Supporting conflict-free replicated data types in opportunistic networks

Frédéric Guidec, Yves Mahéo, Camille Noûs

Supporting Conflict-Free Replicated Data Types in Opportunistic Networks

Frédéric Guidec[1,2], Yves Mahéo[1,2] and Camille Noûs[2]

[1]IRISA, Université Bretagne Sud, France.
[2]Laboratoire Cogitamus, France.

Abstract

Conflict-Free Replicated Data Types (CRDTs) are data types that can be used in distributed systems when optimistic replication is tolerable. Replicas can be updated locally, without coordination, and consistency is obtained eventually by asynchronously propagating updates among replicas. Because CRDTs can tolerate asynchronous transmissions, they can serve as software elements in opportunistic networks (OppNets), where the dissemination of information is dependent on unplanned transient radio contacts between mobile nodes. In this paper we investigate the problem of implementing operation-based, state-based, and delta-state-based CRDTs in OppNets. A contact-driven synchronization algorithm is proposed for each kind of CRDT, and experiments based on realistic tracesets are conducted in order to compare how these algorithms can perform in an OppNet. Experimental results show that delta-state-based CRDTs globally outperforms operation-based and pure state-based CRDTs, especially when considering the number of messages required to ensure the synchronization of replicas.

Keywords: CRDT, optimistic replication, opportunistic networking, ad hoc networks

1 Introduction

In distributed systems, data objects must often be replicated on several hosts. Maintaining the consistency of the resulting replicas can be an issue when these replicas can be updated concurrently. Some systems strive to maintain strong consistency continuously, based on coordination models that prevent or at least severely constrain concurrent updates (e.g., [13, 23]). Other systems adopt a more

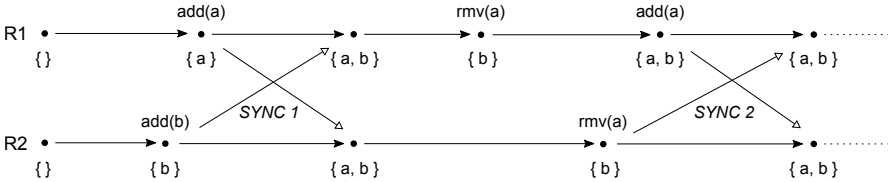


Fig. 1: Example of a run involving an Add-Wins Set replicated in R1 and R2

relaxed approach, allowing replicas to diverge temporarily (optimistic replication), while ensuring eventual consistency, that is, guaranteeing that they will eventually reach a common state.

Conflict-free Replicated Data Types (CRDTs) belong to this second category: any replica can be updated locally, without any coordination with the others. Synchronization occurs episodically between pairs of replicas by exchanging information about past updates. When the same updates have been taken into account by all the replicas, these replicas finally attain the same state.

Many kinds of CRDTs have already been described in the literature, such as counters, registers, sets, maps, lists, graphs, etc. [26, 32]. Each CRDT is characterized by the updates it supports, and by a concurrency semantic to be applied when updates occur concurrently.

A Set CRDT, for example, implements a distributed set. It supports two kinds of updates: *add(x)*, and *rmv(x)*. In the presence of concurrent *add(x)* and *rmv(x)* updates applying to the same element *x*, several concurrency semantics are possible. In an Add-Wins set, priority is given to *add(x)*, while in a Remove-Wins set it is given to *rmv(x)*. Figure 1 shows an example where an Add-Wins set is replicated in two replicas *R1* and *R2*, with initial state { } on both replicas (i.e., both replicas agree that the set is initially empty). In replica *R1*, element *a* is first added locally to the set, while element *b* is added to the set in replica *R2*. At this stage of this execution the state is different in the two replicas, but a synchronization occurs between these replicas (SYNC1), after which they agree that the state of the set is now {*a, b*}. In replica *R1*, element *a* is then removed, and then added again to the set, while in replica *R2* element *a* is only removed from the set. After these operations, both replicas synchronize again, which leads to a situation where they agree that the state of the set is {*a,b*} again. This is because the last *add(a)* in *R1* and the last *rmv(a)* in *R2* occurred concurrently, and priority is given to *add(a)* in and Add-Wins Set. The same scenario with a Remove-Wins set would result in a set with state {*b*} in both replicas, since the priority would then be given to *rmv(a)*.

Two main ways of implementing CRDTs are usually distinguished, resulting in so-called operation-based CRDTs and state-based CRDTs. In an operation-based CRDT, when an operation (update) is applied to a replica, a description of this operation is embedded in a message and sent to all other replicas. Each recipient can then update its own state accordingly. This approach requires a message dissemination layer that guarantees at least reliable broadcast. If the operations that can be applied to the CRDT are not idempotent and do not commute, then the

dissemination layer must additionally guarantee that the messages are delivered exactly once, and (usually) in causal order.

In a state-based CRDT, an operation is applied only to the state of the local replica. Each replica synchronizes periodically with other replicas by sending them its entire state. Upon reception of the state of another replica, this received state is merged with the local state, thanks to a function that deterministically computes the join (least upper bound) of both states. The set of all states then forms a monotonic semilattice, and the state of the CRDT must be defined such that any update monotonically increases this state, according to the same partial order rules as the semilattice [26, 32].

With state-based CRDTs, there is no need for each update to be transmitted to all replicas, unlike with operation-based CRDTs. A sporadic synchronization with a few other replicas is sufficient. If the synchronization graph is connected [26], eventual consistency can be reached. Nevertheless, shipping entire states between replicas can incur a major communication overhead. Delta-state CRDTs (or delta CRDTs for short) have been proposed to alleviate this overhead by passing only partial information about the sender's state. This information typically consists in a representation of the effect of the last update operations performed on the local state [1, 21].

CRDTs are suited to distributed systems that may observe node crashes and network partitions. Provided crashed nodes eventually recover, and partitions merge, the replicas will ultimately converge. Opportunistic networks (OppNets) are inherently networks that exhibit such characteristics. An OppNet is a network whose nodes are mostly mobile, and that operates solely by exploiting temporary direct radio contacts between pairs of nodes [25]. Most of the times these radio contacts cannot be predicted in advance, so they must be exploited opportunistically whenever they occur, hence the opportunistic nature of communication in such a network. Besides, once a contact is established between two nodes, it can be broken as soon as the nodes move away from each other.

Figure 2 illustrates the evolution of a typical OppNet, whose mobile nodes only come into contact every now and then, depending on how they move. The nodes depicted in this figure may be handheld devices carried by human beings, but they may as well be carried by vehicles, animals, drones, or any combination of mobile entities. The edges in the figure depict radio contacts between the nodes. Because the radio transmission range is limited, this network is obviously partitioned. This is not an exceptional situation in an OppNet, but a perfectly normal situation.

We believe that because they tolerate asynchronous communication, which is inherent to OppNets, CRDTs can serve as perfect building blocks for distributed applications devoted to OppNets. In traditional (Internet-like) networks, CRDTS are typically used to implement distributed databases, collaborative editing systems, chat systems, etc. There is no reason why similar use cases could not be implemented in OppNets as well, using CRDTs as a means to deal with asynchronous communication.

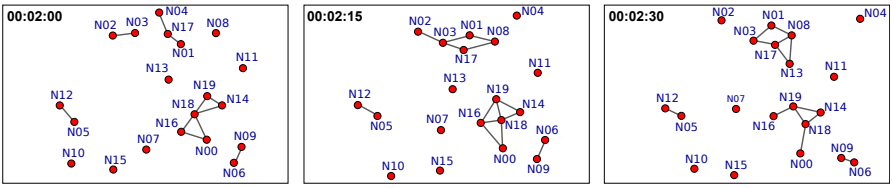


Fig. 2: Illustration of the evolution of an opportunistic network

This does not mean that CRDTs can be used in any kind of OppNet, though. The behavior of CRDT replicas is always guaranteed only within the limits of a clearly defined system model (e.g., reliable broadcast is required for operation-based CRDTs). All OppNets do not necessarily satisfy this model, so when running experiments involving CRDTs we shall make sure that the networking scenario considered is consistent with the use of CRDTs.

In this paper we investigate several implementations of CRDTs in OppNets, showing how the replicas of each kind of CRDT can be synchronized in a networking environment in which information propagation can only rely on unpredicted pair-wise contacts between pairs of mobile nodes. Our objective is also to determine if all kinds of CRDT can perform equally well in an opportunistic networking setting, or if one kind should for example be preferred over the others.

The paper constitutes an original study on the synchronization of CRDTs in OppNets that encompasses operation-based, state-based, and delta-state-based CRDTs. Its main contributions are:

- the definition of OppNet-compatible synchronization algorithms for operation-based, state-based, and delta-state-based CRDTs;
- the definition of a transitive mode for the synchronization of delta-state-based CRDTs, in order to potentially accelerate the convergence of the replicas in stable parts of the OppNet;
- the definition of an experimentation methodology for CRDT synchronization in OppNets;
- the presentation and the analysis of experimental results produced according to this methodology, based on several real-life contact tracesets.

It is worth mentioning that, in the literature, it is commonly assumed that an OppNet is a network in which the main challenge is to route each unicast message toward its destination. The work presented in this paper clearly stands out from this vision, since in the synchronization algorithms we define for state-based and delta-state-based CRDTs, messages are only exchanged directly between neighbor nodes, so no routing algorithm is required to forward messages beyond direct neighbors. Operation-based CRDTs require reliable broadcast, though, but in an OppNet this can be obtained efficiently with an epidemic dissemination of messages.

As mentioned above, CRDTs can only be used in networking environments that fully satisfy their system model. Although a few tracesets of real OppNets

are available in the CRAWDAD database (Community Resource for Archiving Wireless Data At Dartmouth [31]), none of these tracesets can readily be used to run experiments involving CRDTs. This is generally due to the fact that some of the mobile nodes considered in a traceset behave erratically, which would prevent CRDT replicas to ever converge. Rather than designing a purely synthetic scenario that would satisfy the requirements of CRDTs by construction, we show in this paper how satisfactory tracesets can be derived from the raw tracesets available on CRAWDAD.

The remainder of this paper is organized as follows. Related work is presented and discussed in Section 2. The system model we consider in this work is detailed in Section 3. Synchronization algorithms for each kind of CRDT considered in this paper (i.e., operation-based, state-based, and delta-state based) are presented in Section 4. Section 5 presents experimental results we obtained while running each of these algorithms to ensure the synchronization of Add-Wins Sets (AW-Sets) in realistic opportunistic networks. A discussion on the lessons learned from experimentation is developped in Section 6 and Section 7 concludes the paper.

2 Related work

2.1 Operation-based CRDTs in OppNets

Opportunistic computing has been introduced by Conti et al. as an emerging paradigm that moves forward from simple message forwarding in OppNets, with the aim of enabling collaborative computing in such networks where long disconnections and network partitions are the rule [9].

Since they can tolerate asynchronous communication and occasional network partitions, CRDTs are natural candidates to be used in OppNets, in which they could be used to implement opportunistic computing applications. Yet implementing and using CRDTs in OppNets has not been addressed much in the literature so far. Costea et al. formulate two propositions, [10] and [11], in which a CRDT derived from Logoot [34] is used as a means to obtain total order in OppNets. In both of these papers the authors implement operation-based CRDTs, assuming that an opportunistic communication layer is available to ensure the causal dissemination of operations network-wide. Two protocols are considered to ensure this dissemination (pure epidemic [33], and socially-aware interest-based dissemination [8]), and causal order is obtained based either on CBCAST (Causal Broadcast) [10] or on causal barriers [11]. The experimental results presented in both of these papers show that implementing operation-based CRDTs in OppNets puts significant pressure on the caches maintained on each node to store messages. Recognizing this fact, the authors assume that when messages get purged from the caches before being delivered to all potential receivers, there is still a chance for replicas to synchronize during pair-wise contacts, based on data maintained directly in the application layer.

