Weak consistency vs. access control: Distributed path maintenance

Erwan Le Merrer
Technicolor

Gilles Straub
Technicolor

Abstract

The proliferation of connected electronic devices, along with their possible states, rises new challenges for data maintenance in terms of efficiency, scalability and security. In the context of data replication at large, weak consistency has been adopted as a standard in uniformly trusted systems. Recently, a distributed framework has been proposed to handle untrusted systems, by adding access control at the file level. Yet, if no particular care is paid to the effective connectivity of the overlay, with respect to dynamically evolving access rights, updates of replicas may not propagate correctly anymore.

This paper proposes the design and simulation of a middleware application in charge of maintaining the connectivity of the overlay used for update propagation. We propose two techniques improving the resilience of any overlay that may be disconnected as access right policies are modified or as critical nodes crash: *(i)* the first one detects problems due to a node blocking the propagation of updates, and propagates the alert to the application at each node; *(ii)* the second one, allowed to modify the overlay, reacts to a blocking node by rewiring the overlay in order to preserve at least one path between any pairs of replicas, while conserving the structural characteristics of the overlay.

We illustrate the applicability of our middleware through simulations; they show that at a tunable overhead, overlay connectivity is maintained, despite local decisions that would have otherwise disrupted the replication service.

1 Introduction

User or application generated data is at the core of the IT revolution. Data replication problems, be they for resilience or access on multiple terminals, have generated a tremendous interest, which have become de facto a corollary of the ever increasing number of cooperating electronic devices [32]. While the replication of immutable data allows some flexibility in the way it is treated by maintenance mechanisms [31, 21], the case of mutable data maintenance at large has yet to be deepen, in order to provide efficient and scalable solutions.

While traditional commercial and well known replication protocols are running in trusted environments [8, 29], only very recent research has been dedicated to access enforcement in distributed systems where nodes can be differently trusted [28, 34]. Paper [28] addresses the specific problem of how to delegate access control to system nodes while preserving privacy. In a larger context, a recent paper [34] is the first to propose and describe a full authorization framework for weakly consistent replication without uniform trust of nodes. It then combines weak consistency models [27] with distributed access right settings [24]. The goal of this approach is to control the ability of less-trusted nodes to perform certain actions (like modifying or accessing a particular data), while preserving the eventual consistency of replicas at the network scale. This appears to be an important contribution, as in the view of the *cloud* development, data may be stored at differently trusted locations, as opposed to fully trusted environments like home or company networks. We give few practical applications for this kind of problematic:

Data replication A basic example of usage of such a framework is the need to replicate k times a personal data, in order to insure its persistence in the context of dynamic and crash-prone systems. The storage application builds a logical interconnection between nodes hosting the replicas of a given file. This structure, often referred to as an *overlay graph*, is used to propagate or not updates when a replica is modified at one of the nodes. This system could use the authorization framework presented, in order to implement access policies that have been set by the user or the application: updates originating from a replica inside the home environment is replicated, while a replica modified on a non authorized node (*e.g.* in a storage network composed of different legal entities) is discarded. The latter is here only used as a depot to reach a given replication rate k, and this depot is not meant to read or modify this particular data (as it could be compromised for example).

Distributed social networks Other applications include data management in online social networks: the forthcoming project Diaspora [10], aims to be a massively distributed architecture, where each user is in charge of its resources and contacts. The purpose of such applications is sharing with friends, outside of the control of any centralized server, based on local right settings on who should be able to access particular data.

Source code management (SCM) Another example is the distributed version control system called Mercurial [23] (used by projects like Mozilla, OpenOffice.org or Symbian OS for instance). Built totally distributed to avoid the classic central-server bottleneck, users are participating in designing a topology to propagate updates between repositories. Authentication is used in order for a repository owner to restrict read or pushing of updates to a limited set of users. As those right settings are made on a ad-hoc fashion, topology cuts can occur at the global repository connectivity; some programmers can then be excluded from code updates, without even knowing about it due to the fully distributed logic of the application. Similar distributed version control systems include Git [14] (used for the Linux kernel) and Bazaar [4]. While the authorization framework is directly applicable to grant access to hosts (nodes) themselves, SCM protocols grant rights to connections between those nodes, which can be easily emulated.

Unfortunately, as the framework proposed in [34] avoids constraints on the overlay topology, it acknowledges that bad access settings can prevent updates from propagating to all replicas. This may happen as untrusted nodes on the update path may "block" propagation, because they are not allowed to read (and then to propagate) data content. Consider for example an overlay which topology is a line; each node of this line hosts a copy of a replicated file. It is clear that if a user puts a restrictive access to the replica located at the middle of this line, a modification on the left side of the line will not be able to be propagated to the right part of the line. User then "cuts" the overlay. In other words, the existence of at least one path between any two nodes hosting the same mutable data should be maintained in order for data consistency to occur. Recent theoretical approaches [3, 16] consider the repair of a graph under certain assumptions; we are interested in providing a distributed solution for this connectivity problem, while having a small impact on the base overlay.

In weakly consistent systems, *reconciliation* mechanisms [27] are used to resolve conflicts at a node, when concurrent updates have let the replica in an inconsistent state. Those techniques cannot be applied to solve this propagation problem, as concurrent updates may never reach particular replicas due to the partition of the update overlay. This may be a particular difference with systems that can tolerate partitions due to temporary unavailabilities of hosts [30]: in our considered scenario, operations for access right management are clearly long term decisions, that may partition the overlay permanently or at least until user has detected a serious problem on the replication network.

We actually show in this paper that in addition to churn, the concurrent decisions made on access policies can disrupt the replication service; we propose practical solutions for the identified problem.

Contributions Paper [34] leaves this issue open, and thus gives no guarantees nor tries to minimize consistency problems that could occur due to arbitrary right settings. We first formulate the problem, and then reduce it to a graph connectivity issue. By adapting building blocks from the distributed computing community, we design practical solutions for this problem. As we do not aim to constrain the user or application regarding choice on those access rights, we propose in this paper to provide two levels of actions:

(i) a *proactive* mechanism, that distributedly detects cases of topology cuts when they occur (then may prevent propagation of updates), and that rises a flag that will alert the user about this problem, and

(ii) a *reactive* mechanism, where we allow the overlay to be modified, so we can propose a technique that distributedly maintains at least a correct path between any two overlay nodes, by adding edges to bypass blocking nodes.

This is achieved through a middleware, in which nodes (hosting replicas) self-organize on a distributed fashion, in order to alert or avoid bad topological configurations at run time. We thus give a generic mechanism that could be used on top of any overlay design. This approach could constitute an alternative to the systematic building of an overlay from scratch at application launch time (like a Distributed Hash Table [26] or a random graph [5]), and instead respects the initial design, making our solution suitable for a wider range of contexts.

