FIFO and Atomic broadcast algorithms with bounded message size for dynamic systems

Colette Johnen, Luciana Arantes, Pierre Sens

FIFO and Atomic broadcast algorithms with bounded message size for dynamic systems

Colette Johnen†, Luciana Arantes‡ and Pierre Sens§
† INRIA Paris, LIP6, UMR 7606, France
Université de Bordeaux, LaBRI, UMR 5800, France
email : Colette.johnen@labri.fr
‡ Sorbonne Université, CNRS, INRIA, LIP6, France
email : Luciana.Arantes@lip6.fr
§ Sorbonne Université, CNRS, INRIA, LIP6, France
email : Pierre.Sens@lip6.fr

***Abstract*—FIFO broadcast provides application ordering semantics of messages broadcast by the same sender and have been mostly implemented on top of unreliable static networks. In this article, we propose a round-based FIFO broadcast algorithm with both termination detection and bounded message size for dynamic networks with recurrent connectivity (Class $\mathcal{TC}^{\mathcal{R}}$ of Time-varying Graph formalism [1]). Initially, processes only know the number of processes N in the system and their identifier. Due to the dynamics of the network links, messages can be lost. Since no unbounded timestamp is used to identify a message, its size is bounded to $2N + O(log(N)) + msgSize$ bits where $msgSize$ is the bound size in bits of the broadcast data. We also present an FIFO atomic broadcast algorithm in Class $\mathcal{TC}^{\mathcal{R}}$ that uses the proposed FIFO broadcast and deliver primitives.**

***Index Terms*—Theory of computation, Distributed algorithms, dynamic graphs, communication primitive, FIFO broadcast, Causal Broadcast, Atomic Broadcast, bounded message size**

I. Introduction

Many applications in distributed message-passing system (e.g. publish subscriber systems, permission-based mutual exclusion, replicated server systems, etc.) use group communication broadcast primitives to disseminate a message to all processes of the system. They are usually implemented on top of a point-to-point network. Existing broadcast primitives often provide reliability properties, imposing constraints on the order of message delivery by processes. Some examples of message ordering are FIFO, causal, and total order [2]. Another feature that a broadcast can present is termination detection which ensures that if a process broadcasts a message it will be aware that the message has been delivered by all correct processes within a finite time.

In this article we are particularly interested in FIFO broadcast which is a reliable broadcast that guarantees that messages broadcast by the same sender are delivered in the order they were broadcast. FIFO broadcast provides application ordering semantics of messages broadcast by the same sender and are thus used by applications where, for instance, the sender's actions or updates to sender's replicated data must be applied in FIFO order by all processes. Such a message ordering is usually implemented with the sender including in each message a sequence number which is incremented at every new broadcast. By keeping a buffer to temporarily store out of order messages, processes deliver the sender's messages respecting their sequence number order [2]. We should point out that since the sequence number values are not bounded, such an implementation may be unfeasible for long-term applications with high broadcast frequency.

The majority of broadcast algorithms of the literature have been proposed for unreliable static distributed systems whose nodes and/or links can fail but the point-to point communication network is fixed and process membership is known [2], [3], [4], [5]. However, many current distributed computing systems are usually unstructured, self-organizing, and, like MANETs (mobile ad-hoc networks) or Mobile Robot Networks [6], support nodes' mobility. Therefore, the network communication structure varies and the temporal variations in the network topology imply that these systems cannot be viewed as a static connected graph with fixed paths between nodes established before the broadcast of messages. The communication connection graph is dynamic and a path between two nodes, denoted journey, is built dynamically over the time. On the other hand, to guarantee dissemination of messages in dynamic networks, recurrent connectivity among nodes is a fundamental assumption for broadcast algorithms.

By considering an asynchronous systems with fixed constant number N of processes communicating through a dynamic network that ensures recurrent connectivity, we propose in this article a FIFO broadcast algorithm with both termination detection and bounded message structure for controlling sender's messages ordering. Processes execute in synchronous round and, since they do not fail, thus the membership of the system does not change. The number of processes N is known by all processes but not the system membership which they discover by executing the algorithm. Due to the dynamics of the network links, messages can be lost. Every message is identified by a counter plus a label value. The former is bounded to $2N$ and the latter varies from 0 to 2. Since a message, besides the broadcast data, includes the sender's identification, the counter, and a 2-bit timestamp vector with N entries, its requires at most $2N + O(log(N)) + msgSize$ bits where $msgSize$ is the

This study was partially supported by ANR project ESTATE : ANR-16 CE25-0009-03.

size bound in bits of the broadcast data.

In order to model the dynamics of the network, we apply the Class $\mathcal{TC}^{\mathcal{R}}$ (recurrent connectivity) of Time-varying Graph (TVG) formalism [1]. In the class $\mathcal{TC}^{\mathcal{R}}$, at any point in time, every vertex can reach all others through a journey (*i.e.* the equivalent of a path in static graph). Without a journey from the vertex s to r starting after t, s cannot send message to r at or after t whatever is the routing mecanism used. Note that the temporal length of a journey (*i.e.* the equivalent of the path length in static graph) is unbounded in the Class $\mathcal{TC}^{\mathcal{R}}$. So, the broadcast delay cannot be bounded in the Class $\mathcal{TC}^{\mathcal{R}}$. If the dynamic graph belongs to the Class $\mathcal{TC}^{\mathcal{B}}(\Delta)$ (*i.e.*, at any point in time, every vertex can reach all others through a journey whose the temporal length is at most Δ), a broadcast requires at most Δ rounds.

By calling the FIFO broadcast and delivery primitives of the new algorithm, we present a second algorithm that easily implements a FIFO total order (atomic) broadcast.

As discussed in the Related Work Section, some existing works have proposed broadcast algorithms for some classes of dynamic networks [7], [8], [9], [10], [11], [12], [13] for subclasses of $\mathcal{TC}^{\mathcal{R}}$. They mainly focus on the efficiency of information dissemination in terms of number of rounds, giving some lower bounds for different dynamic graph classes. They do not study the delivery order of a sequence of messages from one emitter; neither the size of broadcast messages to ensure FIFO order property.

The rest of the paper is organized as follows. Section II discusses some related work while Section III introduces some concepts and the computational model. Sections V and VI respectively present the proposed FIFO broadcast algorithm and its proofs of correctness. The FIFO, total order algorithm is described in Section VII. Finally, Section VIII concludes the paper.

II. Related Work

The problem of broadcasting messages has been widely studied in the message passing model [14], [15], in wireless networks [16], [17] or overlay network [18], [19] using gossip protocols.

An extensive survey, collecting around sixty total order broadcast algorithms in message passing model, is presented in [3] where the algorithms are classified according to the mechanism they use to order messages: communication history, privilege-based, moving sequencer, fixed sequencer, and destinations agreement. It is worth pointing out that total order broadcast primitives are a fundamental building block for a lot of distributed systems, applications, algorithms and services such as State Machine Replication [20], managing replicated database [21], [22], [23], sequential consistency [24], [25], shared registers implementation with Byzantine failure [26] or continuous churns [27].

In the gossip model, messages are propagated at each round either with a certain probability to all the neighbors or to a subset of neighbors chosen at random. Such a model is different from a dynamic graph model since each edge is randomly chosen and not in a adversarial way. In [28], [29], gossip broadcast in edge-Markovian evolving graph is studied. Markovian evolving graphs are dynamic-graph models where the links among a fixed set of nodes change during time according to an arbitrary Markovian rule.

Fewer works study broadcasting in the presence of dynamic changes. The seminal work of [30] considered the problem of end-to-end message delivery under an assumption of eventual connectivity. Some papers address broadcasting under churn scenario handling constant join and leave of nodes in overlay networks [31], [32] but without considering changes in the network topology.

The work in [7] shows that it is possible to flood a single message to all nodes which eventually stop sending messages in *1-interval*, a weak connectivity model in which the communication graph may change from round to round, nevertheless it stays connected all times. Such a class of dynamic networks is contained in $\mathcal{TC}^{\mathcal{B}}(N) \subsetneq \mathcal{TC}^{\mathcal{R}}$.

