RenPar'20 / SympA'14 / **CFSE 8**
Saint-Malo, France, du 10 au 13 mai 2011

Mechanical Support for Efficient Dissemination on the CAN Overlay Network

Francesco Bongiovanni, Ludovic Henrio

INRIA Sophia Antipolis Méditérranée – CNRS – I3S – Université de Nice Sophia-Antipolis,
2004 Route des Lucioles - BP 93,
06902 Sophia Antipolis, France
firstname.lastname@inria.fr

Abstract. The various algorithms underlying P2P systems are notoriously difficult to design and analyze. Coming up with new *proven* algorithms for such large scale systems is a challenging task. We report on the initial steps of an ongoing work that aims to devise an efficient correct-by-construction broadcast algorithm for the CAN structured overlay network. To rigorously reason about such an algorithm and prove correctness we rely on an interactive theorem prover : Isabelle/HOL. This paper presents a generic reasoning framework which should ease the promotion of formal correctness proofs of existing multicast algorithms and also facilitate the design of new ones.

Keywords : Peer-to-Peer (P2P), CAN, broadcast algorithm, theorem proving, Isabelle/HOL

1. Introduction

Peer-to-Peer (P2P) systems have been recognized as a key communication model to build scalable platforms for distributed applications such as file sharing, distributed storage, etc. P2P systems are broadly classified into *unstructured* and *structured* overlays based on the topology [4]. In the context of this work, we are interested in *Structured Overlays Networks* (SONs) that emerged to alleviate inherent problems of unstructured P2P architectures. In these systems, peers are organized in a well-defined topology (ring, torus, cube, etc.) (e.g., CAN [17], Pastry [20], Chord [21]), where resources (e.g., data, file, etc.) are uniformly stored in a deterministic location using consistent hashing. SONs typically offer a *Distributed Hash Table* (DHT) abstraction for data storage and retrieval which supports efficient *key-based* lookups. The main advantage of SONs is their deterministic behavior in terms of search complexity which is guaranteed, with a high probability, to be logarithmic with respect to the number of nodes and that the data is uniformly distributed among nodes thanks to the use of consistent hashing functions. The nature of some large-scale applications, such as content delivery systems or publish/subscribe systems, built on top of SONs, demands application-level dissemination primitives which do not overwhelm the overlay, i.e. efficient, and which are also reliable. Building such communication primitives in a reliable manner on top of such networks would increase the confidence regarding their behavior prior to deploying them in real settings. In order to come up with real efficient primitives, we take advantage of the underlying geometric topology of the overlay network and we also model the way peers communicate with one another. In this paper, we are interested in the *correct* design of an efficient communication primitive on top of the CAN overlay network. We thus present a reasoning framework that will allow us in the future to define dissemination primitives and *formally* prove their properties.

Contributions

This paper contributes to the correctness of distributed execution environment. We choose to focus on theorem proving techniques to be able to prove generic properties of distributed frameworks and middleware and address large-scale systems. More precisely, our aim is to design an *efficient* (in terms of messages) and *correct* broadcast algorithm for the CAN overlay network. In this paper, we present Isabelle/HOL definitions and theorems to reason on such algorithms, to prove the correctness of existing group communication algorithms, but also to be able, through the abstractions and proofs we propose, to facilitate the design and proofs of new ones. The typical properties we are interested in are : efficiency, reliability, and coverage.

We expect our framework to be general enough to study CAN networks in general, by providing the formalization for the basic blocks composing this specific structured overlay network. We are not interested in formalizing the whole CAN protocol but rather focus on the minimal set of abstractions needed to devise efficient correct-by-construction group communication algorithms on top of such overlay. Therefore our contributions in this paper are the following :

– A formalization of an abstraction of the CAN overlay network with related theorems and correctness proofs.
– A formalization of the interplay between the geometric notions of the CAN and the neighboring and communication aspects ; more precisely correctness proofs about *messages*, *message paths*,. . . used by the algorithm on top of CAN.
– An example explaining how to define formally a broadcast algorithm for a static CAN.

Why Isabelle/HOL ?

In general, formal methods improve the reliability of proposed algorithm and the confidence one has in their properties. In our case we want to see what conditions are necessary to ensure the correctness and other properties of a broadcast algorithm over a CAN. Mechanical proofs will ensure the correctness of the studied protocols, with a much higher confidence than paper proofs which rely too often on "well known" properties or "obvious" steps that could reveal wrong or underspecified. A theorem prover enforces the precise and sound formalization of the studied protocols, and of the hypotheses ensuring their correction and properties.

