicles are in close vicinity of each other ➔ one may assume (i) 1-hop radio links, (ii) synchrony.

➲ Moreover, all future vehicles will be equipped with sensors, radars, etc. ➔ there is some additional "network" (pure signaling) that is separate from a radio-based message passing network ➔ transient communication failures can be compensated.

Note: The coincidental failure of both networks for a given vehicle is to be proved extremely unlikely (this entails on-board redundancy). A vehicle whose all equipments are down is still detectable by neighboring vehicles.

II.2 – DC in VANETs/ITS

Given the above:

- In most safety-critical scenarios, it is impossible to experience unbounded communication delays and/or "long lasting" network(s) partitioning.
- For other scenarios, availability of accurate space-time coordinates (and control laws) permits to compensate for communication failures and/or unbounded communication delays.

Consequently:

- **Impossibility results for "traditional" DC do/may not apply in many (future) VANETs/ITS.**

Conversely:

- Safety properties must hold for some VANETs/ITS applications, not addressed in the traditional DC literature (fault-tolerance does not imply safety).

II.2 – DC in VANETs/ITS

- What about "Byzantine drivers"? It is demonstrably possible to build provably correct Byzantine fault-tolerant embedded systems, to be installed on vehicles. But a driver may behave arbitrarily abnormally, no matter what.

- Societal and legal issues interfere with technical/scientific issues.* For example, is vehicle driving bound to resemble airplane piloting? Namely, when it is provably safer to rely on embedded dependable systems, should it be made ***physically*** impossible for a driver to "intervene"? Who is to be held responsible in case of recurrent accidents:
  - A safety regulation authority which has been unacceptably "conservative"?
  - A company which produces a flawed implementation of a system whose design has been proved correct?
  - A car maker which has contracted such a company?

* *Security issues left out, due to lack of time.*

II.2 – DC & On-Ramp Merging

Ramp Metering

Current solutions not efficient enough.
On-ramp merging should be possible at “high” speed, under high density traffic conditions.

II.2 – DC & On-Ramp Merging

E: entrant car.

SG: group of vehicles on highway, within mutual radio range, which are at least s seconds (or m meters) away from the merging ramp endpoint at time t. Safety regulations ➔ ∃ lower bound for s.

G ⊂ SG, vehicles on rightmost lane L.

G'⊂ SG, vehicles on all lanes, except rightmost one.

A vehicle has a unique name ➔ id(x) for vehicle x.

DC initiated by E at time t:

Bcast M ≡ {start DC (ORM, id(E), …)}

Cars ∈ G': starting when M received, merging with lane L is prohibited until DC terminates (they need not participate in DC).

Cars ∈ {E ∪ G}: must agree on "where" to create a slot for E in lane L ➔ they run DC.

Which DC algorithm?

II.2 – DC & On-Ramp Merging

Let w denote the "winning" car (slot is created just behind)

- Car E appears to be a natural "coordinator" ➔ DC is of type asymmetric, based on the non rotating coordinator scheme (?)
- Non rotating? If car E equipment fails while running DC, and E is aware, car E is stopped on the ramp.
- The opposite scenario is safety-critical: car E equipment fails while running DC, but E is unaware ➔ car E keeps proceeding with merging, despite the fact that cars in lane L may believe that E has stopped on the ramp (since they don't hear from E).
- Therefore, once initiated by E, DC must be executed fully by all cars involved, no matter what, i.e. there must be an agreement on which car is w, ensuring that a slot is created for E.

II.2 – DC & On-Ramp Merging

- But E must be "replaced" in the safety-critical scenario

  ➔ *DC cannot be based on the non rotating coordinator scheme*!

- The group of cars involved is unknown at time t, and learning about the *exact* group membership may conflict with imposed reaction times ➔ *a rotating coordinator scheme is out of question*

  ➔ DC can only be a symmetric algorithm.

- Shortly after time t, some number (?) of cars in (unknown) set G will have received M. Every such car must be able to identify a *convenient* subgroup in charge of executing DC, subgroup denoted $\Gamma(x)$ for car x

  ➔ The choice rule (in DC algorithm) must ensure that *any* $\Gamma(x)$ is *sufficient* approximate knowledge.

II.2 – DC & On-Ramp Merging

- We need to have at least n cars involved in every DC round

  ➔ size of any set $\Gamma(x)$ has lower bound $(n + f)$.