The *T-interval* connectivity model was introduced in [9]. In this model, for every T ($T \geq 1$), consecutive rounds, a stable connected spanning tree exists. A *T-interval* connected dynamic graph is also a *1-interval* connected one. In [9], lower bounds of rounds to disseminate k pieces of information without termination detection to all the nodes in a *T-interval* connected dynamic graph are presented. In [13], the authors propose a variant of *T-interval*, denoted *T-interval k-connected*, where there exists a *k-vertex-connected* stable subgraph during T rounds. Any *k-vertex-connected* graph belongs to the $\mathcal{TC}^{\mathcal{R}}$ class. They study the single-message broadcast problem in this model providing a probabilistic protocol.

The authors in [8], [11] focus on broadcasting with termination detection over three different subclasses of the dynamic TVG class $\mathcal{TC}^{\mathcal{R}}$: eventual, bounded, or periodic reappearance of links. They compare the feasibility of three variants of the broadcast problem minimizing the delivery date at every node, the number of hops, and overall duration of broadcast, respectively.

In [10], Raynal et al. proposed a reliable broadcast algorithm suited to the class of dynamic systems where all links appears and disappears infinitely often (*i.e.*, a subclass of $\mathcal{TC}^{\mathcal{R}}$). The broadcast is based on a spanning-tree on top of which processes forward received messages to their respective neighbors.

Finally, in [12], the authors present a hierarchy of three classes of time-varying graphs and provide a solution for each class to the problem of Terminating Reliable Broadcast. The stronger class studied is $\mathcal{TC}^{\mathcal{B}}(\Delta)$.

III. Background

In this section, we present the FIFO and Total order specifications implemented by our algorithms as well as some concepts about time varying graph and their classes on which are solution is based.

A. Broadcast with termination detection

The following specification of FIFO broadcast is an adaptation of the original one [2] in a system where all processes are correct.

- *Validity*: If a process p invokes $Broadcast(m)$, then it eventually executes $Deliver(id(p), m)$.
- *Agreement*: If a process executes $Deliver(id, m)$, then all other processes eventually execute $Deliver(id, m)$
- *Integrity*: a process delivers at most once the message $(id(p), m)$ and only if p has previously invoked $Broadcast(m)$
- *FIFO order*: If process p invokes $Broadcast(m2)$ after invoking $Broadcast(m1)$ then every process delivers $(id(p), m2)$ after having delivered $(id(p), m1)$.

A broadcast with termination detection requires that, if a process p invokes $Broadcast(m)$, then p eventually detects that all processes have delivered the message $(id(p), m)$ within finite time.

Similarly, the specification of atomic (total order) broadcast can be also adapted when all processes are correct, by applying the above three definitions of *Validity*, *Agreement*, and *Integrity* properties and the *total order* property which states that all processes deliver the broadcast messages in the same order. More precisely:

- *Total order*: If a process delivers (id, m) before delivering (id', m') then all processes deliver (id, m) before delivering (id', m').

The atomic broadcast and consensus are equivalent problems [33].

B. Dynamic Graphs

We model the network dynamics using the dynamic graph (DG for short) paradigm [34].

For any directed graph G, we denote by $V(G)$ its vertex set and by $A(G)$ its set of directed edges. A *dynamic graph* $\mathcal{G}$ *with vertex set* V (DG for short) is an infinite sequence of directed loopless graphs $G_1, G_2, ...$ such that $V(G_i) = V$, for every $i \in \mathbb{N}^*$. For every $i \in \mathbb{N}^*$, we denote by $\mathcal{G}_{i\triangleright}$ the dynamic graph $G_i, G_{i+1}, ...$ with vertex set V, *i.e.*, the suffix of $\mathcal{G}$ starting from position i.

A *journey* $\mathcal{J}$ can be thought as a path over time from a starting vertex p_1 to a destination vertex q_k, *i.e.*, $\mathcal{J}$ is a finite non-empty sequence of pairs $\mathcal{J} = (e_1, t_1), (e_2, t_2), ..., (e_k, t_k)$ where $\forall i \in \{1, \ldots, k\}$, $e_i = (p_i, q_i) \in A(G_{t_i})$ and $i < k \Rightarrow q_i = p_{i+1} \wedge t_i < t_{i+1}$. We respectively denote by $dpt(\mathcal{J})$ and $arr(\mathcal{J})$ the *starting position* t_1 and the *arrival position* t_k of $\mathcal{J}$.

The *temporal length* of $\mathcal{J}$ is equal to $arr(\mathcal{J}) - dpt(\mathcal{J}) + 1$. We denote by $\mathcal{J}_{\mathcal{G}}(p, q)$ the set of journeys from p to q in $\mathcal{G}$. We let $\overset{\mathcal{G}}{\rightsquigarrow}$ be the binary relation over V such that $p \overset{\mathcal{G}}{\rightsquigarrow} q$ if and only if $p = q$ or there exists a journey from p to q in $\mathcal{G}$.

The *temporal distance* from p to q in $\mathcal{G}$, $\hat{d}_{\mathcal{G}}(p, q)$, is defined as follows: $\hat{d}_{\mathcal{G}}(p, q) = 0$ if $p = q$, otherwise $\hat{d}_{\mathcal{G}}(p, q)$ is $\min\{arr(\mathcal{J}) : \mathcal{J} \in \mathcal{J}_{\mathcal{G}}(p, q)\}$ (by convention, we let $\min \emptyset = +\infty$). Roughly speaking, the temporal distance from p to q in $\mathcal{G}$ gives the minimum timespan for p to reach q in $\mathcal{G}$. The *temporal distance from* p *to* q *in* $\mathcal{G}$ *at position* $i \in \mathbb{N}^*$, $\hat{d}_{\mathcal{G},i}(p, q)$, is the temporal distance from p to q in $\mathcal{G}_{i\triangleright}$, *i.e.*, $\hat{d}_{\mathcal{G}_{i\triangleright}}(p, q)$. Similarly, the *temporal diameter* in $\mathcal{G}$ at position $i \in \mathbb{N}^*$ is the maximum temporal distance at position i between any two vertices in $\mathcal{G}$.

For all previous notations, we omit the subscript $\mathcal{G}$ when it is clear from the context.

TVG Classes. A class of dynamic graphs is defined as a particular set of dynamic graphs.

Class $\mathcal{TC}^{\mathcal{R}}$ *(Recurrent Connectivity), denoted by* $\mathcal{C}_5$ *in [1]:* at any point in time, every vertex can reach all the others through a journey. Formally, $\mathcal{G} \in \mathcal{TC}^{\mathcal{R}}$ iff

$$\forall p \in V, \forall q \in V, \forall i \in \mathbb{N}^*, p \overset{\mathcal{G}_{i\triangleright}}{\rightsquigarrow} q.$$

Class $\mathcal{TC}^{\mathcal{B}}(\Delta)$ *(Bounded Temporal Diameter), denoted by* $\mathcal{TC}(\Delta)$ *in [12]:* every vertex is always at temporal distance at most Δ from all other vertices. Formally, $\mathcal{G} \in \mathcal{TC}^{\mathcal{B}}(\Delta)$ iff $\forall p \in V, \forall q \in V, \forall i \in \mathbb{N}^*, \hat{d}_{\mathcal{G},i}(p, q) \leq \Delta$.

IV. Computational Model

We consider the computational model defined in [35], [34]. We assume a *distributed system* composed of a set V of N *processes*. Each process has a local memory, a local sequential and deterministic algorithm, and message exchange capabilities. We assume that each process p has a unique identifier (ID for short), denoted $id(p)$ and taken from an arbitrary domain $IDSET$ totally ordered by $<$. Processes communicate by message passing through an interconnected network that evolves over the time. The dynamic topology of the network is modeled by a dynamic graph $\mathcal{G} = G_1, G_2, ...$ with vertex set V, *i.e.*, the set of processes. Processes execute their local algorithms in *synchronous rounds*. For every $i \in \mathbb{N}^*$, the communication network at *Round* i is defined by G_i, *i.e.*, the graph at position i in $\mathcal{G}$. $\forall p \in V$, $\forall i \in \mathbb{N}^*$, we denote by $\mathcal{IN}(p)^i = \{q \in V : (q, p) \in A(G_i)\}$ the set of p's neighbors at Round i. $\mathcal{IN}(p)^i$ is assumed to be unknown by p, whatever be the value of i. A process q belongs to $\mathcal{IN}(p)^i$ if the message q sent in round i is received by p during the round i, *i.e.*, the communication channel from q to p exists and the message from q is not lost.

