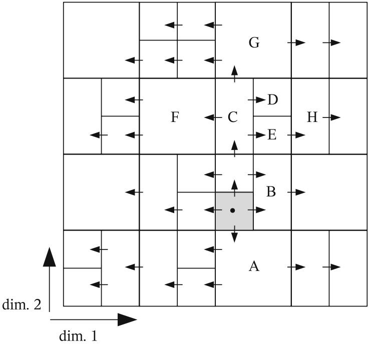
Mechanical Support for Efficient Dissemination on the CAN Overlay Network

- Francesco Bongiovanni -

INRIA Sophia Antipolis
OASIS team
Work done in collaboration with Dr. Ludovic Henrio

12 October 2011



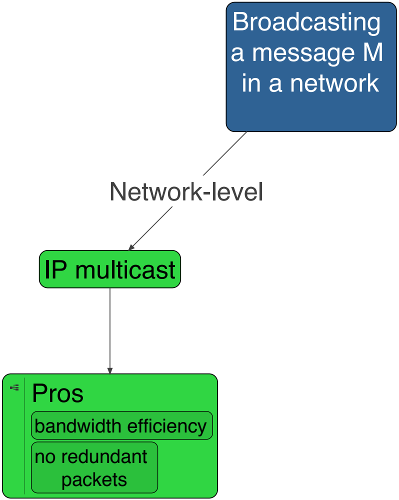
↓

correct-by-construction efficient $\frac{\textit{broadcast}}{\textit{P2P protocol}}$ *