  Note the "natural" self-adaptive phenomenon:
  This lower bound is not met when there are "not enough" cars in set G,
  in which case there are many "natural" slots in L for car E.

- Cars within radio range ➔ synchronous DC.

  ➔ lower bound: $s > (f+1)\,\Delta + \sigma$

  Δ :: round duration σ :: safety margin

- Failures can be crash or omission. We want Uniform DC

  ➔ $f < n/2$.

- Considering IEEE 802.11a broadcast mode, expectable values for n in the order of 5 to 10, and $\sigma = 2\,(f+1)\,\Delta$, s' lower bound is less than 1 second ➔ m less than 40 meters for freeway speed in the order of 140 km/h, which is within radio range capabilities.

II.2 – DC & On-Ramp Merging

- $\forall x \in G$, round 1 at car x is triggered upon receiving M from E.
- Every car wants to be the "winner" w ➔ initial value sent out by car x is id(x).
- The biggest id received in a round by a car is the provisional winner (one could have chosen the smallest).
- Group Γ for a car comprises the n-1+f other highest id's received.

At round r, DC happens to be run by E ∪{Γ'}, Γ' being the union of all Γs "in use" at round r (Γ' is unknown to every single car).

- At the end of round 1, each car (x) computes its own vision of:

  (1) provisional winning id, denoted w'(x), (2) set Γ(x).

- If not in set Γ(x), car x stops participating in DC (safe since there are at least n other cars involved, and f < n/2).

II.2 – DC & On-Ramp Merging

DC code for rounds 2, …

$V_x \leftarrow w'(x)$
when r = 2, …, f+1 do
begin_round
  foreach $j \neq x$: send V_x to j
  record values and id's received during r
  in set *rec_from(r)* until timeout Δ;
  $\Gamma(x) \leftarrow$ (n-1+f) highest id's in set *rec_from(r)*
  $V_x \leftarrow \max [\{V_x\} \cup rec\_from(r)]$;
end_round
return V_x

E merges between car w [which may accelerate mildly] and car that follows w [which slows down, if necessary]

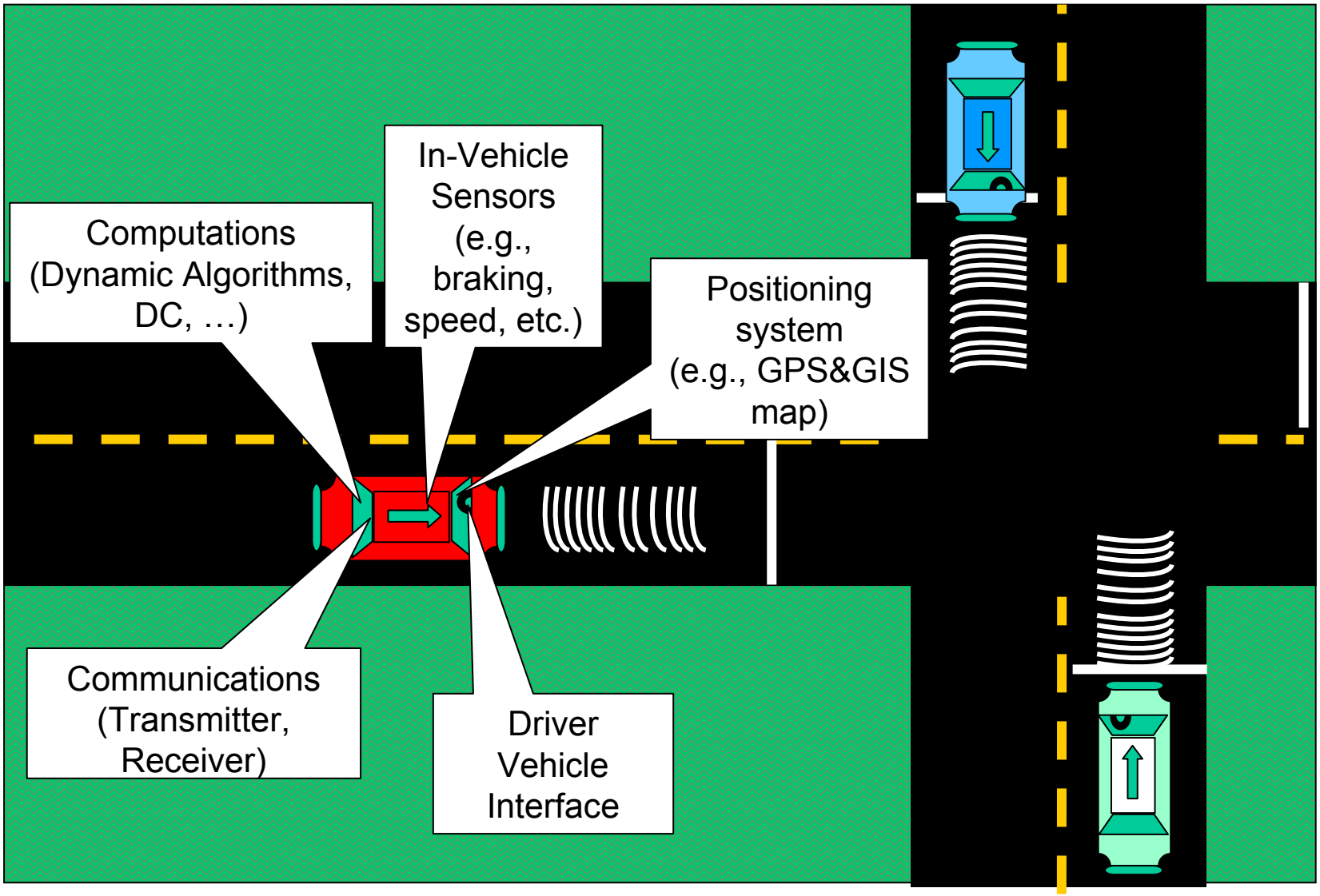
Final value V_x is id(w) w = "winning" car