A *distributed algorithm* $\mathcal{A}$ is a collection of N local algorithms $\mathcal{A}(p)$, one per process $p \in V$ (*n.b.*, different processes may have different codes). At each round, the *state* of each process $p \in V$ in $\mathcal{A}$ is defined by the values of its variables in $\mathcal{A}(p)$. Some variables may be constant in which case their values are predefined and their value cannot be corrupted. In the presented algorithms, process identifier and the number of processes are constant variables.

A *configuration* of $\mathcal{A}$ for V is a vector of N components $\gamma = (s_1, s_2, \ldots, s_N)$, where s_1 to s_N represent the states of the processes in V. Let γ_1 be the initial configuration of $\mathcal{A}$ for V. For any (synchronous) round $i \geq 1$, the system moves from the current configuration γ_i to some configuration γ_{i+1}, where γ_i (resp., γ_{i+1}) is the configuration *at the beginning* (resp., *the end*) of *Round* i. Such a configuration change is atomically performed by every process $p \in V$ according to the following three steps, defined in its local algorithm $\mathcal{A}(p)$:

1) p sends a message consisting of all or a part of its local state in γ_i by calling the primitive `SEND()`. The messages

of others processes in transit at p are also sent by calling the primitive SEND() on them.
2) using Primitive RECEIVE(), p receives all messages sent by processes in $\mathcal{IN}(p)^i$;
3) p computes its state in γ_{i+1}.

An *execution* of a distributed algorithm $\mathcal{A}$ in a dynamic graph $\mathcal{G} = G_1, G_2, ...$ is an infinite sequence of configurations $\gamma_1, \gamma_2, \ldots$ of $\mathcal{A}$ for V such that $\forall i > 0$, γ_{i+1} is obtained by executing a synchronous round of $\mathcal{A}$ on γ_i based on the communication network at Round i, *i.e.*, the graph G_i.

V. FIFO Broadcast with Termination Detection

Algorithm 2 provides FIFO Broadcast primitives with termination detection for Class $\mathcal{TC}^{\mathcal{R}}$ and bounded message size, assuming that the number of processes N of the system is known. Variables and macros are defined in Algorithm 1. We should point out that broadcasting with detection of termination cannot be solved in dynamic networks without additional knowledge (e.g., the number of processes) [11].

The algorithm is based on rounds. At every round, a process s either broadcasts an application message, if the latter exists, or an empty one ($\perp$). In the first case, the message is delivered by the N processes within the current round. A bounded sequence number, denoted label, is assigned to every broadcast message.

As the network is dynamic, every process s keeps the current broadcast message as well as the most recent ones it received from each other process of the system in a local buffer. Every message also contains a N size timestamp vector: if the message was sent by s, the entry corresponding to s keeps the label of the last message it broadcast, otherwise ($r \neq s$) the value of this entry corresponds to the label of the last message broadcast from s received by r, *i.e.*, an acknowledgement from r for the reception of s'message. The local buffer of a process is updated upon reception of a message, keeping, for each other process, only the most recent one. Process s continually sends the messages of its local buffer. The s'current broadcast terminates only after receiving from every other process r a message whose timestamp vector entry related to s acknowledges the reception of the message in question. Then, s starts its next broadcast: either an application message or ($\perp$).

Labels range from the finite set $\{0, 1, 2\}$. Only these three label values are sufficient to distinguish different messages in transit broadcast by a process. In other words, if, at each broadcast of a new message, process s updates the label assigned to the message to $(i + 1)\ mod\ 3$, our algorithm ensures that there will never exist two messages in transit whose timestamp vector entry related to s is equal to i, corresponding to two different messages broadcast by s. In fact, during the $i + 3^{th}$ broadcast by s, all messages from $r \neq s$ in transit were sent at or after the reception by r of the $i + 1^{th}$ message broadcast by s, *i.e.*, there is no message in transit (in a process local buffer) from r which acknowledged a previous s'broadcast message with label i. Consequently, tagging broadcast messages with labels within $\{0, 1, 2\}$ does not induce ambiguity to distinguish message acknowledgments.

A. Description of the algorithm

Algorithm 1: variables and macro of FIFO Broadcast with detection of termination in $\mathcal{TC}^{\mathcal{R}}$ for process p.

Constant Variables:
$N \in \mathbb{N}^*$: number of processes in the DG
$id(p) \in IDSET$: the identifier of p

Local Variables:
- $queueBrdcst(p)$: queue with the messages that p has to broadcast; initially, the queue is empty.
- $ack(p)$: set of processes having received the current broadcast message, initially the set is $\{id(p)\}$.
- $msgB(p)$: message being forwarded, initially, its value is $brdcst\perp$.
- $myCnt(p)$: count the number of timestamp updates during the current broadcast, initially its value is 0.
- $myTS(p)[]$: $myTS$(p)[q] keeps the timestamp of the current or last message broadcast by q and received by p; initially, $myTS(p)[p] = 1$ and $myTS(p)[q] = 0$ if $q \neq p$
- $MsgsT(p)$: set that keeps messages in transit in the format $\langle id, m, updtCnt, TS\rangle \in IDSET \times msgType \times \mathbb{N} \times \{0, 1, 2\}^N$. Initially, the set contains the single message $\langle id(p), msgB(p), myCnt(p), myTS(p)\rangle$.

Macros and Predicates:
- $i \oplus j : (i + j)$ mod 3
- $i^{\oplus} : i$ mod 3
- $(ts1, cpt1) >_{\oplus} (ts2, cpt2)$ iff $(ts1 = ts2 \oplus 1)$ or $((ts1 = ts2)$ and $(cpt1 > cpt2))$

Whenever the application layer of p invokes the procedure **FD_brcst** in order to broadcast a message, the latter is enqueued in $queueBrdcst(p)$. At the end of the current p' broadcast, the algorithm broadcasts the first message of $queueBrdcst$(p) (if-block starting in line 21).

At the beginning of every round, by calling the $SEND$ primitive, p sends all the messages in $MsgsT(p)$ (line 4). p will insert its current message in $MsgsT(p)$ at the end of the round (line 26). This message will then be sent at the beginning of the next round. The content of the message is composed by the current values of p's local variables $msgB$, $myCnt$, and $myTS$ (see local variables description of Algorithm 1). Hence, a message $msg = \langle id(p), m, updtCnt, TS\rangle$ has the following four fields:

- $id(p)$: the identifier of the sender;
- m : the data of the message (application data or $brdcst\perp$). If $m = brdcst\perp$, m is not delivered. In this case, msg contains only acknowledgments;
- $updtCnt$ is the current value of the timestamp update counter;
- TS is a vector of size N that contains both the acknowledgments and the label of the msg. Precisely, the label of $msg = \langle id(p), m, updtCnt, TS\rangle$ sent by p is $TS[p]$, while for $q \neq p$, $TS[q]$ is the label of message sent by q, acknowledged by p.

Before inserting in $MsgsT(p)$ a new message broadcast by q, the current q's message kept in $MsgsT(p)$ is removed from it (lines 9-11 if $q \neq p$, otherwise lines 25-26). Thus, $MsgsT(p)$ contains at most a single message of q at any configuration.

Algorithm 2: Code of FIFO Broadcast with detection of termination in $\mathcal{TC}^{\mathcal{R}}$ for process p.

```
1  Procedure FD_brcst(m)
2      queueBrdcst(p).enqueue(m)

3  Repeat Forever
4      SEND(msg = ⟨id, m, updtCnt, TS⟩ ∈ MsgsT(p))
5      mailbox := RECEIVE()
       // processing of receiving messages of q ≠ p
6      forall tuples msg = ⟨id(q), m, updtCnt, TS⟩ in mailbox do
7          if id(q) ≠ id(p) then
               // keep the latest q-message
8              if msg' = ⟨id(q), −, updtCnt', TS'⟩ ∉ MsgsT(p)
                 or (TS[q], updtCnt) >⊕ (TS'[q], updtCnt') then
9                  if msg' ∈ MsgsT(p) then
10                     remove msg' of MsgsT(p)
11                 insert ⟨id(q), m, updtCnt, TS⟩ in MsgsT(p)
                   // delivery of q-acknowledgment
12                 if TS[p] = myTS(p)[p] then
13                     insert id(q) in ack(p)
                   // new broadcasting from q
14                 if TS[q] = myTS(p)[q] ⊕ 1 then
15                     myTS(p)[q] := myTS(p)[q] ⊕ 1
16                     myCnt(p) + +
17                     if m ≠ brdcst⊥ then FD_dlr(id(q), m)
       // start a new broadcast
18     if |ack(p)| = N then
19         ack(p) := {id(p)} ; myCnt(p) := 0;
20         myTS(p)[p] := myTS(p)[p] ⊕ 1
           // determine the information broadcast
21         if queueBrdcst(p) is not empty then
22             msgB(p) := queueBrdcst(p).dequeue ;
23             if msgB(p) ≠ brdcst⊥ then
                   FD_dlr(id(p), msgB(p))
24         else msgB(p) := brdcst⊥
       // add/update the p's message
25     remove ⟨id(p), −, −, −⟩ of MsgsT(p)
26     insert ⟨id(p), msgB(p), myCnt(p), myTS(p)⟩ in MsgsT(p)
```