Proving properties on distributed algorithms could be done by specific formalisms for distributed systems, like TLA$^+$ [11], however we chose a more general theorem prover to have better support for general reasoning. Indeed, reasoning on the geometry of a CAN requires generic theorems that will be better supported by a general purpose theorem prover like Isabelle or Coq for example. Additionally, all formal methods relying on model-checking work by instantiation on a finite set of states, meaning one can only verify protocols on a small number of processes. Theorem proving on the contrary requires the help of the programmer to prove properties that are valid on an arbitrary number of processes. Consequently, the proof performed in Isabelle/HOL are particularly adequate to study large-scale distributed systems. Among theorem provers, the exact choice of Isabelle/HOL is not crucial here, our framework could be easily written in Coq for example ; however Isabelle/HOL is an interactive theorem prover quite user-friendly.

The rest of the article is organized as follows. Next section presents the CAN overlay network, and motivates the importance of application-level dissemination algorithms with an emphasis on CAN and we briefly present the motivation behind our approach for proving the correctness of those algorithms. Section 3 presents our contribution, focusing on the design choices we made and giving a first feedback on the use of theorem provers for proving distributed algorithms. Section 4 reviews the most relevant related works.

2. Background and motivation

The Content Addressable Network (CAN) [17] is a structured P2P network based on a d-dimensional Cartesian coordinate space labeled $\mathcal{D}$. This space is dynamically partitioned among all peers in the system such that each node is responsible for storing data in a zone in $\mathcal{D}$; stored data consist in *(key, value)* pairs. To store the (k, v) pair (the *insert* operation in Figure 1), the key k is deterministically mapped onto a point i in $\mathcal{D}$ and then the value v is stored by the node responsible for the zone comprising i. The lookup (the *retrieve* operation in Figure 1) for the value corresponding to a key k is achieved by applying the same deterministic function on k to map it onto i. The query processing is an iterative routing process which starts at the query originator and which traverses its adjacent neighbors (a peer only knows those), then the neighbors' adjacent neighbors so on and so forth until it reaches the zone containing the value as depicted by the *retrieve* operation in Figure 1.

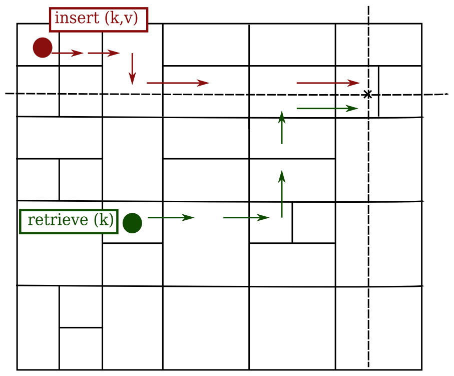
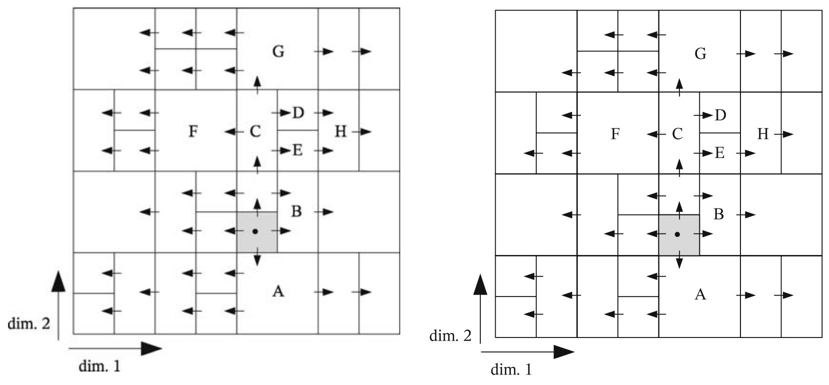


FIGURE 1 – Routing in CAN - example of data storage (*insert(key ,value)*) and retrieval (*retrieve(key)*).

CAN is a practical infrastructure for file sharing, data storage and so on. It can be also very effective when it comes to large scale information dissemination. As a matter of fact, network-layer multicast is still not widely adopted by most commercial ISPs [6] and this prevents the usage of practical native *one-to-many* communication primitives by today's large scale applications. This technical impediment, mainly due to costs issues and bandwidth preservation policies, was overcome with the introduction of *application-level multicast* protocols such as [5]. Scribe [3], Bayeux [24], and more specifically the CAN-based multicast [16] are examples of such application level multicast solutions. Such systems, which are based on SONs, directly leverage the underlying geometrical infrastructure and offer practical group communication abstractions to higher level applications which need to *efficiently* disseminate information to multiple nodes in the network.
The authors of CAN, in [16], give hints of a flooding mechanism which is efficient for a perfectly partitioned coordinate space. However their method does not fully eliminate all duplicates if the space is not perfectly partitioned as depicted in 2(a) (zone H receives twice the information). Figure 2(b) shows on the contrary an optimal dissemination pattern, unfortunately we found no algorithm implementing such an optimal dissemination pattern. The DKS system [8] introduced generic algorithms for building ring-based SONs and also provides generic multicast algorithms. A correlated contribution by the same authors is a generic and efficient broadcast