Following the example of consistency protocols that use well known primitives as building blocks (*e.g.* distributed hash table, anti-entropy mechanisms, gossip-based membership, sloppy quorums, *etc.*, for Amazon's Dynamo [8]), we propose the joint use of distributed techniques and adapt them to design our middleware. In this light, this proposal is an additional block that could be used along with the framework proposed in [34], to add guarantees on the propagation of updates, and also leveraged by protocols that implement distributed update propagation for consistency, as *e.g.* [2, 25].

2 Operational environment

2.1 Authorization framework in a nutshell

We briefly introduce in this subsection the authorization framework for weakly consistent replication presented in [34], which is an example that motivates our work.

In a system in which all nodes mutually trust each other, policy can be enforced by any system node; the decision

made is then accepted by other nodes as the result of the policy application. However, in the addressed case of mutually distrustful nodes in a system, security has to be implemented at a finer grain. Each entity has to perform policy checks when receiving updates, as well as policy management. Proposed solution allows data access policy to be dynamically evolving: rights ("write", "read", "sync") can be added and revoked, and delegation ("own", "control") provides nodes with the right to do both. Framework logic is implemented in a distributed way: nodes with write access to the data can receive updates and commit them, while nodes with read only right can propagate that update. When a node is not entitled to read the particular data concerned by the update, this update is locally discarded and not forwarded. Framework is shown correct thanks to the Microsoft SecPAL prover.

2.2 Replication system model

In this work, we consider the representation of a set of cooperating devices, that can host replicas and access them, as an overlay in the traditional model of distributed systems. The overlay G contains N hosts at the time of observation, each one being uniquely identified (IDs with a total ordering). Each node $i \in G$ is logically connected and communicates directly with some other nodes, called its *neighbors*, V_i, for replica synchronization and right management on data. Communications are bidirectional so that G is undirected, as the framework for implementing access right management works on undirected graphs (any node can push and get updates).

Diameter of the graph, that is the longest of the shortest paths, is noted D_G. We assume that communications are synchronous and reliable, as assured by lower protocol layers; our protocol tolerates node crashes. Finally, we do not make assumption on the base overlay network, except it is initially connected when our middleware is started.

According to our problem, and without loss of generality, we consider two states for a node-file pair: *transit* nodes, through which updates of a file can propagate, and *blocking* nodes in the reverse case. Applied to the model proposed in [34], transit nodes are the one that have a R/W or RO access to that file, while blocking nodes are simply nodes that cannot modify or read that particular data.

Replication sub-network We augment the model of the already pre-existing overlay G, with another overlay constituted by a subset (possibly empty or identical) of nodes from G, along with their connections. This second overlay is noted $G\neg$. A node is removed from $G\neg$ if it is given a restrictive access (*i.e.* no access at all) on a particular file. For clarity, we consider here an overlay $G\neg$ by single replicated object, as we handle access control at the granularity of the file[1]. In other words, $G\neg$ is a subgraph of G only constituted of nodes through which updates of a particular file can propagate (transit nodes); at this level, there is thus no difference between a node that has left the overlay and a blocking node. We then have two logical levels of interests. First, the existing applicative level, that implements the security framework for data replication and consistency (all replicas and all states). The second logical level, where we operate to maintain connectivity, is then constituted only by transit nodes for a given file or group of files.

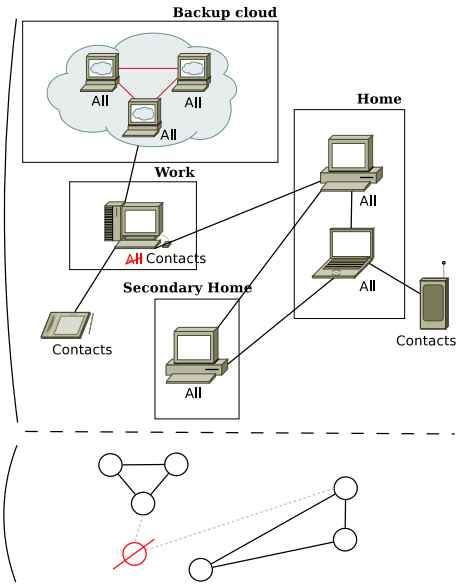


Figure 1: Read access representation of a micro overlay: the upper part is G, as defined by the user. The lower part is $G\neg$, for all other resources than contacts (*i.e.* photos): as computer at work only has access to contacts, the resulting overlay is partitioned, so updates of files cannot be propagated at all locations.

In practice, each node can monitor access rights of its neighbors (as with the access control matrix [34]); there is then no need for a global knowledge of $G\neg$ at any particular node. This logical view is instead constituted from the aggregate of the partial knowledge neighborhood of nodes, which make its maintenance totally distributed. It could be fully retrieved, if needed, using a depth/breadth first search starting at any node, and simply following iteratively neighbors V_i which are in $G\neg$ of the currently inspected node i.

[1] at a higher level, if used for collections of files as introduced in [34], nodes maintain one $G\neg$ for each label or group of files instead

2.3 Legitimate configuration of replication overlay

It is necessary to define the state we want the overlay to stick to. The network is in a *legitimate configuration* (*i.e.* in a correct and working state) for a particular file, if:

- G, the base replication overlay, is connected.
- $G\neg$, composed of all transit nodes for that file, is connected.

Connectivity is defined by the existence of at least a path between any two nodes. Note that as $G\neg$ is a subset, if G is not connected, then $G\neg$ cannot be itself. We are dealing in this paper with network partitions due explicitly to right-management. Temporary unavailabilities of devices, that can also partition the network, are naturally handled by the paradigm of eventual consistency [13].

As seen in [34], Figure 1 presents a micro example of a replication and multi-terminal access use-case. The upper part is constituted by several geographical locations, that are given Read-access rights on different groups of files. According to the authorization framework, transit nodes are all nodes if we consider the group of files named Contacts (the All group includes Contacts), so eventual consistency is possible at all places. Contrariwise, if we consider any other group of files from the All collection, we can observe in the lower part of the figure that $G\neg$ is partitioned, due to a blocking state of the computer at work location. The update flow, that may come from home, is not able to reach the cloud backup area, as the user has changed Read-access rights at work from All to Contacts only. While this configuration problem might have been easy to foresee on this small example, we argue that at a larger scale, where decision on right management can be taken at any network location with no global knowledge, or with a high number of files or group of files, is trickier or infeasible to prevent. In the rest of the paper, we will focus on a single instance of $G\neg$, that could be a single file or a label (a group of related files).