The ith broadcast of a message issued by process s, named $m(s,i)$, takes several rounds which comprise the delay to carry the data kept in $msgB(s)$ to every process plus the delay taken for the reception by p of the acknowledgments regarding $m(s,i)$ from the other processes. Process s starts its $i+1$th broadcast during the round in which its ith broadcast has finished. In this case, s has received an acknowledgment from every other process (*i.e.*, $|ack(s)| = N$ - if-block starting at line 18 -). This round is named $t_{s,i+1}$.

If the application has called the **FD_brcst** primitive, at least $i+1$ times before the beginning of the $i+1$th broadcast by s (*i.e.*, $queueBrdcst(s)$ is not empty) then $m(s,i+1) = msgB(s)$ is a message dequeued from $queueBrdcst(s)$. In this case s delivers $m(s,i+1)$ by calling $\mathbf{FD\_dlr}(id(s), m(s,i+1))$ (if-line 23). Otherwise (*i.e.*, $queueBrdcst(s)$ is empty), $m(s,i+1)$ is set to $brdcst\perp$ (if-else-block starting at line 21).

In both cases, the set $ack(s)$ contains just s, *i.e.*, the only process having delivered $m(s,i+1)$ ($ack(s) = \{s\}$ - line 19). Then, $myTS(s)[s]$ value is updated to $(i+1)^{\oplus}$ (line 20) which will be the label of the messages sent by s during the $i+1$th broadcast.

Let r be a process distinct of s. $myTS(r)[s]$ contains the label of the last s's broadcast data m, received by r. At the beginning of the ith broadcast of s this value is $i \oplus 2 = i - 1 \bmod 3$. When r receives for the first time s's message sent during its ith broadcast, $myTS(r)[s]$ is set to $myTS(r)[s] \oplus 1 = msg.TS[s] = i^{\oplus} = i \bmod 3$. This round is called $t_{s,i}(r)$. Hence, r will verify the if-condition of line 14 and, if $m(s,i)$ is an application data, r will deliver it by calling $\mathbf{FD\_dlr}(id(s), m(s,i))$ (line 17). Then, $myTS(r)[s]$ will be updated to the value $i^{\oplus}$ (line 15) and, therefore, the next messages sent by r, which will have the label $msg.TS[s] = myTS(r)[s] = i^{\oplus}$, acknowledge the reception of $m(s,i)$.

When s receives a message of r acknowledging the reception of $m(s,i)$ (*i.e.*, $TS[r] = i^{\oplus}$), r is inserted in the set $ack(s)$ since, in this case, r has received $m(s,i)$ (line 13).

During the ith broadcast, s does not send at each round the same message. Nevertheless, their field m has the same value $m(s,i) = msgB(s)$ or $m(s,i) = brdcst\perp$ as well as the label value $TS[s] = i^{\oplus} = i \bmod 3$. Both of them are only changed during the execution of the if-block starting at line 18 (*i.e.*, during the round $t_{s,i+1}$, ending the ith broadcast). It is worth pointing out that messages of s sent during two consecutive broadcasts have distinct labels whose values are taken from the finite set $\{0,1,2\}$.

On the other hand, during the ith broadcast of s, the message sent by s changes whenever the acknowledgment array $myTS$ is updated (line block 15-17). At every update, s increments $myCnt$ value (line 16). It is reinitialized to 0 at the beginning of each broadcast (lines 19-24). The label and $updtCnt$ values of messages are used by a process $r \neq s$ to choose which messages of s to keep in its set $MsgsT(r)$ - (if-condition at line 8). In section VI, we prove that a process r keeps in transit, i.e., in $MsgsT(r)$, only the most recent message of s (Lemma 4).

VI. Proof of correctness of FIFO Broadcast primitives in Class $\mathcal{TC}^{\mathcal{R}}$

A. Predicate $WF\_Brdcst(s,i)$

In a configuration γ, a message msg is in transit if it exists a process q, such that $msg \in MsgsT(q)$.

We denote $t_{s,i}$ the round where s starts its ith broadcast.

The verification of predicate $WF\_Brdcst(s,i)$ in the configuration reached after $t_{s,i}$ (*i.e.*, $\gamma_{s,i}$) allows to establish that every process delivers once $m(s,i)$ during the time interval $[t_{s,i}, t_{s,i+1}[$ if $m(s,i) \neq brdcst\perp$, otherwise no s-message is delivered (Theorem 1). $WF\_Brdcst(s,1)$ is verified in the configuration $\gamma_{s,1}$. We will prove that the predicate $WF\_Brdcst(s,i)$ is verified in the configuration $\gamma_{s,i}$ (Lemma 6), for all $i \in \mathbb{N}^*$.

Definition 1 (predicate $WF\_Brdcst(s,i)$): $t_{s,1}$ is 0, a virtual round reaching the configuration γ_1. For $i > 1$, $t_{s,i}$ denotes the round where s executes the line block 19-24 for the i-1th time.

$\gamma_{s,i}$ is the configuration reached after the round $t_{s,i}$.

$m(s,i)$ is the value of $msgB(s)$ in $\gamma_{s,i}$.

The predicate $WF\_Brdcst(s,i)$ is defined as :

a $myTS(s)[s] = i^{\oplus}$ and $ack(s) = \{s\}$ and $myCnt(s) = 0$

b $\langle id(s), msgB(s), myCnt(s), myTS(s)\rangle \in MsgsT(s)$

c $\forall r \in V/\{s\}$, if $\langle id(s), -, -, TS\rangle \in MsgsT(r)$ then $TS[s] = i \oplus 2$

d $\forall r \in V/\{s\}$, $myTS(r)[s] = i \oplus 2$

e $\forall r \in V$, if $\langle id, -, -, TS\rangle \in MsgsT(r)$ then $id = id(s)$ or $TS[s] \neq i^{\oplus}$

Predicate $WF\_Brdcst(s,i).a$ ensures that the local s's variables have a correct value. Predicates $WF\_Brdcst(s,i).b$ and $WF\_Brdcst(s,i).c$ ensure that any s-messages in transit are not old (*i.e.*, their label is $i \oplus 2$ except the one that has the label $i^{\oplus}$, this one carries $m(s,i)$). Predicate $WF\_Brdcst(s,i).d$ ensures that any process $r \neq s$ is ready to deliver a s-message with label $i^{\oplus}$. Let q be a process distinct of s. Predicate $WF\_Brdcst(s,i).e$ ensures that the q-messages in transit are not too old - their s-acknowledgment (*i.e.*, $TS[s]$ value) is not $i^{\oplus}$ -. This last property guarantees that during $[t_{s,i}, t_{s,i+1}[$, the set $ack(s)$ contains only process that have received and handled $m(s,i)$. Thus, the broadcast ends after every process has actually received $m(s,i)$.

In Section VI-B, we prove that, if $m(s,i) \neq brdcst\perp$, and the predicate $WF\_Brdcst(s,i)$ is verified in $\gamma_{s,i}$ then at the end of ith broadcast of s, $m(s,i)$ is delivered by every process. In the Section VI-C, we establish that the ith broadcast of s terminates and also that $WF\_Brdcst(s,i)$ is verified in $\gamma_{s,i}$ for all i values.

We conclude that the broadcast primitives of Algorithm 2 ensure FIFO broadcasts with detection of termination (corollary 3). If the dynamic graph belongs to the Class $\mathcal{TC}^{\mathcal{B}}(\Delta)$, a broadcast requires 2Δ rounds.

In section VI-D, we establish that the value of $updtCnt$ field of any message is bounded by $2N$. Hence a message requires at most $2N + O(\log(N)) + msgSize$ bits with $msgSize$ being the bound of the data size to broadcast (application data or $brdcst\perp$).