algorithm on top of ring-based SONs which eliminates any redundant messages [7]. Finally, authors of Meghdoot [9], a content-based publish/subscribe system based on CAN, provide a way to avoid duplicates in their event propagation algorithm which seems worth checking out *formally*.
The questions that we ask ourselves were the following : "can we devise a broadcast algorithm which is efficient (i.e. without any redundant messages) for CAN, as depicted in Figure 2(b) " and "can we do it and prove its correctness while trying to construct it ?".



(a) Efficient flooding (b) Enhanced efficient flooding

FIGURE 2 – Efficient flooding in CAN (taken from [13])

Using an interactive theorem prover such as Isabelle/HOL [14] and its high-order logic provides us the expressiveness needed for formalizing such a distributed algorithm and answering the above questions. A high order logic naturally supports the formalization of the data structures and the properties a distributed algorithm possesses and provides the reasoning modes to prove them. The expressiveness of Isabelle's logic allows us to reason on an abstraction of the system we design, meaning that we can abstract away some properties and precise details of the CAN overlay and focus on the aspects ensuring the correctness of the dissemination algorithm properties. The benefits of using such an environment is that it gives us the confidence in the correctness of the proofs we construct. There is a strong need for enriching the existing Isabelle libraries with specific reasoning building blocks for distributed systems. CAN is a popular DHT which is used as a distributed substrate for large scale applications. Thus, our motivation is to put forward *proven* abstractions for proving correctness properties of distributed algorithms on top of CAN in order to contribute to the advancement of *correct* distributed algorithms.

3. A Mechanized Model for CAN and Broadcast Algorithms

We describe in this section our formalization of CAN, messages exchanged between CAN nodes, and definition of a generic broadcast algorithm for CAN. We provide the Isabelle/HOL definition of most of the crucial notions, relying on more informal definition for the parts that are not directly related to our purpose. We also provide a few characteristic lemmas. Our objective is to give a precise idea of our approach, but we refer the reader to the source code available

on our Web page [1] for the exhaustive formalization.
We do not present here the Isabelle/HOL syntax in details, the reader can refer to Isabelle/HOL tutorial for a precise description. Note however that most of the syntax is much similar to mathematical notations ; the main differences are the following : $\longrightarrow$ is the implication, :: defines the type of an expression, and $A \Rightarrow B$ is the type of functions from A to B. For manipulating structures Isabelle/HOL provides the following notations : ! accesses an element of a list, # is the list constructor it appends an element at the beginning of the list, and @ appends two lists.

3.1. CAN

A crucial question when formalizing a complex structure like a CAN is which level of abstraction should be used, and which notions of Isabelle/HOL should represent basic notions of CAN networks. We represent a CAN by a set of nodes, a zone for each node, and a neighboring relationship, stating whether one node is neighbor of another. More precisely, a *CAN* is a set of integers identifying the different nodes. A function Z matches each node to a *Zone*, where a zone is a *Tuple set* (a tuple is an array of integers). Note that we abstract away a few constraints of the CAN model which are not useful for us, like zones are rectangular, and we do not relate zones with the neighboring notion as it does not reveal useful for the moment. In Isabelle, a CAN is defined as follows :

```
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′))}
```

Additional constraints state that the set of nodes is finite, the zones cover the whole space, and are disjunct. We define three auxiliary functions *CAN_Nodes*, *CAN_Zones*, and *CAN_neighbours* returning each part of a *CAN*.
We also define a function *intersects* Z Z' that checks whether zone Z intersects zone Z' : it is true if Z and Z' have at least one point (tuple) in common. In the following, we say that "a node intersects a zone Z", if the zone of the node (((*CAN_Zones* C) *node*)) intersects Z. Then we state that a zone is connected [2] if the nodes it intersects are all connected to one another (there is a path of neighbors between any two nodes intersecting the zone). The Isabelle definition of *Connected* is the following, it is a function that takes a *CAN* and a *Zone* and returns a *bool* :

```
definition Connected:: CAN⇒Zone ⇒bool
where Connected C Z ≡ (∀ n∈ CAN_Nodes C. ∀ n′∈ CAN_Nodes C.
   ((intersects (CAN_Zones C n) Z∧intersects (CAN_Zones C n′) Z) ⟶
     (∃ node_list. (node_list!0 = n∧destination_NL node_list = n′ ∧
         distinct node_list ∧
      (∀ i<(length node_list−1). ((node_list!i)∈CAN_Nodes C) ∧ CAN_neighbour C (node_list!i)
(node_list!(i+1)) ∧ intersects (CAN_Zones C (node_list!i)) Z)))))
```

It states that if n and n' are two nodes which zones intersect Z, then there is a list of nodes (all distinct) starting at n, ending at n', only passing by nodes intersecting Z, and for which each node of the list is neighbor of the previous one.