A collaborative editing system for Android that targets delay-tolerant networks is presented in [28]. This work constitutes a first attempt to fully implement a CRDT for OppNets, using IBR-DTN [30] to handle intermittent connections. As it is

derived from Logoot [34] (which defines an operation-based CRDT), the presented prototype also suffers from the scalability drawbacks mentioned above.

2.2 The potential of state-based CRDTs in OppNets

Although the implementation of operation-based CRDTs in OppNets has already been considered in a few papers, the case of state-based CRDTs has not been investigated as yet, to the best of our knowledge. Some papers describe opportunistic computing algorithms that could typically rely on state-based CRDTs, though. For example, in [20] the authors investigate the problem of counting how many mobile nodes are present in an enclosed area, using a distributed algorithm, and based on pair-wise radio contacts only. A GO-Counter (Grow-Only Counter) is a CRDT that can be implemented efficiently as a state-based CRDT, and that could typically be used to count the number of nodes in a network.

The advantage of state-based CRDTs is that, unlike operation-based CRDTs, they can be synchronized over unreliable communication channels. The drawback is that they require shipping the entire state of a replica, which can yield significant communication overhead for container CRDTs (e.g., sets, maps, graphs), which can store large amounts of data [26]. Delta-state based CRDTs (δ-CRDTs) mitigate this issue by only shipping in a synchronization message the change that has been made recently to a replica, rather than its full state [1, 2, 15]. This change is expressed as the join of multiple fine-grained states called deltas. Several deltas can be joined in a delta-group, and shipped together to a remote replica.

This approach requires that each replica keeps track of all the other replicas it is connected to at any time. Besides, the replica must maintain for each peer a communication buffer (δ-buffer) that contains the deltas that have not yet been passed to (and acknowledged by) the peer, thus ensuring FIFO communication between any pair of replicas. When a connection is established between two replicas for the first time, or after the last connection has been disrupted, the replicas must exchange their full state before they can start exchanging deltas.

Δ-CRDTs are a variant of δ-CRDTs that does not resort to delta buffers, and that does not require that pairs of replicas communicate continuously to synchronize their state [21]. Instead, it is assumed that each replica maintains the causal context, typically, as a version vector, that replicas can exchange. By comparing its own causal context with a received one, a replica can determine the minimal delta (Δ) of its state that is missing on the remote replica, and send only this Δ rather than its full state. With the synchronization algorithm described in [21], a replica sends its version vector to a randomly selected peer, either periodically (pull model), or after an update has been applied locally (push model). Several forms of δ-CRDTs have been proposed in the literature, enhancing the original synchronization algorithm proposed in [2] for specific data structures like JSON [5, 27] or text [24].

In most papers dealing with CRDTs, details about the communication layer are usually not discussed much. It is often assumed that the hosts where CRDT replicas are maintained belong to a P2P network, that each replica is somehow aware of its peers at any time, and that connections between peer nodes are available whenever

Code 1 Definitions of basic functions and events offered by the communication layer [Brown code: definition only required for operation-based synchronization]

```
01  def ID: String   # or IMEI, MACaddr, etc.
02  def MSG: ...

03  Initialization:
04    static ID self.id   # id of the local host

05  event new_neighbor(ID neigh_id)
06    # Raised by the networking layer when a new neighbor
07    # has been detected

08  function current_neighbors(): Set<ID>
09    # Returns the identifiers of the current neighbors
10    # of the local host

11  function send(ID dest_id, MSG msg)
12    # Sends the specified msg to the specified destination
13    # (which must be a neighbor of the sender)

14  function broadcast(MSG msg)
15    # Broadcasts the specified msg network-wide

16  event receive(ID src_id, MSG msg)
17    # Called by the networking layer when a msg
18    # has been received from a neighbor
```

needed, although transmissions may actually fail. As a consequence, in most synchronization algorithms, each replica periodically selects one or several peer replicas (usually randomly), and initiates a synchronization session with this/these peer(s). In an OppNet, the situation is quite different, since mobile devices can only communicate during transient contacts, which are usually unexpected and may be broken at any time. Any synchronization algorithm designed for OppNets should take these constraints into account, considering that pair-wise synchronization between replicas must be contact-driven rather than being attempted randomly.

3 System model

In an opportunistic network, interactions are based solely on pair-wise radio contacts between neighbor devices. As a general rule, a radio contact between two devices cannot be planned in advance, so when such a contact occurs it constitutes a transient opportunity for these devices to exchange messages. Likewise, once a contact is established between two nodes it is usually not possible to predict how long this contact will last, so any opportunistic interaction protocol must tolerate communication disruptions.

The synchronization algorithms defined in the next section are meant to run on top of a basic communication layer, whose characteristics are detailed below.

The functions and events offered by the communication layer are presented in Code 1. A mobile device must be able to detect its neighbors, that is, other devices that are in its radio range. In addition, it must be able to exchange messages with any of these neighbors.

Event *new_neighbor()* is raised by the communication layer whenever a contact is established with a new neighbor. As illustrated in Fig. 2, a mobile device may have several neighbors simultaneously, so function *current_neighbors()* can be called to obtain a list of the current neighbors, as perceived by the communication layer. Neighbor discovery may actually be performed in several ways in the communication layer, depending on the characteristics of the actual underlying transmission technology. With Bluetooth, for example, an event is raised automatically whenever a channel is established (or broken) between two peers, so discovering a neighbor is quite straightforward. With Wi-Fi running in ad hoc mode, a neighbor discovery protocol (typically based on the periodic broadcast of "hello" messages) must be implemented in the communication layer.

Function *send()* serves to transmit a message to a direct neighbor of the sender. A number of reasons may incur a transmission failure, for instance if the targeted neighbor has just moved out of range. It should be noticed that function *send()* does not return a status, for it is not assumed that transmission failures can be detected (a missing acknowledgment, for example, is not a guarantee that a message has not reached its destination).

Event *receive()* is automatically raised by the communication layer when a message has been received from a neighbor device. We assume that the communication layer automatically discards corrupted messages.

Note that *send()* and *receive()* are only meant to support message exchange between direct neighbors. With state-based and delta-state-based CRDTs there is no need to propagate messages beyond direct neighbors, since the synchronization of replicas only occurs between neighbor nodes. There is thus no need for a routing algorithm supporting multi-hop message forwarding.

Operation-based CRDTs require that updates disseminate in the whole network, though. For such CRDTs —and only for these— the communication layer must implement a broadcast protocol that makes it possible to send a message to all nodes in the network, via function *broadcast()*. A dissemination protocol operating according to the "store, carry and forward" principle is thus required: each node maintains a cache in which messages can be stored, carried for a while, and forwarded later to neighbor nodes. Epidemic forwarding is typically an effective method to perform this dissemination [12, 33], but other dissemination algorithms may be used, for example in networks where mobility or radio contacts follow regular patterns.

4 Synchronization algorithms for CRDTs in OppNets

A synchronization algorithm designed for opportunistic networks must use radio contacts between mobile nodes as opportunities for these nodes to synchronize the CRDT replicas they hold. In this section we present three algorithms that are meant to support the synchronization of operation-based CRDTs, state-based CRDTs, and delta-state-based CRDTs respectively. We also propose an extension for the synchronization of delta-state-based CRDTs, using transitive forwarding as a means to speed up the dissemination of delta states in an OppNet.

Code 2 Operation-based (OB) synchronization algorithm

```
01 function generate_effector(MUTATOR m): EFFECTOR
02   # Generates the effector of update m

03 function apply_effector(EFFECTOR m)
04   # Applies effector m on the local replica

05 upon update(MUTATOR m)
06   m' = generate_effector(m)
07   broadcast(m')
08   apply_effector(m')

09 upon receive(ID src_id, EFFECTOR m')
10   apply_effector(m')
```

4.1 Operation-based synchronization

In an operation-based CRDT, whenever an operation (update) is performed on a replica, information about this operation is embedded in a message and sent to all other replicas, which can then update their own state accordingly [32].

In practice, any operation-based CRDT must define a generator function and an effector function [26]. The generator function is meant to be executed only in the replica where the update is originally applied (see lines 01-02 in Code 2). It returns an effector that encodes the side-effects of the update. This effector must eventually be executed in all replicas of the CRDT, thus updating each replica state.

In Code 2 we define the signatures of functions *generate_effector()* and *apply_effector()*. How these functions are implemented depends on the kind of CRDT considered.

In any case, once such functions are available, the synchronization of replicas is straightforward, provided a broadcast service is available to propagate effectors between replicas. Whenever an update is applied to a replica (line 05), an effector is created locally on this replica (line 06), and broadcast to all other replicas (line 07). This effector is eventually applied in each replica, regardless of whether it has been produced locally (line 08) or received from another replica (lines 09-10).

Operation-based synchronization requires a broadcast service that guarantees at least reliable broadcast: all effectors must reach all replicas, and be executed there. If the operations (updates) that can be applied to the CRDT considered are not idempotent or do not commute (or both), then the broadcast service must additionally guarantee that the messages carrying effectors are delivered exactly once, and (usually) in causal order.

Broadcasting messages in an OppNet requires resorting to the "store, carry, and forward" principle [16]: each mobile node must serve as a "data mule" for messages it has either produced itself or received recently, storing these messages in a local cache, and carrying them for a while, so they can be forwarded to other nodes whenever new contacts occur. Epidemic forwarding is a broadcasting approach that relies on this general principle [33]. Whenever two mobile nodes get into contact, they first exchange summary vectors. A summary vector is basically a manifest of what is contained in the sender's cache. Upon receiving a neighbor's summary

Code 3 Basic state-based (SB) synchronization algorithm

```
01 def ID: String      # or MACaddr, or IMEI, etc.

02 static ID self.id   # local host's id
03 REPLICA self.state ↩ ⊥

04 function merge(REPLICA t_L, REPLICA t_R): REPLICA
05   # Returns the LUB (Least Upper Bound) of t_L and t_R

06 upon new_neighbor(neigh_id) do
07   if (self.id < neigh_id) then
08     send(neigh_id, self.state)
09   fi

10 upon receive(neigh_id, neigh_state) do
11   if (neigh_state ≠ self.state) then
12     if (self.id > neigh_id) then
13       send(neigh_id, self.state)
14     fi
15     self.state ↩ merge(self.state, neigh_state)
16   fi
```

vector, the receiver can either request copies of messages that are not yet in its own cache (pull mode), or send copies of messages that are not yet in the neighbor's cache (push mode). In variants of this model, each node can be made selective regarding the kind of messages it is willing to store in its cache [12].

Epidemic forwarding per se does not guarantee that messages are delivered in causal order. This can however be obtained with very little overhead, for example by piggy-backing causal barriers in the messages submitted to epidemic dissemination [17].

4.2 State-based synchronization

State-based CRDTs do not necessitate that each update be sent to all other replicas. It is sufficent that each replica synchronizes frequently enough with a few other replicas. Under the assumption that the synchronization graph is connected, eventual consistency is ensured [26].

A basic state-based (SB) synchronization algorithm is presented in Code 3. This algorithm is contact-driven: the synchronization process between two nodes is initiated when these nodes get into radio contact. Whenever such a contact occurs, one of the nodes initiates the synchronization by sending its full state to the peer node (lines 07-08). Upon reception, the receiver compares it with its own local state (line 11). If both states are distinct, and if this node had not initiated the synchronization procedure, it sends its own state to the peer node (line 13). In any case, the received state is merged with the local state (line 15). The implementation of function *merge(),* which returns the LUB (Least Upper Bound) of both states, depends on the kind of CRDT considered.

