HAL
open science

A Mechanized Model for CAN Protocols

Francesco Bongiovanni, Ludovic Henrio

To cite this version:

Francesco Bongiovanni, Ludovic Henrio. A Mechanized Model for CAN Protocols. 16th International Conference on Fundamental Approaches to Software Engineering (FASE'13), Mar 2013, Rome, Italy. hal-00802006

HAL Id: hal-00802006

https://inria.hal.science/hal-00802006v1

Submitted on 18 Mar 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Mechanized Model for CAN Protocols

Francesco Bongiovanni[1] and Ludovic Henrio[2]

[1] Joseph Fourier University and LIG Labs, Grenoble
[2] INRIA-I3S-CNRS, University of Nice Sophia Antipolis

Abstract. Formal reasoning on Peer-to-Peer (P2P) systems is an intimidating task. This paper focuses on broadcast algorithms for Content Addressable Network (CAN). Since these algorithms run on top of complex P2P systems, finding the right level of abstraction in order to prove their functional correctness is difficult. This paper presents a mechanized model for both CAN and broadcast protocols over those networks. We demonstrate that our approach is practical by identifying sufficient conditions for a protocol to be correct and efficient. We also prove the existence of a protocol verifying those properties.

Keywords: Structured P2P, CAN, broadcast algorithm, theorem proving.

1 Introduction

Structured Overlay Networks (SONs) are a class of P2P systems that emerged in the last decade to provide an abstraction of a lookup service over a large number of distributed nodes. These technologies are widespread and can be found at the heart of companies such as Amazon, Facebook and Twitter [4, 9].

Distributed applications operate now at a very large scale, and interactions between computers are no longer limited to *one-to-one* communications. Broadcast communication primitives simplify the application development by avoiding to explicitly write code for dissemination. In the context of a P2P system, communication primitives can be designed to take advantage of the logical topology to communicate efficiently. More important, such generic primitives can be proved to be correct, which increases the confidence the programmer and the user have in the system. In this article we are particularly interested in a broadcast algorithm running on top of the Content Addressable Network (CAN) [14] in which the broadcasted message is received *exactly once* by each node; we say that such a broadcast algorithm is *efficient*. Previous works such as M-CAN [15] and Meghdoot [6] also attempt to leverage CAN in order to build an efficient dissemination infrastructure that can be used by distributed applications.

However, there is no free lunch. Building high-assurance distributed applications which are *correct* remains a very difficult challenge but various methods have emerged over the years to tackle it [1, 2, 5, 11, 19]. Theorem proving seems to be among the best method to prove generic properties on distributed systems of *arbitrary size*; in particular verification techniques based on model-checking are,

in general, limited to systems of a predefined size and suffer from combinatorial explosion. This article promotes the use of theorem proving in the safe design of distributed systems, more particularly concerning the CAN overlay network.

We present here a mechanized model of CAN that will help the design of communication primitives and the formal proof of their properties. We use this model to define the characteristics and the properties of broadcast primitives. Our first contribution is thus a set of abstractions, properties, and theorems regarding the topological and the communication aspects of CAN. We formalize the *CAN network*, the *messages* over such a network, and a notion of *connectivity*: a zone is connected if any two nodes of this zone can communicate, possibly indirectly. Some of these proven abstractions can be reused with resembling topologies such as the Delaunay triangulation protocol (DT protocol) [10] and the Voronoi-based Overlay Network (VON) [8], widely used in online games. Our second contribution is to use our framework to characterize broadcast protocols and *prove the existence of an efficient broadcast* protocol by presenting a naive yet efficient *correct-by-construction* protocol and which directly leverage the topology of the structure; we do not build and maintain an extra layer on top of CAN (such as a broadcast tree) but rather make use of locally available information. Knowledge of the entire network is necessary for proving intermediary lemmas, but we show that such a knowledge is not used by the protocol itself.

We are interested in the proofs of *functional correctness* of broadcast algorithms for CAN, and of the fact that there is a protocol for which each node receives a message exactly once. We present selected Isabelle/HOL definitions and theorems to give an overview on how to reason about such algorithms and prove their properties. Our model is general and flexible enough to study CAN networks, as it provides the formalization of basic building blocks composing it. However, we are not interested in formalizing the full CAN protocol but rather on the minimal set of abstractions needed to reason about communication protocols for CANs. Overall, not only this paper proves that it is possible to design an efficient broadcast protocol for CAN but also that we can formally prove its correctness and efficiency. Compared to a realistic P2P network, we do not consider node churn in this study. Also we assume that reliability of point-to-point communication is an orthogonal concern that should be studied independently.

In §2, we present the CAN overlay network, the existing broadcast primitives for CAN, and the objectives of this paper. In §3 we present a mechanized model of CAN along with the proven abstractions making up the model. Broadcast algorithms are presented in §4. Finally we compare our study to other related work in §5.

2 Background and motivation

A CAN [14] is a structured P2P network based on a d-dimensional Cartesian coordinate space, labelled $\mathcal{D}$. This space is dynamically partitioned among all nodes in the system such that each node is responsible for storing data, in the form of *(key, value)* pairs, in a sub-zone of $\mathcal{D}$. Each node is responsible for

a zone, and the set of zones is disjoint and covers the whole space. To store (k, v)-pairs, the key k is deterministically mapped onto a point in $\mathcal{D}$, using consistent hash functions, and the value v is stored by the node responsible for the zone containing this point. The search for the value corresponding to a key k is achieved by applying the same hash function on k in order to find the node responsible for storing the value. The routing process starts at the query originator and traverses iteratively its neighbors (a node only knows its adjacent neighbors), until the zone responsible for the key to store/retrieve is reached.