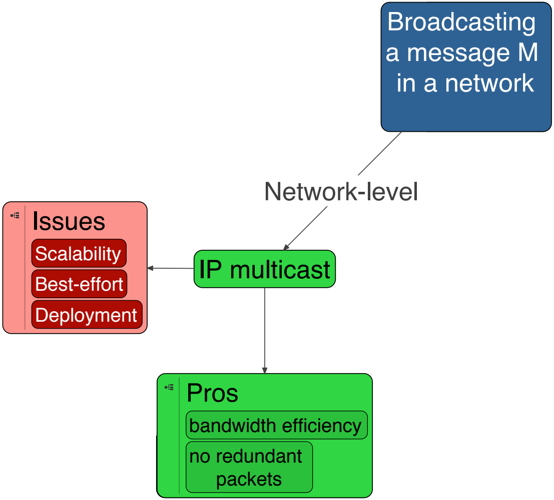
QED

* conditions apply

IP multicast

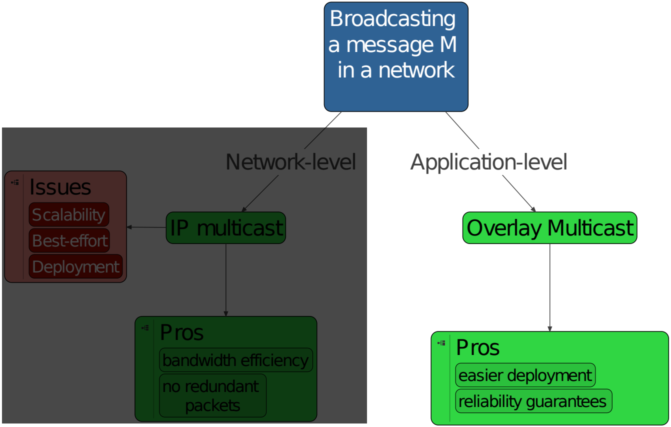


IP multicast Issues

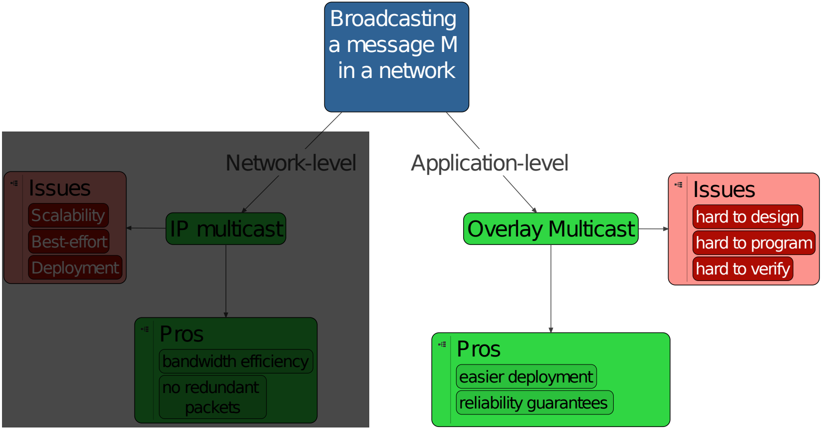


- Can we achieve efficient multi-point delivery without support from the IP layer ?

- Can we achieve efficient multi-point delivery without support from the IP layer ?



- Can we achieve efficient multi-point delivery without support from the IP layer ?



- Can we build such delivery mechanism correctly and formally prove its properties ?

- Can we build such delivery mechanism correctly and formally prove its properties ?

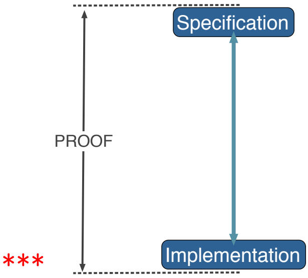
$\longrightarrow$ Using an interactive proof assistant

- Can we build such delivery mechanism correctly and formally prove its properties ? ***

$\longrightarrow$ Using an interactive proof assistant

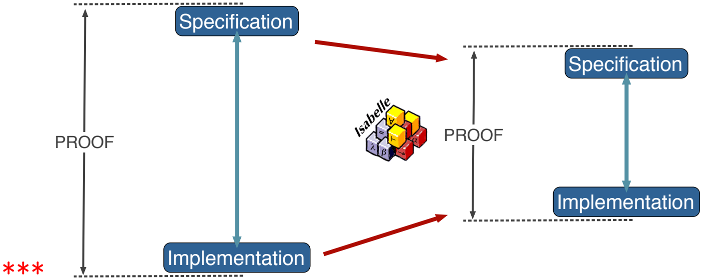
- Can we build such delivery mechanism correctly and formally prove its properties ? ***

$\longrightarrow$ Using an interactive proof assistant



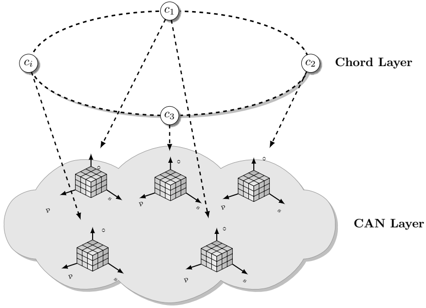
- Can we build such delivery mechanism correctly and formally prove its properties ? ***

⟶ Using an interactive proof assistant



Context - FP7 STREP PLAY

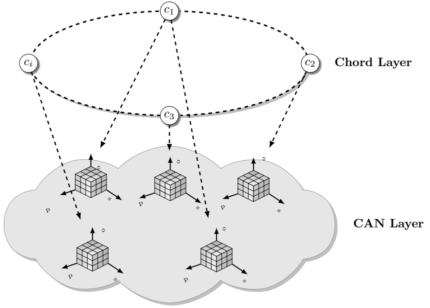
- *Event Cloud* : Publish/Subscribe system for large scale RDF data * processing and storage (based a modified version of CAN).



* *RDF quadruple* $=$"{*subject*, *predicate*, *object*, *context*}"

Context - FP7 STREP PLAY

- *Event Cloud* : Publish/Subscribe system for large scale RDF data * processing and storage (based a modified version of CAN).