Code 4 Signatures of functions required by the delta-state-based synchronization algorithm

```
01 def ID: String      # or MACaddr, or IMEI, etc.
02 def DIGEST: xxx     # Version Vector

03 static ID self.id   # id of the local host
04 REPLICA self.state ↩ ⊥

05 function merge(REPLICA t_L, REPLICA t_R): REPLICA
06   # Returns the LUB (Least Upper Bound) of t_L and t_R

07 function get_digest(REPLICA t): DIGEST
08   # Returns the digest of t

09 function generate_delta(MUTATOR m): REPLICA
10   # Returns the effect of m, expressed as a delta state

11 function get_missing(DIGEST d, REPLICA s): REPLICA
12   # Determines what part of state s would strictly inflate
13   # the remote replica whose digest is d
```

4.3 Delta-state-based synchronization

The problem with state-based CRDTs is that shipping entire states between replicas can yield a major communication overhead. When dealing with CRDTs whose size remains constant or does not grow much, such as counters and registers, state-based synchronization is sufficient. But for container-like CRDTs that can aggregate large amounts of data, such as sets, maps or lists, a better option is to design a more frugal synchronization protocol, that avoids to transmit entire states every time it is possible. In [14] two techniques, namely state-driven synchronization and digest-driven synchronization, have been proposed to reduce the need for bidirectional full state transmission. The algorithm we present in this section typically leverages these techniques, shipping either digests or deltas (partial states) in order to synchronize replicas.

Some functions must be defined for processsing the replica on a node. Their signatures are presented in Code 4. The actual implementation of these functions depends on the kind of CRDT considered.

Function *merge()* must be called to merge the local replica'state with the delta state received from a neighbor. More formally, this function returns the LUB (Least Upper Bound) of both states. It is similar to the function used in Code 3, except that the state passed as argument t_R is a delta state, whose internal representation is similar to that of a full state.

Function *get_digest()* is intended to return the digest of the state of a replica. A digest should express the state concisely, so that the cost of its transmission is far lower than that of the entire state. In the case where the internal representation of a CRDT includes causal context medadata, the digest generally takes the form of a version vector. With this digest, the missing part in each replica can be identified, or, more formally, one can determine the part of a replica's state that would be needed to strictly inflate the other replica's state.

Code 5 Delta-state-based (Δ-SB) synchronization algorithm

```
01  upon new_neighbor(neigh_id) do
02    if (self.id < neigh_id) then
03      send(neigh_id, self.digest)
04    fi

05  upon receive(neigh_id, digest) do
06    D ⟵ get_missing(digest, self.state)
07    if (D ≠ ⊥) then
08      send(neigh_id, D)
09    fi
10    if (digest after self.digest) then
11      send(neigh_id, self.digest)
12    fi

13  upon receive(neigh_id, delta) do
14    Δ_in ⟵ get_missing(self.digest, delta)
15    if (Δ_in ≠ ⊥) then
16      self.state ⟵ merge(self.state, Δ_in)
17      self.digest ⟵ get_digest(self.state)
18    fi

19  upon update(m) do
20    self.digest ⟵ get_digest(self.state)

21  function get_missing(DIGEST d, CRDT s): M
22    M ⟵ { m ∈ s.state, m.timestamp > d }
```

When an update (more formally, a mutator) is applied to the local replica, function *generate_delta()* is executed. It returns a delta that is the expression of the effect of the mutation. The computed delta is intended to be transmitted to another replica, so that this replica can merge it with its own local state.

Function *get_missing()* is meant to be invoked when the digest d of a remote replica has been received. It compares this digest to the local replica's state s, and determines what part of s would be required to strictly inflate the state of the remote replica. The result is a delta state, which captures the data required to inflate the remote replica's state, or $\perp$ if there is no possibility to inflate that state based on s. This function can also be used when a delta state has been received from a remote replica, in order to determine whether all or part of this delta is indeed an inflation to the local state.

A CRDT featuring these functions is actually a Δ-CRDT, as defined in [21].

Code 5 presents the delta-state-based synchronization algorithm we propose. This algorithm is quite similar to that presented in [21]. The main difference is that in [21] the synchronization of replicas is assumed to be performed periodically (pull model), and any update to the local replica's state is pushed immediately to a randomly selected remote replica. In Code 5, synchronization between replicas is only triggered by contacts between mobile nodes. Thus, when two nodes get into contact, and only at that time, one of the nodes sends its current digest to the peer node (lines 01–04).

Upon receiving this digest, the peer node invokes function *get_missing()* to compare the received digest to the local one, and determine if part of the local state

can be sent to the neighbor (line 06 and lines 21-22). If the delta state returned by *get_missing()* is not empty, it is sent to the peer node (lines 07-09).

The local version vector is also compared to that of the peer node. If the local vector is late compared to that of the peer, then this means that the local state could also be inflated based on information available in the peer node. In that case the local digest is sent to the peer (lines 10–12), so it can compute the delta to send back to the local node.

Upon receiving a delta state from a peer node (line 13), the local node uses this digest to determine what part of the received delta can be merged with the local state (line 14). This test may appear as being redundant with respect to that performed in line 06, but since a node may be in contact with several neighbors simultaneously (as shown in Fig. 2), it may occasionally receive deltas whose content overlaps from several neighbors. The test performed in line 14 prevents any attempt to merge redundant information with the local state.

Note that the local digest is adjusted whenever the local state is modified, that is, when an update operation is applied locally (line 20), or when there is a merge between the local state and another state (line 17).

4.4 Enabling transitivity in delta-state-based synchronization

In Code 5, two nodes synchronize their replicas only when they get into contact, that is, when event *new_neighbor()* is raised. But as long as a contact between two neighbor nodes is maintained, these nodes do not attempt to synchronize their replicas anymore. When an update is applied on a node, this node does not attempt to propagate this information to its current neighbors. Likewise, when a node synchronizes its replica with a new neighbor (thus obtaining new information from that neighbor), it does not attempt to propagate this information to its other neighbors either.

Depending on the mobility scenario considered in an OppNet, some mobile nodes may occasionally find themselves in relatively "stable" parts of the network, that is, parts where each node maintains long-lasting contacts with a few neighbors, while no new contact is observed for long periods of time. In such circumstances, it may be interesting to enable the transitive propagation of information between neighbors. The brown lines in Code 6 show how the delta-state-based synchronization algorithm presented in Code 5 can be extended along that line.

At each update operation on the local replica, a delta is produced based on the operation m (formally, the mutator) applied locally (line 20b). This delta is conveyed at once to all the current neighbors of the local host (line 20c). Likewise, when a node has received an input from a neighbor node and determined that all or part this input allows it to inflate its local replica, the inflation is propagated to all its current neighbors, except the one from which the input has just been received, thus avoiding the retro-propagation of information between replicas (lines 17b-17c).

Intuitively, enabling the transitive forwarding of delta states among neighbor nodes should be a way to speed up their dissemination in the network. A side-effect

Code 6 Delta-state-based synchronization algorithm with transitive forwarding (Δ-SB_T) [Brown code: additional lines to the Δ-SB algorithm]

```
13  upon receive(neigh_id, delta) do
14   Δ_in ↩ get_missing(self.digest, delta)
15   if (Δ_in ≠ ⊥) then
16     self.state ↩ merge(self.state, Δ_in)
17     self.digest ↩ get_digest(self.state)
17b    targets = current_neighbors() \ neigh_id
17c    disseminate(targets, Δ_in)
18   fi

19  upon update(m) do
20   self.digest ↩ get_digest(self.state)
20b  Δ_out ↩ generate_delta(m)
20c  disseminate(current_neighbors(), Δ_out)

23  function disseminate(targets, output):
24   forall id in targets do
25     send(id, output)
26   done
```

of this approach is that a host may receive the same input several times from distinct neighbors. For example, in Fig. 2b, an update applied locally on node N03 may be sent as a small delta to nodes [N02, N01, N17], and nodes N01 and N17 may in turn forward this delta to node N08 (which would thus receive the same delta twice). This is the reason why function *get_missing()* must systematically be invoked when processing an input (line 14), so as to discard any redundant information, and thus prevent this redundant information to disseminate further in the network.

Another expected side-effect of forwarding deltas transitively is that the number of delta messages exchanged between neighbors is increased significantly, while the amount of information contained in each delta message is reduced. For example, the delta sent to all neighbors via line 20c will only contain information about the last update applied to the local replica.

5 Experimentation

5.1 Definition of experimentation scenarios

In order to observe how the algorithms presented in Section 4 can perform in realistic conditions, we used LEPTON (Lightweight Emulation PlaTform for Opportunistic Networks [29]) to run experiments involving different kinds of CRDTs we implemented in Java, and different opportunistic networking scenarios. Each experiment requires combining a contact scenario (i.e., how the nodes get into contact) with an application scenario (i.e., what kind of CRDT is considered, and what is the timeline of updates applied to its replicas).

5.1.1 Contact scenarios

Operation-based synchronization is only possible in a network that supports reliable broadcast, while state-based or delta-state-based synchronization requires that the

synchronization graph is connected (i.e., each update is eventually accounted for by each replica) [26].

In a network that relies on the "store, carry, and forward" principle, both kinds of requirements actually amount to the same thing: any message broadcast by a node must disseminate asynchronously in the network, and reach eventually all other nodes.

All opportunistic networking scenarios are not able to meet this constraint, though. In an OppNet the dissemination of any piece of information can only rely on successive contacts between pairs of nodes. Sometimes, there is simply no possible "journey" that would allow a piece of information produced at time t on node n_1 to ever reach another specific node n_2 [6].

Running experiments involving CRDTs in an OppNet that cannot support reliable dissemination would simply not make sense. Special attention must thus be paid to defining networking scenarios that fit the needs of CRDTs.

Experiments involving OppNets are often conducted based on mobility or contact tracesets that have been collected in real-world settings. Several of these tracesets are available in the CRAWDAD database[1]. Yet a thorough analysis of these tracesets shows that they are usually not suitable to run experiments requiring the reliable broadcast (or dissemination) of messages. An alternative approach consists in using a purely synthetic mobility model based on a random walk (e.g., random waypoint, Levy walk), whose parameters can be twisted so as to guarantee that any message can eventually reach any node in the network. But such an approach is often deemed as not being "realistic enough", so an intermediate approach consists in using real contact traces as much as possible, while ensuring that reliable broadcast is indeed possible for all nodes over a reasonably long timespan.

The *cambridge/haggle* dataset is one of the datasets available in the CRAWDAD database [31]. This dataset includes traces of Bluetooth sightings collected by groups of users carrying small devices (iMotes) over several days, on different occasions and settings (in and around laboratories, during conference events, etc.). In the following we focus on traceset *Exp3* (also referred to as *cambridge/haggle/imote/infocom*), which contains data about sightings recorded during the INFOCOM conference in 2005.

This traceset contains the data collected by 41 iMote devices over 3 days. The iMotes were configured to scan the radio channels for about 10 seconds every 2 minutes. A sighting in the traceset therefore characterizes a time event when a Bluetooth device detected the presence of another device in its surroundings. Based on this kind of raw information, it is necessary to extrapolate radio contacts, which are time intervals (a contact has a beginning and an end) rather than just time events. We therefore pre-processed the *Exp3* traceset, assuming that a contact between two devices is established as soon as any of these devices detects the presence of the other device, that this contact is maintained as long as subsequent sightings between the same devices are not more than 2 minutes apart, and that the contact is broken shortly after the last sighting event (with a randomly chosen trailing latency).