To limit the load on the network, we want to minimize the number of messages necessary to perform a broadcast. We first define precisely the hypotheses on the network topology to know exactly on which kind of networks our algorithms are valid. There are several ways to construct a broadcast algorithm depending on the structure of the CAN:

- The appendix of the seminal paper [14] suggests to build a construction tree, where a node is child of another if it joined this other node. This construction tree can be used to perform an efficient broadcast but the root of the tree would be overloaded. The main drawback of this approach is that keeping the tree structure when nodes leave leads to strong constraints (only a leaf node can take the place of a leaving node) and may require huge data transfers.
- In M-CAN [15] and Meghdoot [6], a CAN is defined as a structure where each node is responsible for an hyperrectangle: when a node leaves the network, its neighbour(s) can, for example, extend their own zone with the zone left by the leaving node. The only structure that can be exploited by a broadcast protocol is the graph of neighbors. Maintaining a spanning tree on such a structure with nodes joining and/or leaving the network is far from trivial.
- A CAN can be defined as a structure where each node is responsible for an area with no particular shape. This structure fits with the protocol described in [14] (Sec. 2.3): when a node leaves the network, its neighbour becomes responsible for its area. This is the most general case (it encompasses the two previous ones), and the one used here. Even in this unconstrained setting, we prove that an efficient broadcast algorithm exists.

Thus, we represent the network as a graph and provide an efficient algorithm on such a graph; computing a minimum spanning tree on this potentially evolving graph without global knowledge of the graph seems impossible. However, in a CAN, geometrical information can be exploited to avoid any two nodes from sending the same information to the same node. In this paper, our objective is to rely on a notion: "is a set of points connected?" which can be computed locally and to prove that, from this notion, we can design an efficient broadcast algorithm. The algorithm we propose here does not tolerate changes in the network during the broadcast, but it does support changes in the network between two broadcasts. Considering the case where a peer joins or leaves the CAN, i.e. node churn, is left for future work.

Figure 1 shows an efficient broadcast as ensured by M-CAN [15], the protocol sends messages first vertically (to a single neighbor above or below) and then horizontally, avoiding duplicates by sending only to the node that touches the

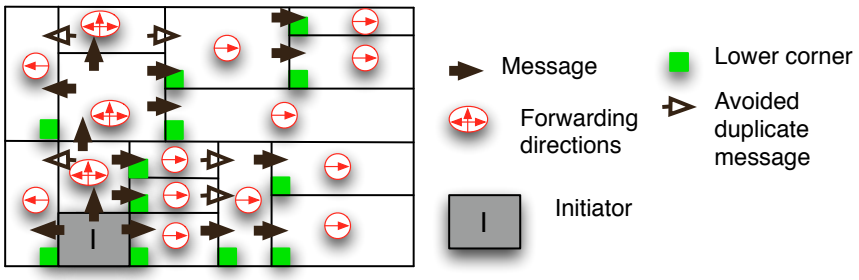

Fig. 1. M-CAN: Efficient flooding in 2 dimensional CAN

lower corner. Unfortunately, this algorithm does not eliminate all duplicates if the dimension is greater than two. In a publish/subscribe context, Meghdoot [6], built atop CAN, also proposes a mechanism to avoid duplicates but requires the broadcast to originate from one corner of the zone to be covered. We underline here that no broadcast algorithm atop CAN has been actually formally specified.

Using an interactive theorem prover such as Isabelle/HOL [13] and its higher-order logic provides us the expressiveness needed to formalize distributed algorithms and reason about them. A higher order logic naturally supports the formalization of the data structures of the algorithms; it also provides the reasoning tools to prove properties on those algorithms and structures. The expressiveness of Isabelle's logic allows us to reason about an abstraction of the system we design, meaning that we can abstract away some details of the CAN overlay and focus on the aspects ensuring the properties of the broadcast algorithm.

Our motivation is to put forward abstractions for proving correctness properties of distributed algorithms on top of CAN, and to promote the development of distributed algorithms *proven correct*. CAN is a difficult setting for proofs, because the structure entails a geometrical structure, which is more difficult to handle in Isabelle than inductive structures. We abstract away most of the geometrical notions, and rely on the notion of connection between nodes, expressed as a neighbor relation. We focus on two properties of broadcast protocols:

- **Efficiency:** a node receives only one message during a broadcast.
- **Coverage:** all the nodes receive the message (correctness property).

We write our specification in such a way that an external reader familiar with basic logic and mathematics can understand it. Indeed, our purpose is to make our results accessible and convincing for the community of distributed systems, including people not familiar with formal methods.

Efficiency vs. Robustness. One can easily argue that duplicating messages can actually increase the robustness of the broadcast in case of communication failures. However, the redundancy provided by a non-efficient broadcast algorithm is not satisfactory. Indeed with an inefficient algorithm, as shown in Figure 1, some nodes receive the message once, while other can receive it more than twice. Relying on an efficient algorithm is a better starting point in order to design robust algorithms in a smarter way. For example, to increase robustness, one

could perform two efficient broadcasts in parallel from two different places in the network and reversing the dimensions of the CAN (considering the first one as the last one); this would ensure that each node receives each message *exactly twice*, most of the time coming from two different directions.

3 Reasoning about a CAN-like structure

This section presents the formalization of CAN written in Isabelle/HOL[3]. We present below the most important definitions and lemmas, expressed in mathematical style; some notions will be defined informally but most of the details are omitted; instead we prefer to give some insights regarding the principles of the formalization. For manipulating structures we use the following notations: ! accesses an element of a list; # adds an element at the beginning of the list; @ appends two lists; *fst* and *snd* access the elements of a pair.