* *RDF quadruple* ="{*subject*, *predicate*, *object*, *context*}"

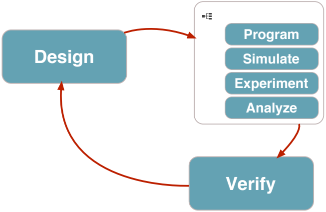
Need for dissemination algorithms for retrieving RDF data *efficiently*

Motivation

- Dissemination algorithms on top of large-scale P2P systems are hard to :

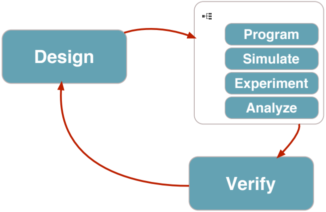
Motivation

- Dissemination algorithms on top of large-scale P2P systems are hard to :



Motivation

- Dissemination algorithms on top of large-scale P2P systems are hard to :



- Distributed Algorithms are subtle & error-prone...yet few have been formally verified
- *Formal methods* to the rescue

Mechanizing formal proofs

What's in it for you ?

- Papers with "just" a description of the algorithm

Mechanizing formal proofs

What's in it for you ?

- Papers with "just" a description of the algorithm
  $\updownarrow$ [Chord, CAN, Pastry,...]

Mechanizing formal proofs

What's in it for you ?

- Papers with "just" a description of the algorithm
  $\updownarrow$ [Chord, CAN, Pastry,...]
- Papers with a more precise description of the algorithm and rough hand proofs of correctness

Mechanizing formal proofs

What's in it for you ?

- Papers with "just" a description of the algorithm
  $\updownarrow$ [Chord, CAN, Pastry,...]
- Papers with a more precise description of the algorithm and rough hand proofs of correctness
  $\updownarrow$ papers with formal hand proofs

Mechanizing formal proofs

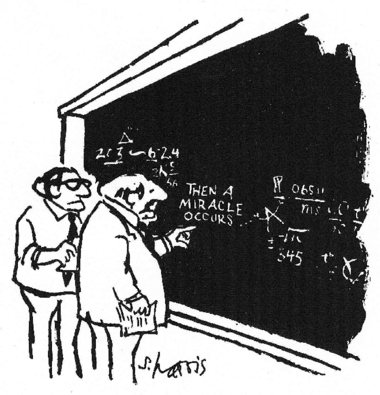
What's in it for you ?

- Papers with "just" a description of the algorithm
  ↕ [Chord, CAN, Pastry,...]
- Papers with a more precise description of the algorithm and rough hand proofs of correctness
  ↕ papers with formal hand proofs
- Papers with machine-checkable proofs ([Charron-Bost & Merz 2009])

Mechanizing formal proofs

It's all about trust...

- Nothing is ever certain, but we can achieve high levels of reliability...
- ...and theorem provers are more reliable than most human hand proofs.

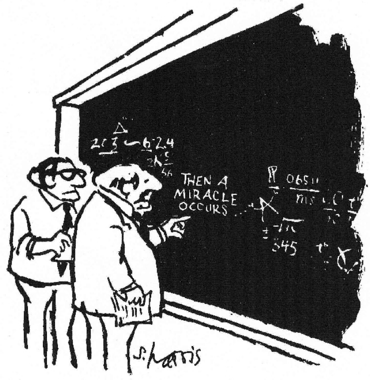


"I think you should be more explicit here in step two."

Mechanizing formal proofs

It's all about trust...

- Nothing is ever certain, but we can achieve high levels of reliability...
- ...and theorem provers are more reliable than most human hand proofs.
- Working in an interactive theorem prover gives you :
    - Confidence in correctness (*assuming the theorem prover is sound)
    - Automatic assistance in tedious parts of the proof



"I think you should be more explicit here in step two."

Contributions

- *Correct* construction of an efficient broadcast algorithm using **Isabelle/HOL** interactive proof assistant.

Contributions

kind of properties to prove

- *Correct* construction of an efficient broadcast algorithm using **Isabelle/HOL** interactive proof assistant.