1. The code can be found here : `http://www-sop.inria.fr/oasis/personnel/Ludovic.Henrio/misc`. We use the *Isabelle2009-2* version.
2. This notion is closed to the geometrical notion of connectivity, or rather path connectivity.

We also proved a few generic lemmas that will be useful for proving properties entailing CAN structure. We detail below two such lemmas illustrating the properties of a CAN that will reveal useful to reason about such a structure.
Let us first mention an induction principle that will allow us to prove a property related to a zone by induction on the size of the zone, or more precisely by induction on the number of nodes intersecting the zone. In Isabelle, $\bigwedge$ stands for "for all" (at the meta-level), and a theorem is expressed by a term of the form ⟦*premise 1 ; premise 2* ⟧ $\Longrightarrow$ *Conclusion*. The induction theorem is thus written as follows :

```
theorem induct_node_zone:    ⟦ P {};
            ⋀n Z. ⟦⋀ Z'. card {node ∈ CAN_Nodes C. intersects Z' (CAN_Zones C node)}=n ⟹ P Z';
                    card {node ∈ CAN_Nodes C. intersects Z (CAN_Zones C node)}=Suc n⟧ ⟹ P Z  ⟧
            ⟹ P Z
```

It states that, if (1) we prove a property P is true for an empty zone, and (2) we prove that if P is true for all zones of size n then it is true for all zones of size $n + 1$; then the property is true for all zones.
Let us additionally mention another useful lemma. It allows us to initiate a path inside a connected zone : if the zone intersects more than one zone, then one can find two nodes, neighbor one of the other, inside the zone :

```
lemma Connected_exists_neighbour:
⟦Connected C Z; card {N∈CAN_Nodes C. intersects (CAN_Zones C N) Z}>1⟧
⟹∃ N ∈ CAN_Nodes C. ∃ N' ∈ CAN_Nodes C.
    intersects (CAN_Zones C N) Z ∧intersects (CAN_Zones C N') Z ∧CAN_neighbour C N N'
```

3.2. Messages and Message Paths

When the structure of the network is defined, we can provide a definition for messages and for the path followed by a message.
A message is made of four pieces of information : an identifier for the message (which could identify uniquely its content for example), a source node, a destination node, and the zone to which it must be transmitted. The Isabelle code defining such a quadruple is very simple :

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

We decided to rely on the notion of zone to be covered to define a broadcast algorithm, because it seems quite adapted to a CAN, and algorithms presented in Section 2 fit easily with such a representation. Also as we are looking for an efficient algorithm, it seems quite reasonable to try to split efficiently the zone to be covered in order to avoid sending a message to the same node twice.
Message_zone, *Message_dest*, and *Message_source* are functions accessing the three first fields. We also define an abbreviation <*m*|*x*,*y*,*Z*> for defining a *Message*, this allow us to easily identify messages inside the definitions and lemmas.
A valid path relatively to a message set is a list of messages (*msgs*), each starting from the arrival node of the previous node. To be able to reason on the longest path, for example inside a zone, we forbid loops inside message paths.

```
definition valid_path:: Message set ⇒Message list ⇒ bool
where valid_path msgs ML ≡  ML≠[] ∧(∀ i<length ML. ML!i∈msgs) ∧
                (∀ i<length ML − 1. Message_dest (ML!i) = Message_source (ML!(Suc i))) ∧
                distinct ML
```

More precisely, the above definition states that, given a list of messages *msgs*, a message list *ML* is a valid path if it is non-empty, and all messages of *ML* belong to *msgs*, and the destination of the message number *i* is the source of the message number $i + 1$; finally all the elements of the list must be distinct, which ensures that no two elements of the *ML* list are equal and thus forbids loops of messages inside *ML*.
The predicate *path_inside_zone* takes a CAN, a set of messages *msgs*, and a zone *Z*, and returns the set of valid message paths formed of messages of *msgs* that are entirely inside the zone *Z*. For this we check that the origin node of the path intersects the zone, and that the destination node of each message of the path intersects *Z*.

```
definition path_inside_zone:: CAN ⇒ Message set ⇒ Zone ⇒ Message list set
where
path_inside_zone C msgs Z ≡
 {MsgL. valid_path msgs MsgL ∧ (intersects (CAN_Zones C (source MsgL)) Z) ∧
     (∀ i<length (MsgL). (intersects (CAN_Zones C (Message_dest (MsgL!i))) Z))}
```