Remark here that maintaining a legitimate configuration in our context is simply targeting a general aim or need for nearly all overlay protocols: keep the overlay connected. This could be done in our context with the help of the user through alert generation in case of critical settings. It could also be done in a more automatic way by the middleware itself, that could modify the overlay to prevent partitions. The path maintenance protocol we propose is then to be leveraged by a practical update dissemination technique, in order to implement eventual consistency (see *e.g.* [2, 25]).

3 Parameters & propagation using gossip

We rely on the *gossip* paradigm for the implementation of three primitives, which are necessary to build up our solution: they are used as inputs to set our protocol parameters. This makes our solution fully distributed, as it does not rely on parameter assumptions about the overlay where it is deployed. Gossip-based techniques are used to compute the network size and diameter, and to broadcast messages. Size and diameter are computed on a periodic basis, so nodes of the system are checking for updates of those estimations. Broadcast is instead launched at demand, even if nodes have to continuously listen for incoming events, in order to propagate them. Protocols are simply sketched, but the interested reader can read details in the cited papers.

Size of the replication overlay In order to provide the estimate $\hat{N}$ at each node, protocol [18] proposes that nodes of $G\neg$ set value 1 locally with some fixed probability (other nodes start with value 0), and tag each instance with current node's ID. Every node then periodically contacts one of its neighbors at random, to swap values and average them (one average is kept for each instance). At the beginning and for each instance, there is then one node with value 1 and all the other at 0; after few rounds of this averaging protocol, values on nodes naturally converge towards $1/N$. The estimate $\hat{N}$ is then directly deduced at each node from local value.

Diameter of the replication overlay Paper [6] computes the diameter estimate $\hat{D}$, using propagation of extrema through gossip. A random real number, with a known distribution, is generated at each node for each protocol execution. A fixed size table at each node records the minimal values that have been observed so far, as nodes periodically gossip their number and table to their neighbors. After convergence, those minimal values at each node are representative of the size of the network. Along with each one of those minimal values also are attached integers. They represent the number of hops that the minimals come from; they are updated at each minimal values change (or equality), as hop count are incremented at each forwarding. Protocol then aggregate values from the whole network (again with gossips) to determine the maximum eccentricity, *i.e.* the diameter ($\hat{D} = max(hops)$).

Message broadcast Reliable broadcast protocols are presented in [12], along with their constraint/performance trade-offs for scalable dissemination. As the update overlay is probably most of the time far from a random network, and as this assumption is used by most reliable broadcast algorithms, efficient broadcast requires the

implementation of a sampling service [20], to provide random contacts.

A practical middleware framework to implement those gossip-based protocols can be found in [7]. We are now ready to present our two protocols.

4 Proactive technique: alert

We define the *alert* of a user by the fact of propagating a simple flag at each node, when the update overlay has been cut by a restrictive access at some node. At the level of applications running on nodes, this flag could be periodically checked, so that an alert to the user is raised if it is set.

The proactive protocol consists in two steps. First, critical-nodes *w.r.t.* the update overlay $G\neg$ have to be identified. Two types of criticality can be considered: *1-connectivity* and a relaxed version, *k-1-connectivity* (both defined hereafter). Each node of which state is declared as critical, and that is subject to a right access change from transit to blocking then propagates an alert flag. We now describe the building blocks of this approach; 1-connectivity is also a component of our second approach.

4.1 1-connectivity and a relaxed version

As we do not assume a particular topology for the update overlay (as it is the case in [3] for instance), and because of the complexity of emerging characteristics of real networks, our protocol periodically "learns" about current topology.

1-connectivity The assessment of 1-connectivity is the maximal criticality for a graph: the removal of just one node can lead the overlay to partition. For example, if connectivity is degrading due to node departures or restrictive access settings, detection of the 1-connectivity state of the overlay is the last stage at which proactive actions could be taken before the graph might be disconnected due to just one node removal. We consider in this paper strict 1-connectivity, as opposed to the general case used in graph theory stating that any connected graph is (at least) 1-connected.

Unfortunately, assessing 1-connectivity in large networks is computationally cumbersome. Hamid et al. [15] propose a distributed technique to test 2-connectivity (and then deduce if the graph is only 1-connected), using an initial spanning tree from a root node, and subsequent tree constructions, excluding sons, to check if more than one path for each root-node pair exists. Technique is executed in $O(\Delta \times N^2)$ steps (with Δ the maximum degree of the graph), plus time initially needed to elect a leader as the root of the tree (see *e.g.* [17]).

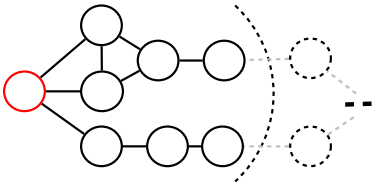


Figure 2: Example of a false positive for a 3-1-connectivity test from the leftmost node i: critical-node flag is raised, while it actually exists a cycle of length $> k$ that prevents disconnection of $G\neg$ in case of a blocking state on i.

As we do not want to restrict our approach to small scale distributed systems, we rely on the periodic computation of overlay size. Based on a predefined threshold, if the estimated size is judged large *w.r.t.* the application, this deterministic assessment of criticality is left out for a relaxed solution based on a local version of the 1-connectivity algorithm previously introduced.

The relaxed k-1-connectivity In order to avoid the need for a single node to know the full topology, and then to lower space and time complexity, a basic solution is to make each node check for connectivity problems it may cause in a limited area: each node now acts as the leader on a ball of radius k, centered at itself.

Definition 1. *For a given graph G and a base node i, k_i-1-connectivity states that removal of node i is disconnecting the subgraph of G constituted by nodes reachable at k-hops away from i.*

Such a local version simply consists in building a spanning tree of depth k (instead of spanning the full graph), rooted at the current node i, this tree thus contains all nodes reachable from i at k hops. If the removal of i causes the disconnection of some reached node from the original graph, then the network is k_i-1-connected. A "danger" flag is raised on i, as its simple removal is enough to split the update overlay. A cut-node is then basically a transit node that blocks the propagation on overlay if set to blocking state.

As every node executes the protocol, all cut-nodes are eventually identified. This is thus a direct application of [15] on small subparts of the update overlay: global $O(\Delta \times N^2)$ complexity is here transformed as N is replaced by the number of nodes reachable from i at k-hops and because no leader election is required anymore (each node thus performs actions of $O(\Delta \times \#reached\_nodes\_at\_i\_hops)$). The obvious aim being that k-1-connectivity, for a reasonable value of k, could reflect 1-connectivity for most overlays used in practice.