A first 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 chose to represent a CAN by a set of nodes, a zone for each node, and a neighboring relationship, stating whether any two nodes are neighbors. More precisely, a *CAN* is a set of integers identifying the different nodes. A function *CZ* matches each node to a *Zone*; a *zone* is simply a set of points, where each point is represented by a tuple of integers: *CZ N* is the zone under the responsibility of the node *N*. Also we require that the set of nodes is finite and the set of their zones partitions the whole space into disjoint zones covering the whole space. Each node is responsible for a zone that never changes and is called the zone of a node; note that our broadcast algorithm will also rely on some zones, i.e. sets of points.

Definition 1 (CAN). *The set of valid CANs is defined as follows:*

$$\begin{aligned} CAN \equiv \{ & (Nodes, CZ, Neighbours).\, Nodes \in \mathcal{P}(\mathbb{N}) \wedge CZ \in \mathbb{N} \rightarrow Zone \wedge Neighbours \in \mathcal{P}(\mathbb{N} \times \mathbb{N}) \\ & \wedge\ symbNodes \text{ is finite} \wedge Neighbours \text{ is finite} \\ & \wedge\ \forall x, y.\, (x, y) \in Neighbours \Rightarrow (y, x) \in Neighbours \wedge \forall x.\, (x, x) \notin Neighbours \\ & \wedge\ \forall tup.\, \exists N \in Nodes.\, tup \in CZ(N) \\ & \wedge\ \forall N, N' \in Nodes.\, CZ(N) \neq \varnothing \wedge N \neq N' \Rightarrow \neg(CZ(N) \text{ intersects } CZ(N')) \ \ \} \end{aligned}$$

We also define three auxiliary functions *Nodes*, *Zones*, and *Neighbour* returning each part of a *CAN*. We adopt a formalization more general and flexible than the strict CAN protocol, more precisely:

- As discussed in Section 2, each node is responsible for a zone of any shape, whereas in most CANs, each node should always be responsible for a single hyperrectangle. We believe that this formalization is better adapted to model a dynamic CAN with nodes joining/leaving the network.
- *We do not relate zones with the concept of neighbor.* This alleviates us from geometric reasoning: we mainly rely on two relationships: "is a node neighbor of another?" and "do two zones intersect?" This allows us to reason at a topological level rather than a geometrical level.

[3] see: `www-sop.inria.fr/oasis/personnel/Ludovic.Henrio/misc`.

We thus define a predicate *Z intersects Z'* that checks whether a zone Z intersects a 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 intersects Z. Then we define *NodesInZone C Z*, the set of nodes whose zones intersect the zone Z:

$$NodesInZone\ C\ Z = \{N \in Nodes(C).\ Zones(C, N)\ intersects\ Z\}$$

We define the size of a zone *ZoneSize C Z* as the number of nodes it contains. Then we define the connectivity of a CAN zone. This notion is close to the geometrical notion of path connectivity but dedicated to the CAN networks. The idea is that a zone is connected if a message can go between any two nodes in the zone passing only through nodes intersecting the zone. In the context of a broadcast algorithm, we will know that in such a zone the broadcast is possible; this notion will also give us a criteria that we have to follow when dividing the area to be covered into smaller zones. We state that a zone is connected if between any two nodes intersecting the zone, there is a path of nodes intersecting the zone where two consecutive nodes of the path are neighbors.

Definition 2 (Connected).

$$\begin{aligned} &Connected\ C\ Z \Leftrightarrow \forall N, N' \in NodesInZone\ C\ Z. \\ &\qquad \exists NL\ a\ list\ of\ distinct\ nodes\ starting\ by\ N\ and\ finishing\ by\ N'. \\ &\qquad \forall i < length(NL) - 1.\ NL!i \in NodesInZone\ C\ Z \wedge Neighbour(C, NL!i, NL!(i+1)) \end{aligned}$$

Lemmas. To reason about CAN structures, we define several generic lemmas related to topology. These lemmas ease the reasoning on connectivity, intersection and nodes. Some typical lemmas prove generic properties on connectivity or intersection on the union or the intersection of zones. These small lemmas will reveal particularly useful for proving the properties of broadcast protocols.

Lemma 1, for instance, states that the union of two connected zones is connected if two nodes of the two zones are neighbors.

Lemma 1 (Connected-union).

$$\frac{\begin{array}{c} Connected\ C\ Z \qquad Connected\ C\ Z' \\ N \in NodesInZone\ C\ Z \qquad N' \in NodesInZone\ C\ Z' \qquad Neighbour(C, N, N') \end{array}}{Connected\ C\ Z \cup Z'}$$

We also prove a symmetric lemma, allowing us to find neighboring nodes in two zones, provided those two zones and their union are connected (and the union contains at least two nodes).

Reasoning by induction on a zone. We also provide induction principles that allow one to prove a property related to a zone by induction on the size of the zone. A trivial induction lemma expresses directly induction on the number of nodes in the zone on which the property is verified. More interestingly, one can prove a property by adding one by one the nodes belonging to the zone of interest; this allows some form of structural induction on a CAN zone.

Theorem 1 (induct-node-zone2).

$$\frac{P(\varnothing) \qquad \forall Z.\, P(Z) \Rightarrow \forall N{\in}Nodes(C).N{\notin}NodesInZone\ C\ Z \Rightarrow \forall Z'.\, NodesInZone\ C\ Z' = \{N\} \Rightarrow P(Z{\cup}Z')}{P(Z)}$$

This theorem states that, if (1) we prove that a property P is true for an empty zone, and (2) we prove that if P is true for a zone then it is true for a zone intersecting one more node; then the property is true for all zones. The proof of this induction principle mainly relies on the fact that the set of all nodes of the CAN C must cover the entire space.