The following lemma can be applied automatically to simplify the *valid_path* predicate by decomposing it on the head and the tail of the list : a path is valid if it appends a new message to a valid path, and the message arrives at the origin node of the first message of *MsgL* (*Message_dest M* = *source MsgL*)).

```
lemma valid_path_cons:
(valid_path msgs (M#MsgL)) = ((MsgL=[]∧M∈msgs)∨(valid_path msgs MsgL
      ∧ M∉set MsgL ∧ (MsgL≠[]⟶Message_dest M = source MsgL) ∧ M∈msgs))
```

path_inside_zone can be proved to be finite provided the set of messages is finite :

```
lemma finite_path_inside_zone: finite msgs ⟹finite (path_inside_zone C msgs Z)
```

3.3. Broadcast Specification

We can now define a broadcast mechanism for the CAN overlay network. It is far from trivial to define an algorithm in a convincing way in Isabelle/HOL. Indeed, the basic language of Isabelle is a pure functional language similar to λ-calculus, which is not the language in which we would usually encounter broadcasting algorithms. Here we want to focus on the way a message triggers other ones, for this we concentrate our specification on the notion of consequences, and on a specification of the *set of messages* used to broadcast an original message.
In our framework *Broadcast* is a triple made of a *CAN*, a message set and initiator node constrained by several well-formedness rules as defined below :

```
typedef Broadcast = {(can,msgs,initiator).
(∀ x y m Z m′ Z′. (<m|x,y,Z>∈msgs ∧ <m′|x,y,Z′>∈msgs) ⟶ (m=m′∧ Z=Z′)) ∧
(initiator ∈ CAN_Nodes can)∧
 (∀ m s d Z. <m|s,d,Z>∈msgs ⟶
   (s∈ CAN_Nodes can ∧ d∈ CAN_Nodes can ∧ CAN_neighbour can s d∧
   (∃ MsgL. valid_path msgs MsgL ∧ destination MsgL = s ∧ source MsgL=initiator)∧
   (∀ m′ d′ Z′. <m′|d,d′,Z′>∈msgs ⟶ (intersects (CAN_Zones can d′) Z ∧ Z′ ⊆ Z  ))  )
)}
```

The constraints expressed in the above definition state that :
– There is a single message between any 2 nodes
– The initiator is a node of the CAN
– All messages are exchanged between neighbor nodes of the CAN

– All messages must originate from a node that has been reached by a list of messages originating from the origin node : *valid_path msgs MsgL* $\wedge$ *destination MsgL* = *s* $\wedge$ *source MsgL*=*initiator*. Requiring the existence of such a valid path ensures that a broadcast only relies on messages transmitted from a node to its neighbor (except for the origin of course). Note that it is not sufficient to require that each message source is the destination of another message, because that would mean that loops of messages not passing by the origin would be allowed.
– Each node d sends only messages ($<m'|d,d',Z'>$) to nodes it has to cover, i.e. node d' must intersect the zone Z that d received in its message. We say that the message $<m'|d,d',Z'>$ sent by d is a consequence of the first one ($<m|s,d,Z>$).
– Finally, the zone of a message must always be bigger than the one of its consequences : a node can only delegate the coverage of a subset of the zone it is responsible for.

We note $<C,M,n>$ such a *Broadcast*, and define functions *BC_CAN*, *BC_msgs*, and *BC_initiator* to access its fields. We can then define a predicate checking whether a broadcast covers the whole CAN (each node of the CAN is either the initiator or the destination of a message) :

definition *Coverage*:: *Broadcast*$\Rightarrow$*bool*
where *Coverage BC* $\equiv$
$\forall n \in (CAN\_Nodes\ (BC\_CAN\ BC)).(n{=}BC\_initiator\ BC \vee (\exists\ m\ s\ Z.\ <m|s,n,Z> \in BC\_msgs\ BC))$

From those definitions, we expect to prove completeness of some specific broadcast algorithm, but also study their optimality. A first lemma we could prove is the following, it states that there is a longest path among all the paths inside a zone. It will be useful in the next proofs of more advanced properties because it will allow reasoning by recurrence on the length of this longest path.

lemma *longest_path_BC*:
$\exists\ M \in (BC\_msgs\ BC).\ (intersects\ (CAN\_Zones\ C\ (Message\_dest\ M))\ Z \wedge$
$intersects\ (CAN\_Zones\ C\ (Message\_source\ M))\ Z)$
$\Longrightarrow \exists\ M \in (path\_inside\_zone\ C\ (BC\_msgs\ BC)\ Z).$
$\forall\ M' \in (path\_inside\_zone\ C\ (BC\_msgs\ BC)\ Z).\ length\ M' \leq length\ M$

