CRDT-based Collaborative Editing in OppNets: a Practical Experiment

Yves Mahéo, Frédéric Guidec, Camille Noûs

CRDT-based Collaborative Editing in OppNets: a Practical Experiment

Yves Mahéo
Irisa, Université Bretagne Sud
Vannes, France
email: yves.maheo@univ-ubs.fr

Frédéric Guidec
Irisa, Université Bretagne Sud
Vannes, France
email: frederic.guidec@univ-ubs.fr

Camille Noûs
Laboratoire Cogitamus
Vannes, France
email: camille.nous@cogitamus.fr

***Abstract*—Developing distributed applications for opportunistic networks (OppNets) has mainly relied on the message passing paradigm so far. Yet the sharing of distributed data is an alternative worth considering, and Conflict-Free Replicated Datatypes (CRDTs) are interesting candidates for this purpose. A CRDT is a distributed data type that supports optimistic replication, and whose global consistency can be maintained based solely on occasional synchronizations of replicas. In an OppNet, these synchronizations can be driven by the contacts between mobile devices, without any need for network-wide routing. CRDTs can thus serve as interesting software building blocks to develop distributed applications for OppNets. In this paper, we demonstrate the feasibility of using CRDTs in OppNets by presenting an experiment conducted in real conditions, involving a collaborative editing application in which communication relies exclusively on opportunistic contacts between laptops. The software elements used in the experiment consist mainly of a CRDT-based text editor, and a communication module that supports the synchronization between laptops, so as to ensure the eventual consistency of the shared document. Experimentation results are detailed, that confirm the viability and usability of this approach.**

***Keywords*—Opportunistic networking; Experiment; CRDT; Collaborative editing.**

I. Introduction

Opportunistic networks (OppNets) are infrastructure-less networks composed of mobile devices that communicate via direct device-to-device radio transmissions. Due to the sparse or irregular distribution of the devices, such networks are often partitioned, and do not provide permanent end-to-end connectivity. Yet, network-wide communication is made possible following the store, carry and forward principle: when a contact occurs between two devices, the opportunity to exchange messages can be seized. Received information can be stored locally, so that, the device that carries it can forward it later to another mobile device at the occasion of new contact. Developing applications for OppNets is challenging because of the inherent asynchronism of the communications, with potential large delays needed for reaching certain devices, from one to the next, according to their movements.

A number of use cases have been envisaged for opportunistic networks, namely when traditional network infrastructure is not available (e.g., communication for disaster relief or in remote areas) or when it is not desirable to use this infrastructure (e.g., for data offloading, or to avoid censorship).

To support such use cases, the research activity over the last two decades has primarily focused on message routing or dissemination in an OppNet, assuming that network-wide message passing is a de facto requirement of any distributed application, and assuming somehow that a message-oriented API is the best API for developers.

Yet, just like the High Performance Computing community uses alternatively the message-passing paradigm (typically via the Message Passing Interface [1]), or the shared-memory paradigm, via OpenMP [2] for example), depending on application needs, the developers of distributed applications for OppNets should not be bound to rely on a message-passing API, but they should also be offered the means to work with shared data structures.

A promising class of distributed shared data structures, called Conflict-Free Replicated Datatypes (CRDTs), has appeared recently, stemming from research on distributed databases and peer-to-peer networks [3]. CRDTs are distributed data types (counters, sets, maps, etc.) that support optimistic replication: replicas can be updated locally without any coordination, and synchronized asynchronously. CRDT replicas may temporarily diverge, but information is exchanged asynchronously between them thanks to a synchronization algorithm running in the background. This algorithm guarantees that all replicas eventually reach the same final state, provided the synchronization graph is connected, that is, provided the history of successive synchronizations is such that any update is eventually taken into account by every replica.

In the literature, the papers dealing with CRDTs in OppNets present simulation results [4][5][6][7][8]. The question whether CRDTs can be of practical use in a real OppNet setting is thus as yet unanswered. The objective of this paper is to contribute to answer this question. It describes the different elements of the setting and the results of a real-world experiment carried out in order to assess the possibility to write a document collaboratively, by relying on the implementation of a CRDT deployed in an OppNet. To our knowledge, it is the first time that the use of a real application involving CRDTs in an OppNet is reported. The choice of collaborative editing as a case study is motivated by the fact that it is a demanding distributed application, for respecting the causality of editing operations is not trivial in an OppNet. Besides, several tested off-the-shelf software elements designed for peer-to-peer wired networks can be reused or adapted for OppNets, which makes it easier to develop a robust solution.

Figure 1 illustrates the kind of OppNet we consider: the contributors to the shared document use their laptops over a few days to edit the shared document collaboratively. Laptops

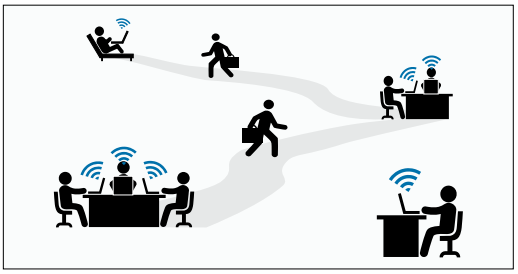

Figure 1. Illustration of collaborative editing in an opportunistic network

are considered here as target devices (rather than smartphones or tablets), because they are more convenient to use for text editing.

A contributor may be isolated while editing the document (e.g., at home), in which case no transmission is possible between his/her laptop and those of other contributors. When several contributors are close to one another (e.g., in adjacent rooms at work), their laptops can synchronize by exchanging messages via direct wireless connections. When a user is on the move, his/her laptop is assumed to be switched off.