[1] https://www.crawdad.org

Metrics	Values (* = min / max / avg / stdev values)
Duration of the scenario	70h37'27"
Nb of nodes	$N_n = 38$
Number of contacts	$N_c = 14\,820$
Durations of contacts	1" / 14h49'19" / 4'51" / 18'18" (*)
Number of inter-contacts	14 124
Durations of inter-contacts	1" / 64h48'42" / 2h33'11" / 6h49'03" (*)

Table 1: Statistics about the Exp3-RB contact scenario (derived from the *cambridge/haggle/imote/infocom2005* traceset)

The result of this kind of pre-processing defines a time-varying graph (TVG) whose nodes model radio-enabled devices, and whose edges model radio contacts between these devices. Such a time-varying graph can be played by a mobile network simulator (e.g, the ONE simulator [19]), or by a mobile network emulator (e.g., LEPTON [29]).

As mentioned above, a contact scenario such as that derived from the *Exp3* traceset does not necessarily guarantee that every node can at any time send information (typically, a message) to all other nodes, even if the "store, carry and forward" model is used on all nodes. Once a scenario is modelled as a time-varying graph, it is possible to use this graph to calculate the horizon of any node at any time. The horizon of a node n_i at time t (noted $H_n(t)$) is, basically, the set of nodes that could eventually receive a message sent by n at time t [7]. If $H_n(t)$ does not include all the other nodes in the graph, then it means that n cannot reliably broadcast a message at time t (or at any time after t).

By computing the horizons of all nodes in the *Exp3* contact scenario, it is possible to identify nodes that should not be considered in any experiment requiring the reliable broadcast of messages. This if for example the case for nodes 36 and 38, which both disappear from the horizon of node 41 at time t_1=03h34'53". Note that this does not necessarily mean that these nodes disappear entirely from the network at that time. It simply means that after t_1, any message broadcast by node 41 will never reach nodes 36 and 38, although these nodes may still be able to receive messages sent by other nodes. As a consequence, nodes 36 and 38 should not be considered as possible targets for reliable broadcast after t_1 (unless node 41 is itself discarded).

It can likewise be determined that the horizon of node 31 loses 15 other nodes at time t_2=17h06'27", so this node should not be considered as a possible source for reliable broadcast after t_2.

Using this approach iteratively, it is possible to discard nodes that are not fit to take part in reliable broadcast over a long timespan. The remaining nodes constitute a subset of nodes among which reliable broadcast remains possible. Based on the original *Exp3* scenario, we thus determined that by removing nodes {31, 36, 38} from the original set of 41 nodes, the remaining subset of 38 nodes can support reliable broadcast from time $t_s = 0$ to time t_f =48h01'30".

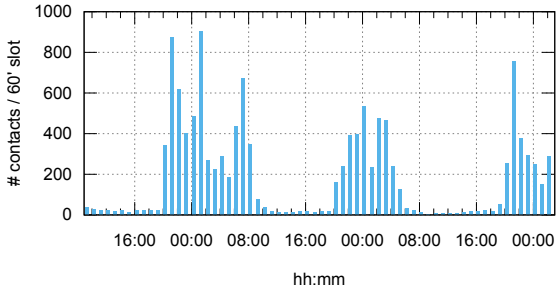


Fig. 3: Evolution of the number of contacts per 60' slot in the Exp3-RB contact scenario

In the remainder of this paper we call *Exp3-RB* (which stands for *Exp3-Reliable-Broadcast*) the contact scenario derived from *Exp3* that only involves these 38 nodes, among which reliable broadcast is possible over 48h01'. Statistics about contact and inter-contact durations in this scenario are given in Table 1, and the evolution of the number of contacts observed over time in this scenario is shown in Fig. 3.

It is worth mentioning that, to the best of our knowledge, none of the opportunistic networking datasets available in the CRAWDAD database is readily suitable to run experiments involving reliable broadcast. Some doctoring of the original traceset (such as that described above) is always required in order to produce a contact scenario in which reliable broadcast is indeed possible over a reasonably long timespan.

We therefore used the same pre-processing method to derive scenarios suitable for reliable broadcast from other traces contained in various datasets, two of which are also considered in this paper:

- The *upmc/rollernet* traceset [3] contains Bluetooth sightings collected during a roller tour in Paris in 2006. From this traceset we derived a *Rollernet-RB* contact scenario that involves 60 nodes among which reliable broadcast is possible over a 2h32' timespan.
- The *ubs/vbn* dataset [18] contains data about the mobility of buses in the urban area of Vannes[2], France. Information about bus lines has been extracted automatically from OpenStreetMap and from the bus company's timetables. From this dataset we derived a *VBN-RB* contact scenario that involves 49 buses among which reliable broadcast is possible (assuming a transmission range of 200 meters) over a 14h14' timespan.

The *Exp3-RB*, *Rollernet-RB*, and *VBN-RB* contact scenarios are all included in the archive file associated with this paper[3].

[2] VBN: Vannes Bus Network.
[3] http://casa-irisa.univ-ubs.fr/download/ACS_1.zip

Metrics	AW-Set/Exp3-RB
Nb of *add* events	5 396
Nb of *rmv* events	5 396
Total nb of events	$N_u = 10\,792$
First event	00h00'22"
Last event	47h49'20"

Table 2: Statistics about the application scenario defined for the Exp3-RB contact scenario

5.1.2 Application scenarios

An application scenario determines what kind of CRDT is considered during an experiment, and what is the timeline of the updates applied to each replica in this experiment. Without loss of generality, in the following we consider scenarios that only involve AW-Sets (Add-Wins Sets). Other kinds of CRDTs (i.e., other kinds of sets, maps, lists, graphs, etc.) would not behave much differently than AW-Sets in the experiments whose results are presented below.

An AW-Set is a distributed set to which items can either be added or removed [26]. Since *add* and *remove* operations do not commute, the causal context must be maintained in the metadata in order to preserve the causality of events. Besides, a rule must be defined to arbitrate between concurrent *add* and *remove*. In an Add-Wins Set, the rule is that *add* takes precedence (i.e., wins) over *remove*. A typical approach to ensure causal ordering in such a CRDT consists in maintaining a version vector in each replica, and tagging each new event with the current value of this vector. The version vector can also serve as a digest of the replica state: by comparing the state of a local replica with the digest (version vector) received from a remote replica, it is possible to determine which events stored in the local replica occurred before, after, or concurrently with the events stored in the remote replica.

Whatever the application scenario considered, it is important to make sure that any update applied to a replica has a chance to propagate fully in the network, and thus reach any other replica. As mentioned in the former section, the *Exp3-RB* contact scenario supports reliable broadcast for 48h01'. Any update applied to a replica after that deadline will reach only a fraction of the other replicas.

Taking this constraint into account, we defined an application scenario whereas all nodes considered in *Exp3-RB* implement an AW-Set. A new item is added by each node to its own replica every 20 minutes, starting as soon as the node appears in the network, but not later than 48h01'. For every *add* event, a corresponding *remove* event occurs 30 minutes later. The resulting application scenario includes 5 396 pairs of *add/remove* events (142 pairs per node), distributed regularly over the 48h01' timespan. The combination of this application scenario with the *Exp3-RB* contact scenario is referred to as scenario *AW-Set/Exp3-RB* in the remainder of this paper. Table 2 provides details about this scenario.

Using the same approach, we also produced application scenarios to be run with the Rollernet-RB and VBN-RB contact scenarios. Details about these scenarios are

available in the appendices (Tables 5 and 8). These three application scenarios are included in the archive file associated with this paper.

5.2 Metrics considered

In order to compare how the synchronization algorithms defined in Section 4 can perform when running the above-mentioned scenarios, we will focus on the following metrics:

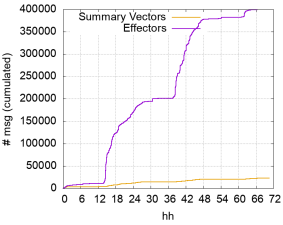
- Number of messages: the number of messages exchanged by neighbor nodes will be considered for each message type, that is:
  - summary vector and effectors when running the OB algorithm;
  - full state messages when running the SB algorithm;
  - digests and delta state messages when running the Δ-SB and Δ-SB_T algorithms.
- Size of messages: for each message type, the size of a message will be expressed in terms of the number of items it contains. This is because container CRDTs such as sets, maps, lists, etc. can be used to store any kind of items (e.g., numerical values, character strings, structured types). Thus, instead of expressing for example the size of an effector message in bytes (which clearly depends on the particular kind of items stored in the CRDT), we consider that this effector carries one item, which is the transcription of the effect of applying one update to the sender's replica. The size of a full state or delta state message will likewise be expressed in terms of the number of updates it contains. Digest messages will be considered as being of constant size (typically O(N) for a digest based on a version vector in a network of N nodes). The size of a summary vector will be expressed as the number of message identifiers it contains, which itself depends on the number of effector messages stored in the sender's cache.
- Cumulated amount of data transferred during an experiment: for each message type, the cumulated amount of data is simply the total number of items transferred via messages of that type during the experiment.
- Time to convergence: for each experiment we will observe how long it takes for all replicas to converge. The idea is to determine if any of the four synchronization methods considered allows the replicas to converge faster.

5.3 Experimentation with the Exp3-RB scenario

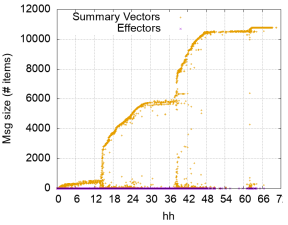
Figure 4 presents the results observed when running the AW-Set/Exp3-RB scenario with the operation-based (OB), state-based (SB), and delta-state-based synchronization algorithms. In the latter case, two variants of delta-state-based synchronization are considered, depending on whether transitive forwarding is disabled (Δ-SB) or enabled (Δ-SB_T).

For each synchronization algorithm the figure shows the evolution of the cumulated number of messages transferred over time (e.g., Fig. 4-OB-a), the size of each message (e.g., Fig. 4-OB-b), and the statistical distribution of message size (e.g., Fig. 4-OB-c, note the log scale along the x axis).