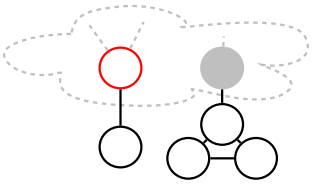


Figure 3: Example of a cut-node (left) and of a critical-node (right).

We will see that k is used for the efficiency/overhead trade-off.

This relaxation however comes at the price of possible *false positives* for the identification of k-1-connectivity at some nodes. Indeed their limited k-hops neighborhood may miss redundant paths, as there might be longer paths that are left out by localized analysis. An example of a false positive is shown on Figure 2. In other words, if the actual shortest cycle (called *girth* in literature) from node i is of length $l = 9$, then the graph looks like a tree from i within a distance of $l/2 - 1$, then here within $k = 4$.

A noticeable emerging property of this local assessment is that the joint operations of all network nodes will eventually give a correct answer at the global scale: if no k-1-connectivity problem is detected at any node, then the full graph is at least 2-connected.

In order to set a value for k, we rely on the gossip estimate of the network diameter, so k is chosen to be $3 \leq k << \hat{D}_{G\neg}$ (there are no trivial cycles on length < 3).

Critical-nodes Cut-nodes are defined in the literature as nodes strictly disconnecting the graph; this for example includes nodes at the "edge" that only disconnect nodes of degree 1 (as left node on Figure 3). As in a practical setting, nodes that become isolated (they have no more neighbors in $G\neg$) can easily notice this new state, a trivial solution is to relaunch the join process to the overlay, in order to reconnect and keep on receiving updates.

To avoid triggering an alert for such nodes of very limited criticality, we use a slightly more selective definition:

Definition 2. *A critical-node is a cut-node whose removal results in a graph with at least* 2 *disconnected components of size* > 1 *each.*

Such a cut-node is depicted as the right node on Figure 3. This distinction is of interest, as contrarily to isolated nodes, nodes in the clique disconnected by the right node cannot locally decide if they are isolated, as each node still have a degree 2 on this example. Such nodes may think that paths exist behind their direct neighbors.

4.2 Monitoring and alert propagation

Be it for 1-connectivity, or for k-1-connectivity, either a critical-node is in charge of monitoring changes in its own access policy regarding the concerned data, or this monitoring action is achieved by its neighbors. If such a node becomes a blocking node, an alert message, tagged with blocking node's ID, is broadcasted to all nodes of the overlay.

Standard broadcast should be used here, because of probably poor connectivity of the overlay. Indeed, even if probabilistic broadcast schemes exist (see *e.g.* [11]), they are likely to fail in providing the alert flag to all other nodes, as cut nodes act as propagation bottlenecks (*lpbcast* requires around $\log N$ random neighbors per node in order to achieve dissemination *w.h.p.*). In other words, reliable dissemination in a poorly connected overlay comes at the price of higher message redundancy.

At the end of the propagation phase, each node of G as its alert flag set; applications are thus aware of a cut that is preventing updates to propagate in the overlay. They can leverage this information according to their particular behavior. If further blocking nodes are declared in the network, other nodes record the received alert flags in a specific structure, thus listing current nodes causing troubles to the system.

Note that as network size and diameter estimates are made available by the gossip mechanisms in background, simple checks on the evolution of the value $\frac{\hat{N}}{\hat{D}}$ could give hints about connectivity of the overlay. If this ratio tends to 1, this means that connectivity gets poorer and poorer, as tending towards the topology of a line (the more fragile overlay, with $D = N - 1$).

4.3 Reintegration of a node in $G\neg$

Reversely, when a node state is changed from blocking to transit, the maintenance algorithm must reintegrate it in $G\neg$. This is a simple operation, as the node to reintegrate was kept in G; its direct neighbors will eventually notice that access rights on the particular data has been set to transit (due to surveillance needed in [34]), and then will put it back in $G\neg$. Same operation is executed when a new node joins the network.

The subsequent operation is to broadcast a message to remove the alert flag for that node's ID, if not a blocking node anymore.

5 Reactive technique: repair

In this second technique, we assume that the overlay can be modified by our middleware, in order to draw new connections when facing potential connectivity issues. The purpose is then to react automatically to a new blocking

Algorithm 1 Reactive rewiring of $G\neg$

1: **At each node i, periodically:**
2: i computes its k_i-1-connectivity
3: **if** i is a critical-node **then**
4: $contacts \leftarrow V_i \setminus j$ where $|V_j| = 1$
5: i sends set $contacts$ to nodes in $contacts$
6: **if** i was a critical-node, but is not one anymore **then**
7: i sends STOP to nodes in $contacts$
8: **After reception of $contacts$ from j at node i:**
9: **if** j changes to blocking state **then**
10: removes j from local view of $G\neg$
11: create an edge to node from $contacts$ whose ID is immediately following, or to lowest ID node otherwise
12: **After reception of STOP from j at node i:**
13: stop monitoring j for possible rewiring

state at a node, by getting around that node thanks to the creation of adequate connections in $G\neg$.

Our simultaneous goals for this reactive rewiring are:

- Keep $G\neg$ connected deterministically, facing successive restrictive access settings on overlay nodes.
- Minimize the number of connections created.
- Connect only close neighbors of the blocking node, to keep the impact localized.

The general purpose is then for a pseudo local action to achieve connectivity maintenance between the remaining set of nodes in $G\neg$, while having a limited impact of that maintenance regarding the original network (respect the original shape and characteristics of the topology it is applied on). As we are proposing a generic mechanism in the perspective of overlay reconnection, we are referring to a node *failure* when a node is given a blocking state.

5.1 Monitoring and reactive rewiring

This technique is also leveraging the criticality detection using k-1-connectivity. A critical-node is in charge of sending its neighbor list V_i (minus nodes of degree 1) to all its neighbors (except nodes of degree 1), and to update them on any change of V_i. Later on, if that node is set to blocking, its neighbors eventually notice this change, as implemented in the security layer of [34]; neighbors are then drawing edges to each other in order to keep $G\neg$ connected. This locality in connectivity maintenance makes sense particularly in applications where there is a certain proximity of acquaintance between close nodes in the overlay. This is the case for instance in the context of social networks, where friends of friends potentially have some proximity of interest, and then are more trusted when backup connections are created (as opposed to purely random connections).

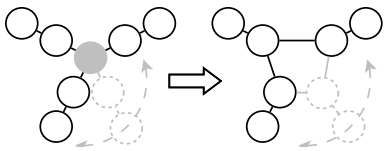
The number of disconnected components that can appear due to a blocking state at a critical-node is order of N, if no rewiring action is taken. The easiest and fastest way to achieve connectivity with certainty is for each neighbor to draw an edge to any other node in V_i; this is nevertheless costly in terms of number of created edges, specially in overlays with a relatively high average degree (this forms a clique with $|V_i|(|V_i|-1)/2$ edges, removing duplicates). Another straightforward approach would be to elect one neighbor (the one with the lowest ID for instance), and to make it responsible to handle all connections of the blocking node; this nevertheless puts all load on one single node, regardless of its capacities.