The remainder of this paper is organized as follows. Section II introduces the concept of CRDT and details the problem of their synchronization, in particular in OppNets. Section III quickly browses the works related to distributed applications targeting OppNets, and in particular on collaborative applications like text editing. In Section IV, we present the different software elements that support the text editing experiment we have set up. The results of this experiment are detailed in Section V. Section VI concludes the paper.

II. Overview of CRDTs

When a data structure must be replicated in a distributed system, there are different ways to maintain the consistency of the replicas of this data structure. Some systems maintain strong consistency at any time by constraining concurrent updates of replicas, or preventing them altogether. Other systems implement optimistic replication, allowing replicas to diverge temporarily, while ensuring that they will eventually reach a common state (eventual consistency).

Conflict-free Replicated Data Types (CRDTs) support optimistic replication: any replica can be updated locally, at any time, without any coordination with the other replicas. Synchronization occurs in the background, usually periodically and between randomly selected pairs of replicas, by exchanging information about past updates. If the synchronization graph is connected (i.e., the consequence of each update is eventually taken into account by each replica), all replicas eventually reach the same state.

Concurrent updates and concurrency semantics

The implementation of traditional data types (counters, registers, sets, maps, lists, graphs, etc.) as CRDTs has already been addressed in the literature [3][9]. For each CRDT, the updates it can support are identified, and a concurrency semantics is defined. Note that several alternative concurrency semantics can often be defined for the same abstract data type, as shown below.

In order to illustrate how a CRDT can be used as a shared data structure, let us consider a basic example. A Set CRDT implements a shared set, and it can typically support updates *add()* and *rmv()*. Since these updates cannot commute when they are applied to the same element, a concurrency semantics must be defined to resolve conflicts between concurrent *add(x)* and *rmv(x)* updates. Figure 2 shows an example where a Set CRDT (initially empty) is replicated in two replicas R1 and R2. Element a is first added locally to the set in replica R1, while element b is added locally in replica R2. The state is thus temporarily different in R1 and R2, but a synchronization occurs between them, after which they agree that the state of the set is now {a, b}. Note that reconciling state $\{a\}$ (from R1) with state $\{b\}$ (from R2) is not an issue, because although *add(a)* and *add(b)* occurred concurrently in R1 and R2, they apply to distinct elements a and b.

After the first synchronization, element a is removed and then added again in R1, while it is only removed in R2. The state is thus different again in R1 and R2, and this time the last *add(a)* on R1 and *rmv(a)* on R2 conflict, as they occurred concurrently *and* apply to the same element a. If both replicas synchronize again, the final state depends on the concurrency semantics chosen for this shared set. A possible option is to give *add(a)* priority over *rmv(a)*, so that the final state is $\{a, b\}$ in both replicas. Another option is to give *rmv(a)* priority over *add(a)*, so the final state is $\{b\}$. A Set CRDT that gives *add()* priority over *rmv()* is called an Add-wins set in the literature, and the opposite is called a Remove-wins set [3].

Synchronization of replicas

In the simple example shown in Figure 2, only two replicas are considered, so synchronization is only required between these two replicas. In a system that involves a large number of replicas, synchronization must be addressed with caution in order to guarantee that all replicas eventually converge, while maintaining the cost of synchronization at a reasonable level.

Several methods of synchronization have been considered in the literature, each method requiring a specific implementation of CRDTs. In an operation-based CRDT, whenever an operation (update) is applied to a replica, a description of this operation is embedded in a message, which is sent to all other replicas. This approach tends to produce a large number of small messages (each message carrying information about a single update). Besides it requires a system that supports reliable network-wide broadcast, and even causal broadcast if the updates do not commute [3].

In a state-based CRDT, each replica must synchronize periodically with other replicas by sending them its entire state. On each receiver the state of the sender is merged with the local state, using a function that deterministically computes the join (least upper bound) of both states. A major advantage of this approach is that is does not require that each update be

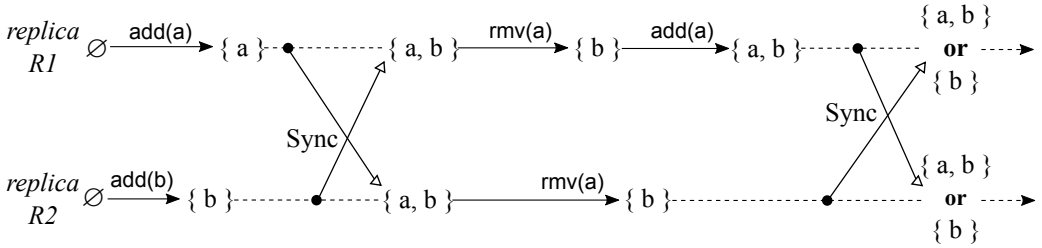

Figure 2. Example of a run involving a Set CRDT replicated in two replicas R1 and R2

transmitted to all replicas, so no broadcast is required. A periodic synchronization of each replica with a few other replicas is sufficient to ensure the eventual convergence of all replicas, as long as the synchronization graph is connected. The main drawback of state-based CRDTs is that shipping entire states between replicas can be costly. Delta-state CRDTs reduce this cost by passing only partial information (a *delta*) about the sender's state (typically, only what is required to allow the target receiver to update its own local state) [10][11][12]. In order to determine what is required by another replica, the sender must first receive a digest of this replica's state, and compare this digest with its own local state. The digest of a state can typically take the form of a state vector.