j: mute variable denoting every member of set $\{E \cup \Gamma(x)\}$

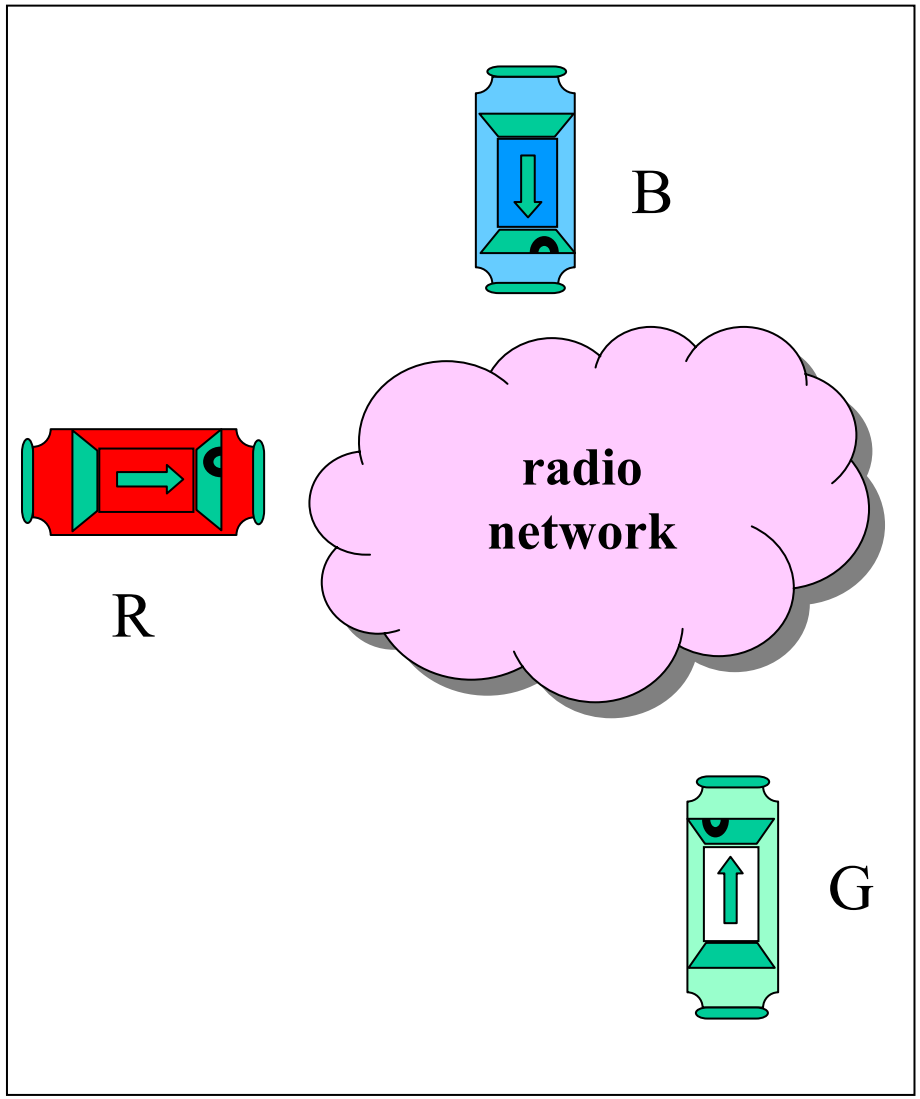
II.2 – DC & On-Ramp Merging

- If required, one may consider a Byzantine fault-tolerant DC algorithm.
- Under the synchronous condition ($f < n/3$), safety is ensured despite up to f arbitrarily malicious cars (masquerading, cheating, etc.). For example, a group Γ of 6 cars (E being the 7^{th} one) would be needed for tolerating up to 2 Byzantine cars (none of which can be w or w's successor).
- It is possible to envision future cars equipped with visual devices that would "display" DC based decisions (drivers can learn about decisions made by other cars' on-board systems).

II.2 – DC & Safe Intersections



II.2 – DC & Safe Intersections



✓no network partitioning and 1 hop between cars

➢ *synchrony assumptions*

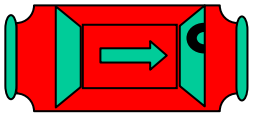
✓n = 3; assume each car hears from itself & from only 1 other car