In order to balance the load, while also limiting the number of edges created, each node is asked to create an edge to the node whose ID is the closest following one in V_i (or to the lowest node ID if current node has the highest ID).

Proposition 1. *If the rewiring process executes fast enough before the removal of a neighbor of the failed node occurs (correlated failure), then Algorithm 5.1 reconnects the graph with certainty.*

Proof. By assumption, two neighbors are not failing together, *i.e.* before neighbors of the first one, i, have repaired (the synchronous model provides a bound for this repair time). All neighbors of i are then available for the rewiring process.

The smallest existing aggregate of nodes that can cause a neighbor node to declare itself as a critical-node is simply two times two nodes linked by an edge, with the critical-node in the middle. From this, one failure triggers at worst $\frac{N-1}{2}$ partitions (case of a critical-node only linking chains of 2 nodes), as depicted on the left part of the following figure.



Remember that we do not consider a single disconnected node to be a partition (a cut-node); we then do not try to reconnect single isolated nodes. The basic way of reconnecting other nodes is by creating a line between each potential partition (represented by neighbors of i). While false positives on the k-1-connectivity of a node can occur due to an insufficient k, false negatives clearly cannot be outputted; then as neighbors of failed node i all know each other (l.3,4&5 of Algorithm 5.1), a simple sort at each node on IDs suffices to identify the ID just

following current node's ID. By drawing an edge to node of following ID, nodes distributedly build the line which reconnects partitions and then nodes altogether. At this stage, the graph is reconnected. Adding one edge from the highest node's ID to the lowest one, to build a cycle, conserves this property (*e.g.* right part of previous Figure). □

It can easily be shown that after one critical-node removal, the distance between any two nodes in $G\neg$ cannot increase by more than the diameter of the cycle created for reconnection; this represents a stretch of maximum $|V_i|/2$, which is, as a function of N, at worst $N/4$ (occurring in the extreme case represented in above figure). We chose the cycle structure for clarity purpose; the stretch can be lowered at the cost of additional edge creation per node, in order to create shortcuts in that cycle (as *e.g.* for the Chord DHT), or by creating a low diameter structure between those nodes.

As a blocking node is removed from $G\neg$, $|V_i|$ edges are deleted; our heuristic adds $|V_i|$ to insure deterministic connectivity. This backup operation then does not result in an increase of the total number of edges in $G\neg$. Contrariwise, new edges drawn in $G\neg$ should be added in G, as sub-overlay $G\neg$ should be maintained as a subpart of the global interaction graph.

6 Simulation on a complex network

In this section, we illustrate the behavior of the reactive repair algorithm, through simulation on a real world network topology. As the proactive technique simply uses k-1-connectivity and a classic distributed broadcast, we focus on advantages and trade-offs of the repair technique.

A network topology is needed as a basis for illustration, in order to model the replication overlay $G\neg$. As we are not aware of a publicly available topology representing a large-scale replication system, that is not built at once from scratch (like a DHT for instance), we use an example of an ad-hoc construction of an emerging structure. The western US power grid network is an example of such a complex structure [33]; it has been studied by Watts & Strogatz and has been shown to exhibit the "small world" phenomenon, a characteristic of networks that are highly clustered, yet have small path lengths.

This network is made of 4941 nodes (generators, transformers and substations) and around 13000 edges (high-voltage transmission lines); distribution of degrees is depicted on Figure 4. Characteristic path length is 18.7 (versus 12.4 for a random graph with same characteristics). Clustering coefficient, measuring the cliquishness of a typical neighborhood is 0.08 (number of existing edges between direct neighbors, over number of possible edges).

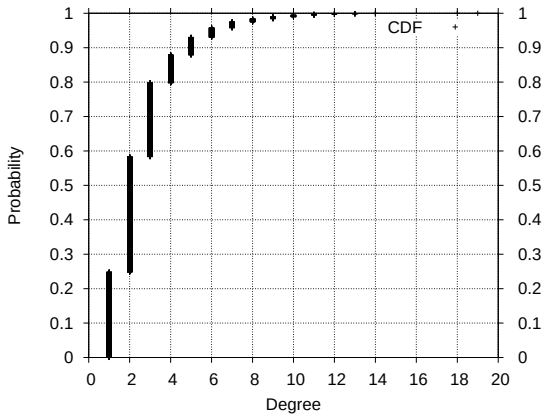


Figure 4: Degree distribution of nodes from the western US power grid.

We believe the characteristics of this network are representative of large scale dynamic and ad-hoc systems construction (see *e.g.* [1], Section II "The Topology of Real Networks"), that may emerge in large-scale applications.

As protocols used as building blocks have been extensively studied and tested in their original paper, we focus in this section on correctness and performance questions that are linked to novel operations. The Peersim simulator [19] is used to perform specific tests on the network representing $G\neg$.

6.1 False positives and k-1-connectivity

We first look at the number of false positives in the network, for a given search depth k. Each node periodically and parallely performs the k-1-connectivity test, and self declares critical if positive. Figure 5 plots the counting of such so considered critical-nodes, based on various k. As expected, the higher k, the lower the number of false positives: nodes have a wider view of network connections, so they could identify alternative paths for reliability. There is a quick drop, from $k = 3$ to 6 in that number of false positives: this indicates that the majority of cycles on this topology could be found relatively close from nodes; this is characteristic of clustered networks. The network diameter is 46; no node in this structure needs to go that deep (*i.e.* discover the full graph) to correctly assess its criticality. In that light, k could be automatically set to a fraction of the diameter, which is computed on a regular basis using gossip (presented in Section 3). We will see later that small values from k provide a good complexity/efficiency trade-off for the protocol.

It could be noticed that the number of basic cut-nodes is substantially higher than the number of critical-nodes (2307 cut-nodes for $k = 3$ for instance, vs. around 400

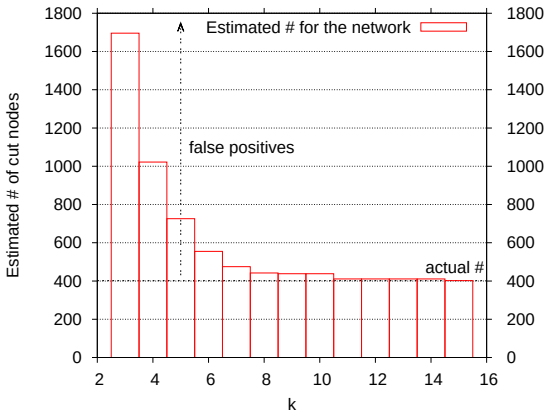


Figure 5: False positives in the computation of the criticality of nodes, given a particular search depth.

here); but again, as the assessment of an isolated state at a node is trivial, they can be left-out from our method, yielding less overhead.