B. Proof of the Algorithm Safety

Lemma 1: Let assume that in the configuration $\gamma_{s,i}$, $WF\_Brdcst(s,i)$ is verified. In the configurations of $[\gamma_{s,i}, \gamma_{s,i+1}[$, every s-message in transit labeled $i^{\oplus}$ contains $m(s,i)$. No s-message in transit has the label $i \oplus 1$ in the configurations of $[\gamma_{s,i}, \gamma_{s,i+1}[$.

Proof: The verification of $WF\_Brdcst(s,i).b$ and of $WF\_Brdcst(s,i).c$ in the configuration $\gamma_{s,i}$ ensure that (1) a single s-message having the label $i^{\oplus}$ is in transit and the data carried by the message is $m(s,i)$; (2) no s-message in transit has the label $i \oplus 1$. A s-message having the label $i \oplus 1$ will be in transit, only after the execution of the if-block of line 18, which happens within $t_{s,i+1}$ round. ■

The mailbox of p contains the message msg during the round t if and only if $\exists q \in \mathcal{IN}(p)^t$ such that $msg \in MsgsT(q)$ in the configuration γ_t.

Let r be a process of $V/\{s\}$. We denote $t_{s,i}(r)$, the first round after $t_{s,i}$ when the r mailbox contains a s-message $msg = \langle id, m, -, TS\rangle$ with label $i^{\oplus}$. We denote $\gamma_{s,i}(r)$ the configuration reached after the round $t_{s,i}(r)$.

Lemma 2: Let assume that in the configuration $\gamma_{s,i}$, $WF\_Brdcst(s,i)$ is verified. In the configurations of $[\gamma_{s,i}, \gamma_{s,i}(r)[$, a s-message in $MsgsT(r)$ has label $i \oplus 2$ and $myTS(r)[s] = i \oplus 2$.
During $t_{s,i}(r)$, r executes the if-block of line 14 when r handles a s-message.
In the configurations of $[\gamma_{s,i}(r), \gamma_{s,i+1}[$, a s-message in $MsgsT(r)$ has thus label $i^{\oplus}$ and $myTS(r)[s] = i^{\oplus}$.

Proof: In the configuration $\gamma_{s,i}$, $myTS(r)[s] = i \oplus 2$ as $WF\_Brdcst(s,i).d$ is verified. During the rounds of time interval $[t_{s,i} + 1, t_{s,i+1}(r)[$, any s-message in r's mailbox has the label $i \oplus 2$ (Lemma 1).

Thus, during $[t_{s,i} + 1, t_{s,i+1}(r)[$, r handles only s-message labeled by $i \oplus 2$. So, r does not execute the if-block of line 14 when handling s-messages. Therefore, in the configurations of $[\gamma_{s,i}, \gamma_{s,i+1}(r)[$, $myTS(r)[s] = i \oplus 2$ and if r has in transit a s-message then it is labeled with $i \oplus 2$.

During $t_{s,i}(r)$, r executes the if-block of line 14 to handle msg, delivering $(id(s), m)$, if $m \neq brdcst\perp$. Then, $myTS(r)[s] = i^{\oplus}$ and it keeps such a value until the r's mailbox contains a s-message labeled $i \oplus 1$. According to Lemma 1, during $[t_{s,i}(r), t_{s,i+1}[$, r's mailbox does not contain any s-message labeled $i \oplus 1$. Consequently, $myTS(r)[s] = i^{\oplus}$ and r handles only s-message with label $i^{\oplus}$. ■

Observation 1: Let $i \in \mathbb{N}^*$. Let also assume that in the configuration $\gamma_{s,i}$, $WF\_Brdcst(s,i)$ is verified. In the configurations of $[\gamma_{s,i}, \gamma_{s,i+1}[$, $myTS(s)[s] = i^{\oplus}$ and $\langle id(s), m(s,i), -, -\rangle$ labeled by $i^{\oplus}$ belongs to $MsgsT(s)$.

Definition 2 (round $t''_{s,i}(r)$): Let r be a process of $V/\{s\}$.
We denote $t''_{s,i}(r)$, the first round after $t_{s,i}$ where s executes the if-block in line 12 to insert $id(r)$ in $ack(s)$.

Lemma 3: We have $t_{s,i} < t_{s,i}(r) < t''_{s,i}(r) \leq t_{s,i+1}$ in $\gamma_{s,i}$, if $WF\_Brdcst(s,i)$ is verified.

Proof: A message $\langle id, -, -, TS\rangle$ acknowledges $m(s,i)$ iff $TS(s)$ has the value $i^{\oplus}$.

Since $WF\_Brdcst(s,i).d$ and $WF\_Brdcst(s,i).e$ are both verified in $\gamma_{s,i}$, there is no r-message in transit in the configurations of $[\gamma_{s,i}, \gamma_{s,i+1}(r)[$ acknowledging $m(s,i)$, as $TS(r)[s] = i \oplus 2$.

We denote $t'_{s,i}(r)$, the first round after $t_{s,i}$ where the s mailbox contains a r-message acknowledging $m(s,i)$. The s mailbox contains a r-message acknowledging $m(s,i)$ in the tth round, only if there exists a r-message acknowledging $m(s,i)$ in transit in the configuration γ_t. We have $t'_{s,i}(r) > t_{s,i}(r)$.

We have $t''_{s,i}(r) \geq t'_{s,i}(r) > t_{s,i}(r)$, as the s mailbox contains a r-message acknowledging $m(s,i)$ during the round $t''_{s,i}(r)$. The verification of $WF\_Brdcst(p,i).e$ in the configuration $\gamma_{s,i}$ ensures that $|ack(s)| \neq N$ during $[t_{s,i}+1, t''_{s,i}(r)[$. Thus, $t''_{s,i}(r) \leq t_{s,i+1}$. We conclude that $t_{s,i} < t_{s,i}(r) < t_{s,i+1}$ as by definition $t_{s,i} < t_{s,i}(r)$. ■

Definition 3 (configuration $\gamma^+_{s,i}$): Let $\gamma^+_{s,i}$ be the configuration before the round $t_{s,i+1}$.

Observation 1, Lemma 1, Lemma 2 and Lemma 3 allows us to establish the following corollary.

Corollary 1: Let $i \in \mathbb{N}^*$ and let r and s be two processes. Assuming that in the configuration $\gamma_{s,i}$, $WF\_Brdcst(s,i)$ is verified. In the configuration $\gamma^+_{s,i}$, $myTS(r)[s] = i^{\oplus}$ and if r has in transit a s-message then this message is labeled by $i^{\oplus}$ and it contains $m(s,i)$. In the configuration $\gamma_{s,i+1}$, $WF\_Brdcst(s,i+1).a$, $WF\_Brdcst(s,i+1).b$, $WF\_Brdcst(s,i+1).c$, and $WF\_Brdcst(s,i+1).d$ are verified.

Observation 2: During $[\gamma_1, \gamma^+_{s,1}]$, every message in transit verifies $TS[s] \neq 2$. $WF\_Brdcst(s,2).e$ is verified in $\gamma_{s,2}$. We have $t_{s,i+1} = max_{r \in V/\{s\}}\{t''_{s,i}(r)\}$.

Theorem 1: Let assume that in the configuration $\gamma_{s,i}$, $WF\_Brdcst(s,i)$ is verified. During the time interval $[t_{s,i}, t_{s,i+1}[$, at most a single s-message is delivered by any process. If $m(s,i) \neq brdcst\bot$, then $(id(s), m(s,i))$ is delivered by every process otherwise no process delivers a s-message during the time interval $[t_{s,i}, t_{s,i+1}[$.

Proof: Process s delivers a s-message only during the execution of the if-block of lines 18-24, so only during the round $t_{s,i}$. According to if-block in line 23, No message is delivered if $m(s,i) = brdcst\bot$. If $m(s,i) \neq brdcst\bot$ then $i > 1$, and s executes the if-else-block of line 21 during the round $t_{s,i}$; hence s delivers $(id(s), m(s,i))$.