CRDTs in OppNets

In the literature, it is commonly assumed that CRDTs are to be deployed in Internet-based peer-to-peer networks. Network-wide message routing or broadcast are thus presumed to be available. In such conditions, each replica can either 1) send each update to all other replicas (in an operation-based CRDT), or 2) select randomly any other replica and synchronize with this replica (in a state-based or delta-state-based CRDT).

In an OppNet, two mobile devices can only communicate as long as they are neighbors. Implementing operation-based CRDTs, which involves sending each update to all other devices, is therefore not trivial as it requires at least epidemic delay-tolerant dissemination. As for state-based and delta-state-based CRDTs, they should rely on contact-driven synchronization, each device synchronizing with its neighbors rather than with randomly selected peers.

Synchronization algorithms for operation-based, state-based, and delta-state-based CRDTs in OppNets have been proposed and evaluated based on simulation scenarios in [4]. The results show that although all synchronization methods ensure the eventual consistency of CRDTs, delta-state-based synchronization clearly outperforms the two other modes of synchronization. Operation-based synchronization is easy to implement on top of an opportunistic communication layer that supports reliable causal broadcast. However, it requires the network-wide dissemination of many small messages (one for each update applied to a replica). This can yield a significant communication overhead in an OppNet, as well as storage issues since each message must be maintained in a local cache on each device. State-based and delta-state-based synchronization can be implemented without any multi-hop routing or network-wide dissemination, using only synchronization between neighbors. The cost of state-based synchronization is significant, though, as it requires exchanging entire states between neighbor devices. Delta-state-based synchronization gets the best of both other methods, as the amount of data transfers required to ensure the synchronization of replicas is kept at a minimum, and as there is no need for message routing or message broadcast.

The work presented in [4] has shown that CRDTs can be deployed in an OppNet, and converge as expected in such an environment ; but the given results have been obtained by running simulations. Whether distributed applications based on CRDTs can be of practical use in a real OppNet setting is still to be demonstrated. The purpose of this paper is to contribute to this task.

III. Related work

Running distributed applications in OppNets has been considered in many papers over the last two decades, but, again, most of these papers only present simulation results. Rare are the papers that present communication systems and applications that have been fully implemented, and tested in real conditions. Among these exceptions are [13] and [14], which present DTN systems aiming at providing Internet-like services in very sparsely populated areas, or in disaster-relief scenarios. Distributed applications for content sharing (files, music, news, software components, etc.) in OppNets have likewise been presented in [15][16][17][18].

In the abovementioned applications, the content shared over the network is considered as immutable. In contrast, collaborative editing (or, more generally, collaborative work) requires to share content that can change over time.

Although Web-based solutions such as wikis, Google Docs, Etherpad, etc. have been available for a long time now, these solutions usually rely on a client-server architecture, with central servers whose role is to store shared documents and ensure that concurrent editing of the same document does not yield inconsistencies.

Recognizing that any solution involving servers is hardly applicable in OppNets, an early solution for shared content editing in such networks has been proposed in [19]. In this proposal, a revision control mechanisms is used to merge contributions whenever possible, but user intervention is still required to solve conflicting contributions. In [20] the problem

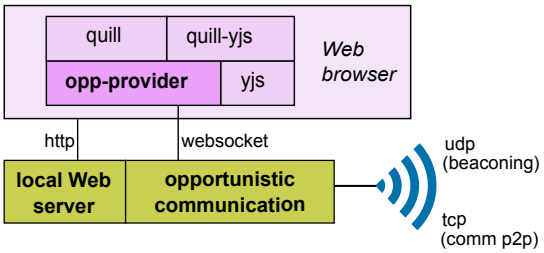

Figure 3. Software elements deployed on each laptop (the elements we have developed are named in boldface)

```
import * as Y from 'yjs'

const ydoc = new Y.doc();

const ytext = ydoc.getText('my document');

ytext.insert(0, 'abcde'); // insert text at pos. 0
ytext.delete(1, 3);       // delete bcd
```

Figure 4. Example of the use of a text CRDT via the Javascript Yjs module

of ensuring total order in OppNets is considered, as a means to support a variant of the Logoot [5] replication algorithm in such networks. A similar approach is presented in [6], using IBR-DTN [21] for opportunistic communication, and a modified LogootSplit [7] algorithm for ensuring consistency in shared content editing. An approach based on OT (Operational Transformation) is considered in [8], and the convergence of an OT-based collaborative editing framework in opportunistic networking scenarios is investigated. Unfortunately, all these papers only present simulation results, and in most cases it is unlikely that a fully-functional collaborative editing solution exists beyond the simulation code.

In the remainder of this paper we describe the implementation of such a fully-functional solution, and demonstrate that it is viable for actual collaborative editing in opportunistic networking conditions.

IV. Design and implementation

The key feature of our experiment is the use of a Conflict-Free Replicated Datatype to guarantee the consistency of the different contributions to the shared document. In our experimental setup, several software elements have been assembled to allow CRDT-based collaborative editing, forming two main layers (see Figure 3). The upper layer runs in a Web browser. It is thus a Web application that combines the editor itself with the implementation of the CRDT that will ensure the consistency of the shared document. The lower layer enables the opportunistic communication between the contributors' laptops, thus allowing the synchronization of the different replicas of the CRDT. The two layers interact via a websocket. In addition, a Web server is deployed locally to supply the code and data of the Web application. Note that the fact that our editing application is based on Web technologies is not a fundamental requirement. It rather results from the choice of an available, extensible and efficient CRDT-based editor.