- The type of props we would like to prove:
  - Efficiency: a node receives the message only once
  - Coverage: all the nodes within a zone must be covered
  - Termination: all the nodes have received the message (only once)

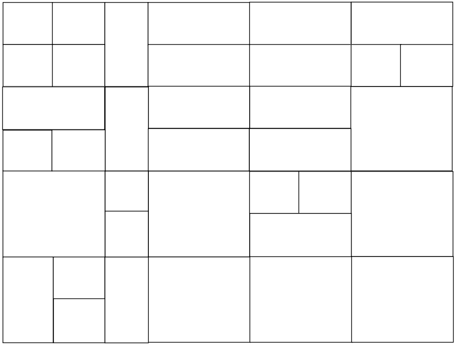
Background

Content **A**ddressable
Network

Background

Content **A**ddressable **N**etwork

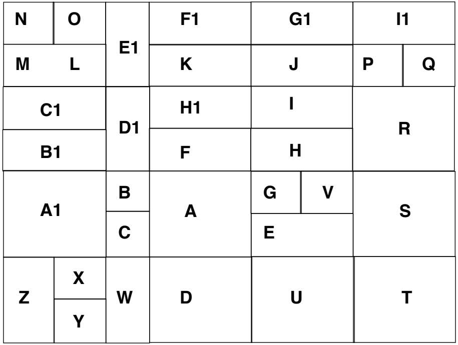
- $d - dimensional$ Cartesian coordinate *space*



Background

Content **A**ddressable **N**etwork

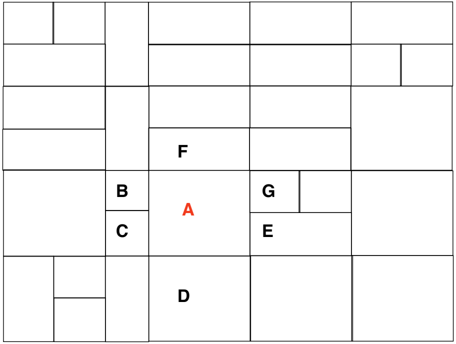
- *d* − *dimensional* Cartesian coordinate *space*
- each peer manages a portion of the *space*



Background

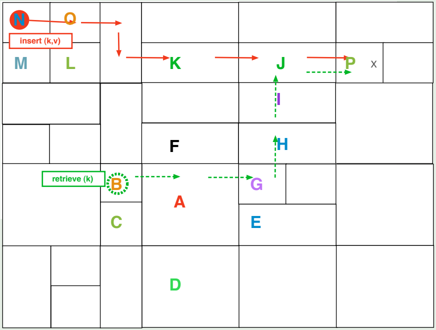
Content **A**ddressable **N**etwork

- *d − dimensional* Cartesian coordinate *space*
- each peer manages a portion of the *space*
- a peer only knows its adjacent neighbors



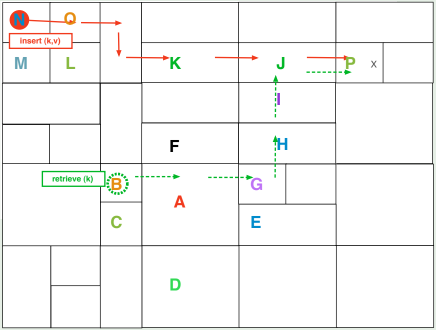
Background

Routing in CAN



Background

Routing in CAN

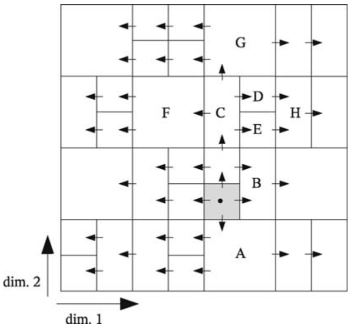


State overhead: $O(d)$

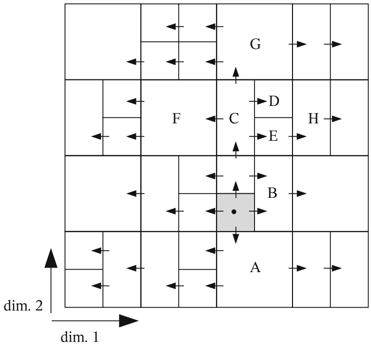
Lookup complexity: $O(dN^{\frac{1}{d}})$