Let $r \neq s$ be a process. During a round where r delivers a s-message, $myTS(r)[s]$ changes its value (if-block of lines 15-17). If $i > 1$ then r does not execute the line block 9-17 during $t_{s,i}$ according to $\gamma^+_{s,i-1}$ properties (Corollary 1) and $t_{s,i}$ properties (Lemma 1). During $t_{s,1} = 0$, no message is delivered. Lemma 2 establishes that $myTS(r)[s]$ changes its value only once during $[t_{s,i}+1, t_{s,i+1}[$, i.e., at round $t_{s,i}(r)$. In the round $t_{s,i}(r)$, the data of the handled s-message is $m(s,i)$ (Lemma 1). If $m(s,i) \neq brdcst\bot$, $(id(s), m(s,i))$ is delivered (see if-line 17) otherwise no s-message is not delivered. ■

C. Proof of the Algorithm Liveness

The ith broadcast of s is terminated, if for any process $r \neq s, t''_{s,i}(r)$ exists, and the configuration $\gamma_{s,i}$ verifies $WF\_Brdcst(s,i)$ (Observation 2). In this Section, we state that any broadcast of a process s terminates.

The timestamp of a s-message $\langle id(s), -, updtCnt, TS\rangle$ is the tuple $(TS[s], updtCnt)$. We note $msg.ts$ the timestamp of message msg.
Let $msg1$ and $msg2$ be two messages sent by s. The comparison $msg1.ts >_{\oplus} msg.ts$ is noted $msg1 >_{\oplus} msg2$.

We denote $msg(r,s,\gamma)$ the s-message in $MsgsT(r)$ in the configuration γ, if such a message exists. We denote $mb(r,t)$ the mailbox of r during round t. $msg \in mb(r,t)$ if and only if $\exists q \in \mathcal{IN}(r)^t$ such that $msg \in MsgsT(q)$ in the configuration before the round t. The set $mb(r,t) \cup \{\forall s \in V, msg(r,s,\gamma_-)\}$ is denoted We denote $mb^+(r,t)$.

The relation $>_{\oplus}$ is used by process r to select which s-message to keep in transit, i.e., during the round t every s-messages of $mb^+(r,t)$ are discarded except one - if-statement in line 8.

Let $msg1$ the s-message sent in round t and $msg2$ the s-message sent in round $t+1$. The algorithm code ensures that $msg2 >_{\oplus} msg1$ or $msg1 = msg2$. However, the relation $>_{\oplus}$ is not transitive. Hence, it is necessary to establish the conditions that allow processes to apply the $>_{\oplus}$ relation in order to compare the sending rounds of two s-messages.

By lack of space, the proof of the following lemma is not presented.

Lemma 4: Let $r \neq s$ be a process. Let γ_- be a configuration in $[\gamma_{s,i}, \gamma^+_{s,i}])$ where $WF\_Brdcst(p,i)$ is verified in $\gamma_{s,i}$. Let t be the round from γ_- reaching γ_+. If $mb^+(r,t)$ contains a s-message then $msg(r,s,\gamma_+)$ is the latest s-message in $mb^+(r,t)$ otherwise $msg(r,s,\gamma_+)$ does not exist.

Even reusing message labels and timestamps, processes may order s-messages in transit according to their sending round during the time interval $[t_{s,i}+1, t_{s,i+1}]$ if $WF\_Brdcst(p,i)$ is verified in $\gamma_{s,i}$. Thus, every process r, distinct from s, keeps in transit among all received s-messages the most recent one during $[t_{s,i}+1, t_{s,i+1}]$ (Lemma 4).

Definition 4 (well-formed configuration): A configuration γ is well-formed if for each process p, it exists $i_p \in \mathbb{N}^*$ such that $\gamma \in [\gamma_{p,i_p}, \gamma_{p,i_p+1}[$ and $WF\_Brdcst(p,i_p)$ is verified in γ_{p,i_p}.
A round of a well-formed configuration is said well-formed.
We name γ_{bad} (resp. t_{bad}) the first configuration (resp. round) that is not well-formed.

According to the definition of well-formed configuration, it exists pb and ib such that $\gamma_{bad} = \gamma_{pb,ib}$ and $WF\_Brdcst(pb,ib)$ is not verified in $\gamma_{pb,ib}$. Moreover, according to Observation 2, $ib > 2$.

By lack of space, the proof of the following lemma is not presented.

Lemma 5: Let p and q be two distinct processes. We name $\gamma_{a+} \leq \gamma_{bad}$ the configuration after the round ta. The message $msg(p,q,\gamma_{a+})$ is sent after td if and only if there exists a journey from q to p whose the departure is after td and its arrival is before or at ta.

Lemma 6: All configurations are well-formed.

Proof: According to the definition of well-formed configuration, it exists pb and ib such that $\gamma_{bad} = \gamma_{pb,ib}$ and $WF\_Brdcst(pb,ib)$ is not verified in $\gamma_{pb,ib}$. More precisely, $WF\_Brdcst(pb,ib).e$ is not verified in $\gamma_{pb,ib}$ (Corollary 1). According to Observation 2, we have $ib > 2$.

Let u and v be two processes with $u \neq pb$. In the following, we prove that $msg(v, u, \gamma_{pb,ib})$ is sent at $t \in \ ]t_{pb,ib-2}(u), t_{pb,ib}]$. Let γ be a configuration of $[\gamma_{pb,ib-2}, \gamma_{pb,ib-1}[$.
Among the u-messages $msg = \langle id(u), -, -, TS\rangle$ in transit in γ, only the messages sent during $]t_{pb,ib-2}(u), t_{pb,i-1}]$ verify $TS[pb] = (ib-2)^{\oplus}$. Therefore, the mailbox of pb, during the round $t''_{pb,i-2}(u)$ contains a u-message sent after $t_{pb,ib-2}(u)$. Thus, there exists a journey from u to pb whose departure is after $t_{pb,ib-2}(u)$ and its arrival is at or before $t''_{pb,ib-2}(u) \leq t_{pb,ib-1}$ according to Lemma 5. If $v \neq pb$ then the message $msg(v, pb, \gamma_{pb,ib-1}(v))$ is sent after $t_{pb,ib-1}$ (Lemma 2). According to Lemma 5, there exists a journey from pb to $v \neq pb$ whose the departure is after $t_{pb,ib-1}$ and its arrival is at or before $t_{pb,ib-1}(v)$. Hence, there is a journey from u to v whose departure is after $t_{pb,ib-2}(u)$ and arrival is at or before $t_{pb,ib-1}(v) \leq t_{pb,ib}$ if $v \neq pb$, otherwise its arrival is at or before $t_{pb,ib-1}$.

Lemma 5 allows us to conclude that $msg(v, u, \gamma_{pb,ib})$ is sent at $t \in \ ]t_{pb,ib-2}(u), t_{pb,ib}]$. As t is well-formed, Lemma 2 establishes that message $msg(v, u, \gamma_{pb,ib}) = \langle id(u), -, -, TS\rangle$ verifies $TS[pb] \in \{(ib-2)^{\oplus}, (ib-1)^{\oplus}\}$. So $WF\_Brdcst(pb, ib).e$ is verified in $\gamma_{pb,ib}$ (*i.e.*, $TS[pb] \neq ib^{\oplus}\ \forall u \in V/\{pb\}$ and $\forall v \in V$). We conclude that γ_{bad} does not exist. ■

The following corollary is a consequence of Lemma 6 and Lemma 5.

Corollary 2: Let s and r be two distinct processes. There exists a journey from s to r whose the departure is after $t_{s,i}$ and arrival is t if and only if $t_{s,i}(r) \leq t$.

There exists a journey from r to s whose the departure is after $t_{s,i}(r)$ and its arrival is t if and only if $t''_{s,i}(r) \leq t$.

The two first statements (*i.e.*, liveness) of the corollary that follows is a consequence of the Observation 2, Corollary 2 while the third statement (*i.e.*, safety) is a consequence of the Theorem 1, and Lemma 6 :

Corollary 3: Let s be a process of V and $i \in \mathbb{N}^*$.

- In a graph of the class $\mathcal{TC}^{\mathcal{R}}$ with the process set V, $\gamma_{s,i}$ exists.
- In a graph of the class $\mathcal{TC}^{\mathcal{B}}(\Delta)$ with the process set V, $t_{s,i+1} \leq t_{s,i} + 2\Delta$.
- $(id(s), m(s,i))$ is delivered once by every process during the time interval $[t_{s,i}, t_{s,i+1}[$ if $m(s,i) \neq brdcst\perp$. Otherwise no process delivers a s-message during this time interval.