Opportunistic communication

The main functions of the opportunistic communication layer we developed are, on the one hand, to ensure neighbor detection (i.e., to detect laptops located within the radio range), and on the other hand to establish a bidirectional communication channel between each pair of neighbors.

Neighbor detection is trivially based on the periodic broadcasting, by each laptop, of HELLO messages that contain the identity of the emitting device (typically the hostname), its IP address, and the TCP port number it listens to. This broadcast is only performed at one hop (i.e., without any routing), so that each host can only detect its 1-hop neighbors. HELLO messages are embedded in UDP datagrams, and addressed to a predefined multicast group.

When a laptop receives a HELLO message from a new neighbor with a lower identity (in lexicographic order), it opens a TCP session (with TLS encryption) with this new neighbor. This session will then serve as a bidirectional channel between the two neighbors, as long as they remain in radio contact. When a contact is lost between two neighbors, it can be reestablished later if they meet again.

CRDT-based editor

We chose the Quill text editor [22] as the editing software. This editor is written in HTML/CSS/Javascript. It can be associated with Yjs [23], through the *quill-yjs* binding module, so that the edited text is maintained internally as a CRDT. Yjs is a Javascript implementation of several types of CRDTs (array, map, text, etc.). It is mainly oriented to linear data structures like text [24]. Figure 4 illustrates the manipulation of a piece of text through the use of the Yjs Javascript library, as could be done by the Quill editor (in this figure, and also in Figures 5 and 6, the functions provided by Yjs are in boldface type).

One of the most interesting characteristics of Yjs is its efficient way of encoding a text CRDT as a double chained list of inserted items (sequences of characters identified by Lamport timestamps) accompanied with a set of deletions (simple set of items), which confers a high efficiency for human-produced text manipulation.

Yjs itself is network agnostic: some extra code is required to ensure the synchronization of replicas. This code must be included in a Yjs *provider*. The providers distributed with Yjs are not suited to opportunistic networking, as they target Internet-based contexts (by using typically WebRTC or centralized servers). We therefore developed our own Yjs provider, called *opp-provider*, which supports delta-state-based synchronization. This task was facilitated by two features of Yjs: *updates* and *state vectors*. An update encodes a series of changes in a document that can serve to modify another document. Updates are commutative, associative, and idempotent. These properties are essential to ensure the eventual convergence of all replicas, whatever the order in which

updates are applied to each replica. A state vector characterizes the state of advancement of all replicas, as perceived by one replica. It is essentially a set of Lamport timestamps that captures the causal context. Updates and state vectors are provided to the programmer as opaque structures encoded in a compressed binary format.

```
// Capture the modification event and
// broadcast the update to the neighbors

ydoc.on('update',
      (upd) => {
        broadcast(upd)
      }
```

```
// Receive the update and
// apply it to the document

• On the reception of update upd
  Y.applyUpdate(ydoc, upd)
```

Figure 5. Synchronization performed on a set of neighbors when a contributor modifies the text (the code on the top runs on the laptop where the text is modified)

The *opp-provider* is notified of editing events (issued by the Yjs module) and neighbor discovery events (issued by the underlying opportunistic communication layer). When a contributor edits the text (by inserting or deleting a character), the *opp-provider* immediately broadcasts the corresponding update to all current neighbor laptops (if any). The receiving neighbors can then apply this update on their own replica (see Figure 5). From a contributor's perspective, collaborative editing operates in real time between his/her laptop and neighbor laptops. The updates exchanged between these laptops are embedded in small messages, as each update only pertains to the last operation performed on the sender.

In contrast, when two laptops get into radio contact (and become neighbors), their local states must be synchronized, which usually requires exchanging larger messages. The two new neighbors first exchange state vectors, and then exchange only what the other laptop needs to reach a common state (see Figure 6).

V. Experiment

Experimental conditions

In order to assess whether collaborative editing based on opportunistic communication can be of practical use, we decided to use this approach to write the latest deliverable of a project our research team is involved in. This project involves six permanent staff members, which all agreed to participate in this experiment. These six participants are not always collocated, though. For example, they all have teaching duties, which do not always occur in the same campus or in the same buildings. Besides, most team members work at home part of the time, so they meet only occasionally. As a general rule, two meetings are organized every week, though,

```
vector = Y.encodeStateVector(ydoc)
send(vector, neighbor)

• On the reception of a state vector v
   delta = Y.encodeStateAsUpdate(ydoc, v)
   send(delta, neighbor)

• On the reception of a delta d
   Y.applyUpdate(ydoc, d)
```

```
• On the reception of a state vector v
   sv = v
   vector = Y.encodeStateVector(ydoc)
   send(vector, neighbor)

• On the reception of a delta d
   Y.applyUpdate(ydoc,d)
   delta = Y.encodeStateAsUpdate(ydoc, sv)
   send(delta, neighbor)
```

Figure 6. Delta-state-based synchronization applied when two neighboring laptops enter in contact (the code on the top is executed by the neighbor with the larger identity)

on Tuesday and Friday afternoons, but not all team members attend every meeting.