Messages and message paths. Once the network is defined, we define messages and paths followed by messages. As we will use messages to perform a broadcast towards a zone, a message is made of four parts: an identifier for the message m (which could represent also its payload), a source node x, a destination node y, and the zone Z to which it must be transmitted. We also define an abbreviation $<m|x,y,Z>$ for defining a *Message*. *Message-source*, *Message-dest*, and *Message-zone* are functions accessing the last three fields.

We believe it is important to provide tools to reason about the path followed by a message. Indeed, communication inside CAN heavily relies on the notion of paths. For this, we define a path as a set of consecutive messages, and provide tools to reason inductively on those path.

Definition 3 (valid-path). *valid-path is a predicate that checks whether the message path ML is valid, i.e. each message is sent from the destination of the previous message.*

$$\begin{aligned} &\textit{valid-path msgs } ML \Leftrightarrow ML \neq [] \wedge \textit{distinct}(ML) \wedge \forall i < \textit{length}(ML).\, ML!i \in \textit{msgs}) \\ &\qquad \forall i < \textit{length}(ML) - 1.\, \textit{Message-dest}(ML!i) = \textit{Message-source}(ML!(i+1)) \end{aligned}$$

For a path to be valid, we additionally require that no two messages are the same; indeed, we will only consider paths of messages among a finite set *msgs* and we would like to consider only paths of finite length, this allows us to reason about the longest path in a zone. We provide various lemmas for building (valid) paths and reasoning about them.

4 Broadcast Algorithms over a CAN

Defining a broadcast in a natural way using Isabelle/HOL is not trivial; here we decide to put an emphasis on the way messages are processed. Our formalization is centered around the specification of messages which are the *consequences* of a given message and on the specification of the *set of messages* used to broadcast the original message. Then we define the way messages are broadcasted by an inductive definition, where messages are "processed" one message after the other sequentially. In our formalization, *Broadcasts* are triple made of a *CAN*, a message set, and an initiator node constrained by several well-formedness rules.

Definition 4 (Broadcast).

$$\begin{aligned} &\textit{Broadcast} \equiv \big\{(C, \textit{msgs}, \textit{init}).\quad \textit{init} \in \textit{Nodes}(C) \\ &\wedge\ \forall m, x, y, Z, m', Z'.\ <m|x, y, Z> \in \textit{msgs} \wedge <m'|x, y, Z'> \in \textit{msgs} \Rightarrow (m = m' \wedge Z = Z') \\ &\wedge\ \forall <m|s, d, Z> \in \textit{msgs}.\big(s \in \textit{Nodes}(C)\ \wedge\ d \in \textit{Nodes}(C)\ \wedge\ \textit{Neighbour}(C, s, d) \wedge \\ &\quad \big(s = \textit{init} \vee \exists ML.(\textit{valid-path msgs } ML \wedge \textit{destination}(ML) = s \wedge \textit{source}(ML) = \textit{init})\big)\big)\big\} \end{aligned}$$

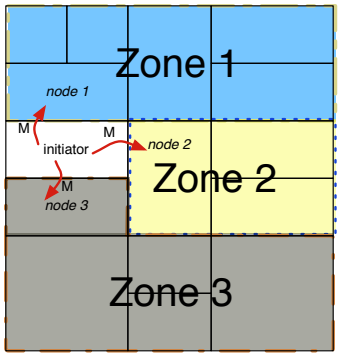

Fig. 2. Zone node list (ZNL) definition

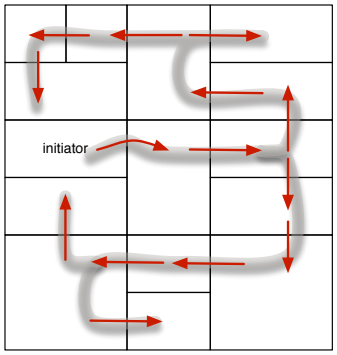

Fig. 3. Naive efficient broadcast

The constraints expressed in the definition state that: (1) There is a single message between any 2 nodes. (2) The initiator is a node of the CAN. (3) All messages are exchanged between neighbor nodes of the CAN, and thus the broadcast pattern respects the CAN protocol. (4) All messages must be sent by a node that has been reached by a list of messages *MsgL* originating from the initiator. The existence of such a valid path ensures that a broadcast only relies on messages transmitted from nodes to nodes, and no message is spontaneously created (except for the initiator of course). Note that it is not sufficient to require that each message source is the destination of another message, because that would allow loops of messages not passing by the initiator. We denote $<C,M,n>$ a *Broadcast*, and define functions *BC-CAN*, *BC-msgs*, and *BC-init* to access its fields. We can then define a predicate checking whether a broadcast covers the whole CAN, i.e., whether each node of the CAN is either the initiator or the destination of a message:

Definition 5 (Coverage).

$$Coverage\ (C, msgs, init) \Leftrightarrow \forall N \in Nodes(C).N = init \vee \exists m, s, Z.\ <m|s, N, Z> \in msgs$$

We decided to rely on the notion of zone to be covered to define a broadcast algorithm because this seems quite adapted to the structure of the CAN. This zone to be covered can have two purposes depending on the algorithm. First it allows the specification of broadcast protocols where only the nodes in a given zone have to receive the message. Also, since we are interested in an efficient algorithm that minimizes the number of messages necessary to broadcast the information, it seems reasonable to split efficiently the zone to be covered in order to avoid sending a message to the same node twice. A broadcast algorithm can be entirely characterized by a function that, given a node N receiving a message and the zone Z to be covered by the message, returns a list of pairs (*Zone*, *Node*), that we call ZNL (*Zone-Node list*). Each pair (Z_i, N_i) of the ZNL consists of Z_i, a sub-zone of Z, and N_i a neighbour of N belonging to Z_i: the message is forwarded to N_i that is now responsible for covering Z_i. The zone Z_i must be connected (as defined above) otherwise it would be impossible to cover

it while staying inside Z_i. Figure 4 illustrates the notion of ZNL for the first step of the broadcast: the message to be broadcasted is sent from the initiator to its neigbhors belonging to three different zones to be covered, which in turn, will be responsible for broadcasting the message to the zone they received.