Another metric is the number of nodes that would be disconnected from the largest remaining component, due to the leaving of a single critical-node. On our example network, that number ranges from 2 to 105 nodes, with an average of 4.95 and an relatively high standard deviation of 6.85. This shows that even for a static structure, built to be reliable, some non neglectable number of nodes may be isolated from the departure of just one node. Back to our initial problem, this means that a blocking setting at such a node may indeed isolate some significant number of nodes from receiving updates.

6.2 Resilience and characteristics of replication overlay

This subsection illustrates the rewiring process of the reactive algorithm. A basic evolution of the states at nodes, from transit to blocking, can be modeled by random node removal (or failure): starting from the connected overlay representing $G\neg$, randomly selected nodes are removed sequentially. At launch time, the frequency of policy evolution is expected to be way smaller than the time needed for nodes to monitor an access change on a neighbor, plus the time needed to initiate new connections for rewiring; furthermore, simultaneous failures are problematic only if they happen to direct neighbors (*i.e.* correlated in space), which has a low chance to occur in real settings. This is why we think it is reasonable to modelize failures as sequential processes[2].

[2]if judged to be a problem, or in case of a network with a very high policy evolution rate, a straightforward solution for protocol resilience is to monitor neighbors not simply at one hop, but at k-hops instead

Resilience Starting from the original network composed of 4941 nodes, Figures 6 and 7 plot size, diameter and clustering coefficient of the resulting network after successive removals. Those two latter measures are traditionally used to characterize the structure of a network; we designed our reactive method to avoid significant side effects on those characteristics. Rewiring technique is applied based on neighborhood change. Finally, those statistics are computed on the largest remaining component.

As expected, first observation is that without repair the network is disrupted quickly; a connected network would draw a line from $(x = 0, y = 4941)$ to $(x = 4941, y = 0)$, meaning that the decrease in size is only due to the randomly removed node at each step. Aggregates of nodes are progressively isolated, until a first major disconnection of more than half the size of the network, around $x = 1400$. This results on Figure 7 by a large increase of the diameter, and a peak at $x = 1400$, just before the massive disconnection. This is due to the lack of redundant paths in the network, which causes the whole topology to sparsify and weaken, and to finally split. The clustering coefficient stays constant until the break, as nodes are removed with their connections, which does not impact the local connectivity of majority of nodes in the network. The weakness of such a network under random failures motivates the need for a maintenance algorithm.

As an illustration, second implementation is our algorithm considering all cut-nodes as critical, so repairs are triggered even to reconnect nodes of degree 1. Size of the resulting network follows a straight line, showing that it remains perfectly connected. Prefered algorithm, based on critical-nodes only, is then implemented: we observe a concave curve, with no break in it. This corresponds to the maintenance of connectivity of the main component, leaving the single disconnected nodes to reconnect through the join process of the application (not shown). Our method thus ensures connectivity of the core network, reaching our first goal of deterministic connectivity, as previously analyzed, while minimizing the number of rewiring reactions on nodes to do so.

For the sake of comparison, a simpler and intuitive rewiring strategy has been implemented. It consists, when a node has detected a failed neighbor, in creating a connection to a random node in the network. Such a rewiring technique implies the use of a random sampling service (as [20]), which is in charge of returning a random node from the network when asked to. Despite probabilistic guarantees, we observe that it manages to preserve perfect connectivity in such scenario.

We now study the characteristics of the resulting network.

Characteristics Diameter and clustering coefficient statistics are presented on Figure 7. Those two metrics

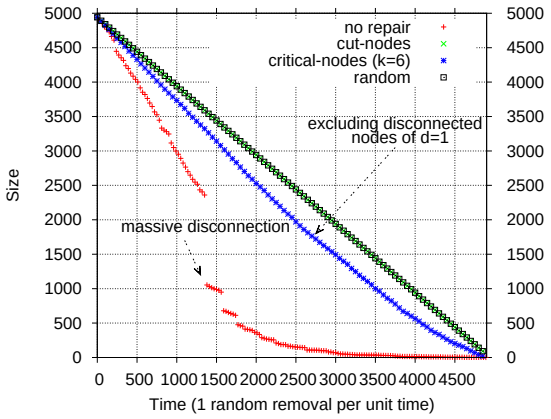


Figure 6: Size of the largest remaining component under sequential setting of blocking state at random nodes.

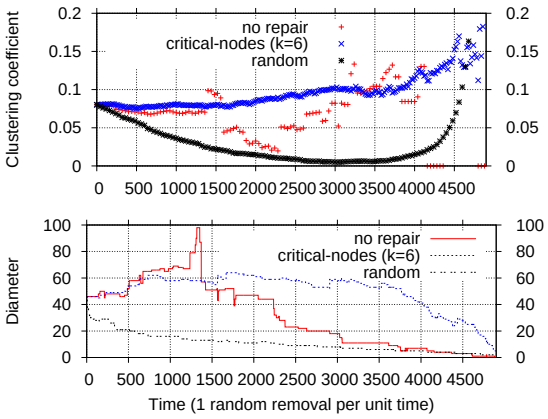


Figure 7: Diameter and clustering of the largest remaining component.

are relevant of the evolution of the considered network, and specially of the impact of the rewiring methods applied to it.

The diameter of the network with no repair implemented (solid line) keeps on increasing before the first massive disconnection, with a peak just before that event. As less and less redundant paths are available due to removals, the network gets sparse, causing this increase. After the break, diameter drops as we are now plotting the diameter of the largest remaining component, but still smaller than before the break. We remark that clustering coefficient stays constant until the break, as simple consecutive edges deletion does not modify deeply the overall clustering trend of the network.

Considering the basic random rewiring strategy, results are coherent with theoretical predictions (see *e.g.* [33]): adding connections to random nodes in the network create shortcuts that cause the diameter to rapidly drop. Clustering coefficient also diminishes very quickly, tending towards the one of a random graph as numerous rewiring actions are performed. Even if in a vast amount of applications, those properties are clear targets for overlay protocols, in our case, it is clear that the base graph is completely transformed, even after few removals (diameter is divided by 2 after 500 removal steps). Original distances from nodes in the base network, as well as local clustering with close neighbors, are then totally changed, leaving with a different type of structure than the one originally designed.