The appropriate software was installed on each participant's laptop (running Linux), and this laptop was configured so as to support opportunistic communication. More specifically:

- A secondary Wi-Fi interface (small form factor USB dongle) was added to each laptop, and this interface was configured so as to operate continuously in Wi-Fi ad hoc mode. The primary builtin Wi-Fi interface of each laptop therefore remained available for daily activities such as Web browsing, Email, etc. The secondary interface was meant to be used only for opportunistic communication, that is, in that case, for collaborative editing.
- The opportunistic communication layer (cf. Figure 3) was installed on each laptop, configured to use the secondary interface (with self-assigned IPv6 addresses), and run in the background as a *systemd* service.
- The desktop settings on each laptop were configured so that the Web browser opened as soon as the user logged in, and the browser itself was configured so that the Quill/Yjs page was loaded automatically. The participants were asked to keep the browser running as much as possible, and to maintain a window or tab on Quill/Yjs in this browser (note that this did not prevent them to browse other Web sites). The motivation was to ensure that Yjs would keep running in the background, thus ensuring automatically the synchronization between the laptops used in this experiment.
- Each laptop was configured so as to log interesting events, such as the laptop being switched on or off, the discovery of a new neighbor, etc.

Overall, each laptop was configured so that his/her owner could keep using it as usual (browsing the Web, sending and receiving Email, etc.), while collaborative editing was

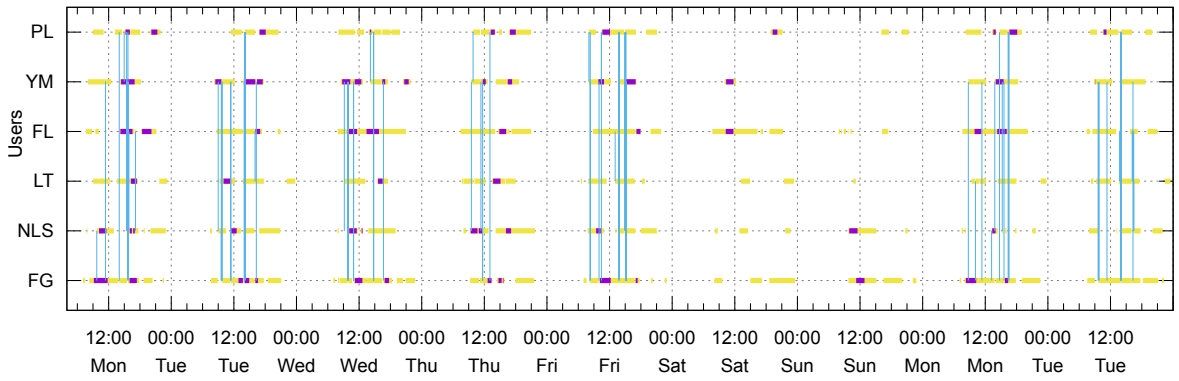

Figure 7. Details of each laptop's activity during the experiment

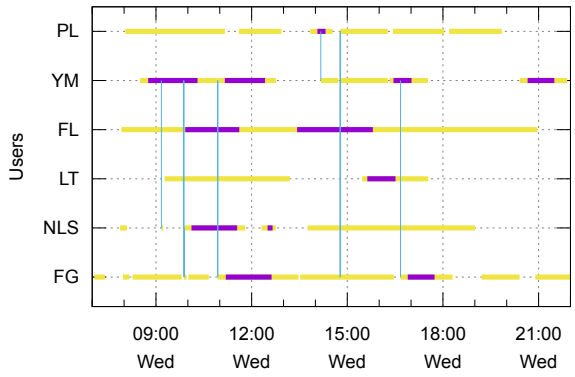

Figure 8. Details of each laptop's activity on Wednesday

performed solely based on opportunistic communication. The general idea was to enlist participants in this experiment without overly disrupting their day-to-day activity.

Results

The whole experiment lasted 9 days, including one weekend (from Monday to the next Tuesday). Figure 7 shows the timelines of each laptop's activity during that period. Each laptop is identified by the initials of its owner's name. In this figure, each horizontal yellow segment corresponds to a period during which the corresponding laptop was up and running. Purple segments mark periods when editing activity was observed on this laptop (i.e., the user was actually editing the shared document). The thin vertical blue lines indicate the beginning of a pairwise radio contact between two laptops, that is, an opportunity for these laptops to synchronize their copies of the shared document.

Figure 8 shows details about the activity observed on a specific day (Wednesday) during the experiment.

Figure 9 shows the evolution of the number of active laptops at any time during the experiment (blue), as well as the number of pairwise connections between these laptops (red). It can be observed that although several laptops were sometimes running at the same time, this does not imply that all these laptops were connected over the wireless ad hoc channel. For example, on Wednesday afternoon all six laptops were up and running most of the time, but only three pairwise connections were observed. This is because, as explained earlier, the participants were not always collocated, so each participant could occasionally use his/her laptop —and possibly edit the shared document— while being disconnected from any other laptop, or while being connected with only one or two other laptops.

Table I
EDITING EVENTS AND SYNCHRONIZATION

Metrics	Values
Size of the final document	102 651 characters (36 pages)
Nb. of editing events	114 612 "ins", 6 821 "del", 104 "cut", 81 "paste"
Nb. of synchronizations upon radio contact	109
Nb. of updates transferred during radio contacts	102 561

Table II
CONTACTS AND INTER-CONTACTS

Metrics	Values (* = min / max / avg / stdev)
Duration of the experiment	8d16h28'41"
Nb of participants (laptops)	6
Number of contacts	109
Durations of contacts	3'16"/4h19'34"/1h19'53"/41'28"*
Number of inter-contacts	94
Durations of inter-contacts	5'06"/95h29'37"/24h26'51"/30h18'35"*

Statistical details about the experiment are presented in Tables I and II. They concern the editing events and the transfers of synchronization messages, and the radio contacts and inter-contacts between laptops.