Finally, we want to illustrate the way we intend to specify broadcast algorithms. The following definitions specifies the set of messages of a broadcast algorithm based on a notion of zone to be covered, it relies on a call to a function *Set_of_Valid_ZNL Params* that returns a list of pairs zone, node to which the message is to be forwarded (details are omitted here, but an example is provided below). The inductive definition of the broadcast is of the form *BC_msgs C Mid init msgs ML*, *C* is the CAN network, *Mid* is the message identifier, and *init* is the initiator node. The inductive definitions put inside the set of messages *msgs* the messages of the broadcast ; it takes a message *M* in the message list *ML* (list of messages to be treated) and computes its consequences before putting *M* inside *msgs*. The first rule below launches the algorithm for the initiator : it computes the messages the initiator has to send by producing a message for each member of the *ZNL* list for the initiator. The list of produced messages is pushed in the list of messages to be treated. The second rules takes a message *M* in the list of messages to be treated, and does a similar operation, producing a list of messages to be treated, and putting *M* in the set of treated messages. At the end, the list of messages to be treated is empty, and the fourth argument contains the list of messages of the broadcast.

Those rules illustrate how we plan to define the message propagation ; even for a broadcast algorithm that would not rely on coverage zones, the inductive structure of the message set definition would be similar.

```
inductive BC_msgs:: CAN ⇒nat ⇒nat⇒Message set⇒Message list ⇒ bool
for C:: CAN and Mid::nat and init:: nat
where
BC_init: ⟦ ZNL∈ Set_of_Valid_ZNL init;
    ML′=map (λ ZN. let Z′=fst ZN in let N′=snd ZN in <Mid::nat | d,N′,Z′− CAN_Zones C N′
>) ZNL⟧
          ⟹ BC_msgs C Mid init {} ML′
|
BC_step: ⟦ BC_msgs C Mid init msgs (M#ML); M=<Mid′ | s,d,Z>; ZNL∈ Set_of_Valid_ZNL s;
    ML′=map (λ ZN. let Z′=fst ZN in let N′=snd ZN in <Mid′::nat | d,N′,Z′− CAN_Zones C N′
>) ZNL⟧
          ⟹ BC_msgs C Mid init (insert M msgs) (ML@ML′)
```

The idea behind the ZNL definition is that a broadcast protocol, to limit the number of duplicates or even remove them, will have to split the geographical space to be covered into (disjoint) zones : each node receives a message together with a zone to be covered ; splits the zone into sub-zones, and finally forwards the message to some of its neighbours with an associated sub-zone. Each selected neighbour should of course intersect the zone to be covered originally.
An optimal but simple algorithm can be provided by the following way to allocate couples (node, zone). Suppose a node receives a message with a given zone Z, where (1) Z is a union of rectangles (thus Z is the finite union of several connected zone, Z_i), and (2) each of the connected zone Z_i intersects one of the neighbour, called N_i of the current node. Then, we forward the message to each of the neighbour N_i, delegating to it the zone Z_i minus the zone of N_i which we just covered ; note that this new zone verifies both (1) and (2), and thus this can be applied recursively until each node receives the message, only once.
It is easy to define formally couples (node, zone) from the above definition. One of our next steps is to encode this definition and prove some properties on this simple algorithm.

3.4. Overview of the Mechanization Process

Overall the current specification and proofs consist of 1800 lines of Isabelle code. As usual, most of the code is dedicated to proofs, but since we are in the early stages of the formalization, and as our framework requires a lot of different notions, the definitions amount for more than 10% of this code. Difficult parts of the reasoning concern the finite sets, and the difficulty to reason by recurrence on a set that is finite but not inductively defined, as illustrated by the induction principle shown in Section 3.1.
The CAN overlay network is a difficult setting for proofs, because the structure entails some (simple) geometrical reasoning, which is more complex than reasoning on structures that could be easily defined by induction. Indeed, Isabelle/HOL support for inductive reasoning is more valuable than for other kind of reasoning ; but this "only" makes the proofs more difficult to perform, and longer.
A crucial part of our approach relies on the fact that the definition and properties are expressed in a formalism that is convincing : it must be easy for an external reader familiar with basic logics and mathematics to understand our formalism, to be convinced by our formulation of a CAN network, and of its properties. Note that we do not plan to extract code, and thus an efficient formalization is not a crucial prerequisite.
It is important for us to have a formalism for expressing the CAN broadcast that is easy to understand ; that is why we presented the sketch of the specification of a broadcast. Although the specification is inductive and thus not in a classical form for a broadcast algorithm, we think it is clear enough to be convincing, and that it is easy to extract an algorithm from it. This way of expressing a broadcast algorithm is not as natural as one would expect because a form of

event-based formulation of the algorithm "when a message M is received, send messages M1, M2, and M3" would be more adapted. However, such an event-like formulation is not well supported in Isabelle/HOL. In the future, we will try to provide abbreviations in Isabelle to allow a formulation closer to the reaction to message reception events.