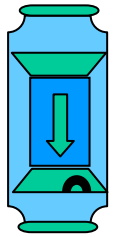
➢ *f = 1 (omission failure)*

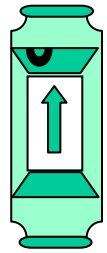
Can Uniform DC be solved?

II.2 – DC & Safe Intersections

Round 1







R: initial value = G
Bcast "G"
receives "G" & "R"
$value_1$ ← "G > R"

B: initial value = B
Bcast "B"
receives "B" & "G"
$value_1$ ← "B > G"

G: initial value = R
Bcast "R"
receives "R" & "B"
$value_1$ ← "B > R"

▪ ▪ ▪ (***choice rule on car id's is « min »***) ▪ ▪ ▪

Round 2

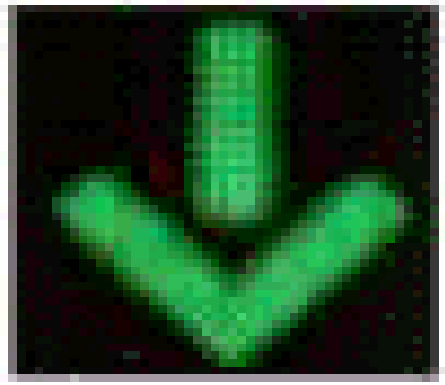
Since 2 cars Bcast "B > ▪ ", $value_2$ ← "B first" everywhere.

Since G appears in 2 $value_1$ messages, $value_2$ ← "G second" everywhere

Since f = 1, this is the last round ➔ Decision = $value_2$ everywhere

➔ car B crosses 1st, car G crosses 2nd, car R crosses 3rd.

II.2 – DC & Reversible Lane Control



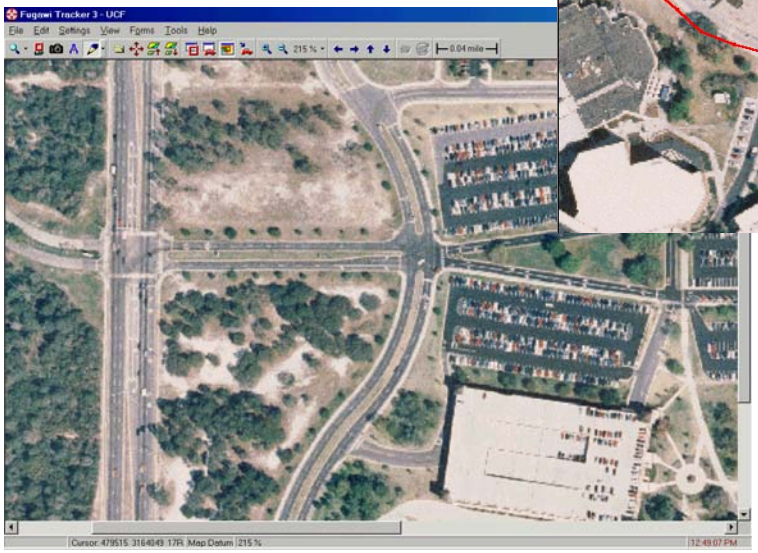
- Use right-of-way more efficiently and economically
- Update to satisfy the dominant flow of traffic