The shared document produced during this experiment is 102 651-character long (36 pages, in plain text). The numbers of *ins*, *del*, *cut*, and *paste* events triggered to produce this document are detailed in Table I. Overall, 109 synchronizations occurred between pairs of laptops upon radio contact, which is consistent with the number of contacts observed (see Table II).

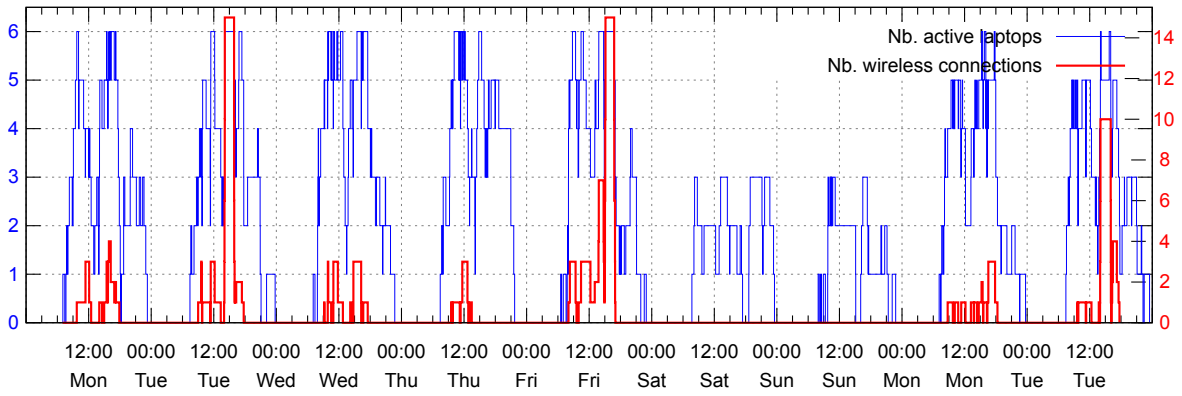

Figure 9. Number of active laptops (blue) and number of pairwise connections between laptops (red)

Each of these synchronizations involved the exchange of state vectors and delta states between two laptops, as shown in Figure 10. The state vectors (at most 700 bytes) were of course far smaller than the deltas transferred upon contacts.

While laptops were in contact, 102 561 transfers of updates were observed. These transfers of updates are actually transfers of deltas pertaining to the last operation applied on the sender, but we distinguish updates from deltas here because updates are significantly smaller. Figure 11 shows the evolution of the size of the messages carrying these updates over time. A large majority of these messages were indeed quite small (about 130 bytes), for they concerned only the insertion or the removal of a single character. Occasionally, larger messages were observed (up to 700 bytes), when a user pasted a piece of text in the shared document. Note that no attempt was made to compress the data transmitted in the messages in this early version of our editing system.

Figure 12 shows the evolution of the size of the shared document on each laptop. It can be observed that this size sometimes trailed behind significantly on some laptops (see for example laptops FL and LT on Friday), as some participants worked far from any other participant, thus preventing their laptop to synchronize with other laptops. The weekly meetings on Tuesday and Friday afternoons allowed most laptops to fully synchronize, although some participants did not attend all meetings and therefore relied on unplanned contacts to get the latest contributions of every other participant. For example, the shared document was finalized (as planned) during the last Tuesday meeting, even though one of the participants did not attend this meeting. A contact with this participant's laptop occurred shortly after this meeting, which allowed this laptop to get the final version of the document.

Outcome of this experiment

The experiment described earlier only involved six participants over a few days. With this small-scale experiment our prime motivation was to verify that collaborative editing based on opportunistic communication is indeed doable and practical. It turns out that editing a shared document in such conditions is actually a near-real-time experience, with synchronizations depending on unpredicted collocation between contributors. At the end of the experimentation period, the participants confirmed that while working on this deliverable they did not perceive opportunistic synchronization as an inconvenience.

Of course opportunistic synchronization requires that all updates eventually reach all replicas. More specifically, the synchronization graph must be connected, which is actually a major requirement for any distributed application involving CRDTs [3]. This experiment shows that this requirement can easily be met in a real-life scenario.

Scalability

Scalability is usually a typical concern in distributed applications. Yet, the question whether an experiment similar to that described above could have been performed with hundreds of participants hardly makes sense, since editing a shared document with so many contributors would probably not be practical anyway.

Yjs can however support other kinds of CRDTs (namely arrays or maps), which can serve as building blocks for a large variety of data structures. The software system used in our experiment could therefore be used to support large-scale CRDT-based collaborative applications. In order to determine if this software system would scale up, we ran additional experiments in emulation mode. In these experiments, the LEPTON platform [25] was used to simulate the mobility and opportunistic contacts of a large number of virtual nodes. For each of these virtual nodes, instead of running Quill/Yjs in a Web browser, we used node.js and replaced the real editor Quill by a dummy editor we developed in order to mimic the editing events a real user would generate over time.

With this architecture we ran scenarios involving up to 200 virtual nodes editing the same shared document concurrently, and did not observe any adverse effect on the eventual convergence of all copies of the shared document.