This formalization provides a set of theorems allowing one to prove the properties of communications algorithms, over CAN-like networks. The outcome of this work will thus be a set of properties for several broadcast algorithms, starting by coverage, (i.e. each node receives the message). As we will define precisely the hypotheses on the network topology and on the algorithm, we will know exactly to which kind of networks those algorithms are applicable.

4. Related Work

Reasoning about concurrent and distributed algorithms at the right level of abstraction has been an active area of research for many years. Common patterns of reasoning in program verification can be found in well known software verification approaches such as temporal logic [15] or in the use of global invariants [1].

A fair amount of work has been done on the verification of distributed systems using theorem provers (e.g., Coq, HOL, Lego, etc.) and other tools. These two works, [10, 22], share the same objective, that is, to practically reason on distributed algorithms and verify their correctness properties. In [22], the author uses HOL to formalize and verify a PIF (propagation of information with feedback) algorithm. With this example, he showed how to build a reasoning infrastructure for distributed algorithms in HOL. In the same line of direction, the work done by Qiao Haiyan [10] reports experiences in verifying distributed algorithms in constructive type theory using the Agda/Alfa proof assistant and provides a methodology which bridges the testing and verification of such algorithms.

Ridge goes further and takes an *operational approach* to distributed systems verification [19]. His main goal is to demonstrate, through a combination of symbolic evaluation and invariant checking using the HOL4 theorem prover, that the verification of distributed system, down to the executable code level, is actually feasible. This work relies on tremendous efforts to reduce the gap between abstract mathematical models of distributed applications and their implementations. The work consists in a rigorous approach to describe network protocols [18], a formal model of the OCaml programming language, and an operational verification of OCaml code.

The verification of DHT protocols is still an active area of research. The most popular of them, Chord [21], was not rigorously and formally proven correct by its designers. Work from [2] used π-calculus to prove some properties of the *pure-join* model of the protocol [21]. Zave, in her work [23], underlines that Chord was never formally proven correct. Using the Alloy analyzer, she proved the protocol in its two models : the *pure-join* and *full*. She provides a rigorous correctness proof of the pure-join model and and proved that the full model of the protocol is indeed not correct using lightweight verification methods. Pastry was also the subject of a verification effort [12] aiming at verifying the correctness and consistency of the protocol which was specified in TLA$^+$ and model checked using TLC. As stated earlier, authors of [9], in the context of a content-based publish/subscribe system, provide a propagation algorithm on top of CAN which prevents repetitive event propagation. However, no formalization nor any form of reasoning regarding correctness properties of such algorithm is presented.

To the best of our knowledge, we are the first ones to formalize and prove some properties of an abstraction of the CAN overlay network using a theorem prover. Such a formalization requires a considerable amount of efforts, and a strong experience in formalization and models for distributed systems. This kind of efforts should greatly increase the correctness and understanding of distributed algorithms, and the confidence the user has in this correctness.

5. Conclusion and Future Works

We presented a framework for reasoning on communications on top of the CAN overlay network, more specifically, we defined formally all the constructs necessary to specify and prove properties of a broadcast algorithm on top of CAN. The formalization is still in an early stage in the sense that all the major constructs have been defined but the properties we have proved are still only the basic blocks that will allow us to prove more important properties. Overall, we formalized constructs mixing basic geometrical aspects with neighboring information and communications. We abstracted away most of the geometrical notions, leading us to an abstract notion of connectivity in order to simplify the reasoning.

Providing a geometry module to formalize more precisely the geometrical aspects of a CAN would be an interesting further work. However, in the future, we first plan on investigating broadcast algorithms on top of CAN, prove their properties and hopefully design an efficient broadcast algorithm and prove its properties. For that, we plan to investigate further the algorithm proposed in [9] and see if it can be proven correct, and under which conditions.

The next steps we envision are the following.