D. Proof of the bounded message size

Theorem 2: For all $s \in V$ and for $i \in \mathbb{N}^*$, in the configurations of $[\gamma_{s,i}, \gamma_{s,i+1}[$, the value of $myCnt(s)$ is bounded by $2N$.

Proof: For all $i \in Positive$, in $\gamma_{s,i}$, $myCnt(s) = 0$ (as s executes the line 19 during $t_{s,i}$).

During rounds of $[t_{s,i}+1, t_{s,i+1}[$, the value of $myCnt(s)$ is only incremented (line 16) when $myTS(s)$ is updated. The value of $myCnt(s)$ is bounded by $2N$ in the configurations of $[\gamma_{s,i}+1, \gamma_{s,i+1}[$, if s does not update 3 times the acknowledgment value associated to a process r distinct of s (*i.e.*, $myTS(s)[r]$).

In the sequel, we assume that s updates 3 times $myTS(s)[r]$ during $[t_{s,i}+1, t_{s,i+1}[$. Theorem 1 establishes that $myTS(s)[r]$ changes of value only one time during $[t_{r,j}+1, t_{r,j+1}[$: at round $t_{r,j}(s)$. We conclude that it exists $j \in \mathbb{N}^*$ such that $t_{s,i} < t_{r,j}(s) < t_{r,j+1}(s) < t_{r,j+2}(s) < t_{s,i+1}$.

Let p be a process distinct of s. There is a journey from s to r starting after $t_{r,j}(s)$ and arriving at or before $t_{r,j+1}$ (Corollary 2). If $p \neq r$, there is a journey from r to p starting after $t_{r,j+1}$ and arriving at or before $t_{r,j+1}(p)$. If $p \neq r$ then $t_{s,i}(p) \leq t_{r,j+1}(p)$, otherwise $t_{s,i}(r) \leq t_{r,j+1}$. If $p \neq r$, there is a journey from p to r starting after $t_{s,i}(p) \leq t_{r,j+1}(p)$ and arriving at or before $t_{r,j+2}$. There is a journey from r to s starting after $t_{r,j+2}$ and arriving at or before $t_{r,j+2}(s)$. So $t''_{s,i}(p) \leq t_{r,j+2}(s)$ according to Corollary 2 for every process p in V. After the processing of messages in s' mailbox during the round $t_{r,j+2}(s)$, we have $receivers(s) = V$, so $t_{r,j+2}(s) = t_{s,i+1}$ (Observation 2).

We conclude that in the configurations of $[\gamma_{s,i}, \gamma_{s,i+1}[$, the value of $myCnt(s)$ is bounded by $2N$. ■

Corollary 4: In any message in transit in any configuration, the value of the field $updtCnt$ is bounded by $2N$.

A label has 3 values, so TS is an array of N entries, each of them of 2-bits size.

The size of a process identifier is $c.log(N)$ bits.

A message size is at most $2N + O(log(N)) + msgSize$ bits.

VII. Causal Total Order Broadcast

Algorithm 3 provides Causal Total Order Broadcast primitives (also called Atomic broadcast primitives) in Class $\mathcal{TC}^{\mathcal{R}}$ assuming that N is known. Algorithm 3 requires a FIFO Broadcast with termination detection algorithm implementation as the one of Algorithm 2 : a process starts a new atomic broadcast after detecting the termination of the previous one, i.e., every process has delivered the previous broadcast message.

Algorithm 3 design is a new variant of communication history schema to establish the delivering order [3]. The processes delay the messages delivery according to a deterministic policy regarding the merging the message streams from each process. More precisely, a process p waits to have received at least a broadcast message from every process to handle a single received message per process (the first received one) - *i.e.* these messages are delivered according to the sender identifier order. To avoid deadlock, each process permanently performs a atomic broadcast. When a process has not data to atomically broadcast, it broadcasts the neutral data (*i.e.* $atomic\perp$) that is not delivered to the application layer.

Algorithm 3 implements the procedure $Atomic\_Broadcast(m)$ by calling the FIFO Broadcast procedure **FD_brcst** i.e. the line 2 is executed. Whenever process p receives a message m broadcast by q, it calls the FIFO Broadcast procedure $\mathbf{FD\_dlr}(id(q), m)$; i.e. p executes line block 4-9.

Algorithm 3: Causal Total Order Broadcast in $\mathcal{TC}^{\mathcal{R}}$ for process p.

Constant Variables:
$N \in \mathbb{N}^*$: number of processes in the DG
$id(p) \in IDSET$: ID of p

Local Variables:
• $receivedMsgs(p)[]$: is an array of N queues. $receivedMsgs(p)[q]$ contains the not delivered messages received from q. They are inserted in $receivedMsgs(p)[q]$ according to their respective receiving round. Initially, all queues are empty.
• $atomicCounter(p)$: counts the number of messages that p should atomically broadcast. Initially, its value is zero.

1 **Procedure** *Atomic_Broadcast(m)*
2 $\quad FD\_brcst$(m); $atomicCounter(p)++$;

3 **Procedure** $FD\_dlr(id(q),m)$
4 $\quad receivedMsgs(p)[q].enqueue(m)$;
5 $\quad$ **if** *all queues of* $receivedMsgs(p)$ *contain a message* **then**
6 $\quad\quad$ **forall** $q \in [1,n]$ **do**
7 $\quad\quad\quad m := receivedMsgs(p)[q].dequeue$;
8 $\quad\quad\quad$ **if** $m \neq atomic\perp$ **then** $Atomic\_Deliver(id(q), m)$;
9 $\quad\quad atomicCounter(p) - -$;

10 **Repeat Forever**
11 $\quad$ **if** $atomicCounter(p) = 0$ **then** Atomic_Broadcast($atomic\perp$);

Each process perpetually executes an atomic broadcast. The broadcast message is either provided by the application or equals to $atomic\perp$. The variable $atomicCounter(p)$ records the current number of messages that p is or are waiting to be atomically broadcast. The value of $atomicCounter(p)$ is incremented at each execution of the procedure $Atomic\_Broadcast$ (line 2) and decremented during the execution of the procedure **FD_dlr** if p ends its current atomic broadcast (line 9 in if-block starting at line 5). If p has no atomically broadcast going-on (*i.e.* $atomicCounter(p) = 0$) then its start a new atomic broadcast of the neutral data by calling Atomic_Broadcast($atomic\perp$) (line 11).

At the invocation of $\mathbf{FD\_dlr}(id(q), m)$, the message m is included in the queue $receivedMsgs(p)[q]$ and thus considered as a waiting message. When every queue $receivedMsgs(p)[q]$ is not empty, i.e., there is a waiting message from every process, the if-condition of line 5 is verified. Then p handles one waiting message per process q; this message is the first one in queue $receivedMsgs(p)[q]$. The N handled messages are removed of their queue and delivered, provided they are not $atomic\perp$.

By lack of space, proof of the correctness of Algorithm 3 are only sketched.

$m(s, j)$ denotes the jth message atomically broadcast by the process s.

$<_{Rlex}$ denotes the reversed lexicographic order: $m(id(p), i) <_{Rlex} m(id(q), j)$ if and only if $(i < j)$ or $(i = j$ and $id(p) < id(q))$.

Let $m(p, i)$ and $m(q, j)$ be two atomically broadcast messages not equal to $atomic\perp$. Algorithm 3 ensures that If $m(p, i) <_{Rlex} m(q, j)$ then $m(p, i)$ is atomically delivered before $m(q, j)$ by every process.

On every process, the delivering order is the reversed lexicographic order of the messages. Hence, algorithm 3 provides FIFO total order Broadcast primitives in Class $\mathcal{TC}^{\mathcal{R}}$ as *Validity*, *Agreement*, and *Integrity*, *Fifo order*, *Total order* properties are verified.

If r atomically delivers $m(s, i)$ before atomically broadcasting $m(r, j)$ then $j > i$. So, algorithm 3 ensures the *Local order* property. (*local order property*) If a process broadcasts a message m and a process delivers m before broadcasting m', then every process delivers m before m'. Hadzilacos and Toueg [2] prove that the property of Causal order is equivalent to the combination of Fifo order and Local order properties. Algorithm 3 provides causal total order broadcast primitives in Class $\mathcal{TC}^{\mathcal{R}}$.

