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 May 2011
Multicast Services over Structured P2P Networks

Fig. 1 An example of the efficient flooding algorithm in a CAN-multicast system. The node responsible for the shadowed zone is the multicast data source.

Fig. 2 An example of the enhanced efficient flooding algorithm in a CAN-multicast system. Note that now there are no double packets in any multicast: Members of a multicast group self-organize in an instance of Chord or DKS(N,k,f). Consequently, a multicast transmission to a certain group just requires to broadcast the Chord or DKS(N,k,f) instance associated to that group. However, this proposal differs from the CAN based multicast in the flooding mechanism. As it is explained in Section 2.1, the efficient flooding algorithm (and also its enhanced version) produces redundant messages. The Chord/DKS flooding algorithm that is called "correcting broadcast" eliminates any redundant packets.

correct-by-construction efficient broadcast

conditions apply

QED
**IP multicast**

Broadcasting a message $M$ in a network

Network-level

IP multicast

Pros
- bandwidth efficiency
- no redundant packets
IP multicast Issues

Broadcasting a message M in a network

Network-level

Pros
- bandwidth efficiency
- no redundant packets

Issues
- Scalability
- Best-effort
- Deployment
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?

→ Using an interactive proof assistant
Can we build such delivery mechanism **correctly** and **formally** prove its properties? *****

→ Using an interactive proof assistant
Can we build such delivery mechanism correctly and formally prove its properties? ***

→ 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

- *Federated Semantic Space*: Hierarchical P2P system for large scale RDF data processing and storage (based on Chord and a modified version of CAN).

* RDF triple = "\{subject, predicate, object\}"
**Federated Semantic Space**: Hierarchical P2P system for large scale RDF data processing and storage (based on Chord and a modified version of CAN).

* RDF triple = \{subject, predicate, object\} *

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:
  - Design
  - Verify
  - Program
  - Simulate
  - Experiment
  - Analyze
Dissemination algorithms on top of large-scale P2P systems are hard to:

- Design
- Verify
- Program
- Simulate
- Experiment
- Analyze

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
  ↩️ [Chord, CAN, Pastry,...]
Mechanizing formal proofs
What’s in it for you?

- Papers with “just” a description of the algorithm
  \[ \uparrow \quad [\text{Chord, CAN, Pastry,}...] \]
- 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
  - [Chord, CAN, Pastry, ...]
- Papers with a more precise description of the algorithm and rough hand proofs of correctness
  - 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.
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.”

- Francesco Bongiovanni -

Mechanical Support for Efficient Dissemination on CAN (9/26)
Contributions

- **Correct** construction of an efficient broadcast algorithm using \textbf{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 Addressable Network
Content Addressable Network

- $d$ – dimensional Cartesian coordinate space
Content Addressable Network

- $d$ – dimensional Cartesian coordinate space
- each peer manages a portion of the space
Content Addressable Network

- $d$ – dimensional Cartesian coordinate space
- each peer manages a portion of the space
- a peer only knows its adjacent neighbors
Background

Routing in CAN

- Francesco Bongiovanni -
Mechanical Support for Efficient Dissemination on CAN (12/26)
Routing in CAN

State overhead: \( O(d) \)
Lookup complexity: \( O(dN^{\frac{1}{d}}) \)
Contributions

Main idea

Informal description of CAN

Sketch of the algorithm

A glimpse of the formalization process

Summary
Contributions

Main idea

From

formally

To
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 - 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

Informal description of CAN
Sketch of the algorithm
A glimpse of the formalization process
Summary
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

P2P Protocol

CAN

(reusable) abstractions

Messages Zones Nodes ...
Contributions

Definitions

Definitional approach
Contributions

Definitions

Definitional approach

- definition of a Node, Space

```plaintext
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**

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

**Abbreviation**

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

Definitions

Definitional approach

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

```plaintext
typedef 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
- ...

```cpp
typedef CAN = "{(nodes::nat set, Z :: nat => Zone, 
neighbours:: (nat x nat) set) .
finite nodes ^
finite neighbours ^
(\forall x, y. (x,y) \in neighbours \rightarrow (y,x) \in neighbours) ^
(\forall x. (x, x) \notin neighbours) ^
(\forall tup. \exists n \in nodes. tup \in (Z n)) ^
(\forall n \in nodes. \forall n' \in nodes. n \neq n' \rightarrow \neg intersects (Z n) (Z n')) ^
(\forall n \in nodes. (Z n) \neq \{\})""
```
Contributions
Definitions

**Definitional approach**

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

```plaintext
typedef CAN = 
  '{(nodes::nat set, Z :: nat => Zone, 
    neighbours:: (nat x nat) set) .

  finite nodes ^
  finite neighbours ^

  (\forall x, y. (x,y)\in neighbours \rightarrow (y,x) \in neighbours) ^

  (\forall x. (x,x)\not\in neighbours) ^

  (\forall tup. \exists n\in nodes. tup \in (Z n)) ^

  (\forall N\in nodes. \forall N'\in nodes. N\neq N' \rightarrow \neg intersects (Z N) (Z N')) ^

  (\forall N\in nodes. (Z N)\neq\{\})}'
```
Contributions

Definitions

**Definitional** approach

- definition of a *Node*, *Space*
- definition of a *Message*
- definition of a *CAN*
- ...
Contributions
the formalization process

P2P Protocol

CAN

(reusable) abstractions

Messages Zones Nodes ...

Finer grain properties + Proofs

Finite Msgs Finite Zones Finite Paths inside Zone ... Neighbors Connected exists neighbor ...
Contributions
On a day-to-day basis

User
- Write a theorem to prove
- Write few lemmas necessary to prove
- Add new lemmas + defs
- Subgoal is too difficult

ITP
- Prove lemmas (set of goals)
- All lemmas proven
- Prove the main theorem
Contributions

formalization process pictured

P2P Protocol

(reusable) abstractions

Finer grain properties + Proofs

Combining Proofs

CAN

Messages

Zones

Nodes

... 

Finite Msgs

Finite Zones

Finite Paths inside Zone

Neighbors

Connected exists neighbor

Coverage

Efficiency

Francesco Bongiovanni

Mechanical Support for Efficient Dissemination on CAN (20/26)
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
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.
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.
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
“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

<table>
<thead>
<tr>
<th></th>
<th>Model Checking</th>
<th>Theorem Proving</th>
</tr>
</thead>
<tbody>
<tr>
<td>State space</td>
<td>Finite</td>
<td>Infinite</td>
</tr>
<tr>
<td>Counter-example</td>
<td>Automatic</td>
<td>Limited automatic</td>
</tr>
<tr>
<td>Verification procedure</td>
<td>Automatic</td>
<td>Not automatic</td>
</tr>
<tr>
<td>Obtaining insight of the system</td>
<td>Tell how the system is incorrect</td>
<td>Tell how the system is correct</td>
</tr>
</tbody>
</table>

- Francesco Bongiovanni -

Mechanical Support for Efficient Dissemination on CAN (26/26)