Contributions

Main idea



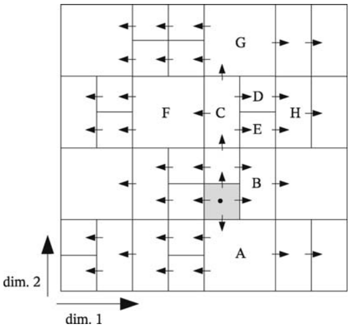
From



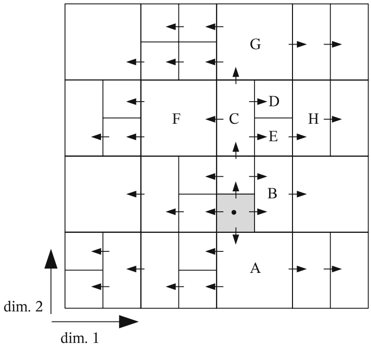
To

Contributions

Main idea



From

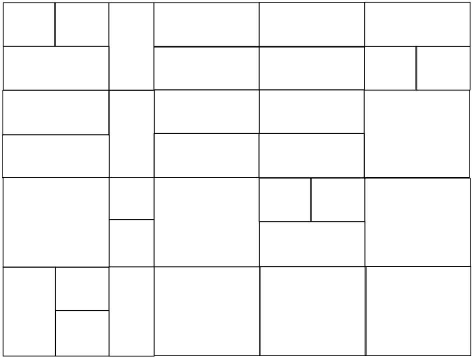


To

formally

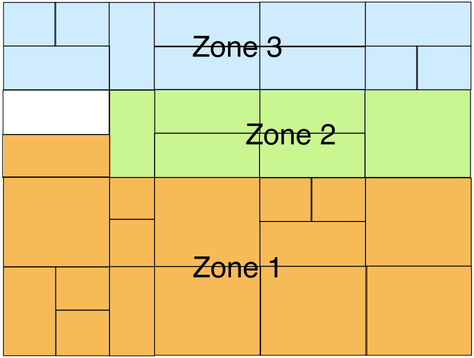
Contributions - intuition

- We split the CAN into *Zones*



Contributions - intuition

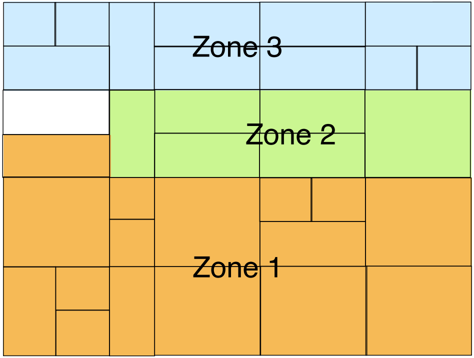
- We split the CAN into *Zones*
- *Zones* do not intersect



* Assuming a method for Zone division
E.g.: Knowing the Space coordinates and its neighbors coordinates, initiator computes the geometrical difference and assigns non overlapping zones to each of its neighbors

Contributions - intuition

- We split the CAN into *Zones*
- *Zones* do not intersect
- There are valid paths within zones (finite)
- Neighbors are connected
- ...