VI. Conclusion

In this paper, we have described the experimental setup and the results of a real-life experiment involving a group of researchers that collaboratively edited a document over nine days, relying exclusively on opportunistic communication

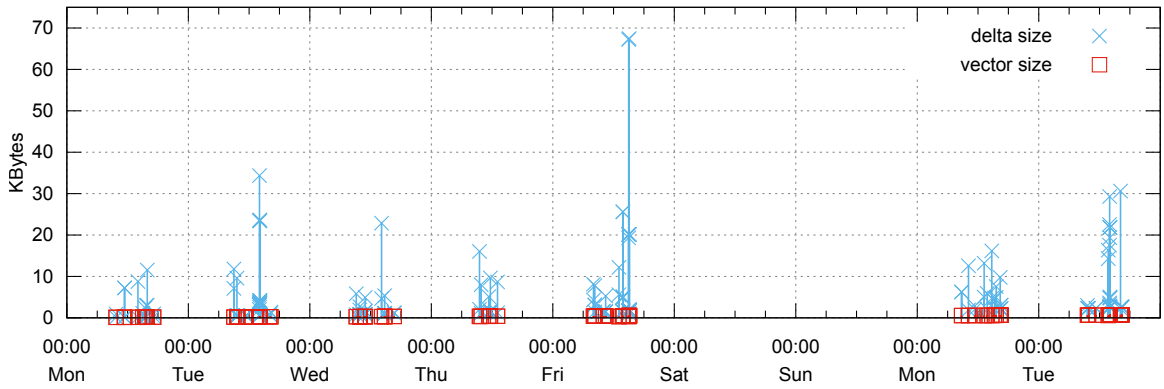

Figure 10. Evolution of the size of the messages carrying state vectors (red) and delta states (blue) transferred between neighbor laptops upon contacts

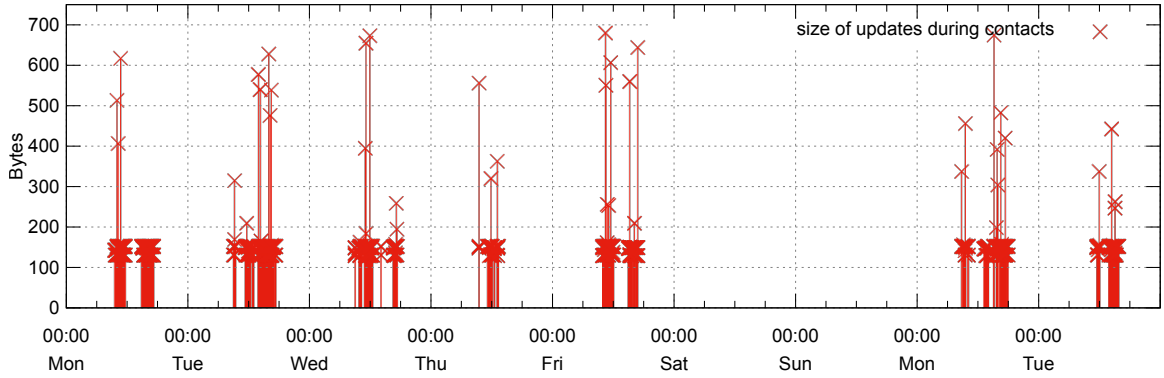

Figure 11. Evolution of the size of the messages carrying updates during contacts

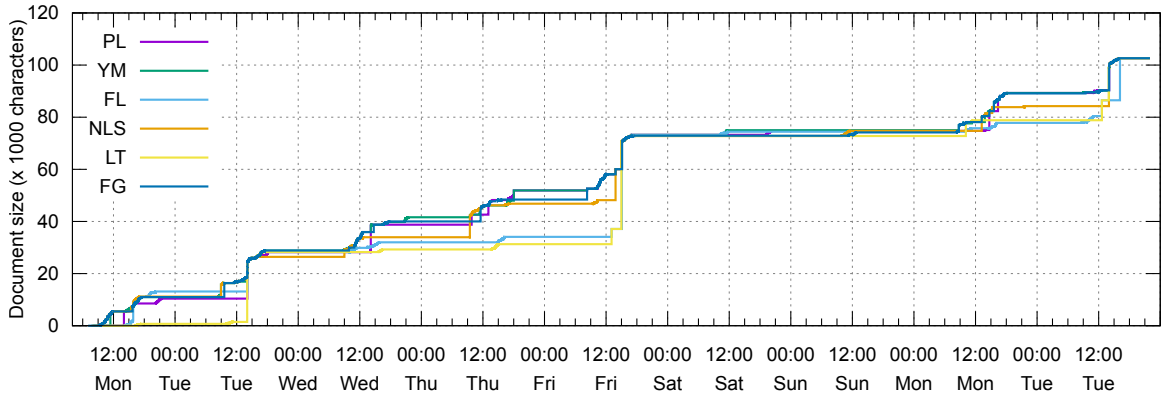

Figure 12. Evolution of the size of the shared document on each laptop

to synchronize their contributions to this document. The text editor deployed on the users' laptops relied on an off-the-shelf implementation of a text CRDT to store the different replicas of the shared document. A specific module ensured a delta-state-based synchronization of the replicas between neighbor laptops.

The analysis of the log files produced during the experiment confirmed the opportunistic nature of the network formed by the laptops, and the ability of the CRDT-based editing system to maintain the consistency of the replicas stored on each laptop.

This small-scale experiment was conducted in an academic setting, avoiding deliberately —and somehow artificially— to rely on the Internet for collaborative editing. It confirms that actual collaborative work in opportunistic networking conditions is indeed viable. Moreover, the mobility of the nodes and the users' behavior are not fundamentally different from those found in other application domains, which let us think that the usefulness of the approach is quite general. Besides, a simulation run involving the same CRDT-based editing system with 200 virtual contributors shows that scalability is not an issue (although having that many contributors edit the same

document simultaneously would probably be useless).

We believe this work paves the way for the deployment and use of distributed collaborative applications in situations where opportunistic communication would be the primary and possibly only option, in remote areas or in a disaster-relief situation for example.