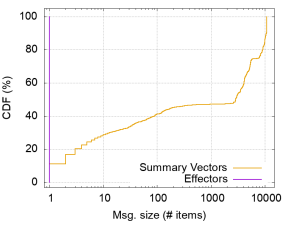
Operation-based synchronization (OB)



(OB-a) Evolution of the number of messages exchanged by neighbor nodes (cumulated)

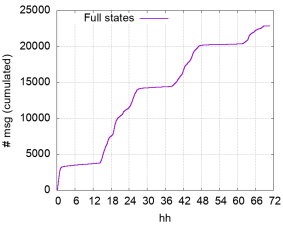


(OB-b) Evolution of message size over time

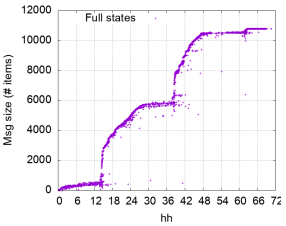


(OB-c) CDF of message size

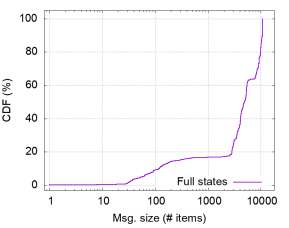
State-based synchronization (SB)



(SB-a) Evolution of the number of messages exchanged by neighbor nodes (cumulated)

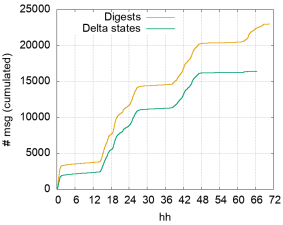


(SB-b) Evolution of message size over time

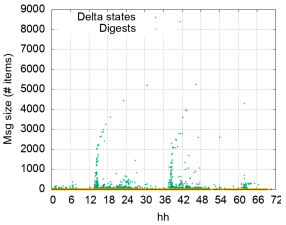


(SB-c) CDF of message size

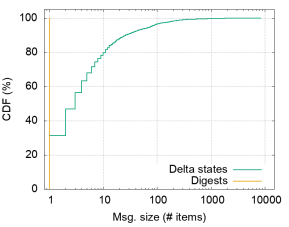
Delta-State-based synchronization (Δ-SB)



(Δ-SB-a) Evolution of the number of messages exchanged by neighbor nodes (cumulated)

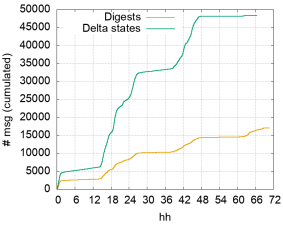


(Δ-SB-b) Evolution of message size over time

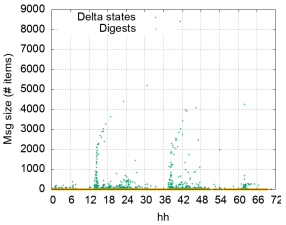


(Δ-SB-c) CDF of message size

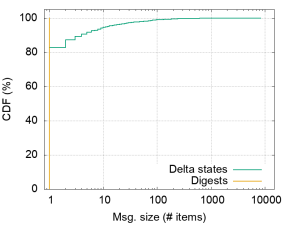
Delta-State-based synchronization with transitivity enabled (Δ-SB$_T$)



(Δ-SB$_T$-a) Evolution of the number of messages exchanged by neighbors nodes (cumulated)



(Δ-SB$_T$-b) Evolution of message size over time



(Δ-SB$_T$-c) CDF of message size

Fig. 4: Synchronizations with the AW-Set/Exp3-RB scenario

Metrics	OB	SB	Δ-SB	Δ-SB$_T$
# summary vectors	22 980	-	-	-
# effectors	399 304	-	-	-
# digests	-	-	22 967	17 005
# full states	-	22 860	-	-
# delta states	-	-	16 365	48 300
Total	422 284	22 860	39 332	65 305

Table 3: Number of messages exchanged while running the AW-Set/Exp3-RB scenario with each kind of synchronization method

Metrics	OB	SB	Δ-SB	Δ-SB$_T$
# items transferred in effectors	399 304	-	-	-
# items transferred in full state messages	-	126 591 078	-	-
# items transferred in delta state messages	-	-	399 304	403 154
Total	399 304	126 591 078	399 304	403 154

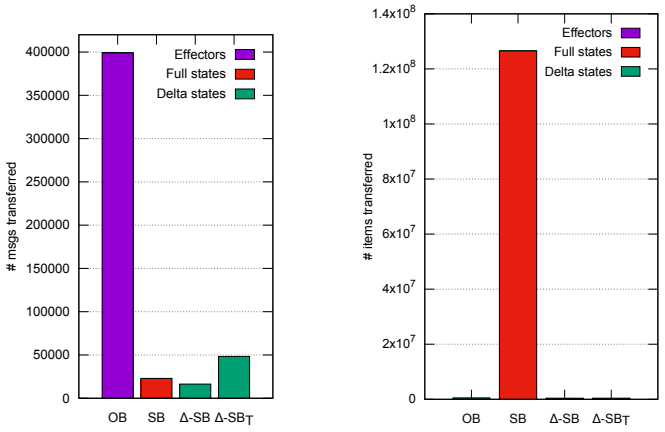
Table 4: Number of items transferred via each kind of message (effector, full state, or delta state) while running the AW-Set/Exp3-RB scenario with each kind of synchronization method

Details about the number of messages transferred with each synchronization method are available in Table 3, and details about the amount of data transferred via each method are available in Table 4. The same information is also presented graphically in Fig. 5.

The convergence of replicas is shown in Fig. 6. More specifically, Fig. 6a shows the evolution of the number of effector messages received by each node when running the operation-based synchronization algorithm, and Fig. 6a shows how long it takes for each node (each replica) to converge. Similar figures are not presented for the SB and Δ-SB algorithms, because they are very similar to what is depicted in Fig. 6. Indeed, the algorithm used to synchronize replicas has very little influence on how fast the replicas converge, as long as the pace of information dissemination in the network is determined by contacts between neighbor nodes. The use of transitive forwarding in the Δ-SB$_T$ algorithm can have an influence, though. This is discussed later in this section.

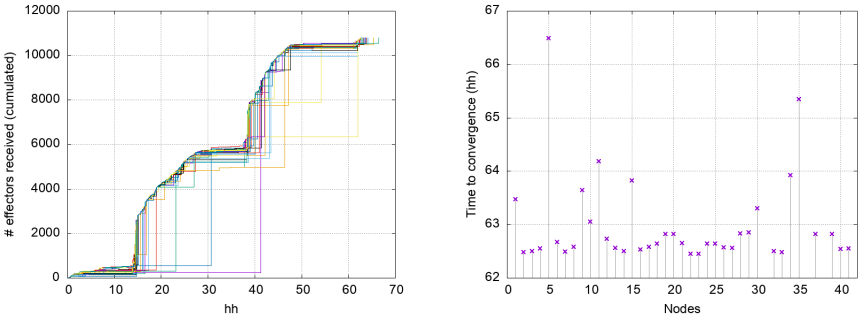
5.3.1 Operation-based synchronization

In the AW-Set/Exp-3RB scenario, the last update applied to a replica occurs at 47h49'20" (see Table 2). In Fig. 4-OB-a it can be observed that the number of messages carrying effectors grows steadily until that time, as new updates are applied to replicas, and effectors are sent accordingly in the network. Once all effectors have been sent, the nodes keep exchanging effectors for a while because the epidemic dissemination of these effectors is still in progress in the network. Once all nodes have received all effectors (which occurs around 66h29' in this particular scenario), the nodes stop exchanging effectors: all replicas of the AW-Set have converged. In fact it can be observed in Fig. 6 that most replicas converge



(a) Number of messages (b) Number of items

Fig. 5: Number of messages and number of items exchanged while running the AW-Set/Exp3-RB scenario with each kind of synchronization method



(a) Evolution of the number of effectors received by each node (b) Time to convergence for each node

Fig. 6: Convergence of replicas when running the AW-Set/Exp3-RB scenario

shortly after 62h30. A few nodes require up to two additional hours to receive all the effectors, and only two nodes (5 and 35) requires far longer delays before they reach convergence. In any case, the delay between the last update is applied to a replica (47h49'20") and the time a replica converges (62h30 or later) is at least 15 hours in this particular scenario. Such delays are not unusual in an opportunistic network, since they are simply the consequence of the way information disseminates in such a network. With a different contact scenario this delay may be significantly shorter, but it may also be far longer.

In Fig 4-OB-a and 4-OB-b it can also be observed that the exchange of effectors between neighbor nodes stops increasing once all replicas have converged, while the traffic pertaining to summary vectors continues in the background.

Operation-based synchronization yields the dissemination of a large number of small messages (effectors). Since each effector message must eventually be received by each node, the number of receive events for effectors is $O(N_n \times N_u)$, where N_n is the number of nodes and N_u the number of updates applied to replicas.

The number of summary vectors exchanged by neighbor nodes is $O(N_c)$, as it depends only on the number of contacts N_c observed in the scenario. The size of summary vectors grows as new effector messages are deposited in each node's cache, but this size is bounded by the number of updates applied to replicas, $O(N_u)$. This is confirmed in Fig. 4-OB-b: the size of summary vectors grows as long as new effectors are sent in the network. Besides, this size does not decrease, for once an effector message is stored in a node's cache, it remains there.

Discussion

While running experiments involving the operation-based (OB) synchronization algorithm, we used a basic epidemic forwarding algorithm that operates in "push" mode. Once a node has received the summary vector of a new neighbor, it determines which effector messages (available in its local cache) can be sent to this neighbor, and then attempts to push all these messages in sequence towards the neighbor. This approach requires very little control traffic, but since each node can be in contact with several neighbors simultaneously, a node may occasionally receive the same message several times from several neighbors. Another approach consists for each node to "pull" the messages it needs from its neighbors. A request is sent specifically for each message, so the requester node can avoid requesting the same message from several neighbors. This approach helps prevent duplicate transmissions, but control traffic is increased significantly.

In the basic epidemic forwarding algorithm we used, the exchange of summary vectors between neighbor nodes occurs whenever two nodes get into contact. This yields background traffic that is maintained even when all replicas have converged. Besides, the size of a summary vector is roughly proportional to the number of messages stored in its sender's cache. As long as new updates are applied to replicas (and thus new effector messages broadcast in the network), the number of messages stored in each node's cache keeps increasing, and the size of summary vectors increases accordingly.

Many variants of the epidemic forwarding algorithm have been proposed in the literature in order to mitigate these problems. The cost of exchanging summary vectors can for example be reduced significantly by having each node maintain a list of the messages it has already exchanged with every other node. Upon a new contact, a node can thus decide what to put exactly in the summary vector sent to its new neighbor. Another approach to reduce the cost of epidemic forwarding is to devise strategies to reduce the number of messages stored in each cache. Each message sent in the network can for example be assigned a set lifetime. Once the lifetime of

a message is over, all copies of this message are purged from the network. Such an approach must however be used with caution, since setting a lifetime too short can prevent a message from reaching all the expected targets. Since the synchronization of operation-based CRDTs requires that all effector messages reach all replicas, assigning a lifetime to these messages is a delicate issue, for this could prevent some of the replicas to ever converge.

5.3.2 State-based synchronization

With state-based synchronization, the only messages exchanged between neighbor nodes when they get into contact are messages that contain the full state of the sender's replica. In Fig. 4-SB-a it can be observed that the nodes keep exchanging the states of their replicas, even after the last update has been applied (47h49'20"). This is because, with pure state-based synchronization, there is no attempt to reduce either the number or the size of the full state messages exchanged by replicas. Indeed, the size of the full state messages exchanged between neighbors grows steadily until all updates have been applied to the AW-Set, afterwards this size stabilizes (see Fig. 4-SB-b).

Discussion

The cost of exchanging full states whenever two nodes meet is clearly visible in Table 4 and in Fig. 5b. Although the number of messages exchanged with the state-based approach is significantly smaller than with the operation-based approach, the total amount of data transferred is far larger. Besides, the nodes keep exchanging full state messages upon every contact, even after all replicas have converged, so if the nodes keep running for a long time, they will also keep exchanging full state messages. The size of each full state message is actually bounded by $O(N_u)$, where N_u is the number of updates applied to replicas, and the total amount of items transferred while running the scenario is $O(N_u \times Nc)$, where N_c is the number of contacts.

5.3.3 Delta-state-based synchronization (no transitive forwarding)

With a delta-state-based CRDT, the synchronization of replicas requires that neighbor nodes exchange digest messages and delta state messages.

In Fig. 4-Δ-SB-a it can be observed that the nodes keep exchanging digest messages after the last update has been applied (47h49'20"). However, since the digest of an AW-Set is typically a version vector, a digest message is a very small control message, as shown in Fig. 4-Δ-SB-b. The exchange of delta state messages stops shortly after the last update has been applied, though. This is the consequence of having the nodes exchange digests prior to any actual delta state message. Besides, the messages carrying delta states are usually far smaller than those carrying full states (in state-based synchronization).