Definition 6 defines an *optimal ZNL*, which underpins an efficient broadcast. To guarantee efficiency, we require that the zones of the list form a partition of the original zone, and that each node belong to the zone it receives. A broadcast will be efficient if every message generates an optimal ZNL.

Definition 6 (Set-of-Optimal-ZNL).

$$\begin{aligned}
&\textit{Set-of-Optimal-ZNL } C\ N\ Z \equiv \big\{ ZNL.\ \bigcup_{i<length(ZNL)} fst(ZNL!i) = Z \\
&\quad \wedge \big(\forall i < length(ZNL).\ Neighbour(C, N, snd(ZNL!i)) \ \wedge\ Connected\ C\ fst(ZNL!i) \\
&\qquad \wedge\ snd(ZNL!i) \in NodesInZone\ C\ fst(ZNL!i) \\
&\qquad \wedge\ \forall j < length(ZNL).\big(j \neq i \Rightarrow\ snd(ZNL!i) \neq snd(ZNL!j) \wedge \\
&\qquad\qquad NoNodeInBothZones\ fst(ZNL!i)\ fst(ZNL!j)\big)\big)\big\}
\end{aligned}$$

Definition 7 specifies the set of messages of a broadcast algorithm based on *ZNLmap*, which is a function that, given a node and zone, returns an optimal ZNL. The set of messages of the broadcast is defined by induction rules expressing how messages are processed. The inductive definition is of the form *ZNL-BC-msgs C Mid init znlmap msgs ML* where C is the CAN network, *Mid* is the message identifier, *init* is the initiator node, and *znlmap* is the *ZNLmap* used by this instance of the algorithm.

Definition 7 (ZNL-BC-msgs).

$$ZNL\text{-}BC\text{-}msgs\ C\ Mid\ init\ znlmap\ \varnothing\ [< Mid|init, init, Entire\text{-}Space\backslash Zones(C, init) >]$$

$$\frac{\begin{array}{c} ZNL\text{-}BC\text{-}msgs\ C\ Mid\ init\ znlmap\ msgs\ \ < Mid'|s, d, Z > \#ML \\ ML' = map\ (\lambda ZN.\ < Mid'|d, snd(ZN), fst(ZN)\backslash Zones(C, snd(ZN)) >)\ (znlmap(Z, d)) \end{array}}{ZNL\text{-}BC\text{-}msgs\ C\ Mid\ init\ znlmap\ (msgs \cup \{< Mid'|s, d, Z >\})\ ML@ML'}$$

The inductive definition works as follows: it takes one by one messages in *ML*, called the list of *messages to be treated*, processes them and put them in the *set of treated messages*, *msgs*. Processing a message consists in using the function *znlmap* to compute the consequences of this message: for each couple (Z_i, N_i) returned by $(znlmap(Z, d))$, a new message with destination N_i and zone Z_i minus the zone of N_i which we just covered is created and put in the new *ML* list. The original message that has been processed is then put in *msgs*. The rules of Definition 7 are applied iteratively treating one message after the other. At the end, *ML*, the list of messages to be treated is empty, and *msgs*, the set of treated messages contains the list of messages of the broadcast.

Relying on a *znlmap* function is important here because it ensures that the broadcast algorithm is built without the knowledge of the messages already sent by the protocol. This ensures that only the knowledge of the current node and the zone to be covered is used to decide which message to send. We will see in the definition of the efficient broadcast how the *znlmap* function can be modified to take into account local information (i.e., neighbors and their zones).

Some design choices have been made in the way messages are actually processed in our formalization. Messages are treated sequentially, but this of course does not correspond to the parallelism that occur in a real system. However this classical simplification has no consequence on the properties of interest here. More important is the fact that the messages to be treated (ML) are represented as a list, this total ordering is artificial and one could improve the representation by defining an equivalence relation allowing reordering of messages. However, the list is a good structure to reason inductively on the messages and to allow rules to be applied iteratively. Concerning treated messages ($msgs$), as we do not perform any further computation on them, a list was not necessary and we considered them to be a set; a multiset is not necessary because it is easy to prevent the same message to be sent twice between the same nodes.