* Assuming a method for Zone division
E.g.: Knowing the Space coordinates and its neighbors coordinates, initiator computes the geometrical difference and assigns non overlapping zones to each of its neighbors

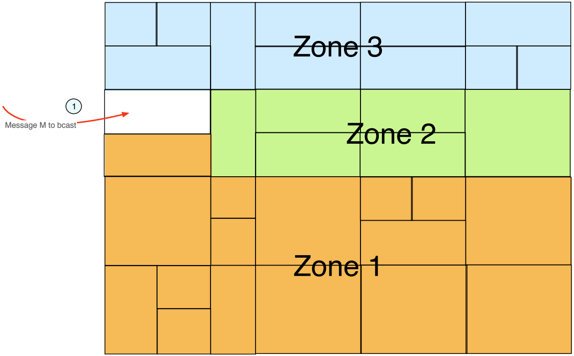
Contributions

Algorithm

Contributions

Algorithm

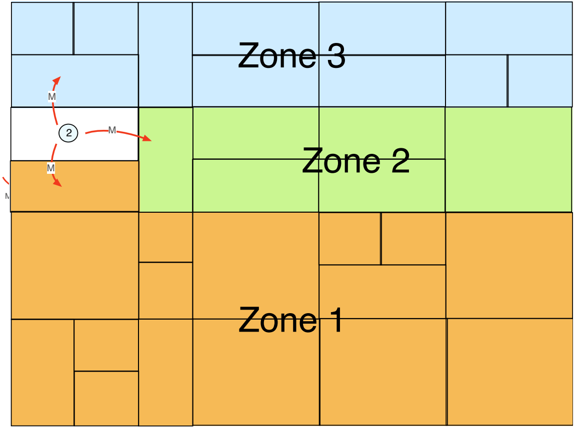
- a message M to bcast is received by a peer (initiator)



Contributions

Algorithm

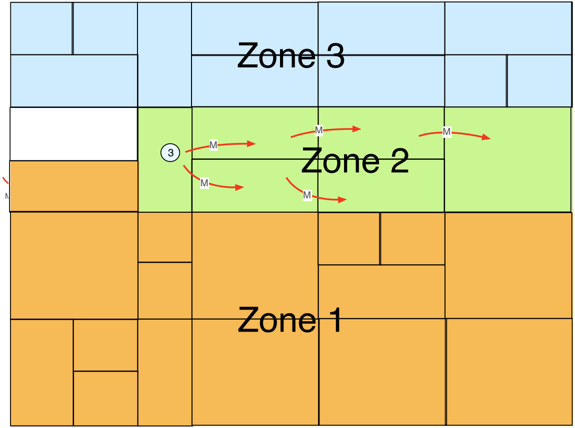
- a message M to bcast is received by a peer (initiator)
- the initiator sends M to all its neighbors



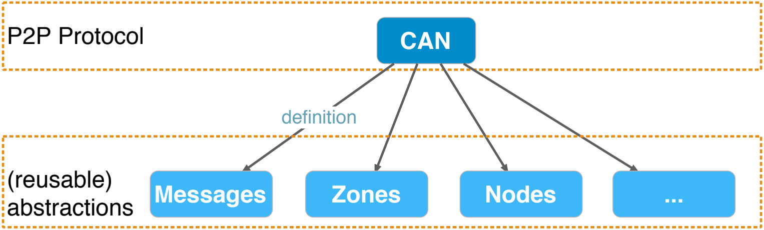
Contributions

Algorithm

- a message M to bcast is received by a peer (initiator)
- the initiator sends M to all its neighbors
- within a zone, M is propagated and stays within the zone



Contributions
the formalization process



Contributions

Definitions

Definitional approach

Contributions

Definitions

Definitional approach

- definition of a *Node*, *Space*

```
typedef Node = "{n :: nat. n <= max_node }"
by auto;

typedef Degrees = "{n :: nat. n <= degree }"
by auto;

consts SIZE_Space :: nat

typedef Space = "{n :: nat. n <= SIZE_Space }"
by auto;
```

Contributions
Definitions

Definitional approach

- definition of a *Node*, *Space*
- definition of a *Message*

```
types Message = "nat × nat × nat × Zone"

abbreviation message::
"nat => nat => nat => Zone => Message"
("</ _|/ _,/ _,/ _>" [0, 0, 0] 70)
where "<m|s,d,Z> ≡ (m,s,d,Z)"
```