The advantage of using the delta-state-based approach is obvious. First, the number of delta state messages transferred between neighbor nodes is smaller than

the number of full state messages transferred with the state-based approach (see Table 4). Second, delta state messages are far smaller than full state messages, as it can be seen when comparing Fig. 4-SB-b and Fig. 4-Δ-SB-b.

With the state-based (SB) approach, about 99% of the state messages exchanged by neighbor nodes contain more than 30 items (each item being the consequence of an update), and about 84% of these messages contain more than 1 000 items (see Fig. 4-SB-c). With the delta-state-based (Δ-SB) approach, 31% of the state messages contain only one item, 78% of these messages contain less than 10 items, and only 4% of them contain more than 100 items (see Fig. 4-Δ-SB-c).

Finally, the total amount of items transferred via delta state messages with the Δ-SB approach is the same as the amount of effectors transferred with the OB approach (see Table 4). The delta-state-based approach is therefore as effective as operation-based approach as far as transmissions are concerned (i.e., no duplicate transmission of items), while requiring a far smaller number of messages (about 9.3% with that particular scenario).

Discussion

The advantage of allowing neighbor nodes to exchange digests (version vectors) before any delta state is transferred is that it prevents the nodes from exchanging duplicate information. This is especially true once all replicas have converged (see Fig. 4-Δ-SB-b), for in that case the only remaining traffic between neighbor nodes is the background traffic composed of digest messages. Before all nodes have converged, the exchange of delta states (instead of full states) also contributes to reduce the amount of data exchanged by neighbor nodes (see Table 4 and Fig. 5b).

The number of digest messages exchanged by neighbor nodes is $O(N_c)$. The number of delta state messages is $O(N_u)$. The size of each delta state message is $O(N_u)$, but in practice delta state messages are far smaller than full state messages, as shown in Fig. 4-Δ-SB-b).

5.3.4 Delta-state-based synchronization with transitive forwarding

The last two columns in Table 3 show the difference in the number of messages transferred between neighbor nodes, depending on whether transitive forwarding is used or not. The number of digest messages exchanged by neighbor nodes is slightly lower with transitive forwarding. This is because of the faster dissemination of deltas, which reduces the probability that when a node receives a digest from a new neighbor, it also needs to send its own digest to this neighbor (lines 10-12 in Code 5). The number of delta state messages is increased significantly, since many small deltas are exchanged whenever updates are applied to replicas. About 82% of the deltas contain only one item when using transitive forwarding, while it is only 31% when transitive forwarding is not used (see Fig. 4-Δ-SB-c and 4-Δ-SB$_T$-c). Overall, the total number of messages exchanged while running the AW-Set/Exp3-RB scenario is increased by about 66% when transitive forwarding is used, while the total number of items transferred is only increased by 1% (meaning that only 1% of these items are duplicates).

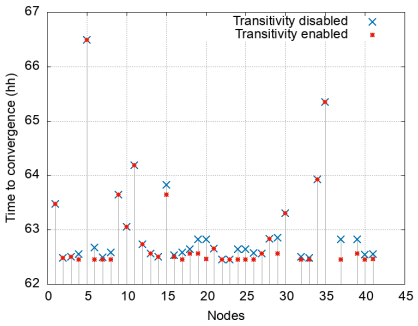


Fig. 7: Influence of transitive forwarding on the time to convergence for each node while running the AW-Set/Exp3-RB scenario

Although transitive forwarding is meant to speed up the dissemination of deltas in the network, thus allowing the replicas to converge faster, it turns out that the benefit of this approach is actually quite limited. Figure 7 shows the difference observed for each node. In most cases, the time to convergence is similar: it takes exactly the same time for a node's local replica to converge, whether deltas are propagated transitively or not. For only a few nodes, a difference can be observed: the local replica reaches the final state a few minutes earlier with transitive forwarding. The difference remains small, though, considering that in this scenario the delay to convergence after the last update is at least 15 hours for all replicas.

5.4 Results observed with the AW-Set/RollerNet-RB and AW-Set/VBN-RB scenarios

The results presented above have been produced with a single contact scenario (namely Exp3-RB). It would be legitimate to assume that the conclusions drawn from these results are not necessarily generalizable. In order to check if similar conclusions can be drawn with other contact scenarios, experiments similar to those conducted with the AW-Set/Exp3-RB scenario have also been conducted using the RollerNet-RB and VBN-RB tracesets mentioned in Section 5.1.

Details about the AW-Set/RollerNet-RB and the AW-Set/VBN-RB experiments are available in Appendices A and B respectively. The three contact scenarios considered in these experiments are quite different in terms of total duration, contacts durations, number of nodes, and frequency and number of events (updates). Yet the general observations made with scenario AW-Set/Exp3-RB can also be made with the two other scenarios. As a general rule, the number of messages exchanged by mobile nodes is far greater with operation-based synchronization than with state-based and delta-state-based synchronization, while the number of items (and thus the amount of data) exchanged by these nodes is exceedingly large with state-based synchronization (see Fig. 11 and 14).

The influence of transitive forwarding in delta-state-based synchronization is clearly negligible with scenario AW-Set/VBN-RB (as shown in Fig. 13). With

scenario AW-Set/RollerNet-RB it may seem at first glance that transitive forwarding has a significant impact (see Fig. 10). Yet it should be noticed that the time to convergence with this scenario is reduced by about 1 minute for most replicas, while the duration of the entire scenario is over 150 minutes.

6 Discussion

The experimental results presented above confirm that the use of CRDTs in an OppNet is indeed doable and practical. This paves the road for the development of distributed applications that rely not only on plain message passing, but also on data sharing

Operation-based synchronization is easy to implement on top of an opportunistic communication layer that supports reliable causal broadcast. Its main advantage is that the total amount of data transferred in the network is kept at a minimum, since each effector produced when applying an update to a replica is eventually received once and once only by every other replica. In contrast, since a new effector message is broadcast for each new update, the number of messages that propagate in the network can be very large. Besides, the information maintained in a CRDT is actually stored twice on each node: once in the local replica's internal state, and once again in the local cache of effector messages maintained by the communication middleware. This may be an issue when the items stored in the CRDT are large (e.g., images, audio files). This space overhead could be mitigated by implementing a communication layer that would directly access the data maintained in the local replica, rather than maintain copies of these data in a message cache. But this approach would prevent application designers from leveraging existing opportunistic communication middleware such as DoDWAN (Document Dissemination in Wireless Ad hoc Networks [22]), IBR-DTN [30], or aDTN (active Delay Tolerant Network [4]).

Pure state-based synchronization should be avoided in opportunistic networks, because the exchange of full state messages whenever two nodes get into contact yields significant transmission overhead. In networks whose topology is relatively stable, such as those based on the standard Internet, the synchronization of replicas can usually be application-driven: the periodicity of full state transmissions is therefore adjusted based on the needs of the application itself. In an opportunistic network, though, the synchronization of replicas must essentially be driven by the contacts between mobile nodes. When these contacts are frequent, state-based synchronization leads to an overuse of the transmission channels.

State-based synchronization can however be a viable approach for simple CRDTs whose size does not grow over time, such as GO-Counters (Grow-Only Counters). For such CRDTs, exchanging the entire state of two replicas when their hosts get into contact is not necessarily more costly than exchanging summary vectors (with the OB approach) or digests (with the Δ-SB approach) upon every contact.

Delta-state-based synchronization provides a good tradeoff between reducing the number of messages exchanged by neighbor nodes and reducing the global

amount of data exchanged by these nodes. It compares with operation-based synchronization as far as the global amount of data (number of items) transferred is concerned, while requiring much fewer messages.

The use of transitive forwarding in delta-state-based synchronization only yields limited benefits. The time to convergence can indeed be reduced for a few replicas, but most of the time this reduction is marginal. Transitive forwarding should only be used for use cases in which it is of prime importance that all nodes reach convergence as soon as possible, or for use cases in which the intermediate states reached by a replica present an interest for the application and should therefore be updated as early as possible.

The experiments reported in this section have been conducted using opportunistic networking scenarios that have been specially tuned so as to guarantee the network-wide dissemination of messages, since this is a formal and explicit requirement of conflict-free replicated datatypes. In future work it would however be interesting to consider relaxing this constraint, investigating how CRDTs (or data structures strongly inspired from CRDTs) could be used in scenarios where all replicas cannot necessarily reach the same final state, though a subset of them actually reach the same state, or get "close enough" to the same state. It would likewise be interesting to consider data structures that do not necessarily ever reach a final state because they are updated continuously over time, but that pass by the same succession of intermediate states.

7 Conclusion

In this paper we have addressed the problem of synchronizing CRDTs (Conflict-Free Replicated Datatypes) in an opportunistic network (OppNet), leveraging transient contacts between mobile nodes to synchronize the replicas maintained on these nodes. Several types of CRDTs have been considered (namely, operation-based, state-based, and delta-state-based CRDTs), and for each type a specific synchronization algorithm has been proposed. For delta-state-based CRDTs, the algorithm can optionally propagate information transitively between neighbor nodes.

Experiments have been conducted by running these algorithms to synchronize Add-Wins Sets in an emulated opportunistic networking setting, using contact scenarios derived from several realistic tracesets. The results show that all forms of synchronization ensure the convergence of replicas in the same time frame. Delta-state-based synchronization globally outperforms operation-based and pure state-based synchronization, though. It compares with operation-based synchronization as far as the global amount of data (number of items) transferred is concerned, while requiring much fewer messages. State-based synchronization yields significant transmission overhead, because it requires exchanging entire states whenever two mobile nodes get into contact. It should therefore only be used for CRDTs whose size is small and almost stable over time, such as GO-Counters.

Using transitive forwarding in delta-state-based synchronization was expected to speed up the convergence of replicas, but results show that it actually only brings

very little benefit: the time to convergence of replicas is only reduced marginally, while the number of delta messages exchanged by neighbor nodes is increased significantly. Transitive forwarding should therefore only be enabled in application that require that replicas converge as fast as possible.

Declarations

Competing interests

The authors have no competing interests to declare that are relevant to the content of this article.

Funding

This work was supported by the French ANR (Agence Nationale de la Recherche) under grant number ANR-16-CE25-0005-02.

Data Availability statement

The datasets generated during and/or analyzed during this study are included in this published article and its supplementary information file "Application and contact scenarios for the Exp3-RB, Rollernet-RB, and VBN-RB experiments" available at https://casa-irisa.univ-ubs.fr/download/ACS_1.zip.