We prove that the set of messages generated by an optimal ZNL constitutes a valid broadcast that covers the entire network:

Theorem 2 (ZNL-BC).

$$\frac{\begin{array}{c}\mathit{ZNL\text{-}BC\text{-}msgs}\ C\ \ \mathit{Mid}\ \mathit{init}\ \mathit{znlmap}\ \mathit{Finalmsg}\ []\qquad \mathit{init} \in \mathit{Nodes}(C)\\ \forall (Z,N) \in \mathit{History}(C, \mathit{init}, \mathit{Finalmsg}, []).\mathit{znlmap}(Z,N) \in (\mathit{Set\text{-}of\text{-}Valid\text{-}ZNL}\ C\ \ N\ \ Z)\end{array}}{(C, \mathit{Finalmsg}, \mathit{init}) \in \mathit{Broadcast}}$$

Theorem 3 (coverage-ZNL).

$$\frac{\begin{array}{c}\mathit{ZNL\text{-}BC\text{-}msgs}\ C\ \ \mathit{Mid}\ \mathit{init}\ \mathit{znlmap}\ \mathit{Finalmsg}\ []\qquad N, \mathit{init} \in \mathit{Nodes}(C)\\ \forall (Z,N) \in \mathit{History}(C, \mathit{init}, \mathit{Finalmsg}, []).\mathit{znlmap}(Z,N) \in (\mathit{Set\text{-}of\text{-}Valid\text{-}ZNL}\ C\ \ N\ \ Z)\end{array}}{\exists s, Z.\ < \mathit{Mid}|s, N, Z > \in \mathit{Finalmsg}}$$

In these two theorems, we consider the last step of induction, when the list of messages to be treated is empty ([]). Also, we require that for each zone and node of *History*, *znlmap* verifies *set-of-Optimal-ZNL*. *History*(...) is the set of pairs (zone, destination node) that influenced the previous treatment of messages, and will influence the processing of the next message if there is one. In other words, it is the set of pairs (zone,destination node) for the set of treated messages ($msgs$), and the next message to be treated (head of ML), plus the couple (initiator node, entire space). We also proved additional lemmas ensuring correctness and easing further proofs, e.g. that the message content is unchanged during the process.

The theorems presented above are not sufficient to prove that the final step of the induction (where no more message is to be treated) is reachable. That is why we proved that ensuring $\forall Z\ N.\ \mathit{znlmap}(Z,N) \in \mathit{Set\text{-}of\text{-}Optimal\text{-}ZNL}\ C\ N\ Z$ is sufficient to ensure progress from each step. However, this is still not sufficient to ensure reachability of the final step while we do not ensure that the broadcast process is finite. Also, the approach presented above requires to compute, from the beginning, the function *znlmap*, which is unrealistic. Below, we prove the existence of an efficient broadcast, built step by step, which ensures reachability of the final step.

An efficient broadcast algorithm A simple but efficient algorithm can be designed by constructing ZNLs as follows: suppose a node N receives a message with a given zone Z to cover, where $Z \cup CAN\text{-}Zones(C, N)$ is connected. We split this zone into several zones Z_i, where each zone is connected, no zone touches another ($i \neq j \Rightarrow Z_i \cup Z_j$ is not connected), and each zone contains a node N_i neighbour

of N. We first prove that such a decomposition necessarily exists (Lemma 2). The decomposition is optimal if the same node does not belong to two zones, which is true because, for any two zones, $Z_i \cup Z_j$ is not connected. We can thus prove that under the conditions mentioned above ($Z \cup CAN\text{-}Zones(C, N)$ is connected), an optimal decomposition exists (Lemma 3).

Lemma 2 (Zone-decomposition).

$$\exists zl. \bigcup_{i<length(zl)} zl!i = Z \wedge \forall i < length(zl).\ Connected\ C\ \ zl!i \wedge \forall i, j < length(zl). i \neq j \Rightarrow \neg Connected\ C\ \ zl!i \cup zl!j$$

Lemma 3 (OptimalZNL-existence).

$$\frac{N \in Nodes(C) \qquad Connected\ C\ (Z \cup Zones(C, N)) \qquad N \notin NodesInZone\ C\ Z}{\exists ZNL.\ ZNL \in Set\text{-}of\text{-}Optimal\text{-}ZNL\ C\ N\ Z}$$

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; note that this new zone verifies the hypothesis above, and thus the decomposition mechanism can be applied recursively until each node receives the message. The drawback of the decomposition lemma is that it seems to require the knowledge of C. More precisely we need to know C to decide whether a zone Z is connected or not. In a real CAN, geometrical connectedness is sufficient to ensure network connectedness, and thus connectedness can be computed without knowing the exact network topology. Consequently, in a real CAN, the decomposition can be computed locally at the node that needs to forward the information.

Remember that, to reason about the whole algorithm, we parametrized *ZNL-BC* by the *znlmap*, supposed to be known originally, not computed at each step. To prove that our algorithm works, we need to compute the optimal *znlmap* at each step with local information. This issue can be resolved in two steps. First, we show that the messages of the broadcast are only sensitive to the values of *znlmap* for the history $History(\dots)$. Second, we show that, provided the history is optimal, we can treat one more message and provide an extended *znlmap* that is still optimal for the new history. Additionally, we prove side conditions on the set of messages treated and to be treated so that, e.g., we can prove that the algorithm terminates.

Theorem 4 (Progress-ZNL).

$$\frac{\begin{array}{c} ZNL\text{-}BC\text{-}msgs\ C\ Mid\ init\ znlmap\ msgs\ MList \qquad init \in Nodes(C) \\ MList \neq [] \qquad \forall (Z, N) \in History(C, init, msgs, MList).znlmap(Z, N) \in (Set\text{-}of\text{-}Valid\text{-}ZNL\ C\ N\ Z) \end{array}}{\begin{array}{c} \exists Mlist', znlmap'.\ msgs \cup set(Mlist) \subseteq msgs \cup \{MList!0\} \cup set(MList') \wedge \\ (Message\text{-}zone(MList!0) = \varnothing \ \vee \\ \bigcup_{i<length(MList')} NodesInZone\ C\ Message\text{-}zone(MList'!i) \subset \bigcup_{i<length(MList)} NodesInZone\ C\ Message\text{-}zone(MList!i)) \wedge \\ \forall (Z, N) \in History(C, init, msgs \cup \{MList'!0\}, MList').znlmap(Z, N) \in (Set\text{-}of\text{-}Valid\text{-}ZNL\ C\ N\ Z) \end{array}}$$

The consequences of those last steps are twofold. First, they allow us to build the znlmap at each step by finding, each time we treat a message, the new zone-node-list corresponding to the new message. Also, the specification above ensures that our algorithm terminates: it can always progress (because of the optimality of the ZNL) and we exhibited a well-founded order along which the set of messages to be treated decreases: at each step, the size of the zone to be covered by all the messages to be treated decreases strictly (i.e. the number of

nodes intersecting it), or it stays the same and the number of messages to be treated decreases strictly.