Contributions
Definitions

Definitional approach

- definition of a *Node*, *Space*
- definition of a *Message*

```
types Message = "nat × nat × nat × Zone"

abbreviation message::
"nat => nat => nat => Zone => Message"
("</ _|/ _,/ _,/ _>" [0, 0, 0] 70)
where "<m|s,d,Z> ≡ (m,s,d,Z)"
```

Contributions

Definitions

Definitional approach

- definition of a *Node*, *Space*
- definition of a *Message*
- definition of a *CAN*
- ...

```
typedef CAN =
"{(nodes::nat set, Z :: nat => Zone,
neighbours:: (nat × nat) set) .

finite nodes ∧
finite neighbours ∧
(∀ x y. (x,y)∈ neighbours ⟶(y,x) ∈ neighbours) ∧
(∀ x. (x,x)∉ neighbours) ∧
(∀tup. ∃n∈nodes. tup ∈ (Z n)) ∧
(∀ N∈nodes. ∀ N'∈nodes. N≠N' ⟶¬ intersects (Z N) (Z N')) ∧
(∀ N∈nodes. (Z N)≠{})}"
```

Contributions
Definitions

Definitional approach

- definition of a *Node*, *Space*
- definition of a *Message*
- definition of a *CAN*
- . . .

```
typedef CAN =
"{(nodes::nat set, Z :: nat => Zone,
neighbours:: (nat × nat) set) .

finite nodes ∧
finite neighbours ∧
(∀ x y. (x,y)∈ neighbours ⟶(y,x) ∈ neighbours) ∧
(∀ x. (x,x)∉ neighbours) ∧
(∀tup. ∃n∈nodes. tup ∈ (Z n)) ∧
(∀ N∈nodes. ∀ N'∈nodes. N≠N' ⟶¬ intersects (Z N) (Z N')) ∧
(∀ N∈nodes. (Z N)≠{})}"
```

Contributions
Definitions

Definitional* approach

- definition of a *Node*, *Space*
- definition of a *Message*
- definition of a *CAN*
- ...

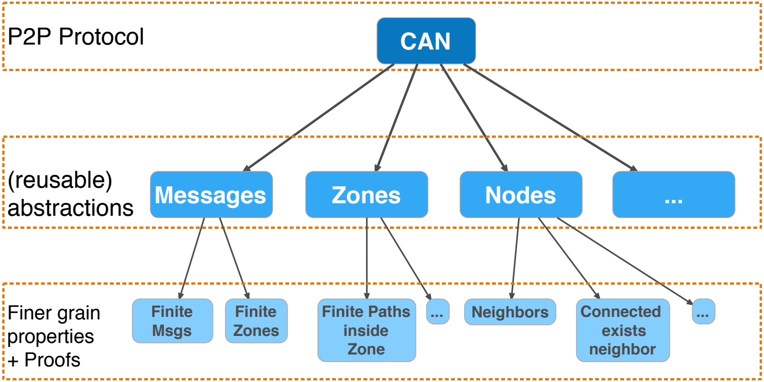
```
typedef CAN =
"{(nodes::nat set, Z :: nat => Zone,
neighbours:: (nat × nat) set) .

finite nodes ∧
finite neighbours ∧
(∀ x y. (x,y)∈ neighbours ⟶(y,x) ∈ neighbours) ∧
(∀ x. (x,x)∉ neighbours) ∧
(∀tup. ∃n∈nodes. tup ∈ (Z n)) ∧
(∀ N∈nodes. ∀ N'∈nodes. N≠N' ⟶¬ intersects (Z N) (Z N')) ∧
(∀ N∈nodes. (Z N)≠{})}"
```