References

[1] Almeida PS, Shoker A, Baquero C (2015) Efficient State-Based CRDTs by Delta-Mutation. In: International Conference on Networked Systems (NETYS 2015), Agadir, Morroco. Springer, pp 62–76, https://doi.org/10.1007/978-3-319-26850-7_5

[2] Almeida PS, Shoker A, Baquero C (2018) Delta State Replicated Data Types. Journal of Parallel and Distributed Computing 111:162–173. https://doi.org/https://doi.org/10.1016/j.jpdc.2017.08.003

[3] Benbadis F, Leguay J (2009) CRAWDAD Dataset upmc/rollernet (v. 2009-02-02). CRAWDAD Wireless Network Data Archive. https://crawdad.org/upmc/rollernet/20090202

[4] Borrego C, Robles S, Fabregues A, et al (2015) A Mobile Code Bundle Extension for Application-Defined Routing in Delay and Disruption Tolerant Networking. Computer Networks 87:59–77. https://doi.org/10.1016/j.comnet.2015.05.017

[5] Brocco A (2021) The Document Chain: a Delta CRDT Framework for Arbitrary JSON Data. In: 29th Italian Symposium on Advanced Database

Systems (SEBD21), Pizzo Calabro (VV), Italy, vol 2994. CEUR Workshop Proceedings, https://doi.org/10.1109/ICTA.2017.8336061

[6] Casteigts A, Chaumette S, Ferreira A (2010) Characterizing Topological Assumptions of Distributed Algorithms in Dynamic Networks. In: Structural Information and Communication Complexity (SIROCCO 2010), Sirince, Turkey, LNCS, vol 5869. Springer, pp 126–140, https://doi.org/10.1007/978-3-642-11476-2_11

[7] Casteigts A, Flocchini P, Santoro N, et al (2012) Time-Varying Graphs and Dynamic Networks. International Journal of Parallel, Emergent and Distributed Systems 27(5):387–408. https://doi.org/10.1080/17445760.2012.668546

[8] Ciobanu RI, Marin RC, Dobre C, et al (2014) ONSIDE: Socially-aware and Interest-based Dissemination in Opportunistic Networks. In: Network Operations and Management Symposium (NOMS), Krakow, Poland. IEEE, https://doi.org/10.1109/NOMS.2014.6838390

[9] Conti M, Giordano S, May M, et al (2010) From Opportunistic Networks to Opportunistic Computing. IEEE Communications Magazine 48(9):126–139. https://doi.org/10.1109/MCOM.2010.5560597

[10] Costea M, Ciobanu RI, Marin RC, et al (2016) Causal and Total Order in Opportunistic Networks, IGI Global, chap Emerging Innovations in Wireless Networks and Broadband Technologies, pp 221–262. https://doi.org/10.4018/978-1-4666-9941-0.ch010

[11] Costea M, Ciobanu RI, Marin RC, et al (2017) Total Order in Opportunistic Networks. Concurrency and Computation: Practice and Experience 29(10). https://doi.org/10.1002/cpe.4056

[12] Datta A, Quarteroni S, Aberer K (2004) Autonomous Gossiping: a Self-Organizing Epidemic Algorithm for Selective Information Dissemination in Mobile Ad-Hoc Networks. In: 1st International Conference on Semantics of a Networked World (ICSNW'04). Springer, Paris, France, no. 3226 in LNCS, pp 126–143, https://doi.org/10.1007/978-3-540-30145-5_8

[13] Dragojević A, Narayanan D, Nightingale EB, et al (2015) No Compromises: Distributed Transactions with Consistency, Availability, and Performance. In: 25th Symposium on Operating Systems Principles (SOSP'15), Copper Mountain Resort, CO, USA. ACM, pp 54–70, https://doi.org/10.1145/2815400.2815425

[14] Enes V, Baquero C, Almeida PS, et al (2016) Join Decompositions for Efficient Synchronization of CRDTs after a Network Partition: Work in

Progress Report. In: 1st Workshop on Programming Models and Languages for Distributed Computing (PMLDC'16), Rome, Italy, https://doi.org/10.1145/2957319.2957374

[15] Enes V, Almeida PS, Baquero C, et al (2019) Efficient Synchronization of State-Based CRDTs. In: 35th International Conference on Data Engineering (ICDE'19), Paris, France. IEEE, pp 148–159, https://doi.org/10.1109/ICDE.2019.00022

[16] Fall K (2004) Messaging in Difficult Environments. Tech. Rep. IRB-TR-04-019, Intel Research Berkeley

[17] Guidec F, Launay P, Mahéo Y (2021) Causal and *Delta*-Causal Broadcast in Opportunistic Networks. Future Generation Computer Systems 118:142–156. https://doi.org/10.1016/j.future.2020.12.024

[18] Guidec F, Launay P, Mahéo Y (2021) CRAWDAD Dataset ubs/vbn (v. 2021-12-16). CRAWDAD Wireless Network Data Archive. https://crawdad.org/ubs/vbn/2021-12-16/vbn_200

[19] Keränen A, Ott J, Kärkkäinen T (2009) The ONE Simulator for DTN Protocol Evaluation. In: 2nd International Conference on Simulation Tools and Techniques (SIMUTools'09), Rome, Italy. ICST, https://doi.org/10.4108/ICST.SIMUTOOLS2009.5674

[20] Li T, Kouyoumdjieva ST, Karlsson G, et al (2019) Data Collection and Node Counting by Opportunistic Communication. In: IFIP Networking Conference (Networking 2019), Warsaw, Poland. IEEE, pp 1–9, https://doi.org/10.23919/IFIPNetworking.2019.8816851

[21] van der Linde A, Leitão J, Preguiça N (2016) *Delta*-CRDTs: Making *delta*-CRDTs Delta-based. In: 2nd Workshop on the Principles and Practice of Consistency for Distributed Data (PaPoC 2016), London, United Kingdom. ACM, https://doi.org/10.1145/2911151.2911163

[22] Mahéo Y, Le Sommer N, Launay P, et al (2012) Beyond Opportunistic Networking Protocols: a Disruption-Tolerant Application Suite for Disconnected MANETs. In: 4th Extreme Conference on Communication (ExtremeCom'12). ACM, Zürich, Switzerland, pp 1–6

[23] Moniz H, Leitão J, Dias RJ, et al (2017) Blotter: Low Latency Transactions for Geo-Replicated Storage. In: 26th International Conference on World Wide Web (WWW'17), Perth, WA, Australia. ACM, https://doi.org/10.1145/3038912.3052603

[24] Nicolaescu P, Jahns K, Derntl M, et al (2016) Near Real-Time Peer-to-Peer Shared Editing on Extensible Data Types. In: 19th International Conference on Supporting Group Work (GROUP'16), Sanibel Island, FL, USA. ACM, pp 39–49, https://doi.org/10.1145/2957276.2957310

[25] Pelusi L, Passarella A, Conti M (2006) Opportunistic Networking: Data Forwarding in Disconnected Mobile Ad Hoc Networks. IEEE Communications Magazine 44(11):134–141. https://doi.org/10.1109/MCOM.2006.248176

[26] Preguiça N (2018) Conflict-free Replicated Data Types: an Overview. Arxiv Preprint. https://arxiv.org/abs/1806.10254

[27] Rinberg A, Solomon T, Shlomo R, et al (2022) DSON: JSON CRDT Using Delta-Mutations for Document Stores. Proceedings of the VLDB Endowment 15(5):1053–1065. https://doi.org/10.14778/3510397.3510403

[28] Robin CEA, Romero VM (2018) DTNDocs: A Delay Tolerant Peer-to-Peer Collaborative Editing System. In: 32nd International Conference on Information Networking (ICOIN), Chiang Mai, Thailand, pp 92–97, https://doi.org/10.1109/ICOIN.2018.8343092

[29] Sánchez-Carmona A, Guidec F, Launay P, et al (2018) Filling in the Missing Link between Simulation and Application in Opportunistic Networking. Journal of Systems and Software 142:57–72. https://doi.org/10.1016/j.jss.2018.04.025

[30] Schildt S, Morgenroth J, Pöttner WB, et al (2011) IBR-DTN: A Lightweight, Modular and Highly Portable Bundle Protocol Implementation. Electronic Communications of the EASST 37:1–11

[31] Scott J, Gass R, Crowcroft J, et al (2009) CRAWDAD Dataset cambridge/haggle (v. 2009-05-29). CRAWDAD Wireless Network Data Archive. https://crawdad.org/cambridge/haggle/20090529/imote

[32] Shapiro M, Preguiça N, Baquero C, et al (2011) A Comprehensive Study of Convergent and Commutative Replicated Data Types. Tech. Rep. 7506, INRIA

[33] Vahdat A, Becker D (2000) Epidemic Routing for Partially Connected Ad Hoc Networks. Tech. Rep. CS-200006, Duke University, Durham, USA

[34] Weiss S, Urso P, Molli P (2009) Logoot: A Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks. In: 29th International Conference on Distributed Computing Systems (ICDCS'09), Montreal, Canada. IEEE, pp 404–412, https://doi.org/10.1109/ICDCS.2009.75

A AW-Set/RollerNet-RB scenario

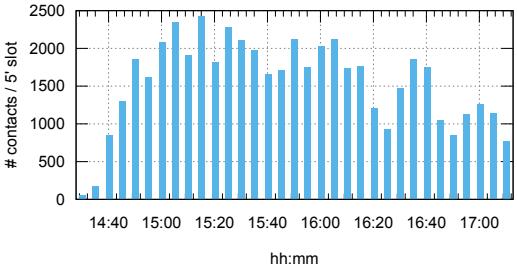


Fig. 8: Evolution of the number of contacts per 15 minute slot in the Rollernet-RB contact scenario

Metrics	Values (* = min / max / avg / stdev)
Duration of the scenario	02h46'30"
Nb of nodes	60
Number of contacts	51 355
Durations of contacts	1" / 8'09" / 0'14" / 0'16" (*)
Number of inter-contacts	49 616
Durations of inter-contacts	1" / 1h53'17" / 3'42" / 8'03" (*)

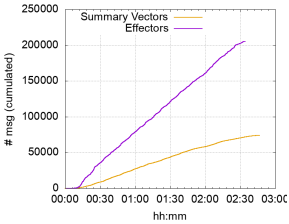
AW-Set/Rollernet-RB	
Nb of *add* events	1 740
Nb of *rmv* events	1 740
Total nb of events	3 480
First event	5"
Last event	2h32'31"

Table 5: Statistics about the Rollernet-RB contact scenario (derived from the *upmc/rollernet* traceset) and details about the defined application scenario

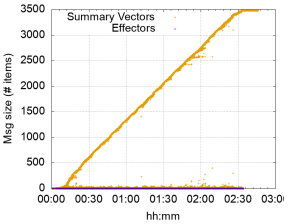
Metrics	OB	SB	Δ-SB	Δ-SB$_T$
# summary vectors	74 166	-	-	-
# effectors	205 320	-	-	-
# digests	-	-	73 896	55 130
# full states	-	74 152	-	-
# delta states	-	-	45 366	110 602
Total	279 486	74 152	119 262	165 732

Table 6: Number of messages exchanged while running the AW-Set/Rollernet-RB scenario with each kind of synchronization method

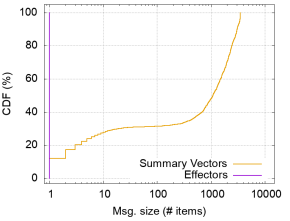
Operation-based synchronization (OB)



(OB-a) Evolution of the number of messages exchanged by neighbor nodes (cumulated)

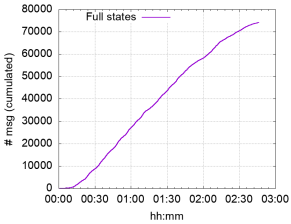


(OB-b) Evolution of message size over time

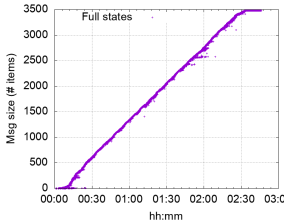


(OB-c) CDF of message size

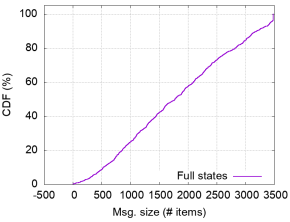
State-based synchronization (SB)



(SB-a) Evolution of the number of messages exchanged by neighbor nodes (cumulated)

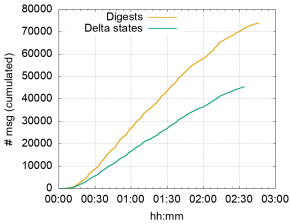


(SB-b) Evolution of message size over time

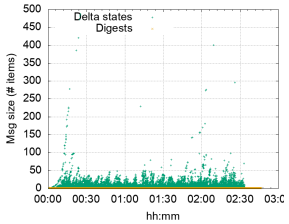


(SB-c) CDF of message size

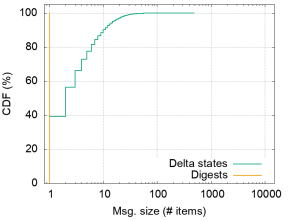
Delta-State-based synchronization (Δ-SB)