Finally, we prove the two following theorems stating that the same node is not reached twice. More precisely, at each step, if a node received a message (it is the destination of a message treated or to be treated) then it is not the destination of another message, and it is not in a zone to be covered. Several intermediate lemmas guarantee the fact that zones to be covered are well-separated and finally, we prove that, at the last step of the broadcast protocol, no two nodes receive the message. Overall, we have proved that there is a broadcast protocol that do not send twice the message to the same node.

Theorem 5 (Efficiency-final-step).

$$\frac{\begin{array}{c}\textit{ZNL-BC-msgs } C \ \textit{Mid init znlmap Finalmsg } [] \qquad \textit{init} \in \textit{Nodes}(C) \qquad M, M' \in \textit{Finalmsg} \\ M \neq M' \qquad \forall (Z, N) \in \textit{History}(C, \textit{init}, \textit{Finalmsg}, []).\textit{znlmap}(Z, N) \in (\textit{Set-of-Valid-ZNL } C \ N \ Z)\end{array}}{\textit{Message-dest}(M) \neq \textit{Message-dest}(M')}$$

Theorem 6 (Efficiency-existence).

$$\frac{\textit{init} \in \textit{Nodes}(C)}{\begin{array}{c}\exists \textit{Finalmsg}, \textit{znlmap}.\ \textit{ZNL-BC-msgs } C \ \textit{Mid init znlmap Finalmsg } [] \\ \wedge\ \forall M, M' \in \textit{Finalmsg}.\ (M \neq M' \Rightarrow \textit{Message-dest}(M) \neq \textit{Message-dest}(M'))\end{array}}$$

All the steps shown above ensure that the *znlmap* function can be built iteratively from local information, and that no knowledge of the message history is necessary to build it. Concerning locality, we rely on the local decidability of the predicate "*Connected C Z*", which can be evaluated without the knowledge of the whole network in the case of a real CAN thanks to geometrical considerations. Relying only on local knowledge is crucial for a P2P algorithm but we cannot prove it formally here. To solve this issue the solution would be to prove that the Lemma 3 is true in a n-dimensional space without knowledge of the whole network; such a proof would require the geometrical formalization of a CAN which is out of the scope of this study.

Concluding Remarks. It is important for us to have a formalism for expressing the CAN broadcast that is easy to understand. Although the specification we showed here 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 implement the program corresponding to our definitions. 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, even if we could try to provide new abbreviations for expressing message transmission more easily. In fact, the definition of *ZNL-BC* distinguishes the messages for which the consequence have already been computed from the others, which is crucial in our proofs, changing this definition to a more implicit set-oriented one would make the proofs even more complex or unfeasible.

In our proofs, we rely on a single axiom that states that the entire space is connected, which is a necessary prerequisite that we are not able to prove since we do not relate neighborhood with any geometrical notion.

The current specification and proofs consist of almost 5000 lines of Isabelle, for more than 150 lemmas and theorems. The length of the proofs is however not uniform: simple properties on the network or the connectivity could take a couple of lines, whether advanced properties on connectivity, and most of the properties of the broadcast algorithm require dealing with a lot of cases, or rely on complex inductions, they necessitate several hundreds of lines.

5 Related Work

A fair amount of work has been done on the verification of distributed systems. This section presents representative works that use theorem provers for reasoning on distributed algorithms, or discuss the formalization of DHT protocols.

Chou [18] proposes a reasoning infrastructure using HOL to formalize and verify distributed algorithms and validates his approach on a propagation of information with feedback algorithm. In the same line of direction, the work done by Qiao Haiyan [7] reports experiences in verifying distributed algorithms in constructive type theory using the Agda/Alfa proof assistant and provides a methodology mixing testing and verification of distributed algorithms. Ridge, in his work [17], takes an *operational approach* to distributed systems verification. His goal is to demonstrate, through a combination of symbolic evaluation and invariant checking, that the verification of distributed system is feasible down to the executable code level. His work relies on previous notable efforts, such as a rigorous approach to describe network protocols [16], a formal model of the OCaml programming language, and an operational verification of OCaml code. Charron-Bost and Merz [3] formally verified a modified version of the Paxos consensus algorithm using Isabelle/HOL. They modeled the consensus algorithm using the Heard-Of model (HO), a round-based computational model for fault-tolerant distributed systems. Their abstractions and proven properties as well as the formalization of the HO model can be used to study a large class of distributed algorithms; however the formalization does not embed any topological aspects of the underlying topology. Contrary to "classical" distributed algorithms such as consensus, distributed algorithms running on top of structured P2P systems, can not abstract away the underlying infrastructure, particularly if the algorithm relies on it. A formalization of an efficient broadcast algorithm on top of CAN has to rely on the needed abstractions and proofs regarding the topological aspects of CAN in order to prove the efficiency of the algorithm. The closest works on distributed algorithms for P2P systems are Meghdoot [6] and M-CAN [15] which are broadcast/multicast algorithms atop CAN which claim partial efficiency in terms of number of exchanged messages; they were briefly described in Section 2. Unfortunately no formalization nor any proofs were presented in these works. On the contrary, our objective is to use formal methods to prove with high confidence the correctness and efficiency of dissemination protocols.