* Basically we define the *needed* abstractions of the protocol

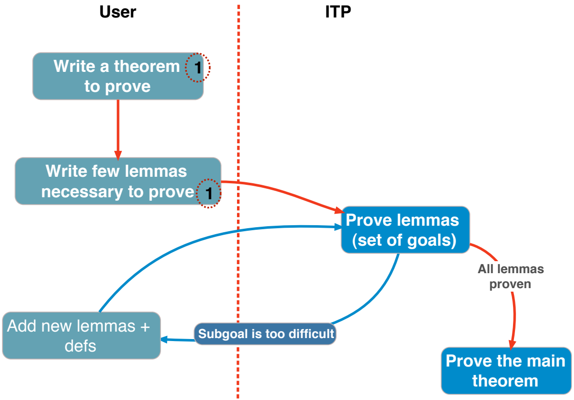
Contributions

the formalization process

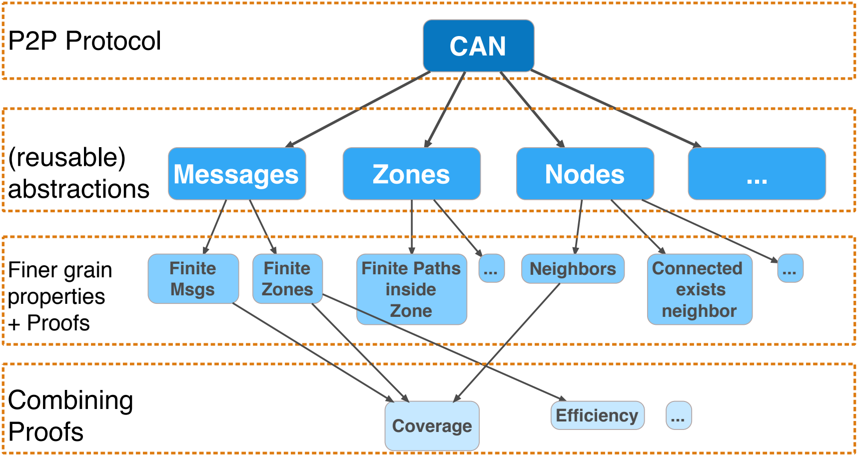


Contributions

On a day-to-day basis



Contributions
formalization process pictured



Contributions

Summary

- What we have done so far:

Contributions
Summary

- What we have done so far:
  - A formalization of an abstraction of CAN overlay network + theorems and correctness proofs.

Contributions

Summary

- What we have done so far:
  - A formalization of an abstraction of CAN overlay network + theorems and correctness proofs.
  - A formalization of abstract geometric notions related to CAN, neighboring and communication aspects + correctness proofs

Contributions
Summary

- What we have done so far:
  - A formalization of an abstraction of CAN overlay network + theorems and correctness proofs.
  - A formalization of abstract geometric notions related to CAN, neighboring and communication aspects + correctness proofs
  - An example explaining how to define formally a broadcast algorithm for a static CAN.

Current spec + proofs : around 2000 lines of Isabelle code

Goals...

- Initial goal:
  to develop the algorithm correctly and prove its correctness properties.

Goals...

- Initial goal:
  to develop the algorithm correctly and prove its correctness properties.
- Additional goal:
  to build a *generic reasoning framework* which will ease the promotion of formal correctness proofs of existing multicast algorithms and also facilitate the design of new ones (which are efficient and fault-tolerant,. . .).

Future work

- Implementation
- Consider a dynamic CAN (*churn*)
- Test different (possibly existing) dissemination schemes
    - multiple initiators, ...
- Fault-tolerant broadcast
- Structured proofs

Take away message

" Programs are not released without being tested, why should algorithms be published without being ~~model checked~~ * ? "

- Leslie Lamport

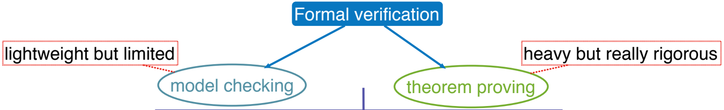
* *proved correct*

Questions...?

Thank you

Backup slides

Model checking VS theorem proving



	model checking	theorem proving
State space	Finite	Infinite
Counter-example	Automatic	Limited automatic
Verification procedure	Automatic	Not automatic
Obtaining insight of the system	Tell how the system is incorrect	Tell how the system is correct