(Δ-SB-a) Evolution of the number of messages exchanged by neighbor nodes (cumulated)

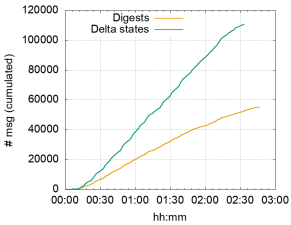


(Δ-SB-b) Evolution of message size over time

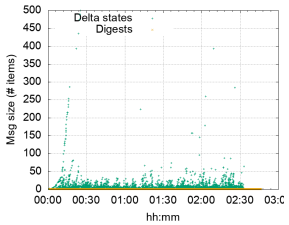


(Δ-SB-c) CDF of message size

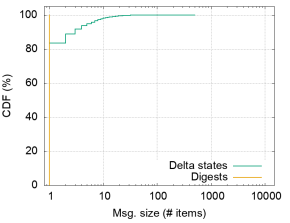
Delta-State-based synchronization with transitivity enabled (Δ-SB$_T$)



(Δ-SB$_T$-a) Evolution of the number of messages exchanged by neighbors nodes (cumulated)



(Δ-SB$_T$-b) Evolution of message size over time



(Δ-SB$_T$-c) CDF of message size

Fig. 9: Synchronizations with the AW-Set/RollerNet-RB scenario

Metrics	OB	SB	Δ-SB	Δ-SB$_T$
# items transferred in effectors	205 320	-	-	-
# items transferred in full state messages	-	134 284 645	-	-
# items transferred in delta state messages	-	-	205 320	205 340
Total	205 320	134 284 645	205 320	205 340

Table 7: Number of items transferred via each kind of message (effector, full state, or delta state) while running the AW-Set/Rollernet-RB scenario with each kind of synchronization method

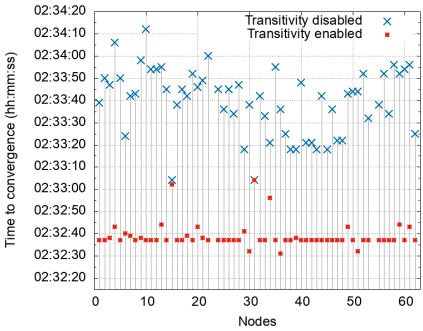
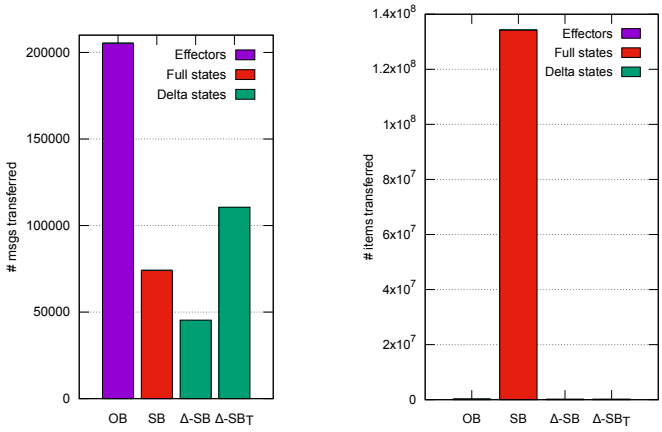


Fig. 10: Influence of transitive forwarding on the time to convergence for each node while running the AW-Set/Rollernet-RB scenario



(a) Number of messages (b) Number of items

Fig. 11: Number of messages and number of items exchanged while running the AW-Set/Rollernet-RB scenario with each kind of synchronization method

B AW-Set/VBN-RB scenario

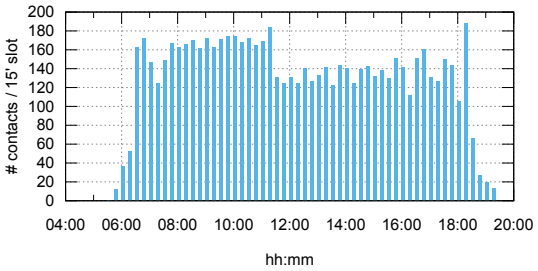


Fig. 12: Evolution of the number of contacts per 15 minute slot in the VBN-RB contact scenario

Metrics	Values (* = min / max / avg / stdev)
Duration of the scenario	14h14'00"
Nb of nodes	49
Number of contacts	7351
Durations of contacts	1" / 4h54'57" / 1'41" / 5'37" (*)
Number of inter-contacts	6 270
Durations of inter-contacts	1" / 12h06'19" / 1h14'52" / 1h34'07" (*)

AW-Set/VBN-RB	
Nb of *add* events	28 493
Nb of *rmv* events	28 493
Total nb of events	56 986
First event	0'04"
Last event	10h09'59"

Table 8: Statistics about the VBN-RB contact scenario (derived from the *ubs/vbn* traceset) and details about the defined application scenario

Metrics	OB	SB	Δ-SB	Δ-SB$_T$
# summary vectors	12 189	-	-	-
# effectors	2 735 328	-	-	-
# digests	-	-	12 078	10 570
# full states	-	12 167	-	-
# delta states	-	-	9 583	88 658
Total	27 747 517	12 167	21 661	99 228

Table 9: Number of messages exchanged while running the AW-Set/VBN-RB scenario with each kind of synchronization method

Metrics	OB	SB	Δ-SB	Δ-SB$_T$
# items transferred in effectors	2 735 328	-	-	-
# items transferred in full state messages	-	310 833 438	-	-
# items transferred in delta state messages	-	-	2 735 328	2 737 483
Total	2 735 328	310 833 438	2 735 328	2 737 483

Table 10: Number of items transferred via each kind of message (effector, full state, or delta state) while running the AW-Set/VBN-RB scenario with each kind of synchronization method

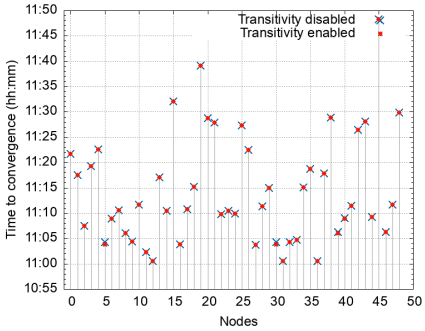
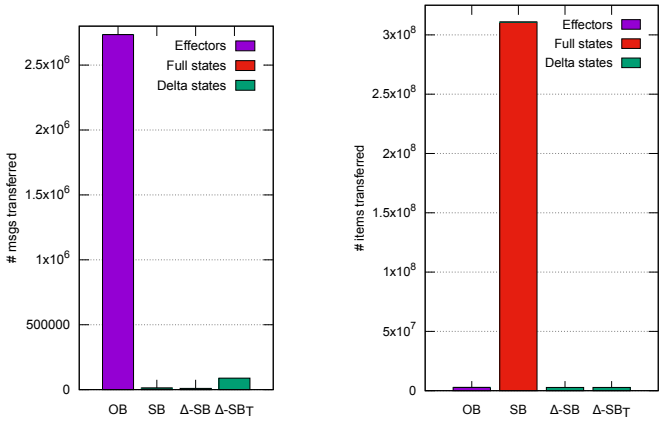


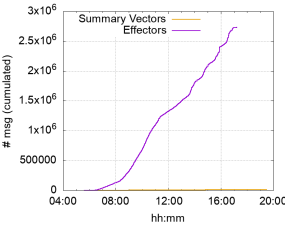
Fig. 13: Influence of transitive forwarding on the time to convergence for each node while running the AW-Set/VBN-RB scenario



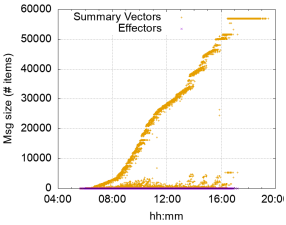
(a) Number of messages (b) Number of items

Fig. 14: Number of messages and number of items exchanged while running the AW-Set/VBN-RB scenario with each kind of synchronization method

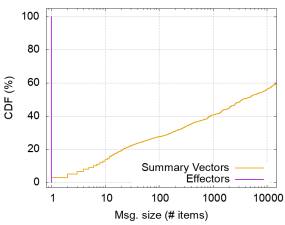
Operation-based synchronization (OB)



(OB-a) Evolution of the number of messages exchanged by neighbor nodes (cumulated)

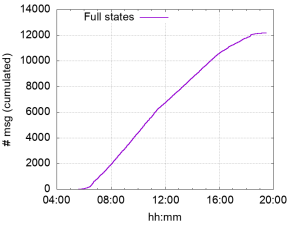


(OB-b) Evolution of message size over time

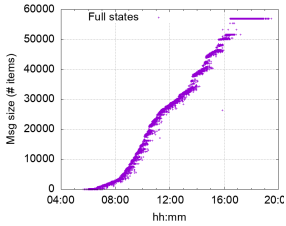


(OB-c) CDF of message size

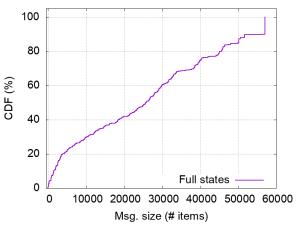
State-based synchronization (SB)



(SB-a) Evolution of the number of messages exchanged by neighbor nodes (cumulated)

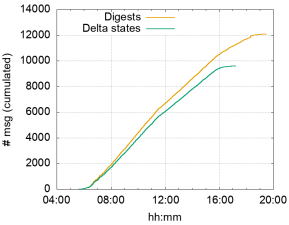


(SB-b) Evolution of message size over time

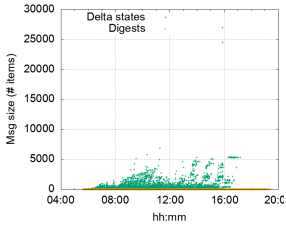


(SB-c) CDF of message size

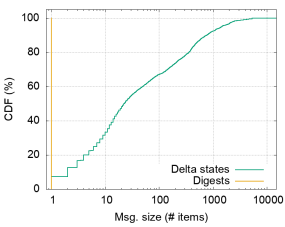
Delta-State-based synchronization (Δ-SB)



(Δ-SB-a) Evolution of the number of messages exchanged by neighbor nodes (cumulated)

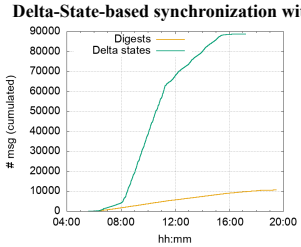


(Δ-SB-b) Evolution of message size over time

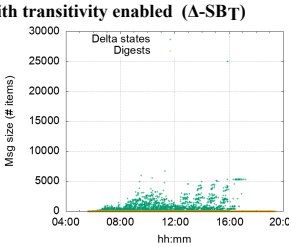


(Δ-SB-c) CDF of message size

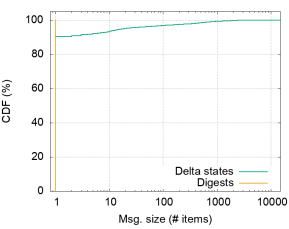
Delta-State-based synchronization with transitivity enabled (Δ-SB$_T$)



(Δ-SB$_T$-a) Evolution of the number of messages exchanged by neighbors nodes (cumulated)



(Δ-SB$_T$-b) Evolution of message size over time



(Δ-SB$_T$-c) CDF of message size

Fig. 15: Synchronizations with the AW-Set/VBN-RB scenario