Our proposal, using a small k ($k = 6$ produces only few false positives on Figure 5), causes the initial increase of the network diameter in the first 500 removal, followed by a steady tendency until close from the end of the simulation. This is predictable, as our local rewiring technique does not on purpose create shortcuts that may reduce the diameter. Our method also causes the clustering coefficient to be stable for a long period of the simulation; this could be explained by the fact that failures are "patched" by local actions (as opposed to random rewiring), which is more respectful of the original network design. It could also be observed that characteristics are very close from "no repair" before disruption, while we ensure resilience in the long run. This confirms that our technique reaches our third requirement of locality, as global clustering and diameter tendencies are preserved.

6.3 Overhead vs optimality trade-off

We now have a look at the overhead needed to run our algorithm, in terms of number of protocol messages. This is measured per node per time unit, and depends on the base network, and on building blocks used. We are interested in the overhead generated and the number of created edges, as a function of k. Overhead per node is averaged on 2500 steps of node removal (following the previously introduced scenario), while the number of edges generated to maintain connectivity is the total number at the network scale. Overhead includes protocol messages needed to implement algorithm 5.1, that is k-1-connectivity for each node at each step[3], the sending of neighborhood tables for critical-nodes, and the number of messages needed to contact nodes to create new connections.

Figure 8 plots this trade-off; note the log-scale on y axis of the overhead curve. As k increases, the number of false positives on nodes decreases, which triggers less connection creations for the same result of keeping the network connected. At the opposite, the number of con-

[3]we use here the complexity upper bound for the computation of twice a DDFS [15], which is basically twice the number of reached edges at k hops from a given node [22])

sumed messages increases fast; this overhead is largely dominated by the k-hop neighborhood exploration around nodes. The highest number of false positives (for $k = 3$ as seen on Figure 5) causes 2518 edges to be created; for a large k, this number is nearly divided by a factor of around 2. Again, small values of k provide satisfying results regarding what can be achieved with larger values; $k = 6$ causes an average of 184 messages per node, while creating only 10% more edges than needed for $k = 15$ for instance.

In such a scenario, a basic technique that would recreate a new connection to another random node each time one of its neighbor has failed would create 7474 edges, which is substancially higher than with our method. Adding around 5 times the number of edges compared to our method is likely to importantly impact characteristics of the overall network structure. On the overhead side, results range from few tens to nearly two thousand of messages per node and per unit time. For high values, as the changes of policies can be expected to be in the order of minutes, time unit can be chosen large enough so that the total number of messages can be diluted in the general network traffic. Here again, low values for k provide a satisfying accuracy/cost trade-off.

A simple solution to divide by two the global overhead is to make nodes cooperate for periodic exploration. Our model (exposed this way for clarity) constitutes a worst case scenario for k values, as each node explores its k-hop neighborhood selfishly. Derbel et al [9], for spanning graph construction, assume that a node exploring k-hops away has in fact a $2k$ view of its neighborhood: this is trivially implementable, as reached nodes are asked to give their own k view, leading to a virtually increased exploration range. That way, our figures evolving as a function of k could be read evolving like $k/2$, yielding an easy optimization in terms of messages and latency.

To sum up, an aggressive reduction of the number of created edges comes at the price of more overhead in terms of protocol messages. In such a distributed application, small values of k still provide satisfying performances with a reasonable overhead, while preserving the integrity of the replication service.

6.4 Setting parameter k

We are not aware of a publicly released dataset capturing access right evolution on a data-management network, probably in the first place because such systems are not yet implemented at large. We have thus illustrated our reactive method by modeling a complex network using successive blocking decisions on random nodes.

One challenging question is how to automatically set k at each node in order to efficiently identify cycles when they exist, without generating an important overhead due to a high value. As it basically depends on the current topology of the replication overlay $G\neg$, one pragmatic solution is to set k at an initial low value (4 to 6 for instance) at each node. Any node that cannot find a cycle will increment k until it reaches a maximum value, judged reasonably large for any scalable distributed system. This value is typically around $\ln N$ in most large scale systems, or simply a fraction of the network's diameter (both N and D are computed on a regular basis through gossip). This heuristic prevents to set an uniform value for k, that would be too large for nodes in some well connected part of the replication overlay, and inefficient in some other.

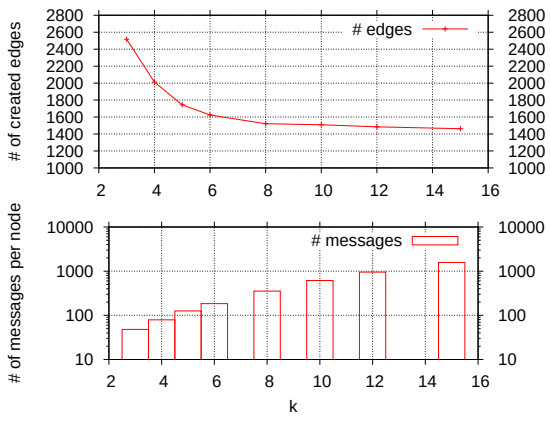


Figure 8: Trade-off between overhead (number of protocol messages) and number of created edges.

7 Related work

On the consistency maintenance side, reconciliation techniques can be used when two different replicas see non ordered modifications to apply, in log-based systems [27]. Unfortunately, such techniques cannot be applied in the described context, as concurrent modifications propagated on the system are not able to reach each other due to the partition of the update overlay.

General approach, regarding topology repair, is to simply minimize the probability of partition, instead of reacting when the event may occur. This is for example done with the building from scratch of resilient topologies, that have a low probability of being partitioned due to churn. Such design of resilient overlay networks has received a tremendous interest in past decade, ranging from fully structured [26] to random networks [5]. This nevertheless imposes that each node participates equally to the building of the overlay, regardless of its original capacities; it also requires nodes to connect to arbitrary nodes, given by the protocol. Finally, only simple connectivity is taken into account, while we also deal with connectivity

at the right management level in our work.

We instead prefer in our context a reactive method, because it respects the original topology, that is free to be designed following any application specific needs, or simply emerging from ad-hoc connection decisions. A recent work is also targeting a reactive repair mechanism: Bansal & Mittal [3] are proposing an operation that leaves the network connected at all times, with limited actions on the topology. Unfortunately this technique applies to a binary tree structure of all processes in the system, and does not tolerate node crashes (fail-stop model), which is restrictive in practice.