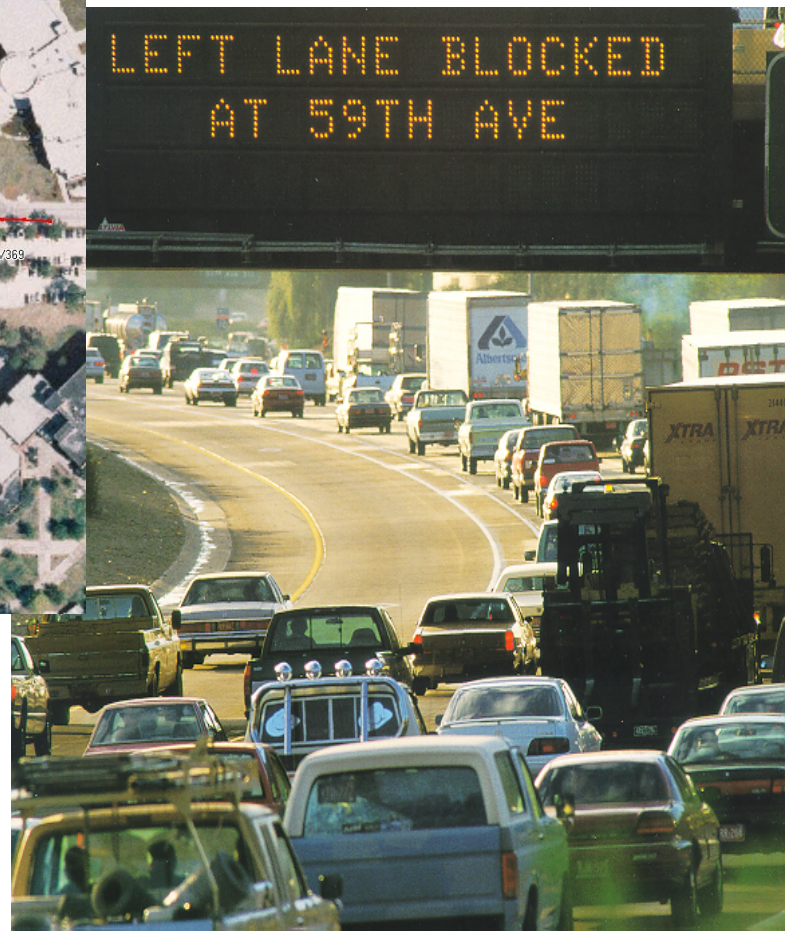
II.2 – DC & Reversible Lane Control

Implies Situational Awareness, i.e. multiple monitoring stations

Currently:

- **A few stations**
- **Human control**





II.2 – DC & Reversible Lane Control

DC with current systems?

➔ Helps in improving safety and efficiency (DC solved with "traditional" algorithms (non mobile stations only are involved)).

Future systems:

- Many static and mobile stations (scattered along highways/roads)
- Fully automated (human supervision ≠ human made decisions)

Stations in charge of monitoring a given set of overlapping segments must reach agreement on various decisions, such as, e.g.:

II.2 – DC & Reversible Lane Control

- Should traffic flows be switched to a new mode?
  (yes/no ➔ Binary Consensus:: initial values are 0 and 1)
- If yes, which lane(s) should be reversed?
- If more than 1 lane at stake, all at once, or one after the other?
- If the latter, in which order, which latency between 2 given consecutive reversals?
- …/…

These are examples of scenarios where DC arises on hybrid networks. Note that some mobile stations might be in charge of enacting decisions made (such as switching traffic lights), such decisions being highly safety-critical.

Of course, it is possible to derive “polarized” DC algorithms from “plain” DC ones.

“Polarization” consists in applying a “choice function” or a “filter” to the value vector built at the end of a round.

Consequently, rather than ending up with a decision which is any initial proposed value, one may bias the decision, according to application/operational semantics.

See, e.g., the On-Ramp Merging scenario, where the choice of which car is the “winner” may be “optimized” (especially useful when many cars want to merge).

The End