REFERENCES

[1] "Message Passing Interface Forum." https://mpi-forum.org/. 2023.08.20.

[2] "The OpenMP API specification for parallel programming." https://www.openmp.org/. 2023.08.20.

[3] N. Preguiça, "Conflict-free Replicated Data Types: an Overview," 2018. Arxiv Preprint https://arxiv.org/abs/1806.10254.

[4] F. Guidec, Y. Mahéo, and C. Noûs, "Supporting conflict-free replicated data types in opportunistic networks," *Peer-to-Peer Networking and Applications*, vol. 16, pp. 395–419, 2023.

[5] S. Weiss, P. Urso, and P. Molli, "Logoot: A Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks," in *29th International Conference on Distributed Computing Systems (ICDCS'09)*, (Montreal, Canada), pp. 404–412, IEEE, June 2009.

[6] C. E. A. Robin and V. M. Romero, "DTNDocs: A delay tolerant peer-to-peer collaborative editing system," in *32nd International Conference on Information Networking (ICOIN 2018)*, (Chiang Mai, Thailand), pp. 92–97, Jan. 2018.

[7] L. André, S. Martin, G. Oster, and C.-L. Ignat, "Supporting adaptable granularity of changes for massive-scale collaborative editing," in *9th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, (Austin, TX, USA), pp. 50–59, IEEE, Oct. 2013.

[8] N. Alsulami and A. Cherif, "Collaborative Editing over Opportunistic Networks: State of the Art and Challenges," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 11, 2017.

[9] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "A Comprehensive Study of Convergent and Commutative Replicated Data Types," Tech. Rep. 7506, INRIA, Jan. 2011.

[10] P. S. Almeida, A. Shoker, and C. Baquero, "Efficient State-Based CRDTs by Delta-Mutation," in *International Conference on Networked Systems (NETYS 2015)*, (Agadir, Morrici), pp. 62–76, Springer, May 2015.

[11] A. van der Linde, J. Leitão, and N. Preguiça, "*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, Apr. 2016.

[12] P. S. Almeida, A. Shoker, and C. Baquero, "Delta state replicated data types," *Journal of Parallel and Distributed Computing*, vol. 111, pp. 162–173, 2018.

[13] A. Lindgren, A. Doria, J. Lindblom, and M. Ek, "Networking in the Land of Northern Lights: Two Years of Experiences from DTN System Deployments," in *Workshop on Wireless Networks and Systems for Developing Regions (WiNS-DR'08)*, (San Francisco, CA, USA), pp. 1–8, ACM, Sept. 2008.

[14] L. Baumgärtner, P. Gardner-Stephen, P. Graubner, J. Lakeman, J. Höchst, P. Lampe, N. Schmidt, S. Schulz, A. Sterz, and B. Freisleben, "An Experimental Evaluation of Delay-Tolerant Networking with Serval," in *Global Humanitarian Technology Conference (GHTC)*, (Seattle, WA, USA), pp. 70–79, IEEE, Oct. 2016.

[15] P. Tennent, M. Hall, B. Brown, M. Chalmers, and S. Sherwood, "Three applications for mobile epidemic algorithms," in *7th International Conference on Human Computer Interaction with Mobile Devices & services (MobileHCI05)*, (Salzburg, Austria), pp. 223–226, ACM, Sept. 2005.

[16] Z. Chen, E. A. Yavuz, and G. Karlsson, "What a juke! A collaborative music sharing system," in *13th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2012)*, (San Francisco, CA, USA), pp. 1–6, IEEE, June 2012.

[17] Y. Mahéo, N. Le Sommer, P. Launay, F. Guidec, and M. Dragone, "Beyond Opportunistic Networking Protocols: a Disruption-Tolerant Application Suite for Disconnected MANETs," in *4th Extreme Conference on Communication (ExtremeCom'12)*, (Zürich, Switzerland), pp. 1–6, ACM, Mar. 2012.

[18] F. Guidec, N. Le Sommer, and Y. Mahéo, "Opportunistic Software Deployment in Disconnected Mobile Ad Hoc Networks," *International Journal of Handheld Computing Research*, vol. 1, no. 1, pp. 24–42, 2010.

[19] T. Kärkkäinen and J. Ott, "Shared Content Editing in Opportunistic Networks," in *9th MobiCom Workshop on Challenged Networks (CHANTS'14)*, (Maui, Hawaii, USA), pp. 61–64, ACM, Sept. 2014.

[20] M. Costea, R.-I. Ciobanu, R.-C. Marin, C. Dobre, C. X. Mavromoustakis, G. Mastorakis, and F. Xhafa, "Total Order in Opportunistic Networks," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 10, 2017.

[21] M. Doering, S. Lahde, J. Morgenroth, and L. Wolf, "IBR-DTN: an efficient implementation for embedded systems," in *4th Workshop on Challenged Networks (CHANTS 2008)*, (San Francisco, CA, USA), pp. 117–120, ACM, Sept. 2008.

[22] "Yjs – A CRDT framework with a powerful abstraction of shared data." https://github.com/yjs/yjs/. 2023.08.20.

[23] "Quill Rich Text Editor." https://quilljs.com/. 2023.08.20.

[24] P. Nicolaescu, K. Jahns, M. Derntl, and R. Klamma, "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)), pp. 39–49, ACM, Nov. 2016.

[25] A. Sánchez-Carmona, F. Guidec, P. Launay, Y. Mahéo, and S. Robles, "Filling in the missing link between simulation and application in opportunistic networking," *Journal of Systems and Software*, vol. 142, pp. 57–72, Aug. 2018.