To the best of our knowledge, the only reactive maintenance technique which can be applied to any topology is presented in paper [16], under an adversarial node removal scenario. Technique used is to react to each failure by reconnecting neighborhood of the failed node; each failure pushes the original graph to get closer to the structure of a half-full tree (that is a particular sort of binary tree). As we consider potential partitions to be the exception and not the rule, we do not trigger the convergence of the base graph towards a particular structure; doing so would imply connecting nodes together that were not directly concerned by failure of a neighbor. Furthermore, we have seen that systematic reconnections on failure trigger more creation of edges; this in turn denaturates the base network, specially in the case of highly clustered networks where alternative paths have a high chance to exist.

8 Conclusion

In today's context of increasing and massive interconnexion of devices, we believe that the study of automated applications for system maintenance are of critical interest. We presented the design of such a middleware that is meant to increase the resilience of systems supporting weakly consistent replication, where access rights are arbitrarily granted by users or applications. We reduced the practical problem of path maintenance in replication networks to a generic problem of graph connectivity, and showed that simple distributed algorithms can preserve connectivity at a reasonable cost, while conserving network characteristics. We hope this contribution will be leveraged by the forthcoming applications implementing reliable storage at large in non uniformly trusted systems.

References

[1] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74(1):47–97, January 2002.

[2] Roberto Baldoni, Rachid Guerraoui, Ron R. Levy, Vivien Quéma, and Sara Tucci Piergiovanni. Unconscious eventual consistency with gossips. In *In Proceedings of the Eight International Symposium on Stabilization, Safety and Security of Distributed Systems (SSS 2006)*. Springer, 2006.

[3] T. Bansal and N. Mittal. A scalable algorithm for maintaining perpetual system connectivity in dynamic distributed systems. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1 –12, 19-23 2010.

[4] http://bazaar.canonical.com/.

[5] Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer. Brahms: Byzantine resilient random membership sampling. *Computer Networks Journal (COMNET), Special Issue on Gossiping in Distributed Systems*, April 2009.

[6] Jorge C. S. Cardoso, Carlos Baquero, and Paulo Sergio Almeida. Probabilistic estimation of network size and diameter. *Dependable Computing, Latin-American Symposium on*, 0:33–40, 2009.

[7] Michael Chow and Robbert van Renesse. A middleware for gossip protocols. In *International Workshop on Peer-to-Peer Systems (IPTPS 2010)*, San Jose, CA, April 2010.

[8] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, 2007.

[9] Bilel Derbel, Cyril Gavoille, David Peleg, and Laurent Viennot. On the locality of distributed sparse spanner construction. In *PODC '08: Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 273–282, New York, NY, USA, 2008. ACM.

[10] http://www.joindiaspora.com/.

[11] Patrick Th. Eugster, Rachid Guerraoui, S. B. Handurukande, Petr Kouznetsov, and Anne-Marie Kermarrec. Lightweight probabilistic broadcast. In *DSN '01: Proceedings of the 2001 International Conference on Dependable Systems and Networks (formerly: FTCS)*, pages 443–452, Washington, DC, USA, 2001. IEEE Computer Society.

[12] Patrick Th. Eugster, Rachid Guerraoui, and Petr Kouznetsov. Delta-reliable broadcast: A probabilistic measure of broadcast reliability. In *In Proceedings of the 24th IEEE International Conference*

on Distributed Computing Systems (ICDCS 2004, pages 24–26, 2004.

[13] Alan Fekete, David Gupta, Victor Luchangco, Nancy Lynch, and Alex Shvartsman. Eventually-serializable data services. In *PODC '96: Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, pages 300–309, New York, NY, USA, 1996. ACM.

[14] http://git-scm.com/.

[15] Brahim Hamid, Le Saëc, and Mohamed Mosbah. Distributed 2-vertex connectivity test of graphs using local knowledge. In *Proceeding of the International Conference on Parallel and Distributed Computing Systems*, pages 71–76, 2007.

[16] Thomas P. Hayes, Jared Saia, and Amitabh Trehan. The forgiving graph: a distributed data structure for low stretch under adversarial attack. In *PODC '09: Proceedings of the 28th ACM symposium on Principles of distributed computing*, pages 121–130, New York, NY, USA, 2009. ACM.

[17] Rebecca Ingram, Patrick Shields, Jennifer E. Walter, and Jennifer L. Welch. An asynchronous leader election algorithm for dynamic networks. *Parallel and Distributed Processing Symposium, International*, 0:1–12, 2009.

[18] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(3):219–252, 2005.

[19] Márk Jelasity, Alberto Montresor, Gian Paolo Jesi, and Spyros Voulgaris. The Peersim simulator, 2004. `http://peersim.sf.net`.

[20] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25(3):8, 2007.

[21] Fangming Liu, Ye Sun, Bo Li, Baochun Li, and Xinyan Zhang. Fs2you: Peer-assisted semipersistent online hosting at a large scale. *Parallel and Distributed Systems, IEEE Transactions on*, 21(10):1442 –1457, oct. 2010.

[22] S. A. M. Makki and George Havas. Distributed algorithms for depth-first search. *Information Processing Letters*, 60(1):7 – 12, 1996.

[23] http://mercurial.selenic.com/.

[24] Stefan Miltchev, Jonathan M. Smith, Vassilis Prevelakis, Angelos Keromytis, and Sotiris Ioannidis. Decentralized access control in distributed file systems. *ACM Comput. Surv.*, 40(3):1–30, 2008.

[25] Karin Petersen, Mike J. Spreitzer, Douglas B. Terry, Marvin M. Theimer, and Alan J. Demers. Flexible update propagation for weakly consistent replication. In *SOSP '97: Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 288–301, New York, NY, USA, 1997. ACM.

[26] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001.

[27] Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, 2005.

[28] M. Sanchez-Artigas. Distributed access enforcement in p2p networks: When privacy comes into play. In *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, pages 1–10, aug. 2010.

[29] Michael L. Scott and Larry L. Peterson, editors. *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003*. ACM, 2003.

[30] Atul Singh, Pedro Fonseca, Petr Kuznetsov, Rodrigo Rodrigues, and Petros Maniatis. Zeno: eventually consistent byzantine-fault tolerance. In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 169–184, Berkeley, CA, USA, 2009.

[31] L. Toka, M. Dell'Amico, and P. Michiardi. Online data backup: A peer-assisted approach. In *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, pages 1 –10, aug. 2010.

[32] Werner Vogels. Eventually consistent. *Queue*, 6(6):14–19, 2008.

[33] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of s̀mall-worldńetworks. *Nature*, 393:440–442, 1998.

[34] Ted Wobber, Thomas L. Rodeheffer, and Douglas B. Terry. Policy-based access control for weakly consistent replication. In *EuroSys '10: Proceedings of the 5th European conference on Computer systems*, pages 293–306, New York, NY, USA, 2010. ACM.