Borgström et al. [2] were interested in the verification of DHT protocols. They formalized and verified a variant of Chord in static settings (i.e. no churn) using

CCS, a process algebra. In a subsequent work, Bakshi et al. [1] used π-calculus to prove the correctness properties of Chord in the pure-join model of the protocol. Zave, in her work [19] proved the Chord protocol in its two models: the *pure-join* and *full*, using the Alloy analyzer. By providing a rigorous correctness proof of the pure-join model and that the full model of the protocol she proved that the original Chord protocol is indeed not correct.

Pastry was also the subject of a recent verification effort [11], which focus was to ensure the correctness of Pastry's algorithms. The *join* and *lookup* protocols were specified using TLA$^+$ and the properties verified using the TLC model checker and the TLAPS proof assistant. Compared to the model-checking in general, theorem proving requires the help of the programmer to prove properties that are valid on an arbitrary number of processes. We focused in this paper on the pure use of Isabelle/HOL because most of our reasoning were about topology and induction, but we do not exclude the use of a proof assistant in conjunction with SMT solvers [12] in the future to ease reasoning, or reason more deeply about liveness properties for example.

6 Conclusion

To the best of our knowledge, we are the first ones to formalize an abstraction of the CAN overlay network using a theorem prover, and provide a framework to reason about those networks. This formalization efforts should increase the understanding of distributed algorithms for structured P2P networks, and the confidence one has in their correctness. We define formally all the constructs necessary to specify and prove properties of a broadcast algorithm on top of CAN. This paper presented the most important notions and support lemmas we designed in order to provide a convenient level of abstraction for reasoning on communication algorithms while abstracting away the geometrical concerns. The structured network represented is more general than a CAN: in a CAN, zones are necessarily hyperrectangles, whereas ours could be any tuple set. We prefer relying on a less restrictive definition of the structure to see which properties of our algorithm are verified in those conditions and also to lay down the groundwork for future challenges such as node churn. Later, requirements on the structure can be added to prove further properties, e.g. an algorithm may only be efficient if the zones are hyperrectangles. Difficult parts of the formalization concern reasoning by induction on a set that is finite but not inductively defined. To ease this kind of reasoning when dealing with zones, we developed an induction principle based on the number of nodes inside a zone (theorem *induct-node-zone-2*, Section 3). The reasoning relies on the topology of the network, defined by a neighbor relation, and related lemmas deal with the notion of *connected* zone, i.e. a node where any two nodes can (indirectly) communicate. We used this framework to describe a class of broadcast algorithms that relies on the notion of "zone to be covered". *We proved that there exists an algorithm that covers the whole network without sending twice a message to the same node.* This development also shows the capabilities of our framework.

In the future, we plan to formalize an algorithm that sends more messages in parallel than the one we exhibited here. Also, a next logical step would be to see how we can take *churns*, i.e. nodes joining/leaving the network, into consideration in our formalization since P2P systems exhibit dynamic behaviors and we can not overlook this feature.

References

1. R. Bakhshi and D. Gurov. Verification of peer-to-peer algorithms: A case study. *Electronic Notes in Theoretical Computer Science*, 181:35–47, June 2007.
2. J. Borgström, U. Nestmann, L. Onana, and D. Gurov. Verifying a structured peer-to-peer overlay network: The static case. In *Global Computing*. Springer, 2005.
3. B. Charron-Bost and S. Merz. Formal verification of a consensus algorithm in the heard-of model. *Int. J. Software and Informatics*, 3(2-3), 2009.
4. G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *SOSP*, pages 205–220. ACM, 2007.
5. H. Guo, M. Wu, L. Zhou, G. Hu, J. Yang, and L. Zhang. Practical software model checking via dynamic interface reduction. In *SOSP*, pages 265–278. ACM, 2011.
6. 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.
7. Q. Haiyan. Testing and proving distributed algorithms in constructive type theory. In *Proceedings of the 1st international conference on Tests and proofs*, 2007.
8. S.Y. Hu, J.F. Chen, and T.H. Chen. VON: a scalable peer-to-peer network for virtual environments. *Network, IEEE*, 20(4):22–31, 2006.
9. A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, April 2010.
10. J. Liebeherr, M. Nahas, and W. Si. Application-layer multicasting with delaunay triangulation overlays. *Selected Areas in Communications, IEEE Journal on*, 20(8):1472–1488, 2002.
11. T. Lu, S. Merz, and C. Weidenbach. Towards Verification of the Pastry Protocol Using TLA$^+$. In *FMOODS/FORTE*, pages 244–258, 2011.
12. S. Merz and H. Vanzetto. Automatic verification of tla+ proof obligations with smt solvers. *Proceedings LPAR-18, LNCS, Mérida, Venezuela*, 2012.
13. 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.
14. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *SIGCOMM*, pages 161–172. ACM, 2001.
15. S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. *Networked Group Communication*, 2001.
16. T. Ridge, M. Norrish, and P. Sewell. A rigorous approach to networking: TCP, from implementation to protocol to service. *FM*, 2008.
17. Thomas Ridge. Verifying distributed systems: the operational approach. In *POPL*, page 429, Savannah, GA, USA, 2009.
18. Ching tsun Chou. Mechanical verification of distributed algorithms in higher-order logic. *The Computer Journal*, 38(2):152, 1995.
19. Pamela Zave. Using lightweight modeling to understand chord. *Computer Communication Review*, 42(2):49–57, 2012.