– Formalization of several state-of-the-art or original broadcast algorithms on a CAN.
– Proof of the properties of those algorithm. We will focus on correctness and liveness, absence of duplicate messages, and properties related to fault-tolerance.
– Our definition of CAN is actually quite abstract, and do not take into account all the geometrical characteristics of the existing CAN implementations. We will study what exact hypotheses of the CAN are necessary, and which of the algorithms also work on another topology (still resembling a CAN).
– Study the impact of churns : nodes arriving and departing cannot be modeled in our current framework, but we can modify the inductive way to define the broadcast algorithm so that the underlying structure of the CAN evolves, and check which properties are preserved. More formally, it would require to change the definition of our broadcast, by allowing the CAN itself to evolve along the inductive definition. For this we intend to interleave usual broadcast steps defined previously, and some CAN structural evolution steps. Properties on a broadcast algorithm with churn will then also depend on how the CAN network evolves.

Acknowledgment : We are indebted to our shepherd, Etienne Riviere, who helped us improve the paper. We thank him warmly along with the reviewers for their useful comments.

References

1. E.A. Ashcroft. Proving assertions about parallel programs*. *Journal of Computer and System Sciences*, 10(1) :110–135, 1975.
2. Rana Bakhshi and Dilian Gurov. Verification of peer-to-peer algorithms : A case study. *Electronic Notes in Theoretical Computer Science*, 181 :35–47, June 2007.
3. M. Castro, P. Druschel, A.M. Kermarrec, and A.I.T. Rowstron. SCRIBE : A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8) :1489–1499, 2002.
4. Miguel Castro, Manuel Costa, and Antony Rowstron. Peer-to-peer overlays : Structured, Unstructured, or Both. Technical Report MSR-TR-2004-73, Microsoft Research, 2004.
5. Y. Chu, S.G. Rao, S. Seshan, and H. Zhang. A case for end system multicast. *Selected Areas in Communications, IEEE Journal on*, 20(8) :1456–1471, 2002.

6. S. E Deering and D. R Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems (TOCS)*, 8(2) :85–110, 1990.
7. Sameh El-Ansary, Luc Alima, Per Brand, and Seif Haridi. Efficient broadcast in structured P2P networks. In *Peer-to-Peer Systems II*, pages 304–314. Springer, 2003.
8. Ali Ghodsi. *Distributed k-ary System : Algorithms for Distributed Hash Tables*. Thesis, KTH - Royal Institute of Technology, 2006.
9. A. Gupta, O.D. Sahin, D. Agrawal, and A.E. Abbadi. Meghdoot : content-based publish/subscribe over P2P networks. In *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 254–273. Springer-Verlag New York, Inc., 2004.
10. Q. Haiyan. Testing and proving distributed algorithms in constructive type theory. In *Proceedings of the 1st international conference on Tests and proofs*, pages 79–94, 2007.
11. L. Lamport and Safari Tech Books Online (Online service). *Specifying systems : The TLA+ language and tools for hardware and software engineers*, volume 14. Addison-Wesley, 2003.
12. Tianxiang Lu, Stephan Merz, and Christoph Weidenbach. Model checking the pastry routing protocol. In *10th International Workshop on Automated Verification of Critical Systems (AVoCS'10)*, September 2010.
13. P. Manzanares-Lopez, J. Malgosa-Sanahuja, J.P. Muñoz-Gea, and J.C. Sanchez-Aarnoutse. Multicast Services over Structured P2P Networks. *Handbook of Peer-to-Peer Networking*, pages 875–896, 2010.
14. T. Nipkow, M. Wenzel, and L.C. Paulson. Isabelle/HOL : a proof assistant for higher-order logic. *Lecture Notes In Computer Science ; Vol. 2283*, page 205, 2002.
15. A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.
16. S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. *Networked Group Communication*, pages 14–29, 2001.
17. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 161–172. ACM, 2001.
18. T. Ridge, M. Norrish, and P. Sewell. A rigorous approach to networking : TCP, from implementation to protocol to service. *FM 2008 : Formal Methods*, pages 294–309, 2008.
19. Thomas Ridge. Verifying distributed systems : the operational approach. In *Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '09*, page 429, Savannah, GA, USA, 2009.
20. A. Rowstron and P. Druschel. Pastry : Scalable, Decentralized Object Location, and Routing for Large-scale Peer-to-Peer systems. In *Middleware*, pages 329–350. Springer, 2001.
21. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord : A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 149–160, New York, NY, USA, 2001. ACM.
22. Ching tsun Chou. Mechanical verification of distributed algorithms in higher-order logic. *The Computer Journal*, 38(2) :152, 1995.
23. Pamela Zave. Lightweight verification of network protocols : The case of chord. Unpublished, `http://www2.research.att.com/~pamela/chord.pdf`, 2009.
24. Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John D. Kubiatowicz. Bayeux : an architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, pages 11–20, Port Jefferson, New York, United States, 2001. ACM.