VIII. Conclusion

In this article we have presented two broadcast algorithms that respectively provide FIFO and total orders for the delivery of broadcast messages on top of dynamic systems with recurrent connectivity. We model the latter with the Class $\mathcal{TC}^{\mathcal{R}}$ of Time-varying Graph formalism [1]. System membership, composed by N processes, is fixed but unknown. The algorithms ensure termination detection and tolerate message losses. The size of their message is bounded to $2N + O(log(N)) + msgSize$ bits where $msgSize$ is a bound size in bits of broadcast data.

The second algorithm provides a causal total order of message delivery.

Futures directions research for our work is to design an algorithm able to deal with fault as process crashes, byzantine fault or transient fault.

References

[1] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro, "Time-varying graphs and dynamic networks," *Inter. J. of Parall., Emergent and Dist. Systems*, vol. 27, no. 5, pp. 387–408, 2012.

[2] V. Hadzilacos and S. Toueg, "A modular approach to fault-tolerant broadcasts and related problems," Cornel University, Tech. Rep., 1994.

[3] X. Défago, A. Schiper, and P. Urbán, "Total order broadcast and multicast algorithms: Taxonomy and survey," *ACM Comput. Surv.*, vol. 36, no. 4, pp. 372–421, 2004.

[4] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to Reliable and Secure Distributed Programming*, 2nd ed. Springer Publishing Company, Incorporated, 2011.

[5] M. K. Aguilera, W. Chen, and S. Toueg, "On quiescent reliable communication," *SIAM J. Comput.*, vol. 29, no. 6, pp. 2040–2073, 2000.

[6] N. Michael, M. M. Zavlanos, V. Kumar, and G. J. Pappas, "Maintaining connectivity in mobile robot networks," in *Experimental Robotics, The Eleventh International Symposium, ISER 2008, July 13-16, 2008, Athens, Greece*, ser. Springer Tracts in Advanced Robotics, O. Khatib, V. Kumar, and G. J. Pappas, Eds., vol. 54. Springer, 2008, pp. 117–126.

[7] R. O'Dell and R. Wattenhofer, "Information dissemination in highly dynamic graphs," in *Proceedings of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing*. ACM, 2005, pp. 104–110.

[8] A. Casteigts, P. Flocchini, B. Mans, and N. Santoro, "Deterministic computations in time-varying graphs: Broadcasting under unstructured mobility," in *6th International Conference on Theoretical Computer Science, TCS 2010*, vol. IFIP AICT, volume 323, 2010, pp. 111–124.

[9] F. Kuhn, N. A. Lynch, and R. Oshman, "Distributed computation in dynamic networks," in *42nd ACM Symposium on Theory of Computing, STOC 2010*, 2010, pp. 513–522.

[10] M. Raynal, J. Stainer, J. Cao, and W. Wu, "A simple broadcast algorithm for recurrent dynamic systems," in *28th IEEE International Conference on Advanced Information Networking and Applications, AINA 2014*, 2014, pp. 933–939.

[11] A. Casteigts, P. Flocchini, B. Mans, and N. Santoro, "Shortest, fastest, and foremost broadcast in dynamic networks," *Int. J. Found. Comput. Sci.*, vol. 26, no. 4, pp. 499–522, 2015.

[12] C. Gómez-Calzado, A. Casteigts, A. Lafuente, and M. Larrea, "A connectivity model for agreement in dynamic systems," in *21st International European Conference on Parallel and Distributed Computing, Euro-Par 2015, Springer LNCS: 9233*, 2015, pp. 333–345.

[13] M. Ahmadi, A. Ghodselahi, F. Kuhn, and A. R. Molla, "The cost of global broadcast in dynamic radio networks," *Theoretical Computer Science*, vol. 806, pp. 363–387, 2020.

[14] P. J. Marandi, M. Primi, N. Schiper, and F. Pedone, "Ring paxos: High-throughput atomic broadcast," *Computer Journal*, vol. 60, no. 6, pp. 866–882, 2017.

[15] O. Lundström, M. Raynal, and E. M. Schiller, "Self-stabilizing uniform reliable broadcast," 2020.

[16] B. Blywis, M. Güneş, F. Juraschek, and S. Hofmann, "Gossip routing in wireless mesh networks," in *21st Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 2010, pp. 1572–1577.

[17] Z. J. Haas, J. Y. Halpern, and L. Li, "Gossip-based ad hoc routing," *IEEE/ACM Trans. Netw.*, vol. 14, no. 3, pp. 479–491, 2006.

[18] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A. Kermarrec, "Lightweight probabilistic broadcast," *ACM Trans. Comput. Syst.*, vol. 21, no. 4, pp. 341–374, 2003.

[19] D. Frey, R. Guerraoui, A. Kermarrec, B. Koldehofe, M. Mogensen, M. Monod, and V. Quéma, "Heterogeneous gossip," in *Middleware 2009, ACM/IFIP/USENIX, 10th International Middleware Conference*, ser. Springer, LNCS: 5896, 2009, pp. 42–61.

[20] S. Rajsbaum and M. Raynal, "60 years of mastering concurrent computing through sequential thinking," *SIGACT News*, vol. 51, no. 2, pp. 59–88, 2020.

[21] D. Agrawal, G. Alonso, A. E. Abbadi, and I. Stanoi, "Exploiting atomic broadcast in replicated databases (extended abstract)," in *Euro-Par '97 Parallel Processing, Third International Euro-Par Conference*, ser. LNCS 1300. Springer, 1997, pp. 496–503.

[22] I. Stanoi, D. Agrawal, and A. E. Abbadi, "Using broadcast primitives in replicated databases," in *18th International Conference on Distributed Computing Systems*. IEEE Computer Society, 1998, pp. 148–155.

[23] F. Pedone, R. Guerraoui, and A. Schiper, "The database state machine approach," *Distributed Parallel Databases*, vol. 14, no. 1, pp. 71–98, 2003.

[24] R. Baldoni, S. Bonomi, and M. Raynal, "Implementing a regular register in an eventually synchronous distributed system prone to continuous churn," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 1, pp. 102–109, 2012.

[25] M. Perrin, M. Petrolia, A. Mostéfaoui, and C. Jard, "On composition and implementation of sequential consistency," in *30th International Symposium Distributed Computing, DISC*, ser. LNCS 9888. Springer, 2016, pp. 284–297.

[26] S. Bonomi and A. S. Nezhad, "Multi-writer regular registers in dynamic distributed systems with byzantine failures," in *Workshop on Theoretical Aspects on Dynamic Distributed Systems, TADDS '11,*. ACM, 2011, pp. 8–12.

[27] H. Attiya, H. C. Chung, F. Ellen, S. Kumar, and J. L. Welch, "Emulating a shared register in a system that never stops changing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 3, pp. 544–559, 2019.

[28] A. E. F. Clementi, A. Monti, F. Pasquale, and R. Silvestri, "Information spreading in stationary markovian evolving graphs," *IEEE Trans. Parallel Distributed Syst.*, vol. 22, no. 9, pp. 1425–1432, 2011.

[29] A. E. F. Clementi, P. Crescenzi, C. Doerr, P. Fraigniaud, F. Pasquale, and R. Silvestri, "Rumor spreading in random evolving graphs," *Random Struct. Algorithms*, vol. 48, no. 2, pp. 290–312, 2016.

[30] B. Awerbuch, Y. Mansour, and N. Shavit, "Polynomial end-to-end communication," in *FOCS89, the 30th Annual IEEE Symposium on Foundations of Computer Science*, 1989, pp. 358–363.

[31] A. Ghodsi, O. A. L., S. El-Ansary, P. Brand, and S. Haridi, "Self-correcting broadcast in distributed hash tables," in *15th International Conference Parallel and Distributed Computing and Systems PDCS 2003*, 2003.

[32] W. Li, S. Chen, P. Zhou, X. Li, and Y. Li, "An efficient broadcast algorithm in distributed hash table under churn," in *3th International Conference on Wireless Communications, Networking and Mobile Computing, WiCom*, 2007, pp. 1929–1932.

[33] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of ACM*, vol. 43, no. 2, p. 225–267, 1996.

[34] B. Charron-Bost and S. Moran, "The firing squad problem revisited," in *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018*, 2018, pp. 20:1–20:14.

[35] M. Barjon, A. Casteigts, S. Chaumette, C. Johnen, and Y. M. Neggaz, "Maintaining a distributed spanning forest in highly dynamic networks," *Computer Journal*, vol. 62, no. 2, pp. 